**POLYTECHNIQUE MONTRÉAL**
affiliée à l'Université de Montréal

# Communication, Coordination and Organization of Practical Robot Swarms

**VIVEK SHANKAR VARADHARAJAN**
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Mai 2022

**POLYTECHNIQUE MONTRÉAL**
affiliée à l'Université de Montréal

Cette thèse intitulée :

**Communication, Coordination and Organization of Practical Robot Swarms**

présentée par **Vivek Shankar VARADHARAJAN**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Marios-Eleftherios **FOKAEFS**, président
Giovanni **BELTRAME**, membre et directeur de recherche
Bram **ADAMS**, membre et codirecteur de recherche
Jérôme **LE NY**, membre
Heiko **HAMANN**, membre externe

## DEDICATION

*This thesis is dedicated to my wife Lalli, daughter Thanvi and son Lashank; they sacrificed the most during my research endeavors.*

## ACKNOWLEDGEMENTS

I would like to start by thanking my supervisor Prof. Giovanni Beltrame, who has provided his complete support and guidance in transforming me from a student into a researcher. I joined MIST Lab in 2017 as a naive student with the only intention of advancing science. Giovanni provided me with the freedom to pick up my interests in swarm robotics and guided me all along the way. Sometimes his swim or sink approach was a bit scary, but that allowed me to navigate my research and become an independent researcher. One other exciting thing to mention about Giovanni is that he always claims the most challenging tasks to be easy. On several occasions, this served as the perfect motivation that I needed. Whenever I came up with an idea to solve a problem, Giovanni has always encouraged me to pursue it. Giovanni has always been available to me; his average response time to my queries is about 5-10 Minutes, except when he goes on his trip into the forest. Apart from supporting my research, Giovanni has helped me with many of my endeavors in life outside my study. He is an outstanding teacher, researcher, and supervisor who has taught me innumerable things. I have picked up many good characteristics from him.

I extend my gratitude to my co-supervisor, Prof. Bram Adams, who has always supported me with my research endeavors, particularly during the initial stages of my research.

Prof. David St-Onge is another important person who has supported me immensely during my research. Many of my field robotic experience and skills started by working with David. He has always been available to me, particularly in the late evenings, and was a great reference to me on many robotics topics. David and I have spent many white nights together, working on research in the lab or online. Two other notable lab members that have supported me with my research are Soma Kartick and Sepand Motaghed. They were both extraordinary and hardworking people who have immensely helped me with my research.

Finally, I would like to thank all my lab mates for their support during my research, to name a few (randomly ordered), Ulrich Dah-Achinanon, Pierre-Yves Lajoie, Yanjun Cao, Marcel Kaufmann, Fang Wu, Rafael Braga, Hassan Ali, Seyed Ehsan Marjani Bajestani, Yann Bouteiller, Benjamin Ramtoula, Jacopo Panerati and Du Wenqiang. Many of the above members were there for me when I wanted to share my problems with someone and ask their opinions. Overall, the members of the MIST lab have made my research time a memorable and fruitful experience that I will cherish for the rest of my life.

## RÉSUMÉ

Depuis quelques années, les systèmes multi-robots gagnent rapidement en popularité dans une foule d'applications, telles que les drones de livraison d'Amazon et les flottes de voitures autonomes de Waymo. Ces systèmes doivent être déployés pendant des périodes prolongées avec un minimum de temps morts pour augmenter rentabilité. Le paradigme le plus établi pour la robotique dans l'industrie est la centralisation, où un seul ordinateur contrôle l'ensemble du système. Cependant, dans ces systèmes une simple faute, ou déconnexion, de l'ordinateur central peut conduire à des défaillances catastrophiques comme des drones tombant du ciel sur des passants. Ces défaillances soulignent l'importance des méthodes décentralisées pour les systèmes multi-robots qui répartissent la collecte de données et la prise de décision entre les robots. Malheureusement, l'utilisation de systèmes décentralisés peut conduire à un manque de connaissance de la situation globale au sein des robots individuels, en raison de la nature locale et dispersée des données. L'absence de méthodes appropriées pour transmettre les objectifs globaux aux robots entrave leur utilisation dans les applications réalistes, ce qui est le but ultime de ce travail.

Les défis entourant le manque de connaissances globales dans les systèmes multi-robots distribués proviennent de l'absence d'outils de gestion de données appropriés pour stocker et gérer efficacement les données, ainsi que du besoin de mécanismes de consensus robustes. Les méthodes actuelles de stockage et de gestion des données dans les essaims de robots sont limitées à un seul état. En revanche, la quantité de données produites par les robots devient de plus en plus grande et importante au fur et à mesure de l'évolution des technologies de capteur (i.e., la densité de pixels qui augmente sans cesse dans les caméras). Les mécanismes actuels de partage des données disponibles dans les systèmes distribués et les réseaux pair à pair ne s'appliquent pas directement aux essaims de robots en raison de leur grande mobilité et de leurs ressources de calcul limitées. Dans ce travail, nous proposons donc SOUL, un mécanisme de partage de données pour le stockage et la gestion de gros blocs de données au sein de grands essaims de robots. SOUL crée un pool de mémoire commun à partir de la mémoire inutilisée des robots de l'essaim. Ce pool de mémoire commun est disponible pour les robots qui ont besoin de stockage supplémentaire. Nous montrons que SOUL n'a qu'un impact minimal sur la mobilité et qu'il s'adapte bien aux essaims de milliers de robots en simulation. Nous démontrons également l'utilisation de SOUL dans plusieurs applications pratiques sur un essaim de 10 robots physiques.

La communication est un autre aspect essentiel de l'amélioration de l'aspect pratique des essaims de robots. Les robots doivent maintenir un lien de communication afin d'obtenir une connaissance situationnelle de l'environment et de pouvoir se coordonner. Les méthodes permettant de maintenir la connectivité dans un système multi-robots sont largement étudiées. Les principaux défis des méthodes existantes incluent des exigences élevées en matière de débit de communication et de calcul, ainsi que le manque de mécanismes robustes pour faire face aux défaillances. Les robots couramment utilisés en dans les essaims sont simples, avec des capacités limitées de communication et de calcul, ainsi que des capteurs peu précis. Des approches qui tiennent compte de ces exigences et de ces limites sont nécessaires pour assurer une communication et une coordination solides entre les robots. C'est pourquoi nous proposons Swarm Relays, un système léger et entièrement décentralisé qui permet de maintenir un essaim connecté même en présence de défaillances. Les relais Swarm utilisent un planificateur pour répondre aux exigences de mobilité des robots et des mécanismes réactifs pour appliquer une contrainte de connectivité locale sur la mobilité. En pratique, l'utilisateur spécifie la structure de communication requise, qui se traduit par des contraintes locales de mobilité inter-robots. Les défaillances des robots sont gérées à l'aide d'un mécanisme d'auto-réparation qui répare de manière autonome les connexions rompues dès qu'elles sont découvertes. Les recherches expérimentales indiquent que l'approche a une complexité temporelle sous-linéaire selon le nombre de robots. Les propriétés d'auto-réparation peuvent aussi résister jusqu'à 80 % des robots en panne. Nous validons également l'approche pour des applications pratiques par des expériences matérielles sur un large éventail de robots.

La coordination entre les robots d'un essaim est également essentielle pour les applications pratiques. Les méthodes actuelles de coordination décentralisée sont limitées à des comportements réactifs qui répondent aux mesures des capteurs et aux positions des voisins. Les méthodes robustes qui permettent de spécifier les tâches et de coordonner l'essaim en fonction de ces exigences sont pratiquement inexistantes. Le contrôle hiérarchique démontré dans la nature peut être appliqué aux essaims de robots pour permettre la spécification globale des tâches. Nous concevons une méthode hiérarchique qui divise l'essaim en plusieurs groupes: les *guides* qui peuvent effectuer des tâches globales et les *travailleurs*, des robots plus simples qui réagissent uniquement aux stimuli. Un grand nombre d'ouvriers agissent comme les muscles du système, répondant aux stimuli créés par un petit nombre de guides, agissant comme le le cerveau du système. Nos évaluations expérimentales indiquent que le système hiérarchique proposé restaure un taux de réussite de 100%, ce qui est nettement supérieur aux essaims égalitaires.

Nous appliquons cette approche hiérarchique à un essaim de particules intelligentes et étudions les propriétés de ce système hiérarchique auto-organisé en ajoutant la capacité des guides à changer les règles opérationnelles des travailleurs, montrant que 32 guides peuvent contrôler jusqu'à 1000 travailleurs.

Comme application cible pour les travailleurs, nous concevons une approche collaborative de transport qui fonctionne sur des objets de forme arbitraire. Notre algorithme de transport collaboratif effectue une étreinte séquentielle de l'objet à transporter et permet aux robots de déterminer la force à appliquer localement selon la forme. Nous étudions l'évolutivité de l'approche en simulation et la vérifions avec un petit essaim de robots réels.

Enfin, le déploiement pratique de comportements collectifs nécessite la maintenance de logiciels sur un grand nombre de robots individuels. Le déploiement de logiciels peut devenir difficile si les mises à jour doivent être effectuées pendant que les robots effectuent leurs tâches. Les approches des réseaux de capteurs sans fil ne peuvent pas prendre en compte la mobilité et la nature collective du script comportemental, ce qui exige de nouvelles méthodes. Alors, nous proposons et évaluons expérimentalement un mécanisme de mise à jour “Over-The-Air” pour les essaims de robots qui est distribué, robuste à la mobilité et aux défaillances. Les résultats, en simulation et sur du matériel physique, indiquent que l'approche s'adapte bien et peut s'appliquer à des essaims pratiques.

## ABSTRACT

Multi-robot systems are becoming more and more pervasive: clear examples are Amazon delivery drones and Waymo self-driving car fleets. These systems need to be deployed for extended periods with minimal downtime to increase profitability. The most established paradigm for robotics in industry is centralized, where a single computer controls the whole system. However, centralized systems have a single point of failure, which can lead to catastrophic failures like drones dropping from the sky on people. These failures stress the importance of decentralized methods for multi-robot systems, which distribute the data collection and decision-making among the robots. Unfortunately, the use of current state-of-the-art decentralized systems can lead to a lack of situational awareness within the robots, due to the local and scattered nature of the information. Communication issues can make coordination challenging, and the lack of proper methods to convey global goals to robots hinders their use in real-world applications, which is the ultimate goal of this work.

The challenges surrounding the lack of situational awareness in distributed multi-robot systems stem from the lack of proper data management tools to store and manage data effectively, as well as robust consensus mechanisms. Common data-sharing mechanisms available in distributed systems and peer-to-peer networks do not apply directly to robot swarms due to their high mobility and limited computational resources. In this work we propose SOUL, a data-sharing mechanism for storing and managing large data blobs among large robot swarms. SOUL creates a common pool of memory from the unused memory of robots in the swarm. This common memory pool is available to robots that require additional storage. We show that SOUL has minimal impact on mobility and scales well to swarms of thousands of robots in simulations and also demonstrate the use of SOUL in several practical applications on a swarm of 10 physical robots.

Communication is another vital aspect of improving the practicality of robot swarms. Robots need to maintain a communication link to attain situational awareness and be able to coordinate. Methods to maintain connectivity in a multi-robot system are widely studied: the main challenges of existing methods include high communication requirements, significant computational footprint, and a lack of robust mechanisms to tackle failures. Robots commonly used in swarm robotics are simple, with limited communication and computational capacity, and with noisy sensors. We propose Swarm Relays, a fully decentralized, lightweight system to maintain a swarm connected even in the presence of failures. Swarm relays uses a path planner to address the mobility requirements of the robots and reactive mechanisms to enforce a local connectivity constraint on mobility. In practice, the user specifies the required communication structure, which translates to local inter-robot constraints on mobility. Robot failures are handled using a self-healing mechanism that autonomously fixes broken connections as soon as they are discovered. Experimental investigations indicate that the approach has a sub-linear increase in time with the increase in the number of robots, and that self-healing properties can withstand to 80% of the robots failing. We also validate the approach for practical applications with hardware experiments on a wide range of robots.

The coordination among robots of a swarm is also critical for practical applications: current decentralized coordination methods are limited to reactive behaviors that respond to sensor measurements and neighbor positions. Robust methods which allow both task specification and coordinating the swarm using these requirements are virtually non-existent. Hierarchical control demonstrated in nature can be applied to robot swarms to allow global task specification. We design a hierarchical method that partitions the swarm into *guides* that can encode global tasks and *workers*, simpler robots that only react to stimuli. Large numbers of workers act as the system's muscles, responding to stimuli created by a small number of guides acting as the brain of the system. Our experimental evaluations indicate that the proposed hierarchical swarm restores a 100% success rate which is quickly lost to an egalitarian swarm as the complexity of its task increases. We apply this hierarchical approach to a *smart particle swarm* and study the properties of this self-organized hierarchical system, adding the ability for the guides to change the operational rules of the workers, showing that 32 guides can control up to 1000 workers.

As a target application for the workers, we design a collaborative transport approach which works on arbitrarily shaped objects. Our collaborative transport algorithm performs a sequential enclosure of an object to be transported and allows the robots to determine the force applied to the object locally. We study the scalability of the approach in simulation and verify it with a small swarm of real robots.

Finally, the practical deployment of collective behaviors requires maintaining software on a large number of individual robots. The challenges in deploying software can become daunting if updates have to be performed while robots are performing their tasks. We propose and experimentally evaluate an Over-The-Air update mechanism for robot swarms that is distributed, robust to mobility and to failures. Results, in simulation and on physical hardware, indicate that the approach scales well and can apply to practical swarms.

# TABLE OF CONTENTS

DEDICATION . . . iii

ACKNOWLEDGEMENTS . . . iv

RÉSUMÉ . . . v

ABSTRACT . . . viii

TABLE OF CONTENTS . . . x

LIST OF TABLES . . . xiii

LIST OF FIGURES . . . xiv

LIST OF ABBREVIATIONS . . . xix

CHAPTER 1 INTRODUCTION . . . 1
- 1.1 Context and Motivation . . . 1
  - 1.1.1 Situational awareness . . . 3
  - 1.1.2 Maintaining communication . . . 3
  - 1.1.3 Coordination . . . 4
- 1.2 Problem Statement . . . 8
- 1.3 Research Objectives . . . 9
- 1.4 Research Contributions . . . 9
- 1.5 Research Impact . . . 11

CHAPTER 2 LITERATURE REVIEW . . . 14
- 2.1 Robot swarms . . . 14
- 2.2 Distributed data storage . . . 17
- 2.3 Connectivity preservation . . . 18
- 2.4 Behavioral design for robot swarms . . . 19

CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION . . . 21
- 3.1 Methodology . . . 21
- 3.2 Document Structure . . . 22

CHAPTER 4 ARTICLE 1: SOUL: DATA SHARING FOR ROBOT SWARMS . . . 24
4.1 Introduction . . . 25
4.2 SOUL model . . . 26
4.2.1 Objectives . . . 27
4.2.2 Reconstruction cost . . . 29
4.3 SOUL implementation . . . 31
4.3.1 SOUL Processes . . . 32
4.3.2 Buzz and SOUL . . . 36
4.3.3 Implementation specifics . . . 36
4.4 Simulation Experiments . . . 38
4.4.1 Scalability analysis . . . 40
4.4.2 Dynamic Analysis . . . 42
4.4.3 Bandwidth consumption . . . 43
4.5 Experiments . . . 44
4.5.1 SOUL backup . . . 44
4.5.2 YOLO Search . . . 46
4.6 Related work . . . 50
4.7 Conclusions . . . 53

CHAPTER 5 ARTICLE 2: SWARM RELAYS: DISTRIBUTED SELF-HEALING GROUND-AND-AIR CONNECTIVITY CHAINS . . . 55
5.1 Introduction . . . 56
5.2 Related work . . . 57
5.3 Preliminaries . . . 59
5.3.1 Communication model . . . 59
5.3.2 Local connectivity preservation . . . 60
5.3.3 Global Connectivity Chains . . . 63
5.3.4 Task allocation problem . . . 63
5.4 Proposed Method . . . 65
5.4.1 Motion Control . . . 67
5.4.2 Self-Healing . . . 68
5.5 Experimental Evaluation . . . 69
5.6 Conclusions . . . 72

CHAPTER 6 ARTICLE 3: HIERARCHICAL CONTROL OF SMART PARTICLE SWARMS . . . 74
6.1 Introduction . . . 75

6.2 Related work . . . . . . . . . . 77
6.3 Model and System . . . . . . . . . . 79
6.4 Methodology . . . . . . . . . . 81
6.4.1 Worker Swarm . . . . . . . . . . 81
6.4.2 Guide Swarm . . . . . . . . . . 83
6.4.3 Theoretical Analysis . . . . . . . . . . 88
6.5 Experiments . . . . . . . . . . 91
6.6 Conclusions . . . . . . . . . . 94

CHAPTER 7 ARTICLE 4: COLLECTIVE TRANSPORT VIA SEQUENTIAL CAGING 96
7.1 Introduction . . . . . . . . . . 97
7.2 Related Work . . . . . . . . . . 98
7.3 Methodology . . . . . . . . . . 99
7.3.1 Problem Formulation . . . . . . . . . . 100
7.3.2 Task Allocation based Caging . . . . . . . . . . 101
7.3.3 Behaviors for pushing and rotating . . . . . . . . . . 102
7.4 Experiments . . . . . . . . . . 106
7.5 Conclusions . . . . . . . . . . 110

CHAPTER 8 ARTICLE 5: OVER-THE-AIR UPDATES FOR ROBOTIC SWARMS 111
8.1 Introduction . . . . . . . . . . 112
8.2 Update Strategy . . . . . . . . . . 113
8.3 Validation . . . . . . . . . . 114
8.3.1 Field Experiments . . . . . . . . . . 117
8.4 Conclusion . . . . . . . . . . 117

CHAPTER 9 GENERAL DISCUSSION . . . . . . . . . . 119
9.1 Data sharing in robot swarms . . . . . . . . . . 119
9.2 Connectivity preservation . . . . . . . . . . 120
9.3 Hierarchical swarms . . . . . . . . . . 121
9.4 Collaborative transport . . . . . . . . . . 122
9.5 OTA behavioral updates . . . . . . . . . . 123

CHAPTER 10 CONCLUSIONS . . . . . . . . . . 125
10.1 Future Work . . . . . . . . . . 126

REFERENCES . . . . . . . . . . 128

## LIST OF TABLES

Table 4.1 Variable definitions. . . . . . . . . . . . . . . . . . . . . . . . . . . 27
Table 4.2 Blob storage container data structure . . . . . . . . . . . . . . . . . . . . 37
Table 7.1 Experimental parameters . . . . . . . . . . . . . . . . . . . . . . . . . 106

# LIST OF FIGURES

Figure 1.1 Some examples of natural swarms are a flock of birds, colony of bees, schools of fish and swarms of ants. Credits: Bee colony CC 2.0 - flickr.com/Sy, Fish school CC 2.0 - iStock.com/armiblue, Army ants CC 2.0 - flickr.com/Axel Rouvin, Ant raft CC 4.0 - wikimedia.org/TheCoz and Starling swarm CC 2.0 - wikimedia.org/Walter Baxter . . . . . . 2

Figure 1.2 Hierarchies in robot swarms. (A) primitive robots on the lower level perform collective behavior providing information and exploiting the distilled information coming from the higher levels, producing a synergistic swarm operation as a whole. (B) homeostasis is one of the tasks performed by the hypothalamus inside the brain: maintaining an optimal body temperature by sending stimulus to muscles, sweat glands and blood vessels. (C) The hierarchy represented by the organs in human biology, with the brain as a whole performing a complex task (maintaining the well-being of the individual), composed of a variety of components performing different sub-tasks, and down to a microscopic scale these components are made of more or less self-similar neurons. (D) a range of commercial-off-the-shelf robots that can be composed into a single swarm with hierarchies. (E) capabilities (processing capacity, memory, traversability clearance, range) for the ground robots shown in (D). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

Figure 4.1 Illustration of three consecutive blob injections on a network of three robots. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

Figure 4.2 The four processes and twelve message types involved in the SOUL mechanism. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

Figure 4.3 Scalability analysis with 10, 100 and 1000 robots. . . . . . . . . . . . 39

Figure 4.4 Datagram distribution in a 100 robot network, as a result of continuous injection of blobs. . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

Figure 4.5 Packet drop rate's influence on the time (control steps) required to allocate datagrams for various density of moving robots (markers are with slight offset for visual clarity). . . . . . . . . . . . . . . . . . . . 41

Figure 4.6 Average bandwidth consumption: left with SOUL, right as a result of one-to-one transmission. . . . . . . . . . . . . . . . . . . . . . . . . . 43

Figure 4.7 Schematic representation of the two experiments conducted on the Khepera robots: SOUL backup to the left and YOLO Search to the right. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
Figure 4.8 Nine wheeled robots patrolling around a box. The LED patterns show their ID. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
Figure 4.9 Storage usage throughout one trial for the SOUL experiment. . . . . 48
Figure 4.10 Evolution of the blob distribution over the swarm on the second trial of the YOLO Search experiment. . . . . . . . . . . . . . . . . . . . . 49
Figure 4.11 Storage usage throughout the second trial for the YOLO Search experiment. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
Figure 5.1 Progressive formation of a chain to complete a task. . . . . . . . . . . 57
Figure 5.2 Illustration of the zones in the communication model. . . . . . . . . . 61
Figure 5.3 Variables related to inter-robot distance and robot position before and after a time step $\Delta$t (a) illustration of combined distance of a connectivity chain before (b) and after (c) the addition of a new robot. . . . 61
Figure 5.4 Control architecture used within the chain construction algorithm. . . 64
Figure 5.5 Interplay between Networkers and Workers. . . . . . . . . . . . . . 66
Figure 5.6 Failure recovery: (a) at the time of failure and (b) after recovery. . . 66
Figure 5.7 Middle: comparison of convergence time with [1], convergence time of the chain construction algorithm by enabling and disabling the waypoint algorithm that allows the robots in a chain to predict the number of robots required. Top: convergence time by introducing one "C" shaped obstacle. Bottom: comparison of failure recovery time by failing different number of consecutive robots in a chain and creating two subgraphs. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
Figure 5.8 Top: time taken by 25 (arena 1), 60 (arena 2) and 100 (arena 3) robots to reach the target with 1, 2, and, 3 communication chains. The traveling time on the bottom indicates how fast a robot can travel the path without considering the chain. Bottom: time taken to build the communication with different percentage of random robot failures. . . 71

Figure 5.9 Top: Illustration of simulated maps with each robot starting position, end position and the safe communication disk region: on the left the smallest arena with flying robots going over an obstructed zone, in the middle a medium-sized arena with ground robots only, and on the right our largest arena, also using only ground robots. Bottom: Real robot experimental arena with start, end, trajectory and end position: on the left 7 khepera IV robots in arena1, in the middle 3 khepera IV robots and 3 CrazyFlie in arena 2 and on the right 6 CrazyFlie in arena 2. . 72
Figure 6.1 The states followed by the guide robots in our system. . . . . . . . . 76
Figure 6.2 Illustration of the control parameters used by the worker swarm to preserve a shape, the control parameters are controlled by the guide robots to change the properties of the worker robots. . . . . . . . . . 82
Figure 6.3 Finite state machine describing the high level guide behavior for all $i \in R^G$. The dotted boxes group the high-level states, inside the box are the internal sub-states. The diagram shows the standard single shape objective edge following and shaping procedure in solid black connections. Additional connections for a multi-shape execution is shown with dotted connections. . . . . . . . . . . . . . . . . . . . . . . . . 84
Figure 6.4 The range of shape configurations simulation used in the experiments, with varying swarm sizes. The snapshots are taken as the shaping mode is complete, before the guides transition to movement mode. . 90
Figure 6.5 The different trajectories undertaken by robots in each shape: clover on the left, dumbbell in the center and circle on the right. Here we can already see the motion of the circle is more direct than the others, with the dumbell showing noticeable vertical and horizontal distortion en route to its destination. . . . . . . . . . . . . . . . . . . . . . . . 91
Figure 6.6 Time taken by the worker robots to transition from one state to another. We can see consistently longer and more varying times in the movement of clover and dumbell shapes. The task allocation, shaping setup, and shaping of clovers and dumbells also appear to be affected by increasing the swarm size. . . . . . . . . . . . . . . . . . . . . . . 91
Figure 6.7 Distortion incurred by the swarm in different states, according to the metric defined in equation 6.12 and section $V$. Here we see how the dumbbell is particularly subject to distortion with smaller worker and guide swarms although reducing with higher sizes, whereas circle and clover shapes seem to increase in distortion with swarm size. . . . . 93

Figure 6.8 Trajectory for shaping and translating a swarm of 6 real robots. Left: simulation; middle: real robots; right: the time taken for the operation. Slightly different actuation and forces affecting the real robot dynamics make for a smoother set of trajectories simulation, albeit subject to minor drift. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
Figure 7.1 Illustration of the process of object transpiration, with the robots starting from a deployment cluster, caging the object and transporting the object. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
Figure 7.2 High level state diagram . . . . . . . . . . . . . . . . . . . . . . . 100
Figure 7.3 Illustration of the process of task allocation based caging (a) illustrate the process of edge following to reach the new target (b) the stopping condition to terminate caging. . . . . . . . . . . . . . . . . . . . . . 101
Figure 7.4 Illustration of the resultant force and the angle of effective robots, the effective pushing positions on the perimeter of the object is shown in green. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105
Figure 7.5 Trajectory taken by the centroid of the object vs the desired path in the three benchmarking paths. . . . . . . . . . . . . . . . . . . . . . 107
Figure 7.6 Time to complete caging (left) and push an object along 3 paths (right).107
Figure 7.7 From top to bottom: the first plot shows the average centroid estimation error, the second shows the object centroid position error, and the third shows the object orientation error; from left to right, the figure shows the results for our 3 benchmark paths. . . . . . . . . . . . . . 108
Figure 7.8 Number of robots that were effective in pushing and rotating at different waypoints of the three benchmark paths. . . . . . . . . . . . . . . 109
Figure 7.9 Trajectories taken by the robots (left) and the inter-robot distance between the two adjacent robot in the cage (right). . . . . . . . . . . 109
Figure 8.1 An automaton view of the Update protocol. . . . . . . . . . . . . . 114
Figure 8.2 Figures top to bottom, on the left: the first figure shows cluster with y-axis in log scale, the second shows scale free topology with y-axis in log scale, and the third shows line with y-axis in log scale. On the right: cluster, scale-free and line topology in linear scale. Each plot show: min, mean and max simulation control steps required for 10, 100 and 1000 robot swarm to apply a new patch of different sizes with 0%, 25% and 75% packet drop. The markers within the plot are slightly offset for visual clarity. . . . . . . . . . . . . . . . . . . . . . . . . . . 116

Figure 8.3 The left plot shows the results for varying the distance and on the right varying patch size. Both figures plot the min, mean and max number of control steps needed during thirty repetitions of the update experiments on the UAVs. . . . . . . . . . . . . . . . . . . . . . . . . 118

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ACO | Ant Colony Optimization |
| BGP | Bid Generation Process |
| BRP | Blob Request Process |
| BUP | Blob Up-date Process |
| BVM | Buzz Virtual Machine |
| CBAA | Consensus-Based Auction Algorithm |
| CLB | Chain Location Bidding |
| COTS | Commercial Off-the-shelf |
| DDS | Data Distribution Service |
| DHT | Distributed Hash Tables |
| EKF | Extended Kalman Filter |
| FoV | Field of View |
| GA | Genetic Algorithm |
| GBplanner | Graph-Based exploration Planner |
| GPS | Global Positioning System |
| IPFS | InterPlanetary File System |
| LJ | Lennard Jones |
| LUP | Lists Update Process |
| M | mode |
| MANET | Mobile Ad Hoc Network |
| MMKP | Multiple-choice Multidimensional Knapsack Problem |
| MOAP | Multihop Over-the-Air Programming |
| OTA | Over The Air |
| OMPL | Open Motion Planning Library |
| PSO | Particle Swarm Optimization |
| P2P | Peer-to-Peer |
| RFov | Reactive Field of view |
| RSSI | Received Signal Strength Indicator |
| ROS | Robot Operating System |
| RVO | Reciprocal Velocity Obstacle |
| SA | Single Allocation |
| SGBA | Swarm Gradient Bug Algorithm |
| SLAM | Simultaneous Localization and Mapping |

| | |
|---|---|
| SM | State Message |
| SOUL | Swarm Oriented Upload of Labeled data |
| TSDF | Truncated Signed Distance Field |
| UAV | Unmanned Aerial Vehicles |
| UWB | Ultra Wide Band |
| VS | Virtual Stigmergy |
| WSN | Wireless Sensor Networks |
| WPKG | Work Packages |
| WP | Way Point |

# CHAPTER 1 INTRODUCTION

"Equipped with his five senses, man explores the universe around him and calls the adventure Science."
Edwin Powell Hubble

"You never fail until you stop trying"
Albert Einstein

This thesis has been written in partial fulfillment of the requirements for the degree of Philosophiæ Doctor in computer engineering. This research work was conducted at MIST Laboratory of Polytechnique Montréal (Montréal, Québec, Canada) between September 2017 and January 2022. The structure of the document is that of a thesis by articles—five published and submitted contributions are presented in the Chapters from 4 to 8.

## 1.1 Context and Motivation

Nature exhibits some fascinating collective behaviors in schools of fish, flocks of birds, and colonies of insects (see Fig. 1.1). The existence of complex behaviors, which are beyond the capacity of simple individuals, is referred to as *swarm intelligence* [2], and the phenomenon of many simple things performing complex activities when working as a group is known as *emergence*. Among many examples, ant colonies are an excellent model for swarm intelligence and show several interesting emergent properties: ants work in parallel and use meager amounts of energy to perform tasks (efficiency), the loss of several ants does not compromise the colony (robustness), and they can overcome complex environmental changes (adaptivity).

Given the interesting properties of swarm intelligence, scientists created the field of artificial swarm intelligence to solve engineering problems. Artificial swarm intelligence was initially applied to virtual agents to solve optimization problems that are otherwise considered very hard. Examples of such computational algorithms are Ant Colony Optimization (ACO) [3] and Particle Swarm Optimization (PSO) [4]. ACO applies the foraging behavior observed in ants to optimization: a group of simulated agents move randomly in the search space (i.e., the space of possible parameters), locate optimal solutions, and lay virtual pheromones (analog to the chemical traces left by real ants) to direct other agents. Similarly, particle swarm optimization uses a group of agents moving in a search space. These techniques have been

Figure 1.1 Some examples of natural swarms are a flock of birds, colony of bees, schools of fish and swarms of ants. Credits: Bee colony CC 2.0 - flickr.com/Sy, Fish school CC 2.0 - iStock.com/armiblue, Army ants CC 2.0 - flickr.com/Axel Rouvin, Ant raft CC 4.0 - wikimedia.org/TheCoz and Starling swarm CC 2.0 - wikimedia.org/Walter Baxter

very successful in many domains like antenna design [5], vehicle routing [6], and scheduling problems [7].

Swarm robotics [8] is the application of artificial swarm intelligence to the design of robotic systems. Giving swarm intelligence to multi-robot systems in the real world is not as straightforward as for virtual agent-based optimization algorithms: robots need to perceive their environment, determine their position, and interact with other robots and the (potentially unstructured) environment. Performing all these activities in a single complex robot is already a daunting challenge, and having them emerge from the interaction of many simple robots requires novel approaches to design and to synthesize robotic systems. This additional complexity means that only a limited number of projects like Swarmanoid project [9], CoCoRo Swarm [10], and swarm gradient bug algorithm (SGBA) [11] have demonstrated out-of-the-laboratory operation capability, and there is no real-world application to date that directly uses swarm robotics design principles [8]. However, swarm robotics is rapidly finding new application domains (logistics, agriculture, space exploration [12], and many others) in which it can provide a definite advantage, and swarm-based real-world applications are bound to happen in the near future. Some of the critical challenges that need to be addressed to increase the practicality of robot swarms are:

- providing situational awareness to the whole swarm,
- ensuring that robots can effectively share information,
- effectively coordinating the swarm towards a global goal.

In this thesis, we attempt to deal with these issues and provide solutions that lead to practical applications of robot swarms.

### 1.1.1 Situational awareness

Situational awareness is the ability of a group of robots to collect and aggregate sufficient data to understand the environment and the state of the swarm. The information collected about the environment and the state of the swarm is potentially large and coming from multiple sources, and yet needs to be represented in a compact and precise manner. Both situational awareness and knowledge representation require the robots to store and manage large amounts of data effectively. As an example, consider the task of distributed map merging [13] and inter-robot loop-closure detection [14], in Simultaneous Localization and Mapping (SLAM), where robots need to exchange potentially extensive map fragments and pose graphs along with specific key-frame images. One compounding factor in performing these data sharing and aggregation tasks on resource-constrained robots is the limited available memory. Memory-saving strategies can require prioritization and data trade-offs, which can lead to the loss of important data. An alternative approach to data prioritization is to relocate the available data onto another robot with available memory. These alternative techniques are sometimes employed by peer-to-peer (P2P) file-sharing mechanisms [15]; applying these techniques directly to robot swarms is challenging. The main factors challenging the application of P2P file-sharing techniques on robots are the mobility involved and the underlying computational requirements. Novel methods addressing these challenges are required to store and manage data effectively in multi-robot systems. These novel techniques could find their applications in increasing situational awareness and achieving consensus among the robots.

### 1.1.2 Maintaining communication

Communication between a group of robots is another critical factor that plays a vital role in achieving coordination and situational awareness between the robots. Without reliable communication among the robots, many coordination algorithms, data storage infrastructures, and robot behaviors might be ineffective. Consider a group of robots exploring an unknown environment: lack of communication might cause robots to explore areas already visited by other robots, losing the critical benefit of task parallelization of swarm systems. The critical aspect of maintaining a coordinated group of robots has given rise to many strategies [1, 16–18] that maintain a connected group of robot. The strategies to maintain connectivity fall under either of the two classes: 1. strict end-to-end [19], and 2. relaxed intermittent [20]. The former type of connectivity demands the robots to maintain at least a single communication link between the robots at all times, and the latter allows the robots to stay out of range from other robots by agreeing on a schedule to meet up and exchange information. The type of connectivity enforced is linked with the task performed by the robots: for applications like exploration [12] and coverage [21], intermittent connectivity might enable a limited number of robots to explore larger spaces with opportunistic information exchanges. On the contrary, strict end-to-end connectivity might be preferred when the task requires the robots to continuously relay information such as video, operator commands, or offloaded computation. Strict end-to-end connectivity trades off flexibility for situational awareness, while intermittent connectivity trades off situational awareness for flexibility. Existing methods [18] available to enforce connectivity constraints on the robots are either computationally intensive or depend on a centralized system. The ability of a connectivity maintenance system to gracefully scale to large swarms without depending on any centralized infrastructure is essential to apply to simple, low-cost robots typically used in robot swarms. On top of these requirements, low-cost robots are fallible, and the sensors on these robots are imprecise, producing enormous amounts of noise. A connected swarm of robots operating in synergy should be capable of tackling these challenges. Tree based methods [1, 22] that rely on dynamically building a tree of connected nodes are found to scale to numerous robots, but these approaches are non-optimal and cannot handle robot failures.

### 1.1.3 Coordination

Effective coordination of robots is critical for achieving collective intelligence. Current methods for achieving coordination and collective behaviors rely on a reactive response to sensor measurements and neighbors' positions. For example, flocking [23] is produced by reactive responses to neighbor positions, self-assembly [24] is obtained by migrating robots based on positional gradients, and circle formation [25] reacting to neighbors in the frontal Field-of-View (FoV) cone. Applying these methods to real-world tasks carries the central problem of conveying the global mission goals to the robots. Hierarchical strategies found in nature [26, 27] demonstrate the ability to effectively perform tasks leveraging the knowledge of some experienced entities. One interesting example of hierarchy found in nature is the migrating flocks of birds; the birds in front of the flock are more experienced and perform considerably more thermal exploration than the individuals in the back, who follow the individuals in the front [26]. Similar approaches can be applied to a robot swarm creating hierarchies that allow global task specifications.

We posit that an information hierarchy, where at least some members of the swarm possess knowledge about the state and goals of the swarm as a whole is required to scale swarm robotics applications beyond a certain level of complexity. Whether the information hierarchy is an emerging property (e.g., brain function emerges from collaborating neurons) or a design feature of a system (e.g., a heterogeneous swarm of robots), we think it is necessary to aggregate information gathered locally and provide global decision-making for the swarm to achieve complex coordinated goals. This organization is akin to a heterogeneous swarm where some members, due to their position, current role, or capacity, are better suited at decision-making for the entire collective, taking the role of leaders.

Overall, swarm robotics has focused on studying low-level emergent behaviors, while collective behaviors operating through hierarchical control – which are abundant in nature [28, 26, 27, 29–31] – have received very little attention in the community. Robot swarms have demonstrated some interesting properties: the ability to scale across large swarms, robustness to cope with failures and parallel task completion capabilities [8]. The actual potential of robot swarms could be much more than what is demonstrated by rudimentary collective behavior, and we believe great potential lies within the swarm when multiple behaviors are composed in a hierarchy.

Taking the migrating flocks of birds as an example, the birds in the front of the flock are more experienced individuals and perform considerably more exploration than the ones in the back of the flock, who tend to follow these leaders [26]. When applying this concept to robots, the leaders could be individuals with enhanced capabilities or with superior knowledge of the environment in comparison to others, naturally giving rise to a hierarchy. There is a large body of works [32–35] that study leader-follower behaviors both in a distributed [32] and centralized settings [36]. The main idea in these works is that all the individuals simply follow one or a few robots with explicit communication [35], or without [37]. In leader-follower behaviors, the concept of robot individuality is lost where the robots simply copy the actions of one or multiple agents. **Most importantly, in a hierarchical organization of robot swarms, the robots at the bottom are capable of functioning (albeit at reduced capacity) even in the absence of robots at the top of the hierarchy**.

Advances in robotics have reached a level of maturity that allows autonomous robotic exploration on extraterrestrial planets [38]. Hierarchical organization leverages these advancements to improve robot swarms: the robots in the higher level of hierarchy use advanced percep-

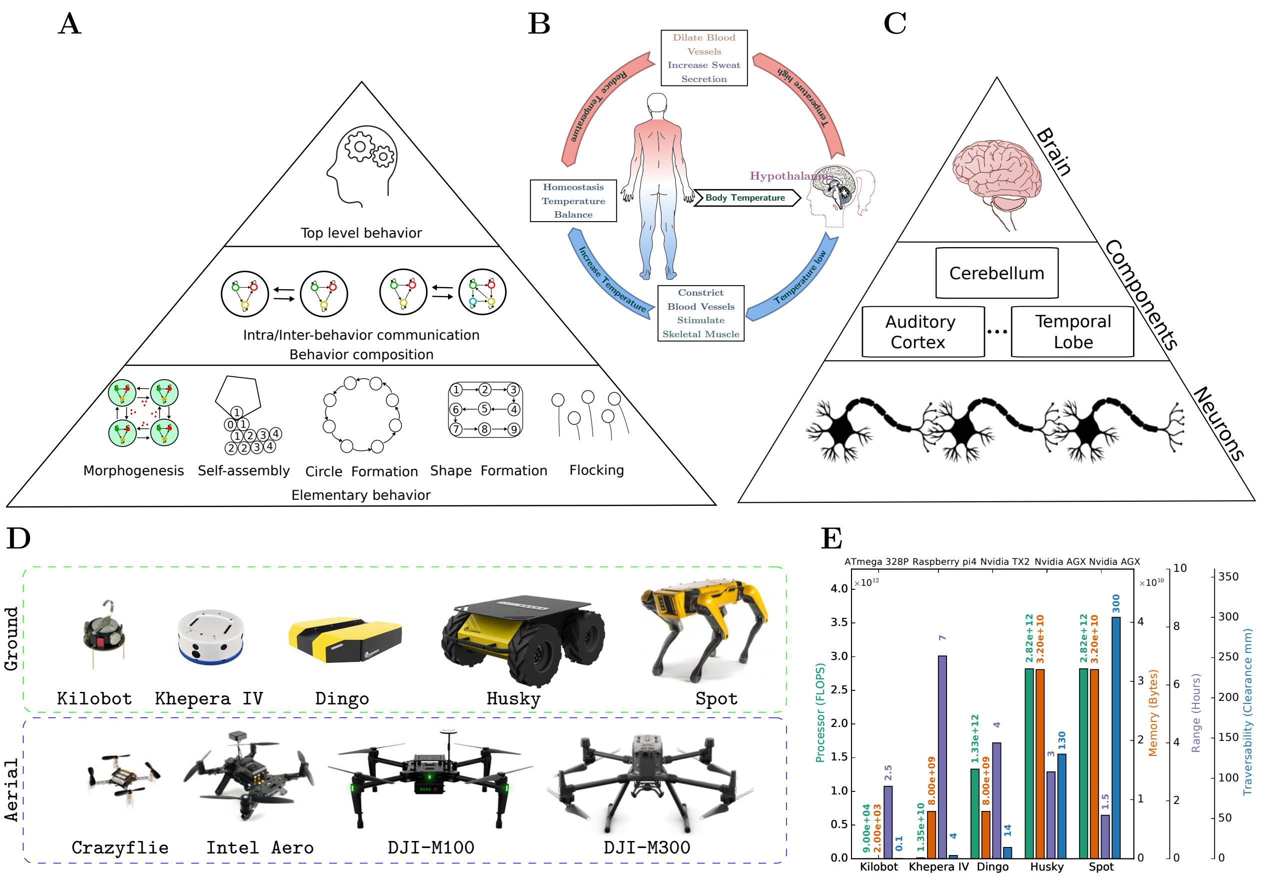


Figure 1.2 Hierarchies in robot swarms. (A) primitive robots on the lower level perform collective behavior providing information and exploiting the distilled information coming from the higher levels, producing a synergistic swarm operation as a whole. (B) homeostasis is one of the tasks performed by the hypothalamus inside the brain: maintaining an optimal body temperature by sending stimulus to muscles, sweat glands and blood vessels. (C) The hierarchy represented by the organs in human biology, with the brain as a whole performing a complex task (maintaining the well-being of the individual), composed of a variety of components performing different sub-tasks, and down to a microscopic scale these components are made of more or less self-similar neurons. (D) a range of commercial-off-the-shelf robots that can be composed into a single swarm with hierarchies. (E) capabilities (processing capacity, memory, traversability clearance, range) for the ground robots shown in (D).

tion and navigation algorithms, and are able to perceive high level mission goals. On the contrary, the robots in the lower level of hierarchy simply make use of the knowledge of the higher level robots, when available. In the absence of a higher level, the low-level robots operate as an egalitarian swarm. Most of the robotic swarm behaviors work on the assumption that robots need to produce a reactive response to certain sensor measurements and neighbor positions. Morphogenesis [39] in robots produce a reaction to virtual morphogen levels, flocking robots [23] react to neighbor positional changes, self-assembly [24] migrates robots based on gradients to create a desired shape, circle formation [25] reacts to neighbors in the field-of-view cone, and the swarm gradient bug algorithm (SGBA) uses reactive routines like wall-following to explore. These behaviors can be seen as microscopic actions performed by the robots that need to be guided at a macroscopic level towards the mission goals. A hierarchy is one natural way to produce such a macroscopic to microscopic mapping. Furthermore, robots used in swarm robotics are generally simple and cost-effective, and hence their limited processing only allows for execution of simple reactive behaviors, which amount to actions associated with stimuli. Assuming this design choice, one way to compose complex behaviors is to have the robots react to stimuli from the robots in the higher level of the hierarchy. Most swarm behaviors can still be used in this way, as long as they incorporate the necessary reaction mechanisms to interact with higher level behaviors from the top-level robots. Fig. 1.2 (A) shows the microscopic low-level behaviors that interact with a macroscopic high-level behavior through inter-behavior composition. This architecture is comparable to that of human physiology (see fig. 1.2 (C)), with one striking example being human homeostasis, maintaining body temperature in humans through hierarchical control. Fig. 1.2 (D) shows a range of commercial off-the-shelf (COTS) robots with a wide range of varying capabilities (see fig. 1.2 (E)) that can be readily used for hierarchical composition and real-world deployments. As shown in fig. 1.2 (D), robots on the left side (i.e., kilo bots, Khepera IV, and Crazyflies) are predominately used in swarm robotics. The choice of simpler robots allows the deployment of cost-effective large swarms and seems intuitively sufficient to design emergent behaviors that follow simple and local interactions. However, robots are becoming more advanced (as seen on the right of fig. 1.2 (D)) and capable of maintaining an enhanced belief of the environment and its states using advanced perception and mapping algorithms. These robots are considerably costly, and deploying a large swarm of such robots might be infeasible due to the associated cost. Using a small swarm of these expensive robots and a large swarm of expandable robots might be feasible using hierarchical organization. This hierarchical organization allows specializing robots based on their capability and uses the fullest potential of each robot towards improving the whole swarm's performance.

Natural swarms are groups made of individuals obeying the same rules: for instance, the behavior of natural swarms is allegedly defined by the gene pairs (DNA) of the organisms. Researchers often use scripts that encode a state machine on robot swarms to define the robots' behavior. The behavior designed for robot swarms is commonly emergent and needs adaptations to various deployment environments. Current practices in deploying behavioral software to robots are either customized explicitly to a given hardware or use complete retrieval to update the software and redeploy. Existing methods in Wireless Sensor Networks (WSN) provide appealing methods to deploy and maintain software. Applying methods from WSN to robot swarms are not robust to mobility and the emergent nature of the behavioral software. These challenges demand novel methods to deploy and maintain software on an already deployed swarm of robots.

## 1.2 Problem Statement

The primary focus of this dissertation is improving the practical applicability of robot swarms to real-world problems. Real-world use of robot swarms requires addressing the challenges surrounding improved situational awareness, maintaining a connected swarm, specification of global tasks, and behavioral design tools. Addressing these challenges increases the real-world applicability and allows for long-term autonomy, requiring minimal human interventions.

In particular, we introduce methods to solve and mitigate the following challenges:

- The nature of data collected on robots (such as sub-maps and point clouds) requires storing and managing large amounts of data for extended periods. The limited memory of the robots creates challenges in prioritizing vital data. Making the robots lose vital mission data used in enhancing situational awareness.
- The lack of simpler methods to enforce connectivity that can be used alongside other behaviors creates challenges in maintaining unified information and task replication in multi-robot systems.
- The lack of methods to enforce global task representation creates challenges in specifying the individual robot's task since swarm robotics behaviors produce a simple reaction to a sensor stimulus. This critical aspect of task representation renders many swarm behaviors challenging to use in real-world missions.
- Unclear aspects of hierarchical control of emergent collectives create challenges in deploying and establishing a truly hierarchically controlled robot swarm, as demonstrated in nature.

- Limited availability of methods to apply to hierarchically organized swarms demands new novel behaviors that can effectively work with hierarchies.
- Infrastructure available for behavior deployment on robot swarms are somewhat opaque; once deployed can not be modified on the fly with updates and patches. The lack of proper deployment tools demands new methods to deploy and maintain behavioral software on robots.

The author believes that addressing and mitigating the above challenges has immense potential in extending the frontiers of knowledge in swarm robotics and collective intelligence on artificial systems.

## 1.3 Research Objectives

The challenges described in the previous section have given rise to methods and solutions presented from Chapter 4 to Chapter 8 by undertaking the following research objectives:

1. Design a method for long-term data storage in robot swarms;
2. Develop fault-tolerant methods to enforce connectivity that will maintain maximal situational awareness and allow methods designed for data storage to function reliably;
3. Design methods to hierarchically organize swarm in such a way that global tasks for the swarm can be specified and implemented in real-world missions;
4. Exploit this hierarchical organization to design solutions to real-world problems, allowing thousands of robots to cooperate in synergy;
5. Implement behaviors that can leverage a hierarchical organization and perform collective tasks suitable for robot swarms;
6. Investigate and implement tools that to change the robots behaviors over time, allowing true long-term deploy-ability of robot swarms.

## 1.4 Research Contributions

To the best of the authors' knowledge, the essential novelty of the research presented in this dissertation are the following notable contributions to the field of swarm robotics, each corresponding to one of our research objectives:

1. A bit-torrent-like data storage system for a group of non-stationary robots that leverages the storage capacity of each robot, with each robot dedicating available unused storage to a robot community pool of memory. Robots can use this memory pool to collect extensive data. This storage system for robot swarms is called Swarm Oriented Upload of Labeled data (SOUL) and it is detailed in Chapter 4. This work was published as:

   V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, "Soul: data sharing for robot swarms," *Autonomous Robots*, vol. 44, no. 3, pp. 377–394, 2020

2. A computationally light connectivity preservation method that can gracefully handle robot failures and provide the capacity to operate alongside other behaviors; The result of this work is Swarm Relays, a fault-tolerant connectivity preservation method described in Chapter 5. Swarm relays initially appeared as an extended abstract and then with detailed formulation as a journal article, leading to the following publications:

   V. S. Varadharajan, B. Adams, and G. Beltrame, "The unbroken telephone game: keeping swarms connected," in *International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 2241–2243 V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, "Swarm Relays: Distributed Self-Healing Ground-and-Air Connectivity Chains," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5347–5354, 2020

3. A method to hierarchically organize and control large groups of robot swarms that can take global task information and operate as a single large system, presented in Chapter 6, which is published as:

   V. Varadharajan, S. Dyanatkar, and G. Beltrame, "Hierarchical control of smart particle swarms," *IEEE Transactions on Robotics*, 2022, under review

4. A behavioral approach to collectively transporting large objects using an ant-inspired behavior, this behavior is designed to be applied to a hierarchically organized swarm. This collaborative transport approach is discussed in Chapter 7 and published as:

   V. Varadharajan, K. Soma, and G. Beltrame, "Collective transport via sequential caging," in *International Symposium Distributed Autonomous Robotic Systems*, 2021, pp. 349–692

5. A behavioral update mechanism specifically designed to update and provide patches to collective behaviors commonly used in robot swarms, presented in Chapter 8 and published as:

   V. Varadharajan, D. St-Onge, C. Guss, and G. Beltrame, “Over-The-Air Updates for Robotic Swarms,” *IEEE Software*, 2018

## 1.5 Research Impact

We believe this work has and will continue to have a significant impact on the field of multi-robot systems. The works in this thesis have collected 56 citations at the time of writing, and the associated software was also used on several missions: the PANGAEA-X and IGLUNA experimental campaigns as well as the ESRIC Prospecting Challenge with the European Space Agency (ESA) and the DARPA Subterranean Challenge with NASA JPL. The results of these field missions are either published or submitted in the following papers at the time of writing (with more to follow):

V. S. Varadharajan, D. St-Onge, I. Švogor, and G. Beltrame, “A software ecosystem for autonomous uav swarms,” in *International Symposium on Aerial Robotics*, 2017 F. Wu, V. S. Varadharajan, and G. Beltrame, “Collision-aware task assignment for multi-robot systems,” in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS).* IEEE, 2019, pp. 30–36 D. St-Onge, V. S. Varadharajan, I. Švogor, and G. Beltrame, “From Design to Deployment: Decentralized Coordination of Heterogeneous Robotic Teams,” *Frontiers in Robotics and AI*, vol. 7, p. 51, 2020 M. Saboia Da Silva, L. Clark, V. Tangavelu, J. Edlund, K. Otsu, G. J. Correa, V. s. Varadharajan, A. Santamaria-Navarro, A. Bouman, S.-K. Kim, T. Vaquero, G. Beltrame, N. Napp, G. Pessin, and A.-a. Agha-mohammadi, “Achord: Communication-aware multi-robot coordination with intermittent connectivity,” *IEEE Robotics and Automation Letters*, 2022, under review

In addition, the work outlined in this thesis led to the $3^{rd}$ prize at the IEEE Research Boost 2017, the ESA PANGAEA Team Achievement award in 2018, and the AAMAS 2019 prize for the Most Innovative Systems Demo. In the future, we also expect:

- The SOUL data storage system will enable long-term autonomy and increased situational awareness in robot swarms. Consider the exploration task; robots can explore for extended periods without running out of memory by archiving the unused segments of the map and retrieving them when required.

- The mission state storage method allows robots to sense imminent failures (like running out of battery) to create a backup of its mission data, allowing other standby robots to replace this robot. This scenario can be commonly seen in the limited flying autonomy of Unmanned Aerial Vehicles (UAV) that requires continuous battery charging after every 20 minutes of usage.

- The lightweight connectivity preservation combined with the data storage enhances swarm situational awareness and has the potential to be used in several real-world missions. For instance, this approach could be directly applied to a chain of robots inspecting radiation disasters locations like the Fukushima nuclear plant without endangering human life.

- Hierarchical approaches have immense potential to revolutionize the domain of swarm robotics, and many notable researchers [8] in the field of swarm robotics acknowledge the potential of hierarchies.

- Behaviors designed for robot swarms often offer limited predictability when deployed on robots. Over-the-Air (OTA) update mechanism designed for robot swarms enables supplying patches to robot behavior after deployment, giving the designer more flexibility to adapt to real-world conditions.

- Finally, these methods and tools designed for enhancing situational awareness and long-term autonomy have the potential to inspire approaches that can be used in the robotics industry. Industrial approaches that stem from these works carry the potential to revolutionize the real-world deployment of robot swarms.

The approaches proposed in this thesis has the potential to improve innumerable venues of application scenarios, some notable ones are: 1. exploration, 2. applying expandable robots to disaster response, and 3. search and rescue. Many of these application scenarios produce some harsh and time-sensitive conditions, consider the application scenarios like disaster response and planetary exploration. In disaster response, all the preexisting infrastructure could be compromised, and in planetary exploration, communication to an operator could take minutes, if not days. The current methods available in robot swarms render these systems challenging to use and maintain in these application scenarios. The approaches designed in this dissertation mainly target these deployment challenges encountered in real-world time-sensitive scenarios. For instance, the hierarchical approach to robot swarms proposed in this dissertation significantly improves the mission success rate compared to traditional techniques.

# CHAPTER 2 LITERATURE REVIEW

This chapter discusses the literature in swarm robotics and details the methods that exist in the literature to the problems surrounding improved situational awareness, maintaining a connected swarm, specification of global tasks, and behavioral design tools.

All of these themes are equally important in addressing the practicality of robot swarms and increasing the autonomy of robot swarms. In each of the sections, appropriate chapters references have been provided to ensure consistency of these sections and locate the novelty of the research within the literature.

## 2.1 Robot swarms

The application of swarm intelligence on robots was first introduced by some pioneer studies [50, 51] proposing nature-inspired algorithms for robots. Nature-inspired algorithms were then studied using robots in projects like swarm-bots [52], and, Swarmanoid [9]. In particular, the swarm-bots project designed a custom modular robot called s-bot that can: 1. Attach to create chains and cross void spaces. 2. Collaborate to carry heavy objects collectively. 3. Attach to mock-up human dolly models and drag them to safety. Swarmanoid's project [9] first introduced heterogeneous robot swarm coordination and operation. Large-scale experiments using up to a thousand miniature hardware robots called kilobots were performed in [24] to study self-assembly. Cellular morphogenesis behaviors were also studied using kilobots [39]. Self-organization has also been applied to some small robots made of deformable and soft structures [53].

Furthermore, several recent studies [54, 25, 23] designed and studied many novel swarm behaviors on robots. These studies have brought some interesting knowledge to swarm robotics: scalability of the robots with decentralized control, ability to compensate for failing robots, and concurrent task performance. Even though swarm robotics is appealing with interesting properties, the main challenge is designing these systems. The design of decentralized behaviors for robot swarms is generally tackled with two approaches: 1. automatic design [55], 2. manual design [56]. In automatic methods, the behavior design problem is formulated as an optimization problem where the parameters for optimal robot behavior are found by searching the design space. Manual methods involve the design of the control software for the robots either by hand using a trial-and-error approach or using the designer's expertise. Manual methods have been predominant for designing robot behaviors because they provide maximal control on the robots' behavioral outcomes, minimizing the unpredictable factors. The main programming paradigms used in programming robots are robot-oriented programming, spatial computing, goal-oriented programming, task-oriented programming, and hybrid approaches.

**Robot oriented** The main focus of robot-oriented programming is to provide the designer with precise control to program every single robot in the group. In robot-oriented programming, the designer focuses on designing an individual robot behavior that will work synchronously to realize the desired group behavior. This type of swarm programming is also known as the bottom-up approach. One of the most common tools used for robot-oriented programming is the Robot Operating System(ROS) [57]. The complexity of managing the ROS-specific details increases exponentially with the number of robots in the system.

**Spatial Computing** focuses on providing programming tools for programming the swarm as a whole rather than considering the individual robot's behavior. Some examples of spatial computing are Proto [58] and Protelis [59]. The robots are assumed to be deployed on a manifold of space called an amorphous medium with a physical and computational state in proto. The robot program defines how the robots interact with the neighbors and the environment to perform a location-specific behavior in the amorphous medium. Spatial computing is a powerful tool for designing swarm behaviors but still loses the robot's individuality and the capability to program each robot in the swarm.

**Goal oriented programming** is considered a bottom-up approach where the individual robots are assigned spatial goals. The global task is broken down into elementary spatial goals and given to robots. The robots coordinate and reach these spatial goals in parallel. Some examples of goal-oriented programming languages are SWARMORPH [60] and Termes [61]. Goal-oriented programming is more suited when the mission requires spatial organization among the robots and has minimal to no robot failures. The approaches do not have contingent mechanisms for robot failures.

**Task oriented programming** In task-oriented programming, the global task is broken down into a set of sub-tasks (such as spatial goals) that can be performed by a single robot and optimally assigned to robots. The robotic swarm is approximated to be a system with parallel machines that can schedule jobs using a scheduler (a system that assigns a resource to a specific task). These types of systems are referred to as deterministic parallel machines in sequencing and scheduling theory [62]. The task of control software design in goal-oriented programming is to formulate the global problem as a scheduling problem and design a scheduling system that will assign jobs (sub-goals) to the robots in a swarm. Task-oriented programming is considered a bottom-up approach since the individual robots are assigned tasks separately. Karma [63] and Voltron [64] are some examples to task-oriented programming. Task-oriented programming is more suited for missions that can be decomposed into a set of sub-tasks and can be performed on a single robot. Task-oriented programming is incapable of performing missions that require active inter-robot coordination (for example, when a sub-task requires two or more robots to complete it).

**Hybrid approaches** Buzz [56] is considered to be a hybrid domain-specific programming language that provides programming primitives similar to robot-oriented programming [57], spatial computing [59] and goal-oriented programming [60]. It allows programmers to maintain desirable levels of abstraction while programming. The swarm can be programmed as a whole (top-down), or individual robot behaviors can be designed (bottom-up) at the same time in single control software. For instance, the language provides support for both setting the actuation commands (bottom-up) and support for neighbors management to consider the swarm as a whole and perform operations in the neighborhoods (top-down). A pure bottom-up approach suffers from scalability issues, and conversely, a top-down approach suffers from an inability to fine-tune individual robot behaviors. A concurrent design used in Buzz allows the designer to pick the right amount of abstraction required at the various stages of the mission.

Buzz satisfies most of the desirable properties of a swarm programming language: the code can be organized as functions and classes (composable), language syntax is intuitive with similarities to Python and Lua (predictable), swarm programming constructs allows concurrent use of heterogeneous robots in a swarm (heterogeneous hardware support) and unified Buzz Virtual Machine (BVM) for use with various hardware platform (hardware-agnostic). These properties make Buzz a promising approach to design control software for robot swarms.

We designed a tool called ROSBuzz [48] that brings together robot-oriented programming (ROS) and the hybrid approach (BUZZ) for commercial off-the-shelf (COTS) robots. The studies presented in chapter 5 to chapter 8 was experimentally evaluated with this tool called ROSBuzz [48] on hardware platforms.

Research has produced promising methods [65] and tools [56] to design swarm behaviors. These tools have been widely used in creating behaviors for robot swarms. However, the current approaches to developing swarm behaviors require a significant design effort and limit their use in many practical applications [8]. In the opinion of the author, the main design effort lies within specifying the specifications of tasks of the individual robots given a global task requirement. On the one hand, robots might have to encode the high-level tasks prescribed. Conversely, the robots might have to operate using local reactive behaviors to achieve simple and local interaction, the primary design principle followed in designing swarm behaviors. Considering nature-inspired flocking [23], robots maintain a prescribed relative position between their neighbors to stay in a formation and self-propel the whole group to a target location. In our hierarchy introduced in the chapter 6, we take inspiration from nature [27, 26] and propose an approach to hierarchically organizing the robots in a group. This approach allows some robots with the ability to perceive the global goals, these robots coordinate and drive the whole swarm towards the mission goal. This approach allows the robots at the top of the hierarchy to perceive global mission goals and other robots to operate based on local reactive rules, maintaining a simple local interaction throughout the swarm.

## 2.2 Distributed data storage

A multi-robot system's sheer amount of data requires novel methods to store and maintain information within the swarm. One common approach to data sharing in distributed systems involves using Distributed Hash Tables (DHT). There are several methods for implementing a DHT [66]. Most of the standard techniques in DHT treat data as tuples with a key and a value. Direct application of these methods to robot swarms is challenging because of a fixed communication topology requirement. The presence of mobility in robot swarms makes the direct use of DHT on robot swarms complicated. Taking inspiration from DHT and nature-inspired stigmergy [67], Pinciroli et al. designed virtual stigmergy [68]. Virtual stigmergy provides a shared space among the robots in a swarm. Each robot can write a value to a table, and these values are replicated on all the other robots using gossip-based communication. In the presence of mobility, gossip-based communication allows robots to adapt to the dynamically changing network topology and achieve consensus on data [68]. Even though virtual stigmergy is a promising approach, the amount of data stored is limited to a single entry(float or integer).

However, methods [15] in peer-to-peer (P2P) networks offer the storage of large amounts of data demonstrating scalability and robustness. The main problem in applying methods in P2P to robot swarms is the lack of reliable communication and limited robot storage capacity. InterPlanetary File System (IPFS) [69] is another interesting approach in P2P that shares data comparably to BitTorrent, the differences from BitTorrent lie in the way the server is replaced with a decentralized system. Applying IPFS directly to robots is also challenging because of the underlying memory usage and computation complexity.

In chapter 4, we propose SOUL that takes inspiration from virtual stigmergy to provide a tuple storage space and IPFS to partition the data into smaller chunks to be stored on robots. SOUL offers a tuple storage space in the form of ⟨key,blob⟩, where each blob (for example, images and maps) is split into smaller chunks and distributed to the robots with available storage. Our approach has already inspired other methods [70] that offer location based storage.

## 2.3 Connectivity preservation

The key aspect of a multi-robot system coordination is achieving unified information across the whole group. Robots might have to agree on unified information to synchronously work on a task. The two main types of connectivity preservation are adopted in multi-robot systems: strict end-to-end connectivity [19], and relaxed intermittent connectivity [20].

Some notable works [71, 17] use spectral graph theory and algebraic connectivity (second non-zero eigenvalue of the Laplacian matrix of a network) to enforce connectivity. Method in [72] takes a distributed approach to compute algebraic connectivity using power iteration. In a decentralized setting, continuous control models [17] are used to achieve structured connectivity by making use of algebraic connectivity. Approach in [73] implement mission oriented control laws on top of continuous control models to enforce connectivity constraints. Some approaches [18] apply k-connectivity from graph theory along with algebraic connectivity to enforce connectivity. Another approach is to design reactive control laws using connectivity as an imposed constraint. For example, in [74] two control laws were developed for rendezvous and formation tasks, maintaining an initially connected configuration and effectively preserving connectivity. The main challenge of applying algebraic connectivity on robots that work in a decentralized manner is to compute the Laplace matrix associated with the robot network. Some known methods like power iteration [72] and wave propagation [75] are slow (multiple computation iterations are required before convergence) and sensitive to noise for distributed implementations.

Another class of methods enforce the desired connectivity among robots by constructing and maintaining a given communication topology. For instance, Schuresko et al. [76] employ a robust and mission-agnostic algorithm that generates a connected spanning tree. Aragues et al. [77] implement an area coverage mission while preserving a minimum spanning tree among robots. This approach maintains connectivity with minimal interference on the area coverage mission. However, this approach requires specific initial conditions such as a fully connected network, which cannot always be guaranteed during field deployment. Majcherczyk et al. [1] treat the connectivity problem as a decentralized deployment problem, where multiple robots are tasked to different target locations while preserving connectivity. Their algorithm assigns roles for robots, such that when a target is specified, a branch of the robot network is deployed and additional robots are supplied to build a structure reaching the target, creating a connected swarm. Panerati et al. [78] proposed a hybrid methodology with a navigation controller enforcing connectivity and a global scheduler to provide the navigation controller with optimal policies. Despite the use of a global scheduler to support the navigation controller, the approach is relatively slow.

On top of the mobility requirements in a multi-robot system that makes connectivity preservation challenging, there are other sources of failures like hardware malfunction and environmental factors. Some works consider robot failures within their controller design [16], but they rely on algebraic connectivity. In Chapter 5, we propose a decentralized approach called swarm relays that is both computationally less demanding than approaches that compute algebraic connectivity [17] and also fault resilient. Swarm relays operate on reactive rules commonly used in swarm robotics and are suitable for robots with limited computational resources.

## 2.4 Behavioral design for robot swarms

Robot swarms behavior design involves designing robot control software given a task specification. The robot controller designer uses one of the tools presented earlier to develop the robots' behavior. In chapter 7, the hybrid approach (Buzz) is used to design a collective behavior that can apply to a hierarchically organized swarm. A collective transport behavior predominantly demonstrated by ants [79] is studied on robots in chapter 7. This section presents all the relevant literature related to collective transport on robot swarms and pertinent approaches to updating these behaviors on robot swarms.

**Collective Transport** The collective transport behavior starts with a process called caging [80], it is a process of enclosing an object to obtain a grip for manipulation. Some approaches [81, 82] make use of object closure or form closure to surround an object to be gripped completely, and this enclosure achieves caging. The main principle behind these approaches is to use an imaginary space using the position and orientation of the object. Computing the relative position of neighboring robots is already a challenging task. On top of that, estimating the pose of the transported object adds additional challenges. A behavioral state-machine that switches between the approach to object and attachment is used by Fink et al. [83]. Similarly, Gerkey et al. [84] employed a strategy that uses some robots to monitor caging and others to perform caging. Other comparable approaches [85] make use of robot group partitioning to assign role-specific tasks to robots. The system's overall performance relies heavily on a few agents that monitor the caging.

The method of caging proposed in chapter 7 makes use of only the relative position of the neighbors and local measurements in a way comparable to [86], the main differences being in the way the robots are placed around the object. The approach proposed by Chen et al. design strategy to identify the position in the occluded region between the goal and the object to place robots, whereas the system in chapter 7 make use of the whole circumference of the thing to be transported, allowing directional changes during motion.

**Behavioral updates** The design of swarm behaviors generally involves a series of trial-and-error iterations to ensure the reliable operation of the robots. The designer has to ensure both the operation of the individual robot and the robot collective is reliable. This design approach demands novel tools to deploy and maintain the behavioral software on the robots. The work on self-assembly [24] designed its specific device for the kilo-bot swarm that broadcasts software to all the robots. Apart from this particular tool in [24], there is a limited number of tools to deploy behavioral scripts onto robot swarms and update them on the fly. However, update tools are abundant in Wireless Sensor Networks (WSN). Pilloni et al. [87] proposed an approach to distributing updates over a WSN with gossip-based routing. In this approach, the authors define the sensing and communication requirements using a middleware that allows the deployment of different applications. The work of Levis et al. [88] proposes an update algorithm for code propagation and maintenance of the new releases in WSN called “trickle”. They use a “polite gossip” policy to periodically broadcast a summary of the code under execution to neighbors. The code summaries are used to maintain the consistency of the software executed across the whole network. Hui et al. [89] proposed a data propagation protocol called “Deluge” for WSN. “Deluge” allows transmission of a large data packet from a source node to all the other nodes in the multihop networ Thanos et al. [90] in their technical report proposed an algorithm called Multihop Over-the-Air Programming (MOAP). The work presents an OTA update mechanism to deploy updates to a WSN through a multihop network. Their work included data dissemination and retransmission strategy for Mica-2 motes [1]. Their data dissemination approach was based on a windowed retransmission tracking scheme.

Some of other approaches to WSN OTA update are Impala [91], ECD [92], XNP [93], MNP [94], and Freshnet [95]. These approaches widely used in the WSN community are valid as long as the nodes are stationary. When applying these approaches to mobile robot nodes, they are challenged by inconstant behaviors on robots and changing robot topology. The method presented in chapter 8 makes use of a consensus mechanism to hold the robots in a safe state before they can be updated to the new behavioral software.

[1]`https://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/mica2.pdf`

# CHAPTER 3 RESEARCH APPROACH AND THESIS ORGANIZATION

This Chapter presents the methodologies used to pursue the objectives outlined in Chapter 1. The research work contains five work packages (WPKGs) that address the problems outlined in Chapter 1. The five main work packages designed in this dissertation are as follows:

1. WPKG1 investigated available methods in multi-robot and distributed systems to store data for extended periods. Design a data storage system for robot swarms considering both the robots' dynamic nature and limited processing power. The contributions from this work package are presented in Chapter 4.
2. WPKG2 develops a light-weighted method to maintain connectivity in robot swarms taking into account the drawbacks of the current methods. The contributions of the WPKG2 is outlined in Chapter 5.
3. WPKG3 designs a novel method to hierarchically organize a swarm of robots that can take in global task information and progress the swarm toward this goal. Chapter 6 discuss and experimentally validate this approach both in simulation and a realistic hardware deployment scenario.
4. WPKG4 develops a collaborative transport method, which can apply to a hierarchically organized swarm proposed in WPKG3. Chapter 7 introduces and experimentally validates the collaborative transport approach.
5. Finally, WPKG5 involves designing and validating an OTA tool to deploy software on robot swarms. Chapter 8 introduces this software deployment tool.

## 3.1 Methodology

The tools and methodologies employed during the research work conducted as a part of this dissertation are outlined in this section.

**ARGoS Robotic Swarm Simulator** All the methods presented as a part of this dissertation used the ARGoS3 [96] simulator to perform experimental evaluations. During the evaluation of these works large-scale simulations were performed on Compute Canada clusters [1] with up to a thousand robots since the simulator gives the flexibility of using a large

[1]`www.computecanada.ca`

number of robots without the complexity of setting up the hardware. The modular nature of the simulator offers the flexibility of customizing the simulator to the needs of the current evaluation performed; this was one of the key reasons to choose the ARGoS3 simulator. ARGoS3 simulator can be considered a few simulators that allow a physics-based engine in the background and still be light-weighted to run simulations with large swarms.

**BUZZ – Domain Specific Programming Language for Swarm Robotics** As introduced in chapter 2, we use Buzz in all the research conducted as a part of this dissertation. The main reason for this choice was the flexibility in programming (i.e., allowing hybrid programming) and the rich feature set apt for swarm robotic applications. In particular, we developed ROSBuzz [48], a tool that connects robot-oriented programming (ROS) and hybrid approach (Buzz) more tightly to benefit from community-developed packages and drivers.

**Hardware Platforms** Methods developed as a part of this dissertation were strictly validated using one of the hardware platforms (i.e., groups of robots) since the authors' motive in this dissertation is to develop methods to improve the real-world deploy-ability of robot swarms. A wide range of robots were used to validate various methods presented in this dissertation.: 1. Khepera IV robots [97], crazyflie robots [2], Spiri robots [3] (PX4 controller equipped robot), Customized Khepera IV robots, and Dji M100 [4]. More specifically, the research in Chapters 4, 5 and 7, used up to 10 Khepera IV robots. The connectivity maintenance approach in Chapter 5 was additionally validated on 4 Spiri, 6 crazyflie, and 3 Dji M100 robots. The hierarchical work presented in Chapters 6 used a customized Khepera IV robots with an Nvidia TX2 [5] onborad computer for some robots called guides and raspberry pi4 [6] for the others. In addition to powerful computing, the guide robots are equipped with 2d lidar and camera systems to perform advanced perception (SLAM) and navigation (exploration and local planning). Finally, the work presented in Chapter 8 used Dji m100 robots to validate and study the performance of the hardware.

## 3.2 Document Structure

This document, with title "Communication, Coordination and Organization of Practical Robot Swarms", is a dissertation submitted to Polytechnique Montréal in partial fulfillment

[2]`https://www.bitcraze.io/products/old-products/crazyflie-2-0/`
[3]`https://spirirobotics.com/`
[4]https://www.dji.com/ca/matrice100
[5]`https://developer.nvidia.com/embedded/jetson-tx2`
[6]`https://www.raspberrypi.com/products/raspberry-pi-4-model-b`

of the requirement for the degree of Philosophiæ Doctor in computer engineering. It follows one of the two layouts permitted by the school for dissertations by articles, that is, the inclusion of published or under review articles in the body of the work, each as a separate chapter. In particular, Chapter 1 provides a short introduction by stating the motivation and the research objectives. Chapter 2 examines the relevant literature, and Chapter 3 describes the methodology used in the research work. Chapter 4 is the work on distributed data storage mechanism for robot swarms corresponding to WPKG1. Chapters 5 detail the connectivity preservation method and discuss the experimental results, corresponds to WPKG2. Chapters 6 detail the approach to hierarchical organization of robot swarms and study the approach though experimental evaluations, corresponds to WPKG3. Chapter 7 propose and evaluate the method of collective transport in robot swarms that add up to WPKG4. Chapter 8 discusses the OTA update to deploy software on robot swarms and checks WPKG5. A discussion surrounding the approaches and the results obtained during the evaluation of the methods is present in chapter 9. Finally, Chapter 10 provides some concluding remarks on the research work.

# CHAPTER 4 ARTICLE 1: SOUL: DATA SHARING FOR ROBOT SWARMS

**Preface:** Data accumulated by robots performing tasks in realistic scenarios can often be extensive and eventually exceed the available storage on the robots. This problem demands novel solutions to store and manage mission data collected by robots. In this chapter, we propose a solution called SOUL; leveraging the available storage on all robots, SOUL creates a shared pool of memory that can be used by any of the robots in the swarm. The approach takes inspiration from bit-torrent and considers the data provided by the robots as blobs. These blobs of data are split into smaller chunks that can be distributed to robots with unused available storage. In this work, we consider a specialized swarm assuming a hierarchical organization with some robots more capable and specialized compared to the majority of the swarm. The specialized robots that generate a large amount of data are called *captureres*. Robots dedicating their storage space to the swarm are called *networkers*, and the robots processing data are called *processors*.
In addition to the data storage method proposed in this chapter, we introduce a method to compress robot mission data that can be stored with SOUL. We study the performance of SOUL using two performance metrics that demonstrate the scalability of the approach. We validate the usability of SOUL in two prototype missions using Khepera IV robots.

**Full Citation:** Varadharajan VS, St-Onge D, Adams B, Beltrame G, “SOUL: data sharing for robot swarms,” *Autonomous Robots*, 2020 Mar;44(3):377-94.

**DOI:** `https://doi.org/10.1007/s10514-019-09855-2`

**Abstract:** Interconnected devices and mobile multi-robot systems are increasingly present in many real-life scenarios. To be effective, these systems need to collect large amounts of data from their environment, and often these data need to be aggregated, shared, and distributed. Many multi-robot systems are designed to share state information and commands, but their communication infrastructure is often too limited for significant data transfers. This paper introduces SOUL (Swarm-Oriented Upload of Labeled data), a mechanism that allows members of a fully distributed system to share data with their peers. We leverage a BitTorrent-like strategy to share data in smaller chunks, or datagrams, with policies that minimize reconstruction time. We performed extensive simulations to study the properties of the system and to demonstrate its scalability. We report experiments conducted with real

robots following two realistic deployment scenarios: searching for objects in a scene, and replacing the full identity of a defective robot.

## 4.1 Introduction

Multi-robot systems can either be controlled from a central node (a cloud system, a ground station for UAVs, etc.) or be decentralized. A decentralized multi-robot system is often referred to as a robot swarm. In such robot swarms, members cooperate with their neighbors and perform several physical tasks simultaneously. An example of an application that would benefit from robot swarms is search-and-rescue, where the robots need to cover a large area. A single Unmanned Aerial Vehicle (UAV) needs extended autonomy and substantial processing power to analyze images of the terrain. Distributing the task over many UAVs requires fewer resources from each robot, and multiple areas can be explored in parallel.

The advantages of a decentralized system can be even more conspicuous when considering a heterogeneous swarm. As an example, flying robot platforms usually have limited payload capacity, but a swarm can be partitioned in sub-swarms according to special attributes (e.g., cluster of robots equipped with cameras, high processing and memory resources, etc.) and assigned specific tasks to better exploit the available resources.

These specialized sub-swarms need to coordinate and exchange information, leading to a new challenge: significant data transfers among sub-swarms become unavoidable. Many applications require such data transfers: for instance, mapping a region after a disaster using aerial photography is still mostly executed on ground stations, and can use large 4k images [98]. Scanning a damaged structure using state-of-the-art RGB-D sensors and GPU-optimized algorithms [99, 100] can be made more efficient when distributed across multiple robots.

Furthermore, in critical missions such as emergency response, swarm intelligence has the great advantage of being able to cope with robot failures, but not without a cost in terms of efficiency or swarm capability. However, if a fault detection mechanism is available, as simple as battery monitoring, each robot can warn the others about its possible imminent failure and upload all of its local data and state variables to the swarm. Using this backup of the failing robots' identity and mission status, another agent can seamlessly replace it.

To address the data sharing requirements of these application scenarios, we developed a strategy for the *Swarm-Oriented Upload of Labeled data* (**SOUL**). The main challenges of SOUL were to: 1- cope with dynamic network topologies, 2- optimize the data fragmentation and reconstruction, and 3- optimize the distribution of the datagrams (chunks of injected data). Since peer-to-peer (P2P) file sharing mechanisms are well established in literature, with ample research to demonstrate their robustness and scalability [15], SOUL leverages some of their strategies (e.g., with the use of distributed hash tables) and integrates additional concepts from decentralized robotic systems. SOUL addresses a problem with current artificial decentralized systems and can be used on top of any swarm behavior.

In previous work [68, 101], we developed strategies to reach consensus on environmental data or swarm-wide state variables. We showed that our mechanism (dubbed Virtual Stigmergy, inspired by biological swarms) was robust to heavy packet loss, scalable, and greatly enhanced the robustness of a swarm when allocating a set of tasks. This mechanism, however, was suitable only for variables or small data structures. In this paper, we extend the Virtual Stigmergy to share, and reach consensus on blobs of data, using a bit torrent like mechanism.

In the following, we describe the SOUL model in Sec. 4.2. From the model, we derive an implementation for the Buzz programming language [102] (Sec. 4.3) and discuss its performance in simulation (Sec. 4.4). In sec. 4.6 we describe the research work that inspired the SOUL design. Finally, we present two realistic experiments for SOUL, executed on a swarm of wheeled robots (Sec. 4.5).

## 4.2 SOUL model

The *Swarm-Oriented Upload of Labeled data* (**SOUL**) mechanism is a collection of discrete policies and bidding algorithms that adapt to the swarm's network topology. Each time a robot shares a file, referred to as a blob, this file is split into a series of smaller chunks of data, referred to as datagrams, spread throughout the swarm. A map $d_b$ holding the datagrams of a blob $b$ is called a datagram map. All variables used in the definitions of this section are listed in Tab. 4.1. The main problem consists of attributing each datagram to the right member of the swarm, or:

> Given a number of blobs $\mathscr{B}$ $=\{b_1,b_2,...,b_k\}$, a team of robots $\mathscr{R} = \{r_1,r_2,...,r_n\}$, a function $C(\mathscr{B},\mathscr{R})$ that specifies the reconstruction cost of storing datagrams of all blob in $\mathscr{B}$ on robots in $\mathscr{R}$, find the allocation configuration $Q = \{b_i \in q_b^r \mid \forall b_i \in b, \forall b \in \mathscr{B}, r \in \mathscr{R}\}$ that minimizes the reconstruction cost from a sub-set of robots $\mathscr{P} \subseteq \mathscr{R}$.

Table 4.1 Variable definitions.

| | |
|---|---|
| $r$ | robot (node) |
| $\mathscr{R}$ | set of all robots |
| $b$ | blob (data file) |
| $\mathscr{B}$ | set of all blobs |
| $d_b$ | set of $b$'s datagrams (blob chunk) |
| $q_b^r$ | datagram set of blob $b$ on robot $r$ |
| $Q$ | allocation configuration containing datagram set of blob |
| $s^r$ | available resource of robot r (such as storage space) |
| $X^r$ | set of proximity to all processors from $r$ |

### 4.2.1 Objectives

While blobs can be continuously injected in the swarm network, each robot $r \in \{r_1, r_2, ..., r_n\}$ has a limited set of resources $s^r$. Fig. 4.1 illustrates a possible distribution of three consecutive blob injections with a representation of the robots' storage resource $s^r$, with $r \in \mathscr{R}$, the list of robots in the swarm. The robot injecting the blob into the swarm network acts as an auctioneer: it collects bids from its neighbors and allocates parts of the blob to different robots based on their bid. The robots' bids include considerations on available storage space $s^r$ and battery level. Given a subset of processor robots $\mathscr{P}$, i.e., robots that requires the blob files, the allocation problem consists of minimizing the time to reconstruct the blob on these units.

The rectangular box above the robots, titled *Storage level*, indicates the occupancy of each robot's memory, with colors and labels relating to each blob. The figure shows $r_2$ injecting blob tuple $\langle$2,blob$\rangle$ at time step $t_1$, the 2 being the blob's ID. Blob 2 ($b2$) gets partially allocated to $r_2$ and $r_3$ considering $s^2, s^3$ . SOULS adds the allocated datagrams to $q_2^2, q_2^3$ at $t_2$. Similarly, SOUL performs the injection of blobs 3 and 4, initiated by $r_3$ and $r_1$ respectively.

Now consider the problem to comprise a heterogeneous group of robots with different resources. A heterogeneous group of robots is defined by the diversity of each of its members' abilities. While identical units are useful for redundancy and scale, it is cumbersome and expensive to deploy one omnipotent type of robot having the full range of required sensing modalities of a mission. In a swarm, sub-groups or sub-swarms can have different specializations and still be managed as a whole in a distributed fashion, as widely demonstrated in nature's insect swarms [103] and robot swarms with Swarm-Oriented programming [102].

For SOUL, one can think of a sub-swarm of robots capturing inputs, i.e., injecting the blobs, a sub-swarm of robots with more processing power and memory to parse and process the

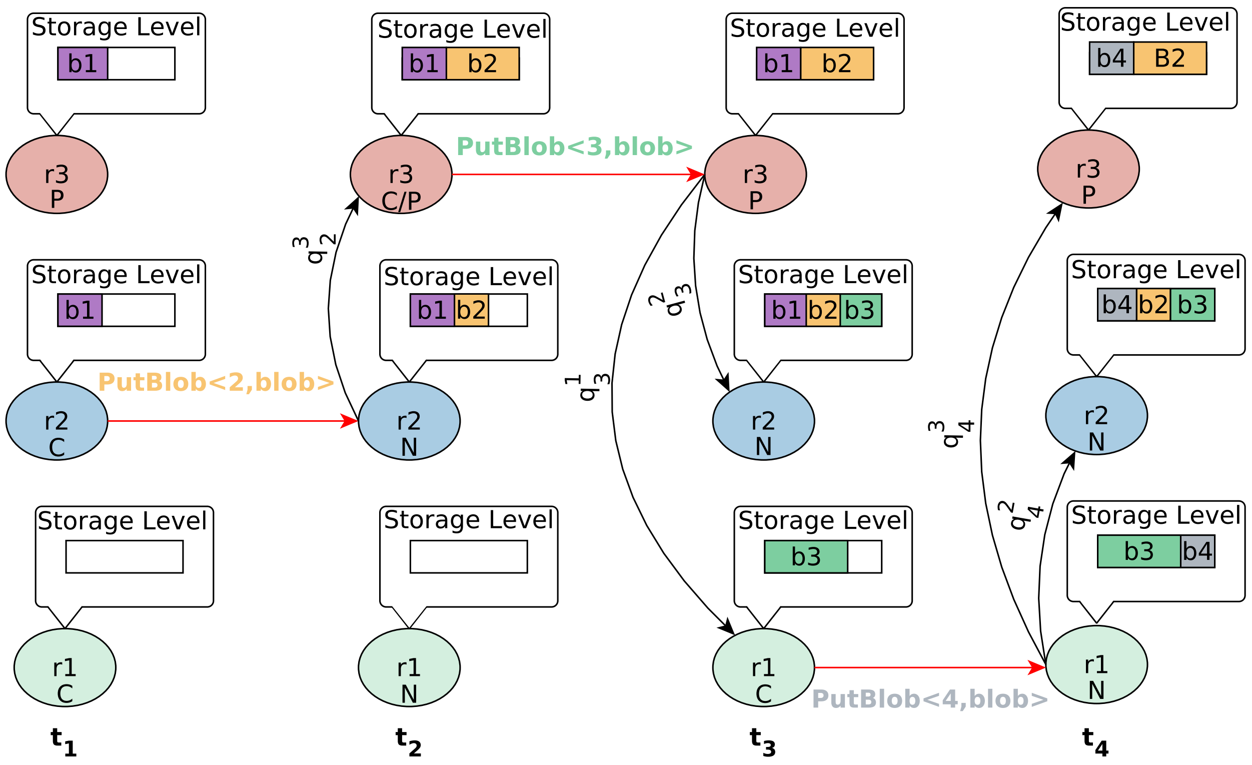


Figure 4.1 Illustration of three consecutive blob injections on a network of three robots.

data, and a generic sub-swarm of robots maintaining the network connectivity of the whole swarm. Consider these three sets of robots: capturers $\mathscr{C}$, processors $\mathscr{P}$, and networkers $\mathscr{N}$, such that $\mathscr{P} \cup \mathscr{C} \cup \mathscr{N} = \mathscr{R}$. Blob datagrams can be stored by robots with any of these three roles.

In general, any robot can take any role depending on the operation performed, set at run time. In this work, we consider fixed roles for robots. During operation, the robots are only aware of the following two pieces of information: the proximity to the processor and their own role. Proximity is measured in number of hops, using periodic gossip broadcasts from the processor. The capturers can be considered as data sources, the networkers buffer the data for processors and the processors are the data sinks.

**Definition 1.** *Given three sets of robots $\mathscr{P} = \{p_1, p_2, ..., p_u\}$, $\mathscr{C} = \{c_1, c_2, ..., c_v\}$ and $\mathscr{N} = \{n_1, n_2, ..., n_w\}$ with $s^r > 0$, $\forall r \in \mathscr{R}$, a set of blobs $\mathscr{B} = \{b_1, b_2, ..., b_k\}$ injected from any $r \in \mathscr{C}$ and the reconstruction cost $C(\mathscr{B}, \mathscr{R})$, an allocation configuration $Q$ exists that minimize the reconstruction cost:*

$$\min_{Q} \sum_{b \in \mathscr{B}} C^p_b(b, \mathscr{R}), \quad \forall p \in \mathscr{P} \tag{4.1}$$

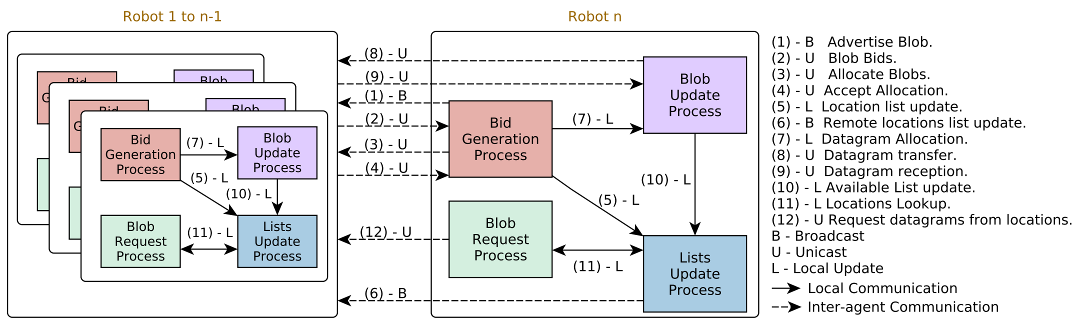


Figure 4.2 The four processes and twelve message types involved in the SOUL mechanism.

*The objective of SOUL is to minimize the cost of reconstructing (Eq. 4.1) all blobs $b \in \mathscr{B}$ for all the robots in set $\mathscr{P}$, by varying the datagram allocations $Q$ to robots in $\mathscr{R}$, with $Q$ containing datagram sets $q_b^r$ of all $r \in \mathscr{R}$ and all $b \in \mathscr{B}$ .*

*the datagram set $q_b^r$ of blob $b$ on robot $r$, is subject to:*

$$|q_b^r| \leq |d_b|, \quad \forall b \in \mathscr{B}, \forall r \in \mathscr{R}, \tag{4.2}$$

$$\sum_{r \in \mathscr{R}} |q_b^r| = |d_b| \,, \quad \forall b \in \mathscr{B}, \tag{4.3}$$

$$s^r \geq 0, \quad \forall r \in \mathscr{R}. \tag{4.4}$$

Eq. 4.2 states that the blobs can be allocated to more than one robot. Eq. 4.3, means that the sum of all the datagrams for a given blob $b$ distributed on all robots in $\mathscr{R}$ is exactly the number of datagrams of blob $b$. In other words, all the datagrams must be allocated, and a datagram can be allocated to only one robot. For the sake of simplicity, despite the fact that the datagrams could be replicated, we assume resource-constrained robots and we do not consider datagrams replication in this work, eq. 4.3 captures the replication constraint. The constraint in Eq. 4.4 sets the maximum number of datagrams held by a robot as its storage capacity.

### 4.2.2 Reconstruction cost

The function to be minimized (Eq. 4.1) is used in a decentralized auction-based algorithm to allocate each blob to the best (i.e., lowest cost) candidate. The proposed solution for allocating the blobs consists of an auctioneer robot that determines how to distribute a blob's datagrams in the network to minimize the cost of reconstruction at each robot $p \in \mathscr{P}$. Each robot can bid on the blob with its resource state vector $s^r$ together with a set of all processors' distance to this robot, $X^r$. The auctioneer takes the bids from all robots and computes the reconstruction cost. This process can be viewed as a market auction soliciting from a set of subcontractors to reach the lowest price. The reconstruction cost $c^r$ of robot $r$ is called the price $p^r$ from here on, because it is computed from a bid and used in an auction. The overall process of blob allocation is solicited as a decentralized auction problem instead of a direct assignment from the capturer because of a lack of global information, i.e., lack of datagram storage cost for a given robot (proximity to the processor) and its current storage capacity. Given the datagrams generated from the blob, the auctioneer assigns them to the robots that submitted bids in a way that will minimize the cost of reconstruction. The processor's proximity is the primary cost metric utilized to label the datagram allocation and reduce the overall cost of reconstructing the blobs at the processor robots.

**Definition 2.** *Given an auctioneer with a set of datagrams generated from a new blob* $b, d_b = \{d_1, d_2, ..., d_k\}$*, which require to be allocated for storage, and a group of subcontractors submitting one service proposal each,* $E = \{e^1, e^2, ..., e^n\}$*: A proposal is a triplet* $e^j = \langle b^j, s^j, X^j \rangle$*, where* $b^j$ *is the blob identifier,* $s^j$ *the storage resource state of robot* $j$ *and* $X^j$*, the proximity of processors, with subcontractor's* price $p^j$ *proportional to* $x^j \in X^j$*.*

**Definition 3.** *Given that the price* $p^j \geq 0\ \forall j \in \{r1, r2, ..., rn\}$*, the winner determination problem is to label the proposals as winning* ($a_j = 1$) *or losing* ($a_j = 0$) *so as to minimize the auctioneer's expense under the constraint that each item (datagram) can be allocated to at most one subcontractor:*

$$\min_{Q} \sum_{b \in \mathscr{B}} \sum_{r \in \mathscr{R}} p^r_b(d_b) \tag{4.5}$$

*subject to constraints of Eq. 4.2, 4.3 and 4.4. SOUL computes the price using*

$$p^r_b(d_b) = \sum_{d \in d_b} \sum_{x^r \in X^r} alloc(d) * x^r, \tag{4.6}$$

in which $alloc(d)$ is 1 if $d$ is allocated to robot $r$ and 0 otherwise. Eq. 4.6 determines the price of storing all the datagrams of blob $b$ within the swarm, where price is in terms of distance. In Eq. 4.5, the sum of all prices for all robots in $\mathscr{R}$ and all blobs in $\mathscr{B}$ determines the cost of reconstructing all the blobs at all processor, assuming one datagram is transmitted every time step. The cost expressed in Eq. 4.5 is a computation of reconstruction time that heavily depends on the network topology and the sub-swarms configuration because of $X^r$ in Eq. 4.6. The SOUL mechanism detailed in the following section, is an algorithmic implementation of definitions 1 to 3.

## 4.3 SOUL implementation

The overall process of blob injection and retrieval can be summarized as the sequence: injection, auction, distribution, and retrieval. When a robot wishes to share a blob within its swarm, the robot acts as the auctioneer for that blob and broadcasts a blob-advertising message. A robot $r$ with available resources responds with a suggested distribution $e^j$, its *bid*, with $s^r$ and $X^r$ defined in the previous section. The proximity to a processor $x^r$ is defined as the hop count, measured using periodic local broadcasts from the processor: all neighbors receiving this broadcast send a proximity of 1 on their bids and re-broadcast to their neighbors, and so on, forming a "hop count gradient" across the swarm. The auction mechanism used within SOUL is similar to [104], with certain modifications and adaptations to fit SOUL.

The auctioneer collects all the bids and sends the allocation request to the robots selected by the auctioneer to store the blob's datagrams. A robot receiving such an allocation message reserves the required space for the incoming blob's datagrams and, if the required size is not above its available space, sends an acceptance reply. Otherwise, the robot refuses the allocation request with a rejection reply. The allocation messages from an auctioneer grant the possibility to handle multiple blobs in parallel with multiple auctioneers auctioning blobs across the swarm. When receiving an acceptance reply, the auctioneer transfers the current datagram in the queue of the datagram map $d_b$. If the auctioneer receives a rejection reply, it tries an allocation request to the next robot in a sorted list of bids. For every allocation accepted, before the blob's datagrams are transferred to another robot, the auctioneer adds the allocation to the shared locations list as a tuple $\langle$robot id,datagram ids$\rangle$ and broadcasts this entry to all the robots in the swarm. If a robot needs to reconstruct a blob, it looks up the shared locations list and requests the corresponding datagrams from the robots holding them. If the auctioneer robot does not receive enough bids to allocate all datagrams within the bid allocation interval, it assumes either the communication is malfunctioning, or there are not enough resources available to handle the new blob.

In this work, network connectivity is expected to be *generally* maintained, for instance with a dedicated controller [105]. Temporary disconnections are dealt with through several retries until the process reaches a timeout and considers the blob lost or the system unable to store an additional blob. We tackle storage resource exhaustion with a set of policies to remove the blob from the network. The policies use local information such as the age of the blob and its priority to decide if it should be removed. In this work, the policy we use is that the blob with lowest priority and highest age gets removed first. The auctioneer robot creates a sorted list of blob indexes with the help of the above policy. In our approach, we start removing blobs once the robots hit their storage limits, but we could also save them to a file, if possible and re-inject them into the network when the resources are available again.

It is worth noting that in this paper we describe SOUL as acting over a whole swarm, but it can seamlessly operate on sub-swarms (e.g., due to disconnection from other sub-swarms in a swarm deployment). Yet, to foster maximal performance, it might be beneficial to maintain at least a spanning tree over the whole swarm.

### 4.3.1 SOUL Processes

The SOUL mechanism consists of four processes: Bid Generation Process (BGP), Blob Update Process (BUP), Lists Update Process (LUP), and Blob Request Process (BRP), as illustrated in Fig. 4.2. Every member of the swarm runs the exact same algorithms and has an identical implementation of these four processes.

#### Bid Generation Process

Algorithm 1 shows the pseudo code of the BGP. When a new blob is injected by a robot $r_i$ using the `putblob` command, this latter executes the procedure on line 1. Robot $r_i$, as an auctioneer, creates at first an MD5 hash with the content of the blob, which is used as the unique internal identifier for the blob as well as during its reconstruction to verify its integrity. Then, the blob is split into $N_{c_q}$ identically-sized datagrams and inserted into the datagram map $d_b$. Following this, the new blob is advertised to other robots and a new auction is created and added to the auction list. The procedure on line 6 is initiated when an advertise blob message is received. The robot $j$ that receives the message checks its current available storage $s^j$ and sends a bid triplet to the auctioneer. The procedure on line 11 collects all new bids in a list when executed on the auctioneer robot. Line 14 shows the datagram allocation procedure: the only periodic procedure executed in the BGP. When the auction list is not empty, the robot looks for any entry that has reached a timeout, and sorts the bidders list in ascending order of cost, such that it minimizes the reconstruction cost (computed using proximity to processor) by solving the Eq. 4.5. The datagrams of blob $b$ are allocated to the robots in the sorted bidders list. The procedures on line 29 and 32 show, respectively, the behavior when a blob allocation is accepted, or rejected by a robot. When an allocation is accepted, the BGP requests the BUP to allocate $s^j$ datagrams to robot $j$. Whenever an allocation is rejected, the datagrams are allocated to the next robot in the sorted list. If there are no more resources to be allocated in the list, one of the blobs is removed using SOUL policies as described in Sec. 4.3.1, and the blob is allocated to the robots that were holding the now-removed blob.

Algorithm 1 The Bid Generation Process (BGP) pseudo code

```
 1: procedure PUTBLOB($k, b$)                                  ▷ New blob $b$ with key $k$
 2:     $(d_b, h_k) \leftarrow SplitHashBlob(b_k)$
 3:     $AdvertiseBlob(b_k, h_k)$
 4:     $\langle Auctionlist \rangle \leftarrow NewAuction(b_k, h_k)$
 5: end procedure
 6: procedure ADVERTISEDBLOB($b_k, h_k$)
 7:     if $s^j > 0$ then                                      ▷ If storage space is not full
 8:         $SendBid(b^j, s^j, x^j)$                           ▷ Bid for the blob
 9:     end if
10: end procedure
11: procedure BIDRECEPTION($b^j, s^j, x^j$)
12:     $\langle Bidlist_k \rangle \leftarrow (b^j, s^j, x^j)$       ▷ Insert to bid list k
13: end procedure
14: procedure DATAGRAMALLOCATION($Auctionlist, Bidlist$)
15:     foreach $A \in Auctionlist$ do
16:         if $A.Time \geq TimeOut$ then
17:             $Bidlist_k \leftarrow SortBid(Bidlist_k)$           ▷ Using Eq. 4.1
18:             $i_k \leftarrow 0$
19:             while $Datagramstoallocate \neq 0$ do
20:                 $r_i \leftarrow Bidlist_k[i_k].r_i$
21:                 $r_i \leftarrow assign(r_i, s^j)$              ▷ $s^j$ datagrams to $r_i$
22:                 $i_k \leftarrow i_k + 1$
23:             end while
24:         else
25:             $A.Time \leftarrow A.Time + 1$
26:         end if
27:     end for
28: end procedure
29: procedure ALLOCATIONACCEPT($b^j, s^j$)
30:     allocatedatagram($s^j$, j)
31: end procedure
32: procedure ALLOCATIONREJECT($b^j, s^j$)
33:     if $i_k < |Bidlist_k|$ then
34:         $r_i \leftarrow Bidlist_k[i_k].r_i$
35:         $r_i \leftarrow a(r_i, B^k_{r_i})$
36:         $i_k \leftarrow i_k + 1$
37:     else
38:         $BR \leftarrow FindBlobToRemove$                      ▷ Using Policies
39:         $r_i \leftarrow BR.r_i$
40:         $r_i \leftarrow aloc(r_i, s^j)$
41:     end if
42: end procedure
```

### Blob Update Process

The BUP's procedures aim at managing blob allocation, and datagram transmission and reception as illustrated in Algorithm 2. The procedure shown in line 1 transfers the next available datagrams from the datagram map to robot $j$ (that has accepted the allocation) by updating the locations list and the available list (more details on these lists can be found in sec. 4.3.1). A robot receiving a datagram $d$ inserts it into the datagram map $q_b^r$ and updates its available list as shown in line 7. If the available list contains all the datagrams of the blob, then the state of the blob is set to be "ready". When receiving a datagram request, the robot checks for the availability of the requested number of datagrams and sends the datagrams to the requesting robot $r_i$.

Algorithm 2 BUP's pseudo code

```
1:  procedure ALLOCATEDATAGRAM($s^j, j$)
2:      $j \leftarrow t(j, d_b[i] : d_b[i + s^j])$            ▷ Transfer $s^j$ datagrams to $j$
3:      $i \leftarrow i + s^j$                                  ▷ Store datagram index
4:      $UpdateAvailableList(d_b)$
5:      $UpdateLocationsList(j, s^j)$
6:  end procedure
7:  procedure RECEIVEDATAGRAM($d_b[a]$)
8:      $\langle d_b \rangle \leftarrow d_b[a]$                  ▷ Insert $d_b[a]$ datagram into $d_b$
9:      $UpdateAvailableList(d_b)$
10:     if AvailableListComplete then
11:         $BlobState \leftarrow Ready$
12:     end if
13: end procedure
14: procedure REQUESTEDDATAGRAM($s^j, r_i$)
15:     if $|d_b| == s^j$ then
16:         $SendDatagram(d_b, r_i)$
17:     end if
18: end procedure
```

### Blob Request Process

When a robot wants to reconstruct a blob, the robot executes the `getblob` function as in Algorithm 3. At first, the robot checks whether all datagrams of the blob are locally available by querying the state of the blob. If the state is "buffered", the blob is reconstructed from the datagram map, verified using the blob's hash, and made available. If not all the datagrams of the blob are available, the auction process may not be complete yet, or the locations list is not up-to-date. In both cases the robot requests the auctioneer for the blob, and the auctioneer, depending on its current state, either sends the remaining datagrams or the updated locations list. The auctioneer robot is identified by storing its id during a blob advertise broadcast. The procedure on line 17 is initiated when a robot requires a blob before the auction is complete. In other words, a robot receives a blob as soon as possible, whenever it is actively looking for it, either during auction or storage.

Algorithm 3 BRP's pseudo code

```
 1: procedure GETBLOB($k$)
 2:     if BlobState == ready then
 3:         $b_k \leftarrow ReconstructBlob(d_b)$
 4:         $b_k \leftarrow VerifyHash(b_k, h_k)$
 5:         return $b_k$
 6:     else
 7:         if LocationsListComplete then
 8:             foreach $j \in Locationslist$ do
 9:                 $RequestDatagrams(j, s^j, r_i)$
10:             end for
11:         else
12:             $AuctineerBlobRequest(b_k, r_i)$
13:         end if
14:         return 0
15:     end if
16: end procedure
17: procedure AUCTINEERBLOBREQUEST($b^k, r_i$)
18:     if AuctionComplete then
19:         foreach $j \in Locationslist$ do
20:             $RequestDatagrams(j, s^j, r_i)$                ▷ Request to $r_i$
21:         end for
22:     else
23:         $TerminateAuction(k)$
24:         $SendDatagram(d_b, r_i)$                      ▷ Send datagram map
25:     end if
26: end procedure
```

### List Update Process

Each robot in the network maintains two lists (the available list and the locations list) and a state for every known blob. The available list contains the index of all available datagrams of the blob, while the locations list contains an entry for each robot holding the datagrams of the blob. Every entry of the locations list contains the robot ID and the IDs' of datagrams placed at that robot. The LUP maintains and updates the lists and the state whenever a change occurs. Lists on each robot are used for quick lookup of the availability of datagrams and their locations.

### 4.3.2 Buzz and SOUL

As stated earlier, SOUL is built on top of Buzz, which is a domain-specific programming language for robotic swarms executed on a custom virtual machine. The Buzz Virtual Machine (BVM) is the core of Buzz that executes byte code compiled from a Buzz script. Buzz provides a range of programming primitives tailored for robotic swarms: for more details on Buzz primitives and virtual machine we refer the reader to [56]. The SOUL implementation is integrated into the BVM, for the swarm research community to use and extend.

In [68], the authors presented the Buzz implementation of the virtual stigmergy, a shared (key,value) pair among the robots in a swarm, with mechanisms to ensure convergence. Inspired by this work, we designed SOUL as a shared `(key,blob)` within Buzz. The interface of SOUL was made to be identical to that of the virtual stigmergy, but its implementation substantially differs (for instance because of the auction and allocation processes). SOUL offers typical get/set routines to manage the blob tuple with a unique key and a set of functions to inquire on the characteristics of a blob (size, status, potential sink, etc.). When the auction is completed, a function (`getblobseqdone`) notifies any interested robot.

### 4.3.3 Implementation specifics

The robots participating in SOUL operations for injecting, storing and reading datagrams, create a SOUL object with a unique identifier. Each SOUL object can share multiple blobs, each identified by its key.

**Blob storage container** Every new blob (advertised or injected), creates a container within the respective SOUL object to locally store certain meta-data associated with the blob on each robot. Each storage container contains the meta-data shown in table 4.2 to identify and manipulate a given blob entry. Each SOUL object contains a map of these local containers associated with its key.

**Writing** Whenever a robot needs to share a blob with the swarm, it executes a `putblob` command with ⟨key,blob⟩ as argument. The `putblob` command first creates and updates the local container with all its associated data. Following this, the blob is advertised with a message containing ⟨PUT, size, key, hash, robot id⟩, any robot receiving this message creates

Table 4.2 Blob storage container data structure

| Data | Type |
|---|---|
| Blob hash | 32 bit unsigned int |
| Blob size | 32 bit unsigned int |
| Priority | 8 bit unsigned int |
| Available list | Array |
| Locations list | Array |
| Robot role | 8 bit enumeration |
| Blob status | 8 bit enumeration |
| Request timer | 16 bit unsigned int |
| Blob datagrams | Hash map |

a blob storage container and updates its key, hash, size, and, blob status. Finally, an auction is initialized to disperse the blob datagrams across the swarm in a way that minimizes the reconstruction cost at all $p \in \mathscr{P}$ following BGP.

**Reading** The approach towards reading a blob is two-fold:

1. updating the blob container, and,
2. updating the datagram container.

At first, the robot has to determine the existence of a blob before attempting to update the datagram container and read the blob. Reading a blob is achieved with a series of commands: `get`, `getstatus` and `getblob`. The need for updating the blob containers is to tackle network disconnections.

Initially, a `get` command is executed with a key to determine the existence of the blob. If the robot is aware of a blob's existence, it returns the hash associated with the blob from the local container, otherwise a NIL value. After every read, a get message containing $\langle$GET, size, key, hash, robot id$\rangle$ is broadcast. Whenever a neighbor receives a NIL hash from a get message and has a valid local hash for the blob, it broadcasts a PUT message to all its neighbors, in an attempt to update the value on a robot actively looking for the blob [68].

A robot with a valid hash and blob container (determined with the `get` command), executes the `getstatus` command along with the associated key, which returns the blob status from the local container and requests for datagrams, from each robot holding the datagrams. If the blob container does not exist, it returns NIL. Every time `getstatus` command is executed the robot broadcasts a get message, in an attempt to update the blob container.

Once the status of the blob switches from buffering to buffered (when all datagrams are locally available), the robot executes a `getblob` command to read the blob. The `getblob` command attempts to reconstruct the blob from the datagram container and verifies its integrity using its hash. In general, executing a `getblob` will result in a similar effect of executing the three commands as in listing 3, however the latter provides more control over the process.

**Inter-agent communication** The inter-agent communications are achieved with two FIFO queues:

1. Broadcast queue, and,
2. unicast queue.

The messages from the broadcast queue are gossip based, propagating in the neighborhoods of the robots. During these broadcasts, duplicate broadcasts are eliminated by a fast look-up hash table and with countdown timers to forget messages recently broadcast. Each message of the unicast queue is sent to a specific receiver. The messages from the queues could be removed and sent to a middleware that handles the communication, allowing hardware-independent deployment. For instance, communication can be implemented with a TCP/IP-based ad-hoc network like batman-adv [106] or with XBee modules with the DigiMesh protocol [107], as both provide message broadcast and unicast with guaranteed packet delivery. In figure 4.2, the broadcast messages are indicated with B, unicasts with U and local updates with L.

## 4.4 Simulation Experiments

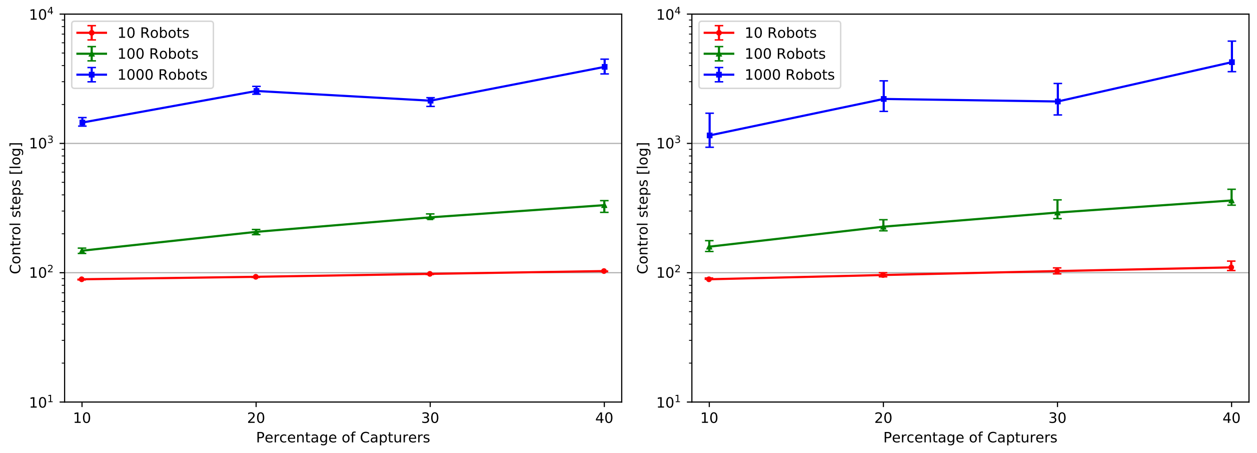
Simulation experiments aim at studying the performance of the SOUL model presented in the previous sections over diverse conditions. Simulations were performed with a physics-based multi-robot simulator, ARGoS3 [96].

**Performance metrics** The performance measure used during the simulations was the time taken for allocating the datagrams of a blob to the robots in the network, i.e., the convergence time of SOUL. The reconstruction time of a blob by any given robot can be computed with the communication delays between the robots holding the datagrams and the robot requiring it. It is desirable to minimize the convergence time to avoid delays in subsequent reconstruction. During simulations, we used the number of control steps as a measure of convergence time, with each control step lasting 100 ms.

**Communication model** A communication model that simulates the peer-to-peer communication [108] was built within ARGoS3. As in any Mobile ad hoc network (MANET), the model provides both broadcast to neighbors within range, and unicast to any given robot in the network. Unicast messages are sent to their destination by computing the shortest path with a local routing table. The messages of SOUL use broadcast in most cases, and unicast in a conservative manner, mostly when handling large datagrams, to minimize contention bandwidth use. We assume any given message between the robots in the network has a probability to be dropped.

We use TCP/IP as the communication model in our simulations (although XBee would be another option), so we assume to have guaranteed delivery of messages and we simulate the effect of packet drop on the model as the delay incurred due to transmission errors and retries. The communication model can be expressed as a weighted adjacency graph: for a network of $N$ robots the adjacency matrix is $A = [\beta_{ij}] \in \mathbb{R}^{N\times N}$ with $\beta_{ij}$, the packet drop probability [109]. The entries in the adjacency matrix evolve over time during the simulation depending on the topology of the network and the packet drop rates.

**Experimental setting** We use two different topological configurations: cluster and scale-free. The former places the robots in a compact cluster with a uniform distribution $\mathcal{U}(-L/2, L/2)$, with $L$ the boundary of the arena. The latter, distributes the robots in small clusters connected by sparse inter-cluster connections using Barabási-Albert's preferential attachment algorithm [110]. For details on these topologies, we refer the reader to [68]. During simulation, we use multiple packet drop probability values $P \in \{0, 0.25, 0.5, 0.75\}$. Each robot has a storage limit of 1,000 datagrams, and injected images of 1.02Kb in size as blobs, these blobs were split in 1,022 datagrams each.



(a) Cluster topology.

(b) Scale-free topology.

Figure 4.3 Scalability analysis with 10, 100 and 1000 robots.

### 4.4.1 Scalability analysis

We studied the dependence of convergence time on network topologies and on the number of robots $N \in \{10, 100, 1000\}$. In particular, we assessed SOUL's scalability using static robots (fixed topology) for the percentage of capturer robots $\mathscr{C} \in \{10, 20, 30, 40\}\%$ and with varying number of robots $N$ over different trials. The percentage of processors $\mathscr{P}$ was set to 10%, and all other robots without a role were by default networkers, $\mathscr{N}$. In general, robots can have different roles for different blobs depending on the operation performed with SOUL, as illustrated in figure 4.1, with different robots injecting blobs. In these simulation experiments, we fix the robot roles by injecting blobs from a given set of robots, to study the performance. For instance, with $\langle N = 100, \mathscr{C} = 40\rangle$, 40 robots are capturers injecting blobs simultaneously, 10 robots are processors, and 50 are networkers. The role assignment was performed in such a way as to obtain random roles for robots over different trials and it was fixed throughout an experiment trial. we repeated each experimental setting with 35 random placement and roles.

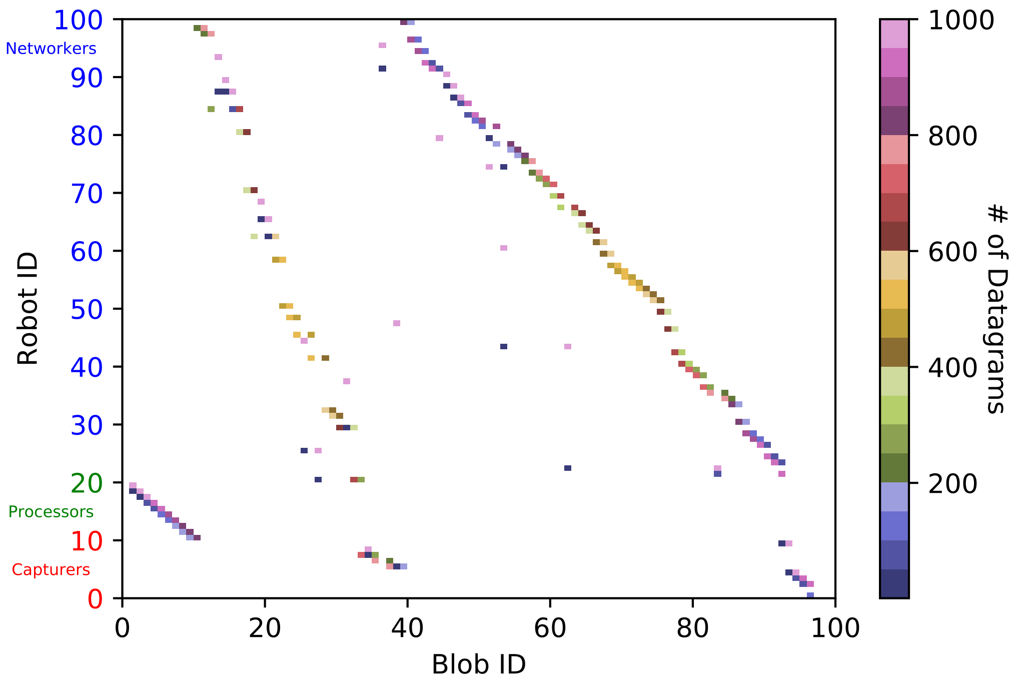


Figure 4.4 Datagram distribution in a 100 robot network, as a result of continuous injection of blobs.

**Results** *SOUL scales with increase in capturers for up to 1,000 robots for both topological distributions tested.* Fig. 4.3 reports the results of our scalability experiment. In both cluster (fig. 4.3a) and scale-free (fig. 4.3b) topologies, the convergence time increases with the percentage of capturers, as more robots were simultaneously injecting blobs. The global trend among both topologies seems similar, however, the time taken has increasing variation between different topologies as the number of robots increases. For instance, with 1,000 robots and 10 percent of capturers the median time is 1,073 control steps with scale-free and 1,450 control steps with cluster topology.

With $N = 10$ both topologies consumed around 100 control steps to converge, whereas with $N = 100$ a steady sub-linear increase in convergence time is observed with the increase in percentage of capturers. With $N = 1,000$, the convergence time incurred large variations in scale-free topology. This can be explained with the change in sparse communication paths for the scale-free topology over different trials. In cluster topology simulations, the variations in convergence time with $N = 1,000$ are smaller compared to scale-free. This topology's dense communication path, which allows persistent data transfer, is most likely the source of stability in the measurements.

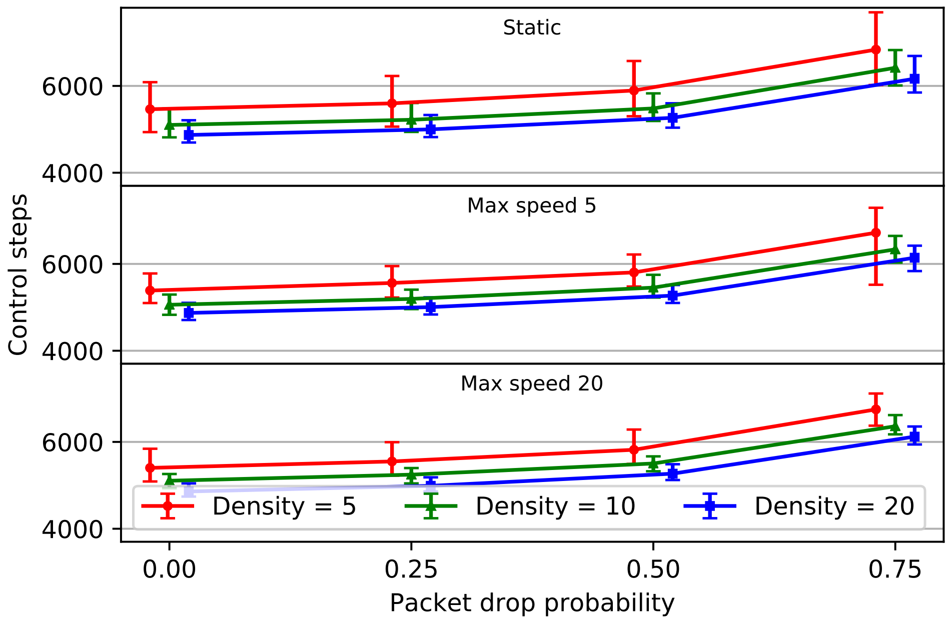


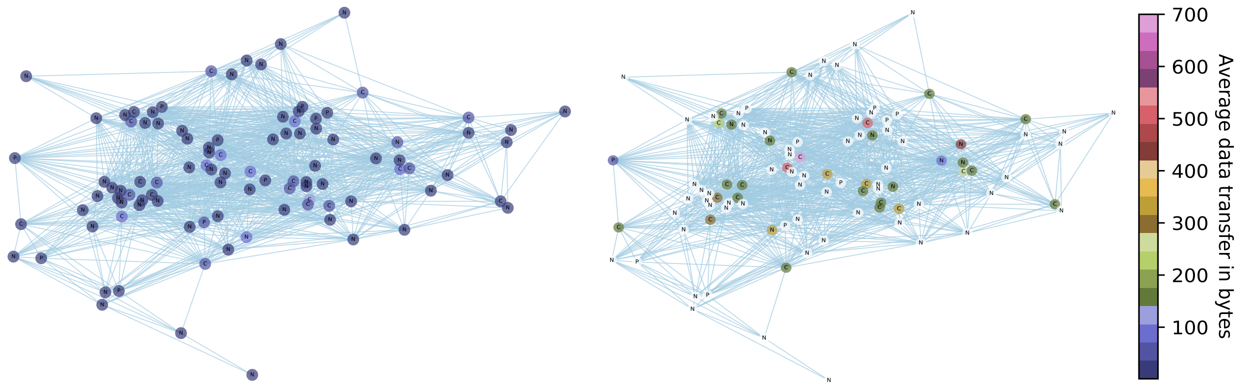
Figure 4.5 Packet drop rate's influence on the time (control steps) required to allocate datagrams for various density of moving robots (markers are with slight offset for visual clarity).

Fig. 4.4 plots the datagram distribution of the robots during one of the experimental trials with $\langle N = 100, \mathscr{C} = 10\% \rangle$. During this trial, the blobs were continuously injected one after the other until the network dropped the first injected blob. The 100-robot network lost the first blob when injecting the 98th blob: the swarm was able to buffer 97 blobs, up to saturation of the available storage space, with each robot storing atleast a datagram. The distribution of datagrams can be grouped into three classes corresponding to the three sloped distributions in Fig. 4.4, from left to right: the first group is composed of processor robots (data sinks) where the first 10 blobs were placed for fast reconstruction; the second group corresponds to the networker/capturer robots that are neighbors of processor robots, thus obtaining the following blobs; the final group contains all other robots having the lowest priority in the network because they are distant from processors. From this three distribution classes stated above, the effect of Eq. 4.5 minimizing the cost of reconstruction is observable, with placement of datagrams to the closest location possible to a processor. It is worth noting that the allocation always starts from the highest robot ID because during the allocation using Eq. 4.5, robots break ties using robot IDs.

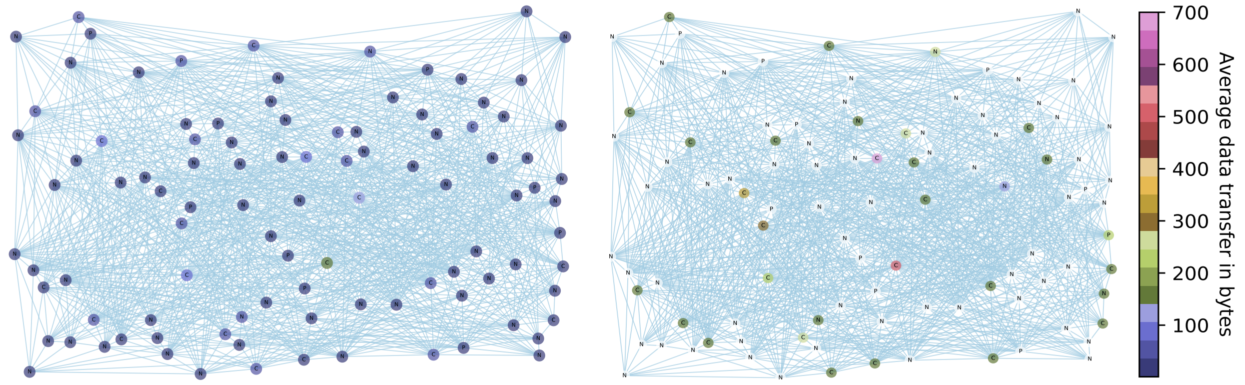
### 4.4.2 Dynamic Analysis

We performed a set of experiments to study the influence of change in topology on convergence time with moving robots. These experiments involve robots following a simple diffusion control algorithm [111] to produce random motion over the arena. Blobs were injected continuously one after the other until the swarm dropped the first injected blob. The time taken to reach an injected blob loss is reported in fig. 4.5. The simulation arena during the motion experiments had four obstacles to stimulate changes in topology. To study the influence of the velocity of change in topology during these experiments, we considered different densities $D \in \{5\text{(loose distribution)},10\text{(medium distribution)},20\text{(tight distribution)}\}$ and maximum speeds for the robots $M \in \{5, 20\}$. The density of the robots was changed by moving the boundaries of the simulation arena [102]. In this set of experiments, $N$ was set to 100 robots and packet drop probability $P \in \{0, 0.25, 0.5, 0.75\}$.

**Results** *The change in topology caused by moving robots has minimal influence with SOULs' allocation mechanism.* Fig. 4.5 reports the time steps required to remove the first blob with static robots $M = 0$ (top), then moving at $M = 5$ (middle), and then moving at $M = 20$ (bottom). The static robot case is analogous to the cluster topology reported in Sec. 4.4.1. In all cases, the time to lose a blob follows a similar trend and arise by saturating the available storage space, i.e., after injecting 97 blobs into the swarm as in previous set of experiments. With packet drop probability of up to 0.5, the network's performance degraded slowly, taking less than 6,000 control steps to lose a blob. For P = 0.75 the time to lose a blob quickly increased over 6,000 control steps. The static topology with $M = 0$ and dynamic topology $M = 5$ reported large variability in convergence time with respect to the dynamic topology $M = 20$, probably due to the fixed and slow communication link variations for a given trial.



(a) Scale-free topology.



(b) Cluster topology.

Figure 4.6 Average bandwidth consumption: left with SOUL, right as a result of one-to-one transmission.

### 4.4.3 Bandwidth consumption

**Results** *SOUL lowered the average bandwidth consumed and evenly distributed the bandwidth requirements over the network.* Fig. 4.6a and 4.6b report the average bandwidth consumed by scale-free and cluster topologies respectively, when compared with unicasting the blobs from a single robot. In this simulation setup, the blobs are injected from 10 different robots in the network with $\langle N = 100, \mathscr{C} = 10, \mathscr{P} = 10$, and the maximum usable bandwidth is set to 700 bytes per step. These experiments allow to measure the average bandwidth consumed until the first blob is lost, i.e., after injecting 97 blobs. With a scale-free topology, the average bandwidth consumed by all the robots using SOUL is less than 100 bytes per step. In comparison, the unicast average bandwidth usage was over 100 bytes per step, but with some robots using the maximum available bandwidth while the majority of the robots use nothing. A similar effect happens with the cluster topology.

## 4.5 Experiments

SOUL was studied under two realistic experimental scenarios with a swarm of 10 Khepera IV robots [97]. Our experimental platform consists of an Optitrack system (an IR camera based tracking system), a centralized wifi-based software communication hub and a swarm of 10 Khepera robots. Our SOUL implementation requires situated communication [112], a common concept of swarms providing each inter-robot message with the relative distance and bearing of the sender. Situated communication is emulated by transmitting all robots' messages to each other swarm member through the software communication hub, named Blabbermouth. For a more detailed schematics of blabbermouth, we refer the reader to fig. 8 of [113]. Blabbermouth also computes the relative position from the tracking system and appends it to each message. A laptop was running Blabbermouth dealing with situated communication and a peer-to-peer network path using TCP/IP over wifi. This configuration by default generates a fully connected network topology, but the transmission range can easily be limited in software using each robot's position to create different topologies. The implementation of SOUL and the experimental buzz scripts can be found in our repository [1].

In order to test and demonstrate the range of applications SOUL can be fit for, we designed two experiments: one to backup a failing robot (SOUL backup) and the second to manage larger data transfer in a heterogeneous mission (YOLO Search). Fig. 4.7 illustrates both experimental scenarios: SOUL backup on the left and YOLO Search on the right.

### 4.5.1 SOUL backup

For this first experimental scenario, the robots were performing a cordon around an object of interest. Patrol and cordon exercises are frequent in emergency response, and swarms are well fitted for the task [114]. Yet, maintaining the surveillance perimeter around a sensible structure is challenging in terms of robot autonomy. As depicted in fig. 4.7, when one of the robots predicts an incoming failure (e.g., low battery), it serializes its current state and injects it into the swarm with SOUL. The robot identity is then available to be picked by any other member of the swarm.

On top of the state of its current task, this backup includes the unique robot id, all of its global variables and the key data structures of its BVM. In [102], the authors state that describing the dynamics of a swarm through a finite state machine allows the developers to focus on the swarm behavior design rather than the individual robot behavior. Copying the key data structures of this finite state machine is what allows this backup-and-resume

[1]https://github.com/vivek-shankar/SOUL.git

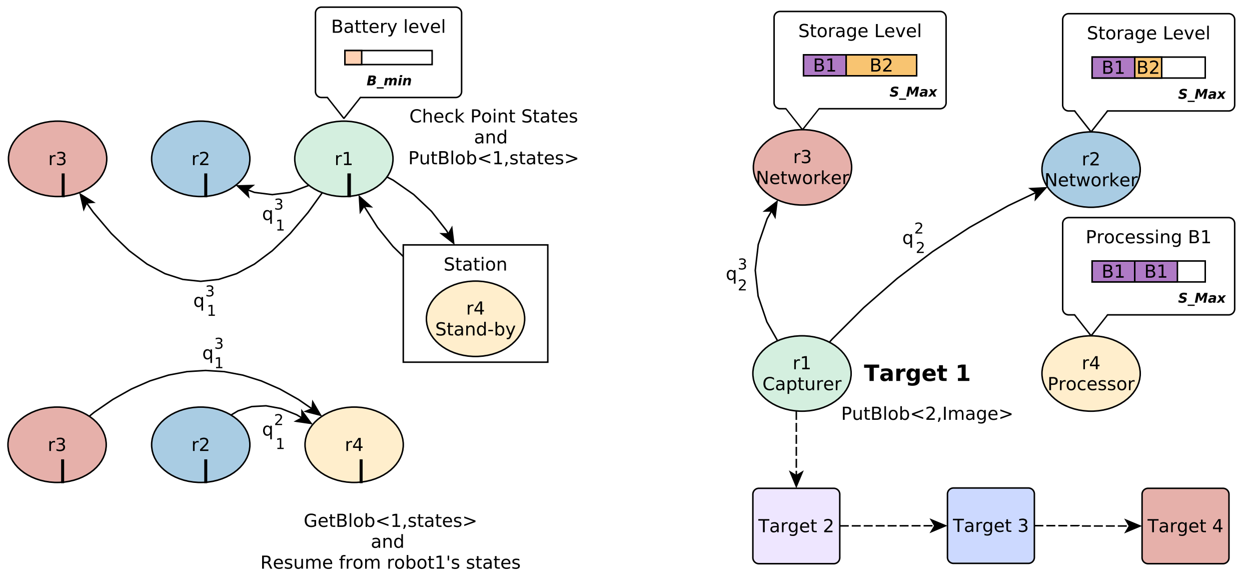


Figure 4.7 Schematic representation of the two experiments conducted on the Khepera robots: SOUL backup to the left and YOLO Search to the right.

mechanism with a new robot. In the experiment, a standby robot waiting aside the formation is asked to replace the failing one and obtain its backup from SOUL. It can then seamlessly resume the mission and be perceived by the swarm as the fallen member. The chronology of the process is illustrated in fig. 4.9 with regards to the datagrams stored in the swarm, while Algorithm 4 shows the control rules followed by the robots.

As shown in fig. 4.8, the cordon is created with nine Khepera IV robots (numbered r1 to r9) in order to surround a box. A stationary robot (r10) waits aside the arena, as a replacement unit. Each robot blinks with a LED pattern indicating their unique ID and a frequency increasing with the mission time. When r6, the failing robot, observes a critical level of its battery, it triggers its identity backup (ID, states and BVM structure) and injects it as a serialized data blob into SOUL.

Moreover, with swarms of robot, a failing robot will be inherently compensated by the swarm with a cost on mission performance. The backup experiment shows how one can use SOUL to minimize the loss of performance by replacing the failing or failed robot with a robot that exactly matches its state before the failure. This is particularly useful in cases where robots need to leave the swarm to replace their batteries: a charged robot continues from the point where the “failing” robot left. Robot r10 gets the blob datagrams and rebuilds it using SOUL. It then sets its current state with blob’s content, seamlessly taking r6’s place in the swarm. Once robot r10 resumes the mission of r6, it joins the formation and start to patrol. This experiment was repeated ten times to confirm a consistent behavior.

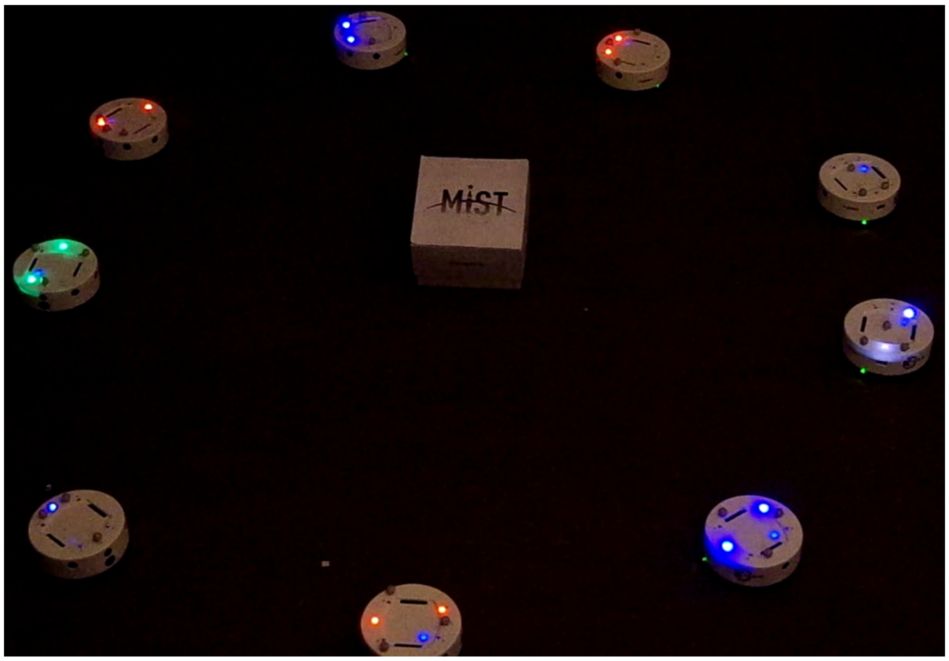


Figure 4.8 Nine wheeled robots patrolling around a box. The LED patterns show their ID.

**Results** *For all experiment trials, the robots were able to successfully backup their identity and resume the mission consistently.* Fig. 4.9 reports the number of datagrams of each robot on one of the trials. When r6 was close to fail it created a backup blob of five datagrams at 1.1s. The datagrams of this blob were then allocated to robot r2, r3 and r4, which were the closest to the replacement robot r10. r10 obtained all the datagrams from these three robots at 4s, and, after reconstruction, removed the blob, allowing the other robots to remove their datagrams for this blob.

Considering all trials, it took an average of 2.3s to create, inject and distribute the blob with SOUL. The full removal process took an average of 3.2s. The backup blob was always distributed over three robots, since the storage resource limit was set to two datagrams for all robots in this experiment and backup blobs were 5 datagrams in size.

### 4.5.2 YOLO Search

The second experimental scenario represents a search mission executed by a swarm of wheeled robots. As shown in fig. 4.7, we arranged four targets, spread around the robots' arena. The goal was to take a picture and detect a key object in it at each location. However, no robot had both the camera and the processing power required, and the swarm was heterogeneous: following their on-board hardware, the robots were assigned different roles. One robot had a

Algorithm 4 SOUL backup experiment pseudo-code

```
 1: procedure Mission Step
 2:     if !depot then                                          ▷ Main mission loop
 3:         if Swarm.size() $\geq$ required_robots then
 4:             if Battery.capacity $<$ 30 % then
 5:                 $blob = get\_mission\_data()$
 6:                 $SOUL\_obj.putblob(blobid, blob)$
 7:                 $Swarm.leave()$
 8:                 $Navigate\_to\_depot()$
 9:                 $depot = 1$
10:             else
11:                 $Circle\_pursuit()$
12:             end if
13:         else if Swarm.Size() $<$ required_robots then
14:             $return$                                     ▷ Wait for required no. of robots
15:         end if
16:     else                                             ▷ Executed by the replacement robot
17:         if Swarm.size() $<$ required_robots then
18:             if $blobid$ exists then
19:                 if $blobid$ is buffered then
20:                     $blob = SOUL\_obj.getblob(blobid)$
21:                     $set\_mission\_data(blob)$
22:                     $SOUL\_obj.putblob(blobid, nil)$
23:                     $Swarm.join()$
24:                 end if
25:             end if
26:         end if
27:     end if
28: end procedure
```

working camera, r4, and was attributed the capturer role. One other robot had a powerful processing GPU required to analyze the data, r10, and was attributed the processor role. All other robots were the networkers, moving around randomly to create a dynamic network topology. In order to test the behavior with dynamic topology, the communication range of the robots was limited to 1.1m. The processor was also moving randomly while waiting for images to process using its GPU-optimized algorithm software: YOLO [115], a popular deep learning object detection system with state-of-the-art accuracy. The capturer was moving to pre-defined target positions, taking pictures at each location.

Ten Khepera IV robots were used: one capturer, one processor and eight networkers, and all robots were using a rule set as in Algorithm 5. The arena was 2m×2m with four tar-

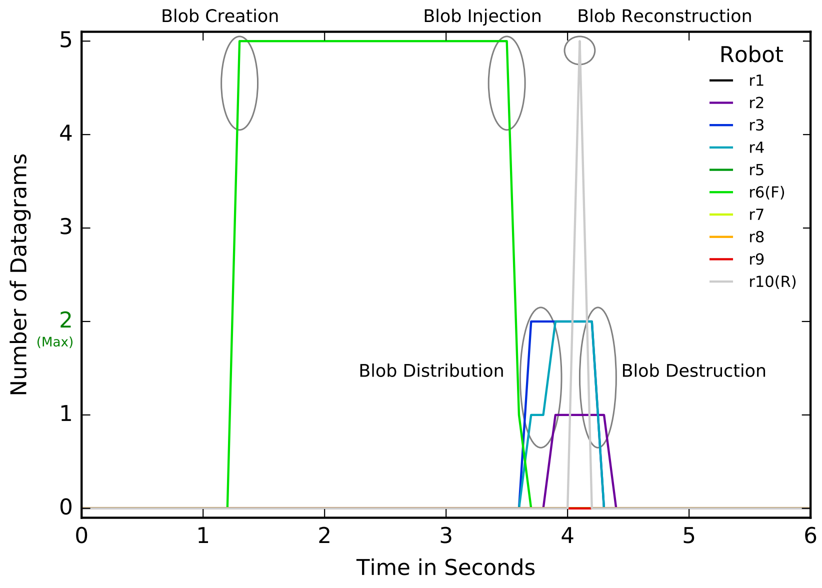


Figure 4.9 Storage usage throughout one trial for the SOUL experiment.

get locations, as illustrated in fig. 4.10. Each location had a different object, selected from the training set of YOLO3. The processor had a Nvidia TX1 board to process the images. All robots were initially placed on the boundaries of the arena. The capturer moved from target to target, each time injecting a blob with SOUL. The processor and the networkers moved randomly in the arena. All robots were equipped with a collision avoidance mechanism, reacting to near obstacles with an incremental steering velocity [116]. The experiment was repeated ten times, each time generating different trajectories and undergoing different completion time due to the random motions and the reactive collision avoidance mechanism.

The storage limit on all networkers was set to 200 datagrams while the images produced were around 500 datagrams. The capturer, r4, injected blobs into the swarm, as soon as it captured an image of the target. The processor, r10, waited for a blob to be available in the stigmergy whenever it was not processing an image already. As soon as the processor was done with an image, it removed it from the stigmergy, consequently removing the datagrams on all robots. The networkers were used to store the datagrams of the blob, when the processor was busy with processing a previous blob.

**Results** *For each trial, the images were successfully captured, injected, reconstructed and processed from all four target locations.* Fig. 4.11 shows the datagram distribution over the whole swarm for one experiment. At the first target, r4 injects the first image blob. Since r10 is initially free and is actively looking for new blob, it gets the image immediately, bypassing

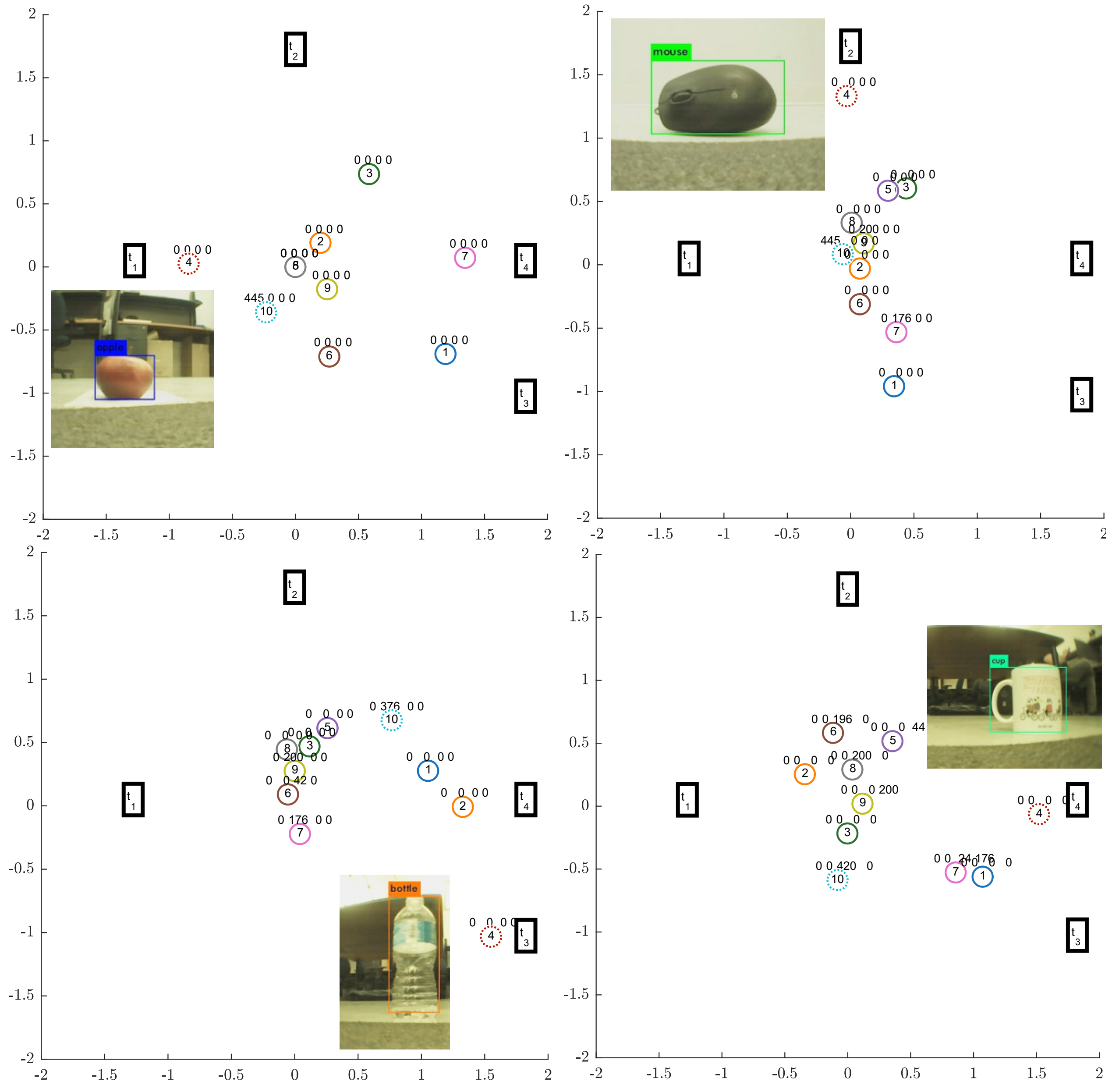


Figure 4.10 Evolution of the blob distribution over the swarm on the second trial of the YOLO Search experiment.

all the auction and the blob distribution, as discussed in sec. 4.3.1. For the other three targets, the creation of a blob happens while r10 is busy. It thus requires the datagrams to be allocated to networkers in the proximity of the processor.

Fig. 4.10 shows the top view of the experimental arena with the four targets (rectangles) and the ten robots (circles). Each robot has four numbers above it, corresponding to the

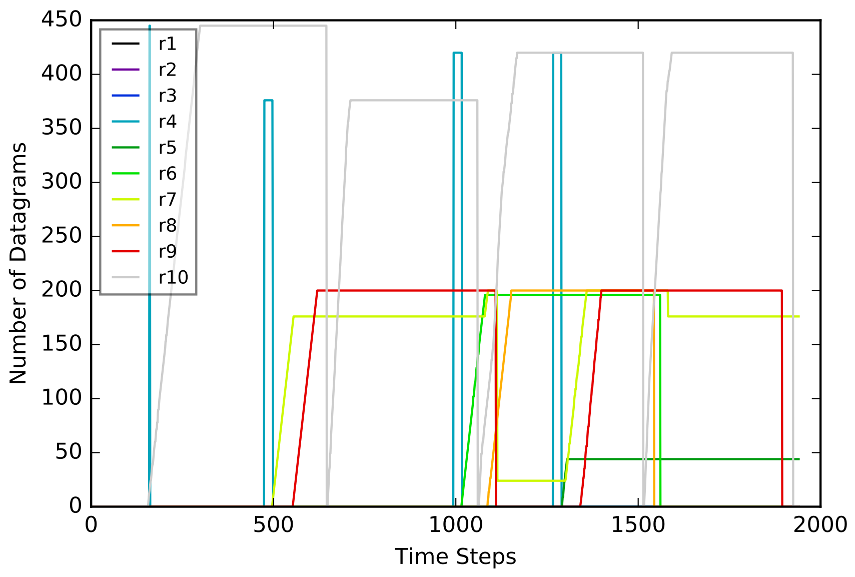


Figure 4.11 Storage usage throughout the second trial for the YOLO Search experiment.

datagrams they hold of blob 1 to 4 (left to right). The capturer and processor are illustrated with dashed lines. Each snapshot corresponds to the moment a blob is injected in the swarm. Both fig. 4.11 and 4.10 represent the same trial. We can observe that the second datagram is attributed to robot r7 and r9, while the third is stored on robots r6, r7 and r8 and the last on r5, r7 and r9.

On average, over the ten trials, 40 blobs were created, from which 21 were sent from the capturer to the processor directly. Over the remaining 19 that had to be distributed over the networkers (since the processor was busy), the average time to allocate the datagrams is 2.7s (min:2.3s, max:7.4s) and the average time to distribute them following their allocation is 12.9s (min: 8s, max: 18.3s). These variations are primarily influenced by the network topology.

## 4.6 Related work

Swarm intelligence can provide both new challenges and new strategies for file sharing [117]. This section gives a brief overview of the related research that inspired SOUL, either in terms of application context (swarm tasks and backup mechanisms) or for their implementation strategy (P2P, consensus and bidding).

**P2P** HTTP is unavoidably the most popular distributed query solution used for databases, Algorithm 5 YOLO Search pseudo-code

```
procedure MISSION STEP
    if camera then                                          ▷ Capturer
        if target $T_i$ reached then
            $Correct\ orientation$
            $blob = Take\_Picture()$
            $SOUL\_obj.putblob(blobid, blob)$
            $blobid += 1$
            $T_i += 1$
        else
            $MovetoTargetT_i$
        end if
    else if GPU then                                        ▷ Processor
        if GPU_free then
            if $blobid$ exists then
                if $blobid$ is buffered then
                    $blob = SOUL\_obj.getblob(blobid)$
                    $obj\_found = Yolo\_Process(blob)$
                    $Propagate(blob\_id, obj\_found)$
                    $SOUL\_obj.putblob(blobid, nil)$
                    $blobid += 1$
                end if
            end if
        end if
    else                                                    ▷ Networker
        $Move\ to\ a\ random\ location$
    end if
end procedure
```

but it still is a one-way client-server approach that does not exploit the number of units available in a swarm [69]. However, as in HTTP/1.0, SOUL manages each request in a separate connection. More recent P2P solutions targeting large files are more suitable for the distribution of large files over a network [15]. The fragmentation, distribution, and reconstruction of the data injected in SOUL is greatly inspired from the approach of the InterPlanetary File System (IPFS) [69]. In fact, distributed hash tables and torrent exchanges are the building blocks of our system, as for many P2P database solutions [118]. However, as opposed to most web-based applications of P2P file sharing mechanisms [119], in a robotic swarm we expect the up- and download times to be equivalent, and we assume that the swarm is cooperative (i.e., no agent is greedy in terms of bandwidth usage), thus removing the need for a fair compensation [120]. Moreover, communication aware task planning [121] could minimize bandwidth usage among the robots.

**Consensus** Consensus-based approaches have been proposed for a number of multi-UAV coordination problems such as resource and task allocation [122], formation control [123, 124], and determination of coordination variables [125]. However, these approaches are specific and need to be implemented again in each new swarm architecture. The swarm-oriented programming language Buzz allows for an easier manipulation of the consensus strategy [102]. The work of [109] proposed a shared table to reach consensus on the state of a swarm of UAVs. Their approach uses a single list to which each robot appends its own value, and the entries on the robots get updated in a query-and-response fashion. However, that approach was not meant to cope with large files and does not support revising the appended values. Similarly, SOUL does not yet provide an update mechanism for the shared data, but earlier work made that possible for state variables, based on the use of Lamport Clocks [101].

**Auction-Based Task Allocation** Auction-based allocation is a popular solution to task allocation problems in robotic swarms, covering centralized, decentralized, and hybrid approaches [126, 127]. Our solution is decentralized, but as described by [127], it uses opportunistic centralization to locally manage each auction related to a new data blob injected in the network. Market-based bidding has found its use in software-agent research, with sophisticated algorithms for winner determination [128]. Computer scientists working in the area generally refer to an auctioneer looking to *sell* the available tasks at the highest price, with various definitions of cost, sometimes including a measure of the tasks' utility [129]. SOUL reverses the problem: it is no longer a means to maximize the auctioneer revenue, but rather to find the lowest-priced subcontractor for each available job (e.g., to store a datagram). The cost submitted by subcontractors considers a set of their available resources, as in [104] for a typical *buyer* strategy. Multiple items in an auction could be sold, sequentially or parallel. Sequential auction algorithms might be beneficial in maximizing agent participation in harsh communication environments, while parallel approaches minimize overall negotiation rounds required during the auction [130]. SOUL leverages a resource aware auction in parallel as proposed by [104].

**Hierarchical tasks for swarms** While most works focus on homogeneous groups of robots, SOUL targets heterogeneous swarms, with the goal of using the full capabilities of each robot. However, as bids often include a resource vector in their computation [104], they are already well-fitted for heterogeneous scenarios. Somewhat along the path of cost computation in the bidding mechanism, one can use decision networks to attribute the tasks following each robot utility calculation [131]. We consider three hierarchical types of tasks: 1- the global swarm task, 2- the sub-swarm tasks, and 3- the individual tasks. For instance, in a search and rescue mission, the overall mission is the task assignment to the swarm, and one sub-swarm may be assigned an exploration task, within which some individuals are tasked to take pictures of particular hotspots. This view is oriented top-down, as opposed to the classical swarm robotics task implementation [132].

**Backup and update** [133] state that distributed systems are complex to update safely while in operation, partly because when a failure is detected, a unit must have access to a backup image to resume. The resource constraints of swarm robotics, and also those of Wireless Sensor Networks (WSN), render the availability of large backups difficult to implement.

In [134], the authors proposed a mechanism to maximize data recovery in WSNs deployed in hazardous environments with high probability of failure: they propose methods to combine data collected by neighboring nodes with the node's data using a set of logical operations, and to store them in the place of the nodes data. This allows the retrieval of data from failing nodes. However, this approach was not meant for large data blobs. Nevertheless, a similar system could be applied within SOUL to increase data persistence.

Large data sharing mechanisms could be found useful for software updates during missions, in continuous deployment. For instance, "hidden changes" providing feature toggles, is a common strategy that generates large patches to be transferred over the network, which can be a serious issue for continuous deployment mechanisms [135]. In fact, this work was partly inspired from the need of large data transfers within a swarm software update system proposed in one of our previous works [45]. There, we propose a systematic approach to perform Over-The-Air updates for robotic swarms during a mission. This update system is made possible by the work in the current paper , by providing a solution for feature toggles and incremental deployment.

## 4.7 Conclusions

This paper introduces a new mechanism, called "Swarm-Oriented Upload of Labeled data" (SOUL), to share data files in a swarm of robots. SOUL can be used in a wide range of applications that need to share data in a fully decentralized manner. SOUL uses an auction-based algorithm to distribute data files on multiple robots, leveraging a BitTorrent-like mechanism. We demonstrate SOUL's scalability and robustness to failures in simulation with different swarm sizes, packet drop rate, network topologies and heterogeneous configurations. We also demonstrate the potential of SOUL under two realistic experimental scenarios with a swarm of ten Khepera IV robots: one to backup a failing robot (SOUL backup) and one to manage data transfers in an heterogeneous area coverage mission (YOLO Search). Both show that

SOUL has consistent performance and potential for practical uses.

As future work, we plan to extend SOUL with versioning, redundancy, a reallocation strategy, and security protocols. Once the datagrams of a blob are allocated, its metadata can be extended to include a git-like track of code changes. As robot movement changes the network topology, the auction attributing the datagrams should be run periodically to maintain the cost of reconstruction minimal. As for security, we will implement stream cipher encryption of datasets.

We believe SOUL or a similar solution is among the essential building blocks of a robust field deployment of robotic swarms in critical scenarios.

# CHAPTER 5 ARTICLE 2: SWARM RELAYS: DISTRIBUTED SELF-HEALING GROUND-AND-AIR CONNECTIVITY CHAINS

**Preface:** Connectivity maintenance in multi-robot systems is widely studied. It plays a vital role in ensuring the robots' synchronous operation and allowing them to coordinate with one another during task performance. The main challenges in applying available methods to robot swarms are large computational footprint, intensive communication requirements, and lack of robust mechanisms to tackle failures. The approach discussed in this chapter considers these challenges and designs a swarm behavior to mitigate these challenges on robot deployments. The approach presented in this chapter was initially published as a short paper [41].

The method to enforce connectivity constraints on a robot swarm proposed in the short paper demonstrated promising results with the ability to scale to a large number of robots. The ability to scale on robots promoted the further extension of the method with more rigorous formulation and detailed considerations to robot failures. The main principle behind this approach is to use local reactive rules on the robots to enforce a globally desired communication structure. The approach provides a small computational footprint by resorting to reactive mechanisms and robustness to failures with the ability to recreate a broken connection with new communication links. Furthermore, the approach provides methods to enforce various levels of connectivity, commonly known as k-connectivity in graph theory, by allowing the user to specify the desired number of links required. Extensive experiments were performed on this method to study its performance on simulated and real-world robots.

In this work, we assume a specialized robot swarm that will utilize the potential of all the robots in a swarm and allows the application of a hierarchy. The swarm is partitioned as the worker and networker. Workers visit an assigned target location and perform path planning and traversability analysis, generally assumed to be robots with enhanced capacity. Networkers secure a communication link between two robots, generally considered simple robots.

**Full Citation:** Varadharajan VS, St-Onge D, Adams B, Beltrame G, "Swarm relays: distributed self-healing ground-and-air connectivity chains," *IEEE Robotics and Automation Letters*, 2020 Jul 2;5(4):5347-54. **DOI:** `https://doi.org/10.1109/LRA.2020.3006793`

**Abstract:** The coordination of robot swarms – large decentralized teams of robots – generally relies on robust and efficient inter-robot communication. Maintaining communication between robots is particularly challenging in field deployments where robot motion, unstructured environments, limited computational resources, low bandwidth, and robot failures add to the complexity of the problem. In this paper we propose a novel lightweight algorithm that lets a heterogeneous group of robots navigate to a target in complex 3D environments while maintaining connectivity with a ground station by building a chain of robots. The fully decentralized algorithm is robust to robot failures, can heal broken communication links, and exploits heterogeneous swarms: when a target is unreachable by ground robots, the chain is extended with flying robots. We test the performance of our algorithm using up to 100 robots in a physics-based simulator with three mazes and several robot failure scenarios. We then validate the algorithm with physical platforms: 7 wheeled robots and 6 flying ones, in homogeneous and heterogeneous scenarios in the lab and on the field.

## 5.1 Introduction

Swarm robotics is a field of engineering studying the use of large groups of simple robots to perform complex tasks [136]. Ideally, a single robot failure in a swarm should not compromise the overall mission because of the inherent redundancy of the swarm [136]. With robustness and scalability, robotic swarms are foreseen as cost effective solution for spatially distributed tasks. In many such applications, the ability of the swarm to coordinate depends largely on its ability to communicate. A reliable communication infrastructure allows the robots to exchange information at any time. However, real deployments include many potential sources of failures (environmental factors, mobility, wear and tear, etc) that can break connectivity and compromise the mission. In this work, we address complete robot failures caused by a robot's inability to communicate or to be detected by its neighbors. In addition, most realistic applications require the robots to remain connected with an operator at a ground station, to either provide information (e.g. in a disaster response scenario) or to receive new commands (e.g. planetary exploration). To maintain connectivity, we propose to progressively use the available robots of a swarm to form a self-healing communication chain from a ground station to a target (illustrated in Fig. 5.1). The operator sets the target and the desired number of redundant links. We target application scenarios including video relay and tele-operation for exploration tasks, where a consistent end-to-end link is required. Our algorithm uses a common path planner to generate a viable path, and builds a chain of robots towards the target. This work extends [41], which briefly presented the self healing aspects of this work without modeling and most experiments. The contributions of this work are: 1. a mathematical formulation of our self-healing chain formation algorithm with configurable redundancy; 2. the performance evaluation of our algorithm in simulated environments of various complexity and integrating robot failures; 3. the validation of the algorithm with a

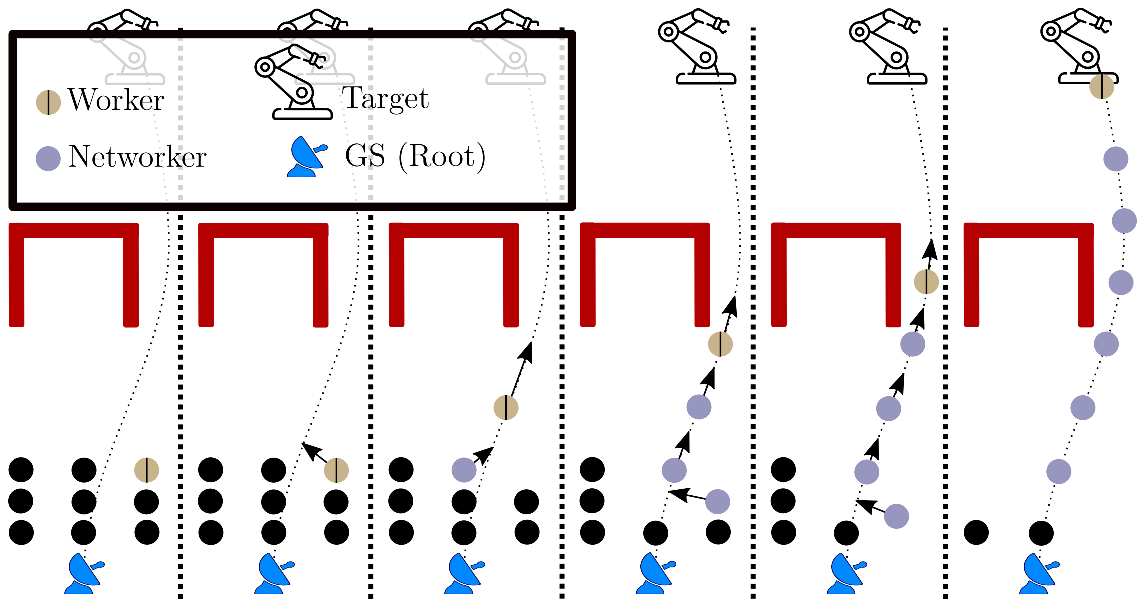


Figure 5.1 Progressive formation of a chain to complete a task.

physical swarm of 7 wheeled robots and 6 flying robots, in homogeneous and heterogeneous configurations.

## 5.2 Related work

There are two general approaches to connectivity maintenance in multi-robot systems: strict end-to-end connectivity [19, 1], or relaxed intermittent connectivity [20, 137, 138]. While the first demands to maintain a link from the source to the sinks (the ends of each branch of the network graph), the latter allows for momentary local breaks in the communication topology. When the mission requires to continuously relay information, like video, operators commands or offloaded computation, strict end-to-end connectivity might be preferred; this is the approach we use in this work.

**Algebraic Connectivity** The problem of preserving end-to-end connectivity is widely discussed in recent literature [19, 1, 16]. Some works use continuous control models with algebraic connectivity [17] and implement mission-related control laws [73], considering robot failures. De Gennaro et al. [71] derive a gradient-based control law from Laplacian matrices' features such as the Fiedler vector to maximize connectivity. Stump et al. [18] also use Fiedler value and k-connectivity to create a bridge between a station and a robot moving towards a target in walled environments. However, the centralized computation of the Laplacian is hardly scalable, and distributed estimations require significant communication bandwidth and are sensitive to noise [78].

**Hybrid approaches** Connectivity can also be maintained by merging continuous motion controllers and discrete optimization for packet routing [139]. INSPIRE [140] uses a two layer control to preserve connectivity: the first layer applies a potential field local controller to maintain a connected configuration and the second layer optimizes the routing. These methods have long convergence time, and are generally not suitable for large robot groups. Ji and Egerstedt [74] integrate connectivity preservation in two control laws for rendezvous and pattern formation. Their controller does not explicitly take into account obstacles and hence cannot easily be adapted to cluttered environments.

**Tree-based approaches** Majcherczyk et al. [1] deploy multiple robots towards different targets while preserving connectivity in a decentralized method based on tree construction. However, they carry all available robots along the path using a virtual communication force field. This approach can lead to unwanted redundant sub-structures and recurrent reconfiguration. Hung et al. [22] propose a decentralized global network integrity preservation strategy that performs strategic edge addition and removal to the network to maintain connectivity while reaching its targets. This approach considers coverage missions and decomposes the space into cells from which to select targets. However, distant and sparse targets increase convergence time significantly.

**Planner Based Approaches** Approaches leveraging a common path planner (centralized [141] or hybrid [19]) determine the optimal communication points for reliable connectivity. These works use a variant of RRT [142] that integrates a communication model to estimate connectivity levels. These methods have realistic communication models but they rely on a centralized solution (a mission planner) and are computationally heavy (they solve a second-order cone program – SOCP). Our approach is fully distributed: we estimate a path using an elected robot whenever a new target becomes available, and navigate while preserving connectivity (similar to [143]). Our method also allows for intermediate robot failures and changes in the environment, which are not considered in the above solutions.

**Robustness to Failure** Connectivity maintenance for multi-robot systems is a widely covered domain but very few works address robot failures. Some approaches [16] take into account a robustness factor to tackle robot failures but cannot recover a completely disconnected network. A few other approaches consider disconnection and recovery due to environmental mismatches [144] but do not consider complete robot failures. The Wireless Actor Networks domain has several works that address failure recovery [145], e.g. using dynamic programming [146] to reconnect a disjoint graph. These approaches disregard motion planning and have limited applicability in cluttered environments.

**Task allocation** Allocation of a fixed number of tasks to a set of robots is a well know combinatorial problem and requires heuristics [147] to approximate to a polynomial solution. Decentralized methods use local planners along with consensus algorithms to agree on the context of the task plan [148]. Other approaches compute a plan on each robot and use consensus algorithms to agree on the global assignment [149]. We use an approach similar to the latter to assign roles: a local bid on each robot serves to achieve consensus on the assignments. Our work leverages the ideas in [22] and [1] to dynamically build structures towards the targets. We sequentially add edges to a tree in a distributed manner by using the path from a standard path planner, as in [19], and reactively enforce connectivity, as in [143]. The final contribution of our approach is the ability to recover from simultaneous robot failures while navigating complex environments with obstacles and preserving a network structure with a configurable number of redundant links. On top of this, our approach uses minimal computation and communication load on the robots, a key aspect for the deployment of other behaviors on top of connectivity maintenance. To the best of the authors' knowledge, this is the first approach that studies this problem in a holistic manner: a path planner, a failure recovery mechanism and a task allocation mechanism to dynamically assign tasks.

## 5.3 Preliminaries

Consider a team of $N_r$ robots with their positions denoted by $X = \{x_1, x_2, ..., x_n\}, \forall x_i \in \mathbb{R}^3$. The evolving position of the robots at time $t_i$ can be denoted as $X(t_i) \in \mathbb{R}^{3N}$. Given a set of target locations $\mathbb{T} = \{\tau_1, \tau_2, ..., \tau_n\}, \forall \tau_i \in \mathbb{R}^3$, our objective is to drive the robots to a formation that ensures (a) at least one communication path between any two robots; and (b) each target is within range of at least one robot $\|x_w - \tau_i\| \leq \delta_{tol} \forall \tau_i \in \mathbb{T}$. We consider a single integrator robot model ($\dot{x}_i(t) = u_i$) and assume that the robots are fully controllable with $u_i$. Taking into account that the robots' workspace, $\mathbb{X} \in \mathbb{R}^3$, is divided into obstacles $\mathbb{X}_{obs}$ and free space $\mathbb{X}_{free}$, we derive the control inputs $u_i(t)$ for all robots, avoiding obstacles and other robots.

### 5.3.1 Communication model

We assume the robots are equipped with a wireless communication device (e.g. 2GHz, 5GHz or 900MHz). The received signal strength is influenced by three main factors: path-loss, shadowing, and fading [150]. We approximate signal strength as a generic function of distance.

Inter-agent communication in a group can be then modeled as a weighted undirected graph

$\mathcal{G} = (\nu, \epsilon, A)$, with the node set $\nu = \{r_1, ..r_N\}$ representing the robots, and the edge set $\epsilon = \{e_{ij} | i, j \in \nu, i \neq j\}$, representing communication links. A common approach to working with a communication graph is to use its adjacency matrix $A$, in which entries $e_{ij}$ represent the probability of robot $i$ decoding $j$'s packets. We aim at maintaining $e_{ij} > e_{min}, \forall e_{ij} \in \epsilon$, that is to guarantee a minimum signal quality. We consider the robots capable of broadcasting and relaying messages to their neighbors over a limited spherical communication range $Z$. The robots estimate $e_{ij}$ for their neighbors using local information, if $d_{ij} > Z$ then $e_{ij} = 0$, whereas if $d_{ij} < \delta$ then $e_{ij} = 1$ and $e^{\frac{-5*d_{ij}}{Z}}$ otherwise. $\delta$ is a small constant, slightly larger than the radius of the robot. The approximation of connectivity using $e_{ij}$ allows for modeling additive white noise in sensing. The surroundings of robot $i$ are divided into communication zones as illustrated in fig. 5.2:

- the safe zone $Z_i^s$, in which neighbors are considered to have a reliable network link ($e_{ij} > e_{min}$) up to the limit distance $d_s$ with connectivity $e_{ij} = e_{min}$;
- the critical zone $Z_i^c$, in which neighbors are getting close to the limit of the communication range. In this zone, $e_c \triangleq d_c$ - $d_s > 0$ is defined as the *critical tolerance distance*. At the critical distance $d_c$, $e_{ij} < e_{min}$;
- the break-away zone $Z_i^b$, in which neighbors are expected to break their network link. In this zone, $e_b \triangleq d_b - d_c > 0$ is defined as the *break-away tolerance distance*. At the break-away distance $d_b$, $e_{ij} \approx 0$ and at Z, $e_{ij} = 0$.

Let $N_i$ be the neighbor set of robot $i$, which is divided into: $N_i = N_i^s \cup N_i^c \cup N_i^b$. Robot $j$ is called a safe zone robot of robot $i$ if $x_j \in Z_i^s$, with $x_j$ its position vector. The set of all neighbors within the safe communication zone of robot $i$ is: $N_i^s = \{j | x_j \in Z_i^s, \forall j \in N_i\}$. From this, we define the *safe connectivity set* as the entries $e_{ij}$ of the adjacency matrix $\forall j \in N_i^s$. Similarly, we can define the *critical* and *break-away* connectivity sets with the corresponding entries of the adjacency matrix. These sets allow us to derive control inputs that guarantee the preservation of local connectivity.

### 5.3.2 Local connectivity preservation

We formulate the constraint for the preservation of connectivity among the robots using geometric arguments as in [22]. However, our solution continuously adds edges to the local graph until specific robots with specific roles reach the targets. Our approach reduces the graph construction time, computational load and communication rounds required to determine which edges to remove in [22]. Fig. 5.3(a) shows the initial position of robot $i$ ($x_i(t)$)

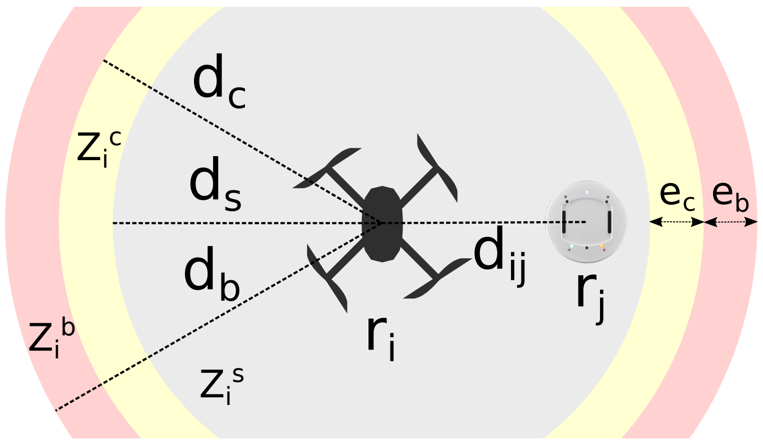
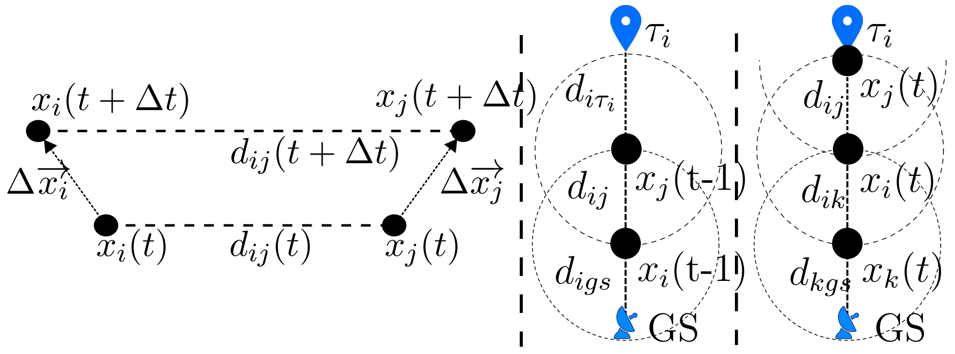


Figure 5.2 Illustration of the zones in the communication model.



(a) (b) (c)

Figure 5.3 Variables related to inter-robot distance and robot position before and after a time step Δt (a) illustration of combined distance of a connectivity chain before (b) and after (c) the addition of a new robot.

and robot $j$ ($x_j(t)$) with their relative distance $d_{ij}(t)$, together with their new position and relative distance after a time $\Delta t$. To guarantee the preservation of the communication link, the control must ensure:

$$(\|\Delta x_i\| + \left\|\Delta x_j\right\|) \leq d_b - d_{ij}, \tag{5.1}$$

with $d_b$, the break-away distance where $e_{ij} = 0$, at which the connectivity breaks. In our implementation we choose to split the responsibility of respecting the available margin $d_b - d_{ij}$ equally among the two robots. The change in position of robot i is $\Delta x_i = \|x_i(t + \Delta t)\| - \|x_i(t)\|$. The connectivity constraint $e_{ij} \geq e_{min}$ between two robots $i$ and $j$ is preserved if the change in position of the robots satisfies the following condition:

$$\|\Delta x_i\| \leq \frac{d_b - d_{ij}}{2} \quad and \quad \left\|\Delta x_j\right\| \leq \frac{d_b - d_{ij}}{2}. \tag{5.2}$$

With $d_{ij}(t + \Delta t)$ the distance between two robots $i$ and $j$ after a time step $\Delta t$ defined as $d_{ij}(t+\Delta t) = \left\|x_i(t + \Delta t) - x_j(t + \Delta t)\right\| \leq d_{ij}(t) + \left\|\Delta x_i + \Delta x_j\right\|$ . When we apply (5.2) we get:

$$d_{ij}(t + \Delta t) \leq d_b. \tag{5.3}$$

proving that robots $i$ and $j$ stay connected.

We can extend the following remarks considering the three neighborhood sets (safe, critical, break-away):

1. A robot $j$ can be in $N_i^s$ if and only if, its position lies within the safe zone $Z_i^s$ of robot $i$. This implies that $d_b - d_{ij} \geq e_c + e_b$. If we choose control inputs that allow $\Delta x_i$ and $\Delta x_j$ to satisfy the condition of (5.1), with $d_s$ as the safe limit, the robots will always stay within the safe communication distance:

$$\Delta x_i \leq \frac{d_s - d_{ij}}{2} \quad and \quad \Delta x_j \leq \frac{d_s - d_{ij}}{2} \tag{5.4}$$

2. For robots $j \in N_i^c$, located in the critical communication zone $Z_i^c$, if we choose control inputs that allow $\Delta x_i$ and $\Delta x_j$ to satisfy the condition of (5.1), with $d_s$ again as the safe limit, the robots tend to regain safe connectivity:

$$\Delta x_j \leq (d_s - d_{ij}) \quad and \quad \Delta x_i = 0 \tag{5.5}$$

3. We can apply an identical reasoning for robots $j \in N_i^b$ with their position within and break-away communication zone $Z_i^b$, have one robot stationary and the other apply a control input $\Delta x_i$ to satisfy the condition of (5.1).

By applying control inputs satisfying the condition of Eq. (5.4) robots in critical/break-away connectivity move towards each other to regain safe connectivity. We use the critical tolerance $e_c$ and break-away tolerance $e_b$ to account for control errors when applying Eq. 5.5.

### 5.3.3 Global Connectivity Chains

Given a group of robots, we build a chain from a ground station to a target location. To build the chain, we assign roles and relationships to the robots. Robots in the chain are assigned parent-child relationships, to manage the construction of the chain towards a target while maintaining connectivity.

*Definition 1*: A connectivity chain is a tree $C$ represented as a partially ordered set $(C, <) = \{c_1, c_2, ..., c_n\}$ with $|C| = n$. All vertices $c_i \in C$, have a parent $c_{i-1}$ and a child $c_{i+1}$, except for $c_1$, the *root*, that is without parent and $c_n$, the *worker*, which is without children. All other $c_i$ are *networkers*. The edge set $\epsilon = \{e_{c_i, c_{i+1}} | \forall c_i \in C\}$.

*Proposition 1:* Given a connectivity chain $(C, <)$ and the distance of the worker $c_n$ to a target $d_{i\tau_i}$, the addition of a new robot between any two robots $c_i$ and $c_{i+1}$ in the chain decreases the distance to the target $d_{i\tau_i}$.

Let $d(t-1)$ denote the sum of the distances between all the robots in a chain before the addition of robot $c_k$, considering the distances of the robots $d_{n-1,n} = d_s \forall n \in (C, <)$. The distance $d$(t) after the addition of $c_k$ into chain is $d(t) = d(t-1) + d_s$, adding a slack of $d_s$ into the chain, which in turn allows the worker to move towards the target, decreasing $d_{i\tau_i}$, as shown in Fig. 5.3(b-c). The desired network structure from the root robot to a worker robot is specified as the minimum number of mandatory communication links $C_n$ and the algorithm enforces $C_n$ individual connectivity chains.

### 5.3.4 Task allocation problem

Let the set of free robots $N_f = N_r / N_c$, with $N_r$ the set of all the available robots and $N_c$ the set of robots already in a connectivity chain. The goal of the task allocation algorithm is to assign the set of tasks $\mathbb{T}$ to free robots in set $N_f$. The class of problem called multiple-choice multidimensional knapsack problem (MMKP) [151], assume n sets of mutually exclusive items. The goal of MMKP is to select exactly one item from each set, minimizing the cost without violating a set of constraints. The task allocation problem considered in this work resembles MMKP. The task allocation problem where every robot can be assigned more than one task is often referred to as a multi-assignment problem [149]. A special case of a multi-assignment problem, where every single robot is assigned exactly one task, is called a single-assignment problem [149]. The problem considered in this work is a single-assignment problem, where exactly one task from $\mathbb{T}$ has to be assigned to robots in $N_f$. The integer programing problem takes the following form of optimization problem:

$$\min \sum_{i=1}^{|N_f|} \sum_{j=1}^{|\mathbb{T}|} c_{ij} a_{ij} \tag{5.6}$$

$$\text{subject to} \sum_{j=1}^{|\mathbb{T}|} a_{ij} \leq 1, \forall i \in N_f \tag{5.7}$$

$$\sum_{i=1}^{|N_f|} a_{ij} \leq 1, \forall j \in \mathbb{T} \tag{5.8}$$

$$\sum_{i=1}^{|N_f|} \sum_{j=1}^{|\mathbb{T}|} a_{ij} = |\mathbb{T}| \tag{5.9}$$

where $c_{ij}(x_i) = ||x_i - \tau_j||$ denotes the cost of robot i at position $x_i$ performing task j and $a_{ij} \in \{0, 1\}$ is the assignment variable indicating the assignment of task j to robot i. The first constraint indicates that each robot can be assigned at most one task and second indicating no two robots get the same task. Consensus-based auction algorithm (CBAA) [149] uses an auction to obtain initial bids from robots and unifies the assignment using a consensus phase. CBAA guarantees convergence and provides a bound on optimality when the cost function follows a diminishing marginal gain [149]. In this work, we use Virtual Sigmergy [68] to share task assignments in a decentralized manner. Virtual Stigmergy is a decentralized information sharing mechanism: once a value is written in the Virtual Stigmergy by a robot, it can be accessed from all other robots.

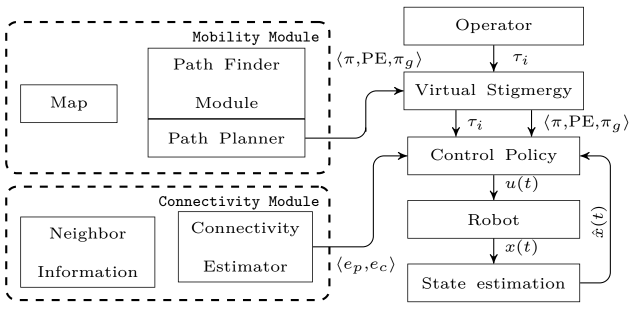


Figure 5.4 Control architecture used within the chain construction algorithm.

## 5.4 Proposed Method

Fig. 5.4 shows the modules of our controller: mobility, connectivity and control policy. The mobility module computes a viable path optimized for path length and stores the result in the Virtual Stigmergy. The connectivity module continuously estimates the current connectivity between the assigned parent ($e_p$) and child ($e_c$). The connectivity $e_p$ and $e_c$ is computed using the piecewise function introduced in sec. 5.3.2. The control policy computes the control inputs using path and connectivity information.

We consider two types of robots in the swarm: *ground robots* with state space $\mathbb{X} \in \mathbb{R}^2$, and *flying robots* with $\mathbb{X} \in \mathbb{R}^3$. We assume the map of the environment is known. The mobility module consists of a path finder and an optimal path planner. The path finder scans for a viable path by splitting the workspace in cells at a given resolution $R$, and building a graph $\mathcal{G}_{\mathcal{PE}} = (N, E)$ of the cells. It then performs a depth-first-search in this graph to verify the reachability of target $\tau_i$ [152]. If it is found reachable, the module raises the $PE$ flag. Ground robots and flying robots can then be used to build a chain on the resulting path $\pi$.

If a path to $\tau_i$ is not found, it either means that the target is unreachable or that the check was done in $\mathbb{R}^2$, but a solution exists in $\mathbb{R}^3$. The path finder can be set to always check in $\mathbb{R}^3$, but that would create overhead for simple cases in $\mathbb{R}^2$. Nevertheless, the resulting path in $\mathbb{R}^3$ has a portion, $\pi_g \subset \pi$, that can be traveled by ground robots: the projection of the 3D path $\pi$ on the ground plane. The path planner module computes an optimized path (continuous vector-valued function) $\sigma : [0, T] \rightarrow \mathbb{X}_{free}$ such that $\sigma(0) = x_0$ and $\sigma(T) = \tau_i$. Similar path planning problems have led to several well-studied approaches [142, 153]. When optimizing for the shortest path length, sample-based approaches can quickly get an approximation that is then optimized.

We selected $RRT^*$ because it was shown to be faster in large environments and to perform well in cluttered spaces [142]. However, our control architecture is agnostic to the path planner algorithm, and we have tested it with several other planners (SST, $BIT^*$ and $PRM^*$) with comparable results, as discussed in 5.5. The chain construction algorithm is implemented in Buzz [56], a programming language for robot swarms, which eases behavior design efforts by providing several swarm programming primitives. For the planners, we integrated classes of Open Motion Planning Library (OMPL) [154] in our architecture to be accessible as Buzz functions. We assigned one of three roles to the robots: root, worker(s), and networkers. A detailed presentation of the task allocation strategy is given as supplementary material [1]. At launch, the root is either pre-determined (ground station) or elected using gradient algorithm

[1] `https://mistlab.ca/papers/SwarmRelays`

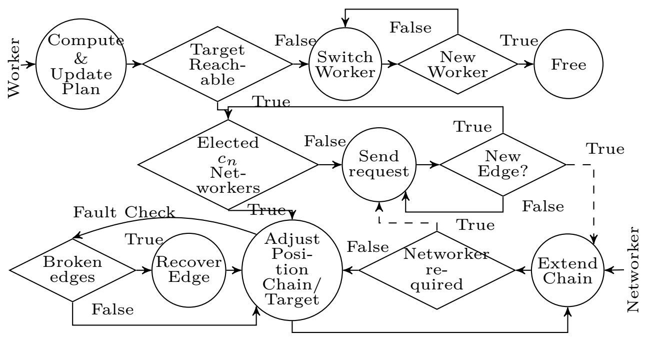


Figure 5.5 Interplay between Networkers and Workers.

(detailed in the supplementary material 1). Each target also gets a worker robot elected with the same strategy. The gradient algorithm is implemented using Virtual Stigmergy [68]. Fig. 5.5 shows the interaction of the workers and networkers in the chain formation. The worker robot checks for the existence of a path tuple $\langle \pi, PE, \pi_g \rangle$ in the Virtual Stigmergy; if not available, it computes the path and shares it. The motivation behind using a single robot to compute the plan is twofold: 1. if multiple robots are computing a plan then all plans must be exchanged to reach consensus – a high load of messages/bandwidth; 2. assigning the task to the worker robot also leads to more efficient updates, as the worker will be the first to encounter any obstacles in the environment. In case of a mismatch between the computed path and the explored environment, the path can be adapted.

The path's tuple includes $\pi$, the path defined as a sequence of points (states), the $PE$ flag and $\pi_g$, the projection of $\pi$ on the ground plane. Following its locomotion type, the robot knows which sections of $\pi$ it can reach. If the elected worker robot does not have the required locomotion type, the swarm elects a more suitable robot, as shown in Fig. 5.5. The worker robot determines the number of links $C_n$ required to reach the target and elects the necessary

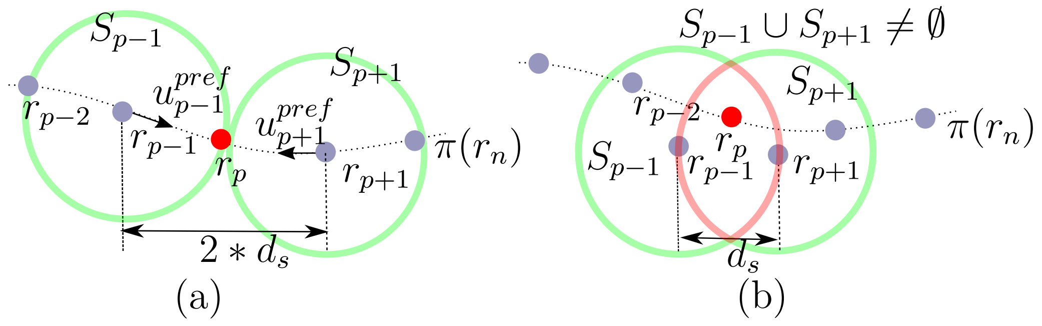


Figure 5.6 Failure recovery: (a) at the time of failure and (b) after recovery.

networkers. If more edges are required than the number of robots surrounding the worker, the request is passed down the networkers towards the root, until a free robot (future edge) is found. An optional module, the Way-Point (WP) prediction, grants each robot in the chain to autonomously elect robots to expand the chain without a signal from their child. The number of robots required in a chain $n = ceil(\frac{d_{path}}{d_s})$ is estimated on each robot using the current plan length $d_{path}$.

### 5.4.1 Motion Control

The motion commands are computed by networkers and workers (the root is fixed) using the available path to extend the chain towards the target(s). The robots that do not belong to any chain are called free robots, and they wait close to the root until they get elected as a worker or a networker. The robots compute the preferred velocity as:

$$u_i^{pref} = \begin{cases} f_c(d_i^C) u_i^{path}(\pi, F) & \text{if } d_i^P \leq d_s \\ u_i^{path}(\pi, B) & \text{otherwise} \end{cases} \tag{5.10}$$

$$f_c(d_i^C) = \begin{cases} 1 & \text{if } d_i^C \leq d_s \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

where $u_i^{path}(\pi, F)$ computes velocity commands to move the robot towards the target and $u_i^{path}(\pi, B)$ computes velocity commands to move the robot towards the root, both based on the path $\pi$. $f_c(d_i^C)$ stops the movement when a child reached the critical communication distance from its parent. In other words, a chain retracts if a robot is in the critical communication zone and expands otherwise. The inputs $u_i^{pref}$ are computed from set point control and altered by a collision avoidance algorithm. Most reactive decentralized collision avoidance can be used within our architecture. We used the Reciprocal Velocity Obstacle (RVO) algorithm [155], which selects the best velocity considering all future potential collisions. The resulting control law is:

$$u_i = \arg \min_{u_i' \in Au_i} \sum_{j \in N_i^{close}} \alpha \frac{1}{tvo_j(u_i')} + \left\| u_i^{pref} - u_i' \right\| \tag{5.12}$$

$$Au_i = \{u_i' \mid \left\| u_i' \right\| < u_i^{conn}\}, \; u_i^{conn} = \min_{j \in P \cup C} \frac{(d_s - d_{ij})}{2\Delta t}$$

Equation 5.12 computes the control input from the velocity within the admissible set $Au_i$ that minimize the risk of collision while maximizing the fit to the path. The function $tvo_j(u_i')$

computes the penalty of future collision between $i$ and $j$ while the norm is a penalty for deviation from the preferred velocity on the path $u_i^{pref}$. $\alpha$ is a scaling factor to balance the penalty terms (set to 1 in our tests). $N_i^{close} = \{j \mid d_{ij} < r_{col} \forall\ j\ \in N_i\}$ is the set of neighbors close enough to be inside the collision radius $r_{col}$. $u_i^{conn}$ is the maximum velocity allowed, a bound to maintain a safe communication distance according to Equation 5.1.

### 5.4.2 Self-Healing

We consider two types of robot failures in our work: due to sensing errors and complete robot failures. Both cases create two or more disconnected groups of robots that then need to be reconnected. Dealing with complete failures is harder than sensing errors since no information transfer is possible between each side of the disconnection. Without any means of communication, it is impossible to agree on the reconnection strategy between chain segments. In this work, we get the information required to reconnect the chain in both cases using a periodic broadcast message (detailed in the the supplementary material [2]). Every robot $r_i$ in a chain $C$ uses a common plan $\pi(r_n)$ updated by worker $r_n$ and shared through Virtual Stigmergy. They are aware of $k$ local edges of their chain $C$ and its current depth through the chain link messages (a sequence of IDs obtained through broadcasts). Every robot can reconstruct the $k * 2$ immediate portion of the chain $C_{local} \subseteq C$ by concatenating parent and child chain messages. If a robot in the communication path fails, the communication chain is broken and the chain cannot be completed. In this case, the parent and the child of the failed robot attempt to bridge the broken link by navigating toward each other using the available chain information as illustrated in the Fig. 5.6. Without loss of generality, we consider an arbitrary robot $r_p$ in a chain failing with a parent $r_{p-1}$ and child $r_{p+1}$. The result is two disconnected chains $C_1 = \{r_1, ..., r_{p-1}\}$ and $C_2 = \{r_{p+1}, ..., r_n\}$. The robots $r_{p-1}$ and $r_{p+1}$ will attempt to bridge the gap. The safe connectivity workspace $S_i = \{y \in \mathbb{R}^n | \|y - x_i\| \leq d_s\}$ of robot $i$ contains all the states within the safe communication distance $d_s$. At the time of failure $\{x_{p-1}\} \cap S_{p+1} = \emptyset$ and $\{x_{p+1}\} \cap S_{p-1} = \emptyset$, formally indicating the disconnection. Robots $r_{p-1}$ and $r_{p+1}$ already share consensus on the plan $\pi(r_n)$, when the robot is executing the current segment of the plan. This allows the robots $p-1$ and $p+1$ to compute control inputs $u_{p+1}^{pref} = u_{p+1}^{path}(\pi(r_n), B)$ and $u_{p-1}^{pref} = u_{p-1}^{path}(\pi(r_n), F)$ respectively. When using an identical plan $\pi(r_n)$ and applying $u_i^{path}$, it is guarantied that the robots will regain safe communication distance ($d_{ij} \leq d_s$) with $x_i \in S_j$. The same reasoning applies to $n$-consecutive robot failures, with the inter-chain distance between two disconnected chains being $(n + 1) * d_s$ instead of $2 * d_s$. The two bordering robots in the disconnected chain will compute and apply the

[2] `https://mistlab.ca/papers/SwarmRelays`

control input $u_i^{pref}$ based on $\pi(r_n)$. The boundary robot that lost a child's connection acts as a temporary worker until the chain is healed or it becomes the worker.

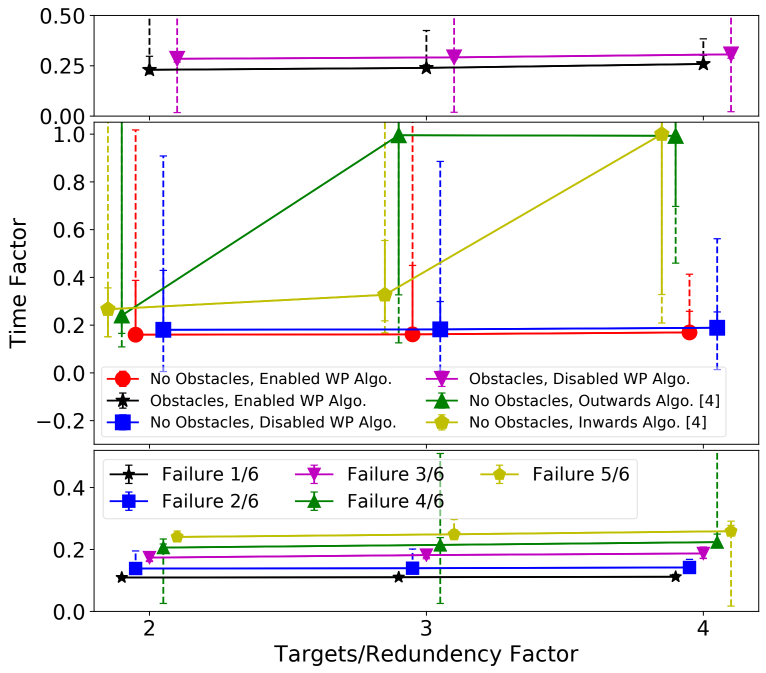


Figure 5.7 Middle: comparison of convergence time with [1], convergence time of the chain construction algorithm by enabling and disabling the way-point algorithm that allows the robots in a chain to predict the number of robots required. Top: convergence time by introducing one "C" shaped obstacle. Bottom: comparison of failure recovery time by failing different number of consecutive robots in a chain and creating two subgraphs.

## 5.5 Experimental Evaluation

**Performance comparison** We use ARGoS3 [96] physics-based simulator to assess the performance of our method. We compare our results with [1], the closest approach to ours. We set up a simple open environment to compare our algorithm with [1] in 12 conditions: 2 to 4 targets uniformly distributed on a circle ($r = 10m$), with and without obstacles (C-shaped) between the robots and the targets, and with or without the WP prediction algorithm mentioned in Section 5.4. Each conditions was repeated 35 times with random initialization. We set the algorithm's parameters to: $d_s = 1.4m$, $d_c = 1.6m$, $d_b = 1.8m$, $v_{max} = 50cm/s$, $\alpha = 1$ and $r_{col} = 25cm$, with a fixed planning time of 2s, a bidding time for the gradient algorithm of 10s, a forgetting time for status messages to 3s and a link failure declaration time of 5s. Fig. 5.7 shows the resulting comparison of performance. The metric is *time factor*, i.e. the time taken by the algorithm divided by the traversal time [1]. Our method outperforms both the inwards and outwards algorithms proposed in [1], particularly when increasing the number of targets or redundancy factor. This can be attributed to the fact that all robots are part of the large virtual force field in [1], whereas we form a directed chain with a near-optimal number of robots.

Unlike [1], our chain construction scales well with the number of targets, showing the ability to reach separate targets in parallel. The time factor decreases by 2% without obstacles and 5% with obstacles when using WP prediction.

**Robustness** We then run the same configuration as above, but inducing up to 80% consecutive robot failures. Fig. 5.7 (bottom) shows the time factor required by the robots to recover from these broken links. There were in average 6 intermediate networker robots connecting the root and the worker. The failure factor denotes the number of consecutive robots disabled after the chain reached the targets, for instance a factor of 5/6 denotes failing 5 out of 6 robots in a row. Experiments show an increase in 5% of the time ratio for every additional robot failing in the chain. Simultaneously failing a constant fraction of robots leads to almost constant recovery time, as reconnections occur in parallel. The supplementary material[3] provides additional experiments that show our method's robustness to noise. To further assess the robustness of our method, we used a motion planning benchmarking dataset [156]: 1. small (dragon age/arena), 2. medium (dragon age/den203d), and, 3. large (dragon age/arena2). , using the same parameters as above (except for $d_s = 9.5m$, $d_c = 9.7m$, $d_b = 10m$ to fit the environment) but injecting random robot failures at every control step with an occurrence probably of $p = 0.0005$ and bounded to a maximum percentage of robots failures $F$ from the set $F \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, with $0 \leq F \leq 1$. Fig. 5.8 (bottom) shows the resulting time taken to build the chain. $F \leq 0.3$ does not have significant impact on the algorithm performance. However, with more failures, completion time increases as well as the performance variability. As more robots fail, the links have to be repaired before proceeding to the target, increasing the convergence time with $F$.

**Scalability** We use again the benchmarking dataset [156], but adding a wall with a small window above the ground to the first arena to force the use of flying robots (top leftmost map of Fig. 5.9). We used 20 ground robots and 5 flying robots for arena 1, 60 ground robots for arena 2, and 100 ground robots for arena 3. For each of the arenas, we enforced 1, 2, and, 3 communication links, except for the 100 robot runs (arena 3) covering only 2 and 3 links. Fig 5.9 (top) shows that the robots stayed within the safe communication zone even when navigating in narrow corridors. Fig. 5.8 (top) plots the time taken by the robots to build 1, 2, and, 3 links in each of the arenas, along with the reference traveling times computed

[3] `https://mistlab.ca/papers/SwarmRelays`

using the path length and the maximum velocity of the robots (set again to 10 cm/s). As expected, the time taken to build a chain increases with the size of the arena, but the average time to build different number of links is similar across different arenas. In particular, when computing the time factor, the median time ratio for N=60 is 0.106 and 0.102 for N=100, for all types of links. This proves our claim that the proposed approach spends on average equal amount of time reaching multiple targets with single or multiple links. However, in arena 1 the chain construction time increases with the number of links: this is due to the need of a specific type of robot (flying) to cross a section of the arena. On average, traversal time increases by 2% for every additional heterogeneous robot chain.

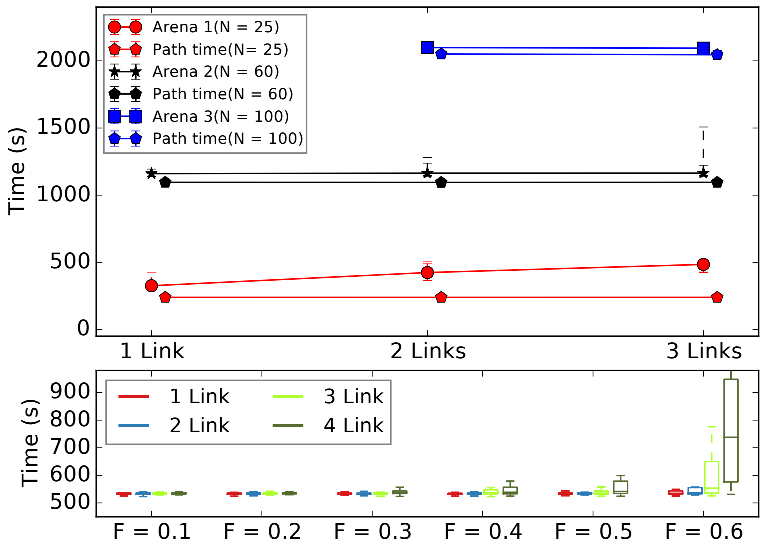


Figure 5.8 Top: time taken by 25 (arena 1), 60 (arena 2) and 100 (arena 3) robots to reach the target with 1, 2, and, 3 communication chains. The traveling time on the bottom indicates how fast a robot can travel the path without considering the chain. Bottom: time taken to build the communication with different percentage of random robot failures.

**Real robot experiments** We validated the performance of our implementation using 6 Crazyflie flying robots and 7 Khepera IV ground robots. We performed experiments in three different settings with two different arena configurations as shown in Fig. 5.9: 1. 7 ground robots in arena 1, 2. 3 ground robots and 3 flying robots in arena 2; arena 2 has a wall containing a hole before the target and can be only reached by a flying robot, 3. 6 flying robots in arena 2). The ground robots run an instance of the Buzz Virtual Machine (BVM) on-board to perform both control and path planning, whereas the flying robots use PyBuzz, a decentralized infrastructure detailed in [157]. We use a motion capture camera system emulating situated communication for state estimation. For these experiments, we set the design parameters to $d_s = 1.2m$, $d_c = 1.4m$, $d_b = 1.5m$ and $v_{max} = 1.5m/s$ to

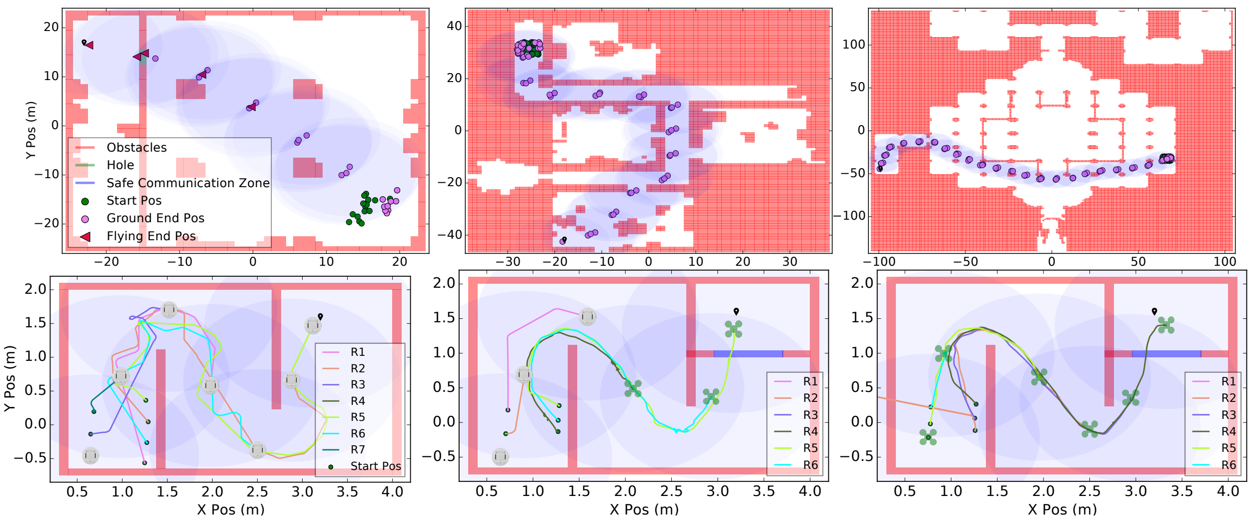


Figure 5.9 Top: Illustration of simulated maps with each robot starting position, end position and the safe communication disk region: on the left the smallest arena with flying robots going over an obstructed zone, in the middle a medium-sized arena with ground robots only, and on the right our largest arena, also using only ground robots. Bottom: Real robot experimental arena with start, end, trajectory and end position: on the left 7 khepera IV robots in arena1, in the middle 3 khepera IV robots and 3 CrazyFlie in arena 2 and on the right 6 CrazyFlie in arena 2.

match the size of our robots and arena. Fig. 5.9 (bottom) shows one of the 10 test runs we performed in each of the three different settings. The figure shows the starting point, the trajectory, and the end position of the robots. On all runs the robots were able to build a communication chain and reach the target. Ground robots, heterogeneous team and flying robots on an average took 100s, 50s and 40s respectively to reach the targets (as discussed in the supplementary material [4]). Finally, we ported the algorithm to our ROS based infrastructure [48] for an outdoor test on a group of 4 larger quadcopters. The outdoor robots used GPS and communicated through an Xbee mesh network. This deployment was made to show the practical use of our algorithm on commercial hardware. The supplementary video[5], features an excerpt of these three settings.

## 5.6 Conclusions

We present a communication chain construction algorithm for a heterogeneous swarm of robots. Our approach is completely decentralized: it requires only relative and local information from neighbors. To tackle with robot failures, we exchange information and bridge

[4]https://mistlab.ca/papers/SwarmRelays
[5]https://youtu.be/HdueQhKV33I

the chain as soon as it is broken. We assess the algorithm performance with extensive simulations on up to 100 robots in five different arenas. Real robot experiments with flying and ground robots demonstrated the usability and robustness of the approach.

# CHAPTER 6 ARTICLE 3: HIERARCHICAL CONTROL OF SMART PARTICLE SWARMS

**Preface:** The hierarchical organization of robot swarms has the potential to encode the global mission requirements and achieve a significant success rate in comparison to a traditional approach. Applying the hierarchical organization, we designed a particle swarm with the identical partitioning of two types of robot specialization, where top-level robots called the guides can change the operational rules of lower-level worker robots. The robots in the lower level are completely self-organizing based on the instruction from the top-level robots, and top-level robots perceive the mission goals, driving the whole swarm. The particle swarm solves a variant of the shepherding problem called formation-shepherding, introduced and addressed in this work. The formation-shepherding problem involves creating a shape with a large swarm of robots and moving this shape to a target location. This particle swarm demonstrates scalability with up to a thousand robots governed by 32 top-level robots. We performed extensive simulation experiments to understand the properties like distortion and time involved in forming and moving various shapes.

**Full Citation:** Varadharajan VS, Dyanatkar S, Beltrame G, “Hierarchical Control of Smart Particle Swarms”, *IEEE Transactions on Robotics*, 2022 (Submitted).

**Abstract:** We present a method for the control of robot swarms which allows the shaping and the translation of patterns of simple robots (“smart particles”), using two types of devices. These two types represent a hierarchy: a larger group of simple, oblivious robots (which we call *the workers*) that is governed by simple local attraction forces, and a smaller group (the *guides*) with sufficient mission knowledge to create and maintain a desired pattern by operating on the local forces of the former. This framework exploits the knowledge of the guides, which coordinate to shape the workers like smart particles by changing their interaction parameters. We study the approach with a large scale simulation experiment in a physics based simulator with up to 1000 robots forming three different patterns. Our experiments reveal that the approach scales well with increasing robot numbers, and presents little pattern distortion for a set of target moving shapes. We evaluate the approach on a physical swarm of robots that use visual inertial odometry to compute their relative positions and obtain results that are comparable with simulation. This work lays foundation for designing and coordinating configurable smart particles, with applications in smart materials

and nanomedicine.

## 6.1 Introduction

Decentralized behaviors have been increasingly studied in the past two decade mainly because they have many desirable properties like scalability, robustness, and flexibility. The study of decentralized behaviors on robots took initial inspiration from natural swarms of ants and bees [8], which exhibit complex collective behaviors through simple local interactions. Even after decades of research, the deployment of collective behavior on robots that cooperatively perform a real-world task remains challenging. Equipping robots with swarm intelligence is one of the most promising applications of artificial intelligence, but it is yet to find its way to solving real-world problems, and it is considered one of the grand challenges of the decade in robotics [158]. Robot swarms are generally composed of a team of robots that are simple and inexpensive, but when working together, they are capable of performing tasks reserved to more complex and expensive robots. Some futuristic views on swarm robotics [65] suggest that engineering methods for heterogeneous, hierarchical self-organization is required in robot swarms to transition from laboratories to real-world applications. This work is a step in this direction that leverages self-organization through local interactions in a two-level robot hierarchy. Hierarchies in robot swarms can be of two types: information hierarchies and intelligence hierarchies. The former (information hierarchies) considers identical robots with varying level of mission-related information: the robots with more information can make better mission decisions and are placed at the top of the hierarchy. For example, consider the task of exploring an unknown environment: some robots might have a better map and are therefore able to make better decisions about where to go next, placing them at the top of the information hierarchy. Intelligence hierarchies consider heterogeneous swarms with some robots having higher sensor and computational capabilities (while others are more limited) that lead to enhanced intelligence and decision making, thus placing them at top of the hierarchy.

Most of the swarm intelligence techniques applied to robot swarms [39, 24, 8] focus on a single level of hierarchy and employ a uniform behavior across the swarm. Heterogeneity is generally addressed by assigning task-specific roles to robots [102, 159] or allowing roles to emerge [160]. Hierarchical approaches to swarm intelligence have been investigated in the context of particle swarm optimization [161, 162] but they have received very little attention in robotics. The shepherding problem [163–165] considers a 2-level hierarchy with two types of robots: *sheep* that need to be herded by robot *dogs*. The problem considered in this work is comparable with the shepherding problem: we use smarter *guide* robots to shape and

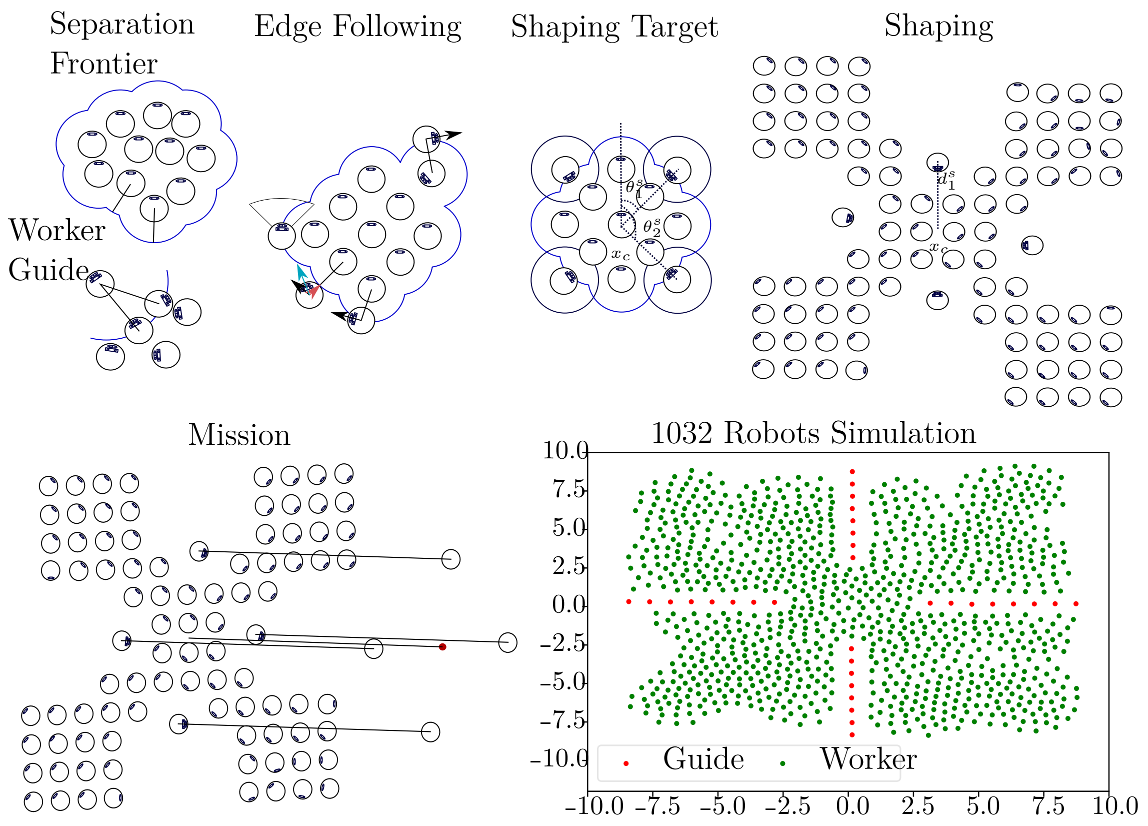


Figure 6.1 The states followed by the guide robots in our system.

transport a group of *workers*. The the main difference with shepherding is that our *guides* (or dogs in the shepherding analogy) shape and maintain the pattern made by *workers* (a.k.a. the sheep) during motion by **operating on the rules that govern the workers swarm**.

To shape the swarm, the guides take up certain locations around the workers and, using a task allocation mechanism [42], they shape the swarm using virtual potentials. Once a desired shape is attained, the workers are "locked" in position and translated by the guides using virtual forces. The guides coordinate with each other by tracking the overall movement of the swarm, the orientation its members towards the target, as well as the position of the center of mass of the formation. The guides interact with the workers by operating on four control parameters: density ($\rho$), field of view ($FoV$), mode ($M$) and reactive field of view ($RFov$). These parameters are used to influence the formation and the bonding between the workers. This approach resembles smart particle design [166] and finds potential applications in smart materials and targeted drug delivery. Just like smart particles can be designed to be dynamically programmed to change their attraction forces, a few intelligent robot guides can order the workers to a desired shape and move them to a target location by operating on the attractive and repulsive forces between the workers.

We evaluate our approach of hierarchical swarm control using a physics-based simulator (ARGoS3 [96]) with varying number of robots (50, 100, 300 and 1000), drawing three different patterns. We empirically study the convergence time for shaping and transport of the swarm, as well as the distortion incurred by the swarm after reaching its target location. We also evaluate our approach with a small swarm of 6 robots using visual inertial odometry to estimate the relative position between the robots, without any external positioning system, obtaining a performance comparable to the simulations.

## 6.2 Related work

Patterns in robot swarms are generally achieved by designing collective behaviors that emerge as a result of local interactions with neighbors [39]. Some examples of pattern formation are morphogenisis [39], self-assembly [24] and progressive shape formation [113]. These methods create stable structures using only local interactions.

**Flocking.** In some applications, robots are required to move while maintaining a pattern or formation. This behavior is usually referred to as flocking [23, 167], taking inspiration from the behavior of birds. Flocking is extensively studied in literature, with hundreds of different approaches [168] inspired from the basic microscopic model [169] of attraction, repulsion and cohesion. Methods for flocking vary based on the way the neighbors are selected, for example, using the $L^2$ [170] or Voronoi distance [171]. Flocking has been applied to aerial [23, 172] and ground robots [170]. However, flocking does not allow for the creation of arbitrary structures: generally flocks are limited to maintaining a common heading and inter-robot distance either in unbounded [172] or bounded spaces [23].

**Leader-follower methods.** Some approaches use a leader-follower strategy for flocking, where a few robots act as leaders [173] that share metrics and positions with the other robots, which in turn use this information to coordinate their motion. The leader-follower problem is widely studied [174, 33, 35, 34] and we can only take a small representative set from literature for discussion. Meng et. al. [35] proposed an optimal control technique using algebraic graph theory for coordinating a group of robots to follow the motion of the leaders, with focus on controllability and control optimality. The approach in [35] and [175] uses Hilbert space projection and dynamic programming. These algorithms are centralized for the leaders and require access to a global positioning system. In a decentralized setting, with robots that have limited visibility and communication constraints, Panagou et al. [34] employ a tractor-trail method to have the followers form a single file and follow the leader in a known environment with obstacles. Some methods consider the robots in a leader-follower setting to be virtual springs [176] that needs to maintain formation. Ji et. al. [177] proposed a two-phase method that first moves leaders to a fixed goal and then moves the followers within the convex-hull generated by the leaders position. Most of these methods in leader-followers are computation intensive, with requirements that increases with the number of robots, with rare exceptions [176] that rely on local information and do not suffer from scalability issues.

**Shepherding.** The leader-follower approach has its natural extension to the shepherding problem: simple robots (the 'sheep') that are herded by one [178] or a group of robots (the 'shepherds') [179]. One common approach to shepherding is to approximate a cluster of sheep as a circle or ellipse that needs to be maintained over motion [179, 164, 165] by placing the shepherds behind or around the sheep herd. The approach in [165] used simple computation free robots with a single line-of-sight sensor to evolve a shepherding controller using evolutionary optimization. This approach requires a large number of shepherds to encircle the sheep to succeed. Strömbom et. al [180] proposed a two-phase approach that first gathers a group of unwilling agents into a cluster and moves them to a target location using a simple attraction/repulsion model. Hu et. al. [163] proposed occlusion-based herding that places the shepherds in the occluded region to the goal behind the sheep: this approach decomposes the control input into various contributions of attraction and repulsion between sheep, and repulsion from shepherds. There is a wide body of other comparable approaches that use Hilbert space filling and path planning [181], repulsive force [182] and connected-component labeling [164]. A large number of these approaches require a reliable communication between the shepherd and sheep, except a few that rely on local measurements [165]. Similarly to flocking, most shepherding approaches do not focus on maintaining a pattern during the translation to a goal.

We consider a variation to the shepherding problem, with two main differences: first, we maintain the pattern during movement to the goal. Maintaining the pattern is important for applications like nanomedicine or smart materials, as the relative position of the particles might have important functional implications. Second, the shepherds (the guides in our parlance) have the ability to change the interaction forces between the sheep (i.e., the workers), which truly establishes a hierarchical control system on the swarm.

## 6.3 Model and System

Consider a group of $n$ robots with each robot $i$ having the following dynamics

$$\dot{x}_i = v_i, v_i = u_i, i \in R, R = \{r_1, ..., r_n\} \tag{6.1}$$

with $x_i$, $v_i$ and $u_i \in \mathbb{R}^2$ being the position, velocity and control input respectively. The relative position ($x_{ij}$) and velocity ($v_{ij}$) of two robots $i$ and $j$ are $x_{ij} = x_i - x_j$ and $v_{ij} = v_i - v_j$. We define the set of relative inter-agent distances $D = \{d_{ij} = \left\|x_i - x_j\right\| \in \mathbb{R} | i, j = 1, ..., N, i \neq j\}$ and the communication graph formed between the robots as $\mathcal{G} = (R, E, W)$ with $R = \{r_1, ..., r_n\}$ the set of vertices, $E = \{e_{ij} | i, j \in R, i \neq j\}$ the edge set and $W = \{w_{ij} \in \mathbb{R}_{>0} | e_{ij} \in E\}$ are weights assigned to the edges. Each robot $i \in R$ forms a neighbors set $N_i = \{j \in R | \left\|x_{ij}\right\| \leq d_{com}\}$ as dictated by $E$, with $d_{com}$ representing the communication range of the robots.

Our swarm has two types of robots: guides and workers, further dividing the graph vertex set $R$ into $R^G$ (vertices corresponding to guides) and $R^W$ (vertices corresponding to workers), with $R^G \cap R^W = \emptyset$. The guide robots are assumed to be equipped with sophisticated sensors capable of performing localization, path planning, traversability estimation, etc. The worker robots are simple and oblivious, being capable of only communicating and measuring the relative position $x_{ij}$ of their neighbors. The enhanced capabilities of the guide robots place them in a leadership role: they perceive mission goals and provide directives to the worker robots both indirectly (by moving to a different position) and directly (by ordering a change to the attractive/repulsive forces among the workers). Just like a smart material or social insects, the workers perform a task without them realizing the significance of their individual contribution.

The neighbor set $N_i$ of robot $i$ can be divided into guide neighbors $N_i^G = N_i \cap R^G$ and worker neighbors $N_i^W = N_i \cap R^W$. The center of mass of the worker robots is defined as $x^c = \frac{1}{|R^W|}\sum_{i \in R^W} x_i$, and assumes a circular approximation of the worker cluster. The robots are only capable of measuring relative positions $x_{ij}$ and hence the center of mass perceived by robot $i$ is $x_i^c = \frac{1}{|N_i^W|}\sum_{j \in N_i^W} x_{ij}$.

Fig. 6.1 illustrates the different states of the guide robots. To shape a cluster of workers, the guides are given $k$ shaping targets, each specified by an angle ($\theta^s$) and a distance ($d^s$) with respect to the worker center of mass ($x^c$). Consider the shape angle set $\Theta^s = \{\theta_1^s, ..., \theta_k^s\}$. and shape distance set $D^s = \{d_1^s, ..., d_k^s\}$. A guide robot $i \in R^G$ is assigned a shaping target $\theta_i \in \Theta^s$ and $d_i^s \in D^s$ through a decentralized task allocation mechanism.

To reach its target angle $-\theta_\delta < \angle(x_i - x^c) - \theta_i^s < \theta_\delta$, with $\theta_\delta$ being a small tolerance, a guide $i$ follows the edge of the worker cluster. Upon reaching $\theta_i^s$, the robots translate to ensure a distance $d_i^s$ from the center of mass of the workers cluster, withing a small tolerance $d_\delta$: $d_i^s - \|x_i^c\| < d_\delta$. Once a shaping target is reached (both angle and distance), the guides wait for all the other guides to reach their respective shape targets using a mechanism called a barrier [48].

Once at their shaping targets, the guides translate to a target distance $d^{sp}$ from the center of mass of the worker cluster, thus forcing the workers into the desired shape. We use again a barrier to ensure all the guide robots have reached their destination, upon which the guides start moving to a sequence of targets $\mathbb{T} = \{x_1, ..., x_n\}, \forall x_i \in \mathbb{R}^2$ maintaining the worker formation obtained through shaping. A guide interacts with the workers through simple potentials by changing its position which in turn causes a repulsive action on the workers. The guides can also manipulate the parameters of the potential fields that control the workers. These parameters can be propagated across the swarm and change the workers reactions to the actions of guides. The available parameters are density ($\rho$), field of view ($FoV$), mode ($M$) and reactive field of view ($RFov$). Fig. 6.2 shows the effect of these parameters.

**Definition 4.** *Given a graph $\mathcal{G}(t) = (R(t), E(t), W(t))$ formed by a group of robots at time $t$, the robots are said to be in a formation if and only if the link between $\{v_i, v_j\} \in E(t)$ is preserved $\forall t \in (t_0, \infty)$. $W = d_{ij}, w : E \to \mathbb{R}_+$ is the weights of the graph encoding the distance between the robots and defining the formation parameters.*

The positions of the robots are continuously evolving over time because of motion. If motion is edge-consistent (i.e. if the distance $\left\|x_{ij}\right\|$ is constant $\forall e_{ij} \in E$ of a graph $\mathcal{G}$ [183]), we say that the formation is maintained. We define a graph to be an even-edged graph of distance $d$ if $\forall \left\|x_{ij}\right\| \approx d, \forall e_{ij} \in E$. The worker robots maintain a even-edged graph with distance $d_\rho$ defined by the current density parameter $\rho$.

**Definition 5.** *The field of view neighbor set of robot $i$ is $N_i^{FoV} = \{j \in R^W | d_{ij} \leq d_{FoV}\}$, with $d_{FoV}$ being the field of view distance and $R^W$ being workers set. Similarly, a reactive field of view neighbor set $N_i^{RFoV} = \{\forall j \in R^G | d_{ij} \leq d_{RFoV}\}$, with $d_{RFoV}$ being the reactive field of view distance and $R^G$ being the guides set.*

The field of view distance $d_{FoV}$ is used to control the field of view neighbor set $N^{FoV}$ and influence the strength of a robots position in a formation. Reaction distance to a guide robot $d_{RFoV}$ is defined by $RFoV$, influence the distance at which the workers will start responding to guide robots. The mode of the worker robots $M \in \{Loose, Shape, Rigid\}$ defines the graph behavior of the worker robots, i.e., maintaining even-edged graph or edge-consistent graph. The density parameter $\rho$ is utilized both in Loose and Shape mode to maintain an even-edged graph, and an edge-consistent graph in Rigid mode.

## 6.4 Methodology

### 6.4.1 Worker Swarm

The control parameters that dictate the behavior of the worker robots are density ($\rho$), field of view ($FoV$), mode ($M$) and reactive field of view ($RFov$). $\rho$ defines the desired inter-robot distance $d_\rho$ to preserve with the neighbors in field of view ($N_i^{FoV}$). $d_{RFoV}$ is the distance at which the robots start reacting to guide robots. The four control parameters of the worker robots are set by the guide robots using virtual stigmergy [68], a conflict-free replicated data structure that acts as a global distributed shared memory for the swarm. Virtual stigmergy uses gossip-based communication that propagates through the network updating the local copies of information on the robots, and was shown to converge if a communication path exists (connected graph).

A common method to design a control law for the sheep in shepherding [163] or flocking [184] is to use artificial potential functions that produce attraction and repulsion between the robots. Unlike the shepherding problem, we need to maintain the desired formation during its translation. Potential functions used in modeling the bonding structure of atomic molecules [185] are some interesting candidates for enforcing a bond between two robots. We considered various potentials like Morse potential (used in modeling diatomic molecules), Lennard-Jones potential (used in modeling rare gases), Buckingham potential (used in modeling ceramic materials) and the harmonic approximation (used in modeling crystalline lattice structures) [185]. Morse, Lennard-Jones, and Buckingham potentials produce uneven attraction and repulsion forces when applied on robots, for instance produce stronger repulsion and weaker attraction becomes a problem when these parameters are dynamically configured by

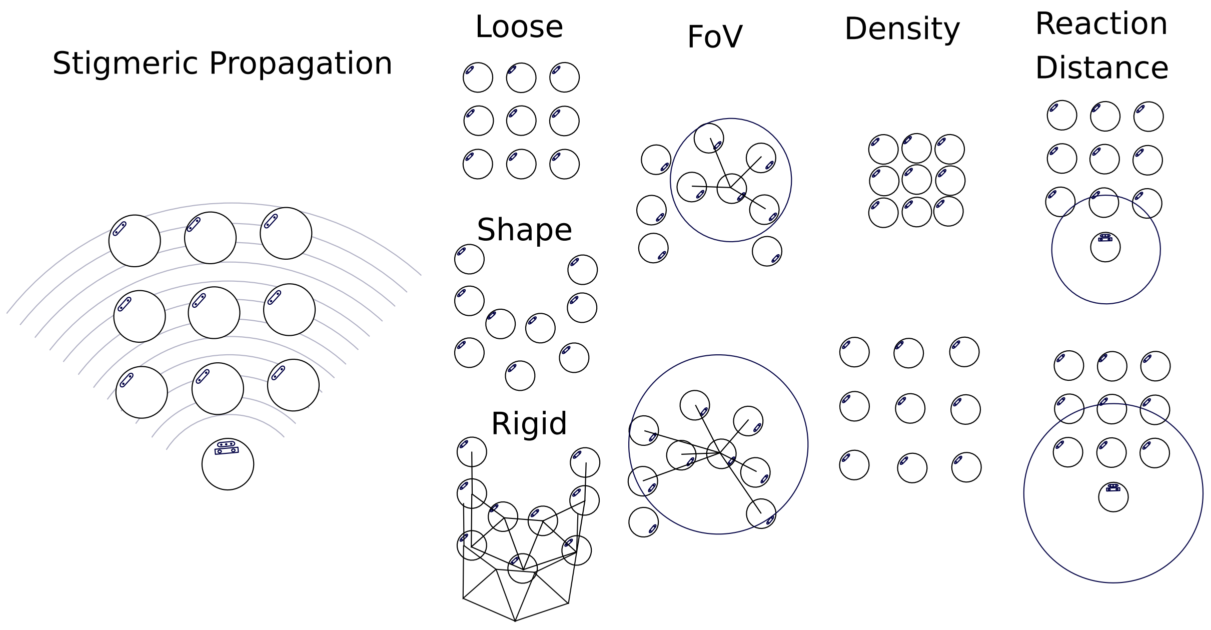


Figure 6.2 Illustration of the control parameters used by the worker swarm to preserve a shape, the control parameters are controlled by the guide robots to change the properties of the worker robots.

the guide robots. In this work, we choose the harmonic approximation to define the bonding force between the worker robots. A standardized Harmonic potential is of the following form $\phi(x) = \frac{1}{2}k(x)^2$, we empirically shape the potential to obtain a desired force response for a given distance and obtain the potential function:

$$\phi(d) = a_0 + \frac{1}{2}k(d)^2 \tag{6.2}$$

with $a_0$ and $k$ being design constants. The feedback control contribution for maintaining the formation between other worker robots within the field of view $FoV$ is $w_i$ and the control contribution for producing a repulsive reaction from the guide robots within the reactive field of view $RFoV$ is $g_i$

$$u_i = \alpha w_i + \beta g_i \tag{6.3}$$

$$w_i = \frac{1}{|N_i^{FoV}|} \sum_{j \in N_i^{FoV}} \phi(d_{ij} - d_\rho) \frac{x_{ij}}{\left\| x_{ij} \right\|} \tag{6.4}$$

$$g_i = \frac{1}{|N_i^{RFoV}|} \sum_{j \in N_i^{RFoV}} \phi^+(d_{ij} - d_\rho) \frac{x_{ij}}{\left\| x_{ij} \right\|} \tag{6.5}$$

The potential function $\phi(d_{ij} - d_\rho)$ provides the magnitude and $\frac{x_{ij}}{\left\| x_{ij} \right\|}$ determines the direction of each $w_{ij}$ and $g_{ij}$ contribution. $\alpha$ and $\beta$ are design parameters that define the relative importance of maintaining formation with respect to reacting to the guides in proximity. The function $\phi^{+}(d) \rightarrow \mathbb{R}^{+}$ produces only repulsion mapping to positive real values if and only if the input domain is $d \in [0, \infty]$. Similarly, the function $\phi(d) \rightarrow \mathbb{R}$ produces real values when $d \in [-\infty, \infty]$. Both functions $\phi^{+}(.)$ and $\phi(.)$ are Lipschitz continuous and belong to class function $\phi^{+}(.) \in S^{+}$ and $\phi(.) \in S$. The class functions are defined as $S^{+} = \{f : [0, \infty) \rightarrow \mathbb{R} | f(d) = 0, \forall d \geq 0, f(d) > 0, \forall d < 0\}$ and $S = \{f : (-\infty, \infty) \rightarrow \mathbb{R} | f(d) < 0, \forall d > 0, f(d) > 0, \forall d < 0\}$. From the class definition and equ. 6.3, an attraction force exists between all the robots with distance $d_{ij} > d_{\rho}, \forall i, j \in E$ and a repulsive force acts on the robots with distance $d_{ij} < d_{\rho}$. The equilibrium point with $u_i = 0$ is when all the neighboring robots of robot $i$ are at distance $d_{\rho}$ and with no guide robots within $d_{RFoV}$.

A worker swarm takes one of the three modes: 1. Loose, 2. Shape and 3. Rigid, each resulting in distinct swarm behavior. In the loose mode, the worker robots maintain a formation with distance $d_{\rho}$, allowing them to create new edges when a new robot appears in its neighbor set. The worker robots behave like a fluid in loose mode since the robots are trying to maintain an even-edged graph, a perturbation caused by the guide robot will cause the workers to act as a fluid to the movement of the guide. The mode transition from any other mode to shape formation creates two new sets: a distance lookup set $D_i^{lo} = \{d_{ij} | d_{ij} < d_{FoV}; \forall j \in N_i^{FoV}(t)\}$ containing the distance of all worker neighbors, lookup set $N_i^{lo} = \{j | \forall j \in N_i^{FoV}(t)\}$ containing the id of worker neighbors, with $t$ being the time at mode transition. The worker robots use $D_i^{lo}$ in shape mode to maintain a formation only with the respective robots in $N_i^{lo}$ and remove robots from sets $N_i^{lo}$ and $D_i^{lo}$ if the distance $d_{ij} > d_{FoV}$, allows link breakages with robots outside $d_{FoV}$. The link breakage in a formation allows the worker robot cluster to create a desired shape, when a guide robot forces itself into the cluster. As in shape mode, the mode transition to rigid creates the sets $D_i^{lo}$ and $N_i^{lo}$, unlike in shape mode, the rigid mode do not allow link breakage. Assuming $d_{com} > d_{\rho}$, worker robots maintain the formation only with robots in set $N_i^{lo}$ and continue to maintain formation with distance $d_{ij} \in D_i^{lo}$ instead of $d_{\rho}$ until the robot is in the communication range $d_{ij} < d_{com}, j \in N_i^{lo}$. Rigid mode enforces a rigid graph to the graph $\mathcal{G}$ formed by the robot group with $E$ being edge-consistent, as defined in [183].

### 6.4.2 Guide Swarm

The guide swarm are the brains of the whole robot group controlling the worker swarm, the muscles that do the work. Guide swarm has a series of internal information and states, in addition to the workers' configuration control parameters. Guides are initialized with

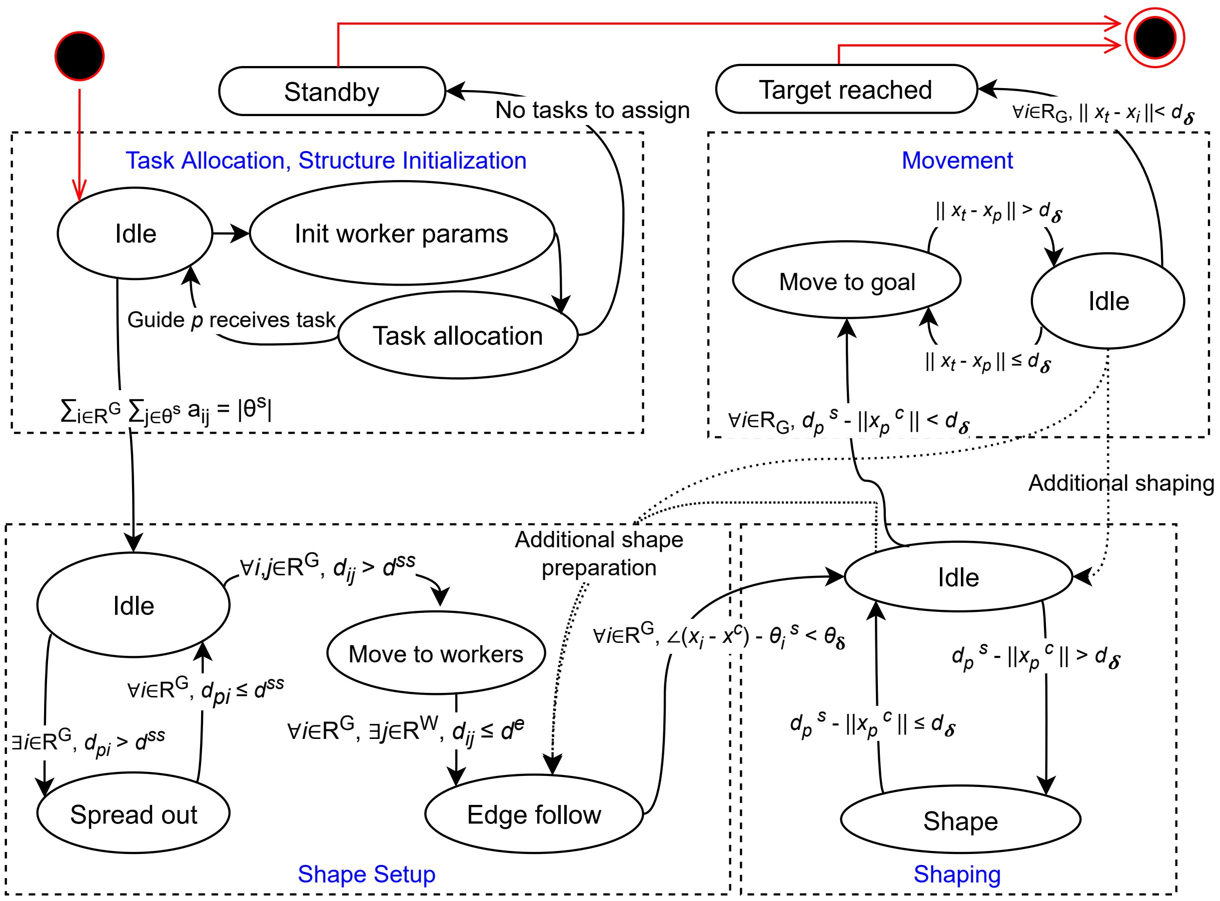


Figure 6.3 Finite state machine describing the high level guide behavior for all $i \in R^G$. The dotted boxes group the high-level states, inside the box are the internal sub-states. The diagram shows the standard single shape objective edge following and shaping procedure in solid black connections. Additional connections for a multi-shape execution is shown with dotted connections.

the overall set of shaping parameters ($D^s$, $\Theta^s$ and $d^{sp}$) and movement targets ($\mathbb{T}$), and are controlled internally by their own state machine. The set of movement targets $\mathbb{T}$ can be easily obtained using any standard robotic path planner. Similarly, the shaping parameters could be obtained by a planner that outputs these parameters for a given shape. The system is designed in a way that allows for dynamic changes to the shaping parameters for a multi-shape movement. In this study, we only consider a fixed parameter set. The parameters for the workers are set through virtual stigmergy, which also defines the properties of the shape and hence it is considered to be part of the shape dependent parameters.

Guides have the following high-level states: 1. Task Allocation, 2. Shaping setup, 3. Shaping, and 4. Movement. The high level state machine of the guides is depicted in fig. 6.3. Our depiction continues to assume a single shape set of shaping parameters, however we show the ability to do multiple shapes in a single execution with dotted connections. All the state transitions between the 4 high-level states occur only with an agreement with all the guides. We use a barrier [48], a decentralized method which holds the state of the robots in a standby state until all the robots satisfy the conditions for the state transition. We implemented the barrier using virtual stigmergy, which provides a tuple space storage.

**Task Allocation** The multi-robot task allocation problem [186] is a combinatorial optimization problem that is formulated as integer programming problem. The combinatorial nature of the problem requires heuristics to obtain an approximate solution in polynomial time. The specific task allocation problem that we have in our setup is a Single Assignment (SA) problem and can be solved using auction or consensus. The SA problem used by the guide robots to assign $t$ to $|R^G|$ robots has the form:

$$\begin{aligned} min \sum_{i\in R^G}\sum_{j\in\Theta^s} c_{ij}(x_i)a_{ij} \\ st. \sum_{j\in\Theta^s} a_{ij} \leq 1, \forall i \in R^G \\ \sum_{i\in R^G} a_{ij} \leq 1, \forall j \in \Theta^s \\ \sum_{i\in R^G}\sum_{j\in\Theta^s} a_{ij} = |\Theta^s| \end{aligned} \tag{6.6}$$

with $c_{ij} = \|.\|$ being the cost function of assigning task $j$ to robot $i$ and $a_{ij} \in \{0,1\}$ being the binary assignment variable. We used the distance to the task estimated using the worker center of mass as the cost. We solve the single assignment problem of equation 6.6 using the decentralized consensus procedure described in [42].

**Shaping setup and Shaping** The task of shaping involves the guides separating from the guide cluster, the robots apply a similar potential in equ. 6.2 to separate and reach a distance $d^{ss}$. Guides edge-follow the worker cluster until the shape angle assigned to them is reached. Algorithm  shows the pseudo-code of the edge-following procedure used by the guide robots reach the shape target angle $\theta_i^s$, the algorithm is partly inspired from [24] but has its difference in maintaining a last followed neighbor list to escape concave edge areas (otherwise stuck with approach in [24]). The problem of guide robots reaching the assigned shaping target angles $\theta^s$ is solved using algorithm 6. The edge following algorithm loops through all the neighbors and finds the closest neighbor to edge follow, if the robot has to yield to a neighbor in front, and updates a list of old neighbors. The old neighbors list is used to avoid following 10 previously followed neighbors. The current move_to vector determining the control inputs for the robot in edge following is computed in line 19. The components of the current move_vec are the perpendicular vector to the edge following neighbor averaged with the distance error in maintain the required edge following distance $d^e$.

In shaping, the guide robots are required to perform a motion by forcing themselves into the worker cluster. The guide robot move in unison at a certain speed to attain the required shape. The speed of shaping for the guide robots are defined using a velocity they will adopt during the shaping process. we define the speed of shaping to be $v^s$. Certain formations require the worker robots to stay behind one another, the shaping with a guide robot in front is performed by closely monitoring the speed of the guide in front.

**Movement** The task of translating in formation is what we define as the formation shepherding problem. The robots have to preserve the edges $E$ of a graph $G$ as per definition 4.

**Problem 1.** *Given a group of n robots with the capability to measure relative position of its neighbors $x_{ij} \in \mathbb{R}^n$, find a control strategy $f(x_{ij}) \rightarrow u_i \in \mathbb{R}^n$ that will allow them to perform motion maintaining edge-consistent graph $\mathbb{G}$ defined by the robot network.*

Problem 1 is solved through hierarchical organization of the robot swarm, guide robots coordinately apply a force on the worker robots in formation to move them to a target. The coordination between the guide robots happen by measuring the center of mass of the worker cluster and their angle to the movement target vector $x_i^g$. Guide swarm apply the control

Algorithm 6 Edge following algorithm

```
1: cur_distance=0, cur_nei=0, old_nei=0
2: Last_seen = circular_buffer(10)
3: **step loop()**
4: Yield = false
5: **foreach** j $\in N_i$ **do**
6:     **if** $\angle x_j < \frac{\pi}{4}$ and $j \in R_i^G$ **then**
7:         Yield = true
8:         break
9:     **end if**
10:    **if** not in Last_seen and $d_j <$ cur_distance and $j \in R_i^W$ **then**
11:        cur_nei = j
12:        cur_edge = j
13:        **if** old_nei $\neq$ cur_nei **then**
14:            Last_seen.push_back(j)
15:        **end if**
16:    **end if**
17: **end for**
18: **if** not Yield **then**
19:    move_vec=$\perp x_{cur\_nei} + (d_{cur\_nei} - d^e) * x_{cur\_nei}$
20:    goto(move_vec)
21: **else**
22:    goto($[0, 0]^T$)
23: **end if**
24: old_nei = cur_nei
```

law defined by the equ. 6.7 to solve problem 1.

$$u_i = \alpha_g g_i + \beta_g c_i + \gamma_g a_i \tag{6.7}$$

$$a_i = p(\theta_i^{tc}) \left( \perp \frac{x_i^c}{\|x_i^c\|} \right) \tag{6.8}$$

$$p(\theta) = \begin{cases} -1 & , \theta < \pi \ and \ \theta < (\theta_t + \pi) \ and \ not(\theta < \theta_t) \\ 1 & , (\theta < \pi \ and \ \theta < (\theta_t + \pi)) \ or \ \theta > \theta_t \\ -1 & , otherwise \end{cases} \tag{6.9}$$

with $\theta_i^{tc} = \angle(x_i^c - x_t)$. The parameters $\alpha_g$, $\beta_g$ and $\gamma_g$ are design parameters respectively that define the importance of moving towards the target, conserving the desired center of mass distance to the workers and maintaining the orientation between the target and worker center of mass. The control input during the movement phase has three contributions: 1. contribution to move the robot to the target $g_i$, 2. contribution to maintain a distance from worker center of mass $c_i$, and, 3. contribution to maintain the orientation with worker center of mass and target $a_i$. Both $g_i$ and $c_i$ use the harmonic function potential defined in equ. 6.2 as in 6.4. Whereas the contribution to maintain the orientation is performed using equ. 6.8.

### 6.4.3 Theoretical Analysis

To show that the robots do not collide or diverge from a given formation, we define the energy of the system $J(x, v) = U(x) + T(v)$, which is a combination of potential energy $U(x) = \sum_{i \in V} U_i(x)$ and kinetic energy $T(v) = \sum_{i \in V} T_i$.

$$U_i(x) = \frac{1}{2} \sum_{j \in N_i^{FoV}} \int_{d_{ij}}^{d_\rho} \phi(x) dx + \sum_{j \in N_i^{RFoV}} \int_{d_{ij}}^{d_{RFov}} \phi^+(x) dx \tag{6.10}$$

$$T_i(v) = \frac{1}{2} \sum_{j \in N_i^{FoV}} \left\| v_j - v_i \right\|^2 + \sum_{j \in N_i^{RFoV}} \left\| v_j - v_i \right\|^2 \tag{6.11}$$

The potential energy of a worker robot $U_i = U_i^W + U_i^G, i \in R^W$ can be divided in two components $U_i^W$ and $U_i^G$. $U_i^W$ is the potential energy of robot $i$ due to the force from other worker robots and $U_i^G$ is potential energy from the force from the guides.

Consider the radius of the robot to be $r < \delta$, with $\delta$ the collision range. Similarly, consider $d_{FoV}$ the distance at which the link between two robots in a formation is broken. This means that a robot with only one neighbor will be disconnected from the formation if this link is broken. Note that links are broken during shaping to create the desired formation. At collision the potential energy between agents is $U_{ij}^{col} = \int_{r}^{\delta} \phi(x)dx$ and the potential energy between two agents when breaking a link is $U_{ij}^{brk} = \int_{d_\rho}^{d_{FoV}} \phi(x)dx$.

**Lemma 1.** *Consider an actuated group of robots 6.1 using 6.3 with potential $\phi(d_{ij} - d_\rho) = \phi(d_{ji} - d_\rho) \in S, \forall i, j \in E$, the energy of the group is $J(x, v) = U(x) + T(v)$ with potential energy at collision $U_{ij}^{col} > 0, \forall i, j \in E$. The potential energy between the two agents $U_{ij} = 0$ when $d_{ij} = d_\rho$. The agent will get to the equilibrium distance $d_\rho$ by applying 6.4 for all distance $d_{ij} < d_\rho$.*

The potential function $\phi(d_{ji} - d_\rho)$ is strictly positive when $d_\delta < d_{ij} < d_\rho$, because $\phi \in S$. The potential energy of the system follows $U_{ij}^{col} > U_{ij}, \forall d_{ij} > d_\delta$ and hence the robot is repelled to $d_\rho$, avoiding a collision. The potential energy of two robots is $U_{ij} = 0$, because $\phi(d_{ij} - d_\rho) = \phi(d_{ji} - d_\rho), \forall i, j \in E$.

Applying Lemma 1 to the case $d_\rho > d_{ij} < d_{FoV}$ results in a potential energy $U_{ij} < U_{ij}^{brk}$ when applying equ. 6.4. Therefore robot $i$ and $j$ will be attracted towards one another. Note that $U_{ij} < 0, \forall d_{ij} > d_\rho$.

**Lemma 2.** *Consider a group of actuated robots using the control law in equ. 6.1 and applying the potential in equ. 6.2. The potential energy caused by the guide is larger than the worker $U_i^G > U_i^W$ when there is at least one guide robot $j$ within the worker's field of view $d_{ij} < d_{RFoV}$.*

Applying the potential function $\phi(d_{ij} - d_\rho)$ with $d_{ij} = d_\rho$ gives $\phi = 0$ according to the class function $S$ and $\phi \in S$, hence $U_i^W = 0$. As in the previous proof, any two given robots are driven to the equilibrium of the potentials, i.e. $d_{ij} = d_\rho$, whenever $d_{ij} > d_\rho$. When a guide robot $j$ is within the reaction distance ($d_{RFoV}$) of robot $i$, then the potential $\phi^+$ is applied to the guide robot's force contribution $U_i^G > 0$ because $\phi^+$ is strictly positive for all distance $d < 0$. Therefore, with at least one worker $j$ with $d_{ij} < d_{RFoV}$, the potential energy $U_i^W > 0$ with $U_i^G > U_i^W$.

**Lemma 3.** *Consider the same group of actuated robots using the control law in equ. 6.1. We saw in lemma 1 that the workers do not collide and in lemma 2 that the potential from the guides allows shaping. Additionally, the control law also ensures that the guides do not collide with or cause collisions within the worker swarm, regardless of mode.*

The guide approaches and interacts with workers using the control law from 6.7. The workers repel with 6.5 but also try to maintain a safe distance with 6.4. Using lemma 2, workers repel the guides at stability $U_i^G > U_i^W$ imposes a risk of collision with $\exists j \in R^W, d_{ij} \leq d_\delta$ as the worker move closer. If guide $i$ follows equ. 6.7, $c_i$ continues increasing as it approaches the worker COM $x_i^c$. As it does, the attracting $g_i$ and $a_i$ terms remain stable, diminish, or remain zero from the inherent motion in direction of $x_t$ and $\theta_i^{tc}$. The $c_i$ potential following equ. 6.2 is quadratically increasing as $d \to 0$, and thus $c_i > g_i + a_i$. Hence, the guide is repelled and cannot collide with or cause collisions within the worker swarm.

According to Lemma 2, whenever a worker robot is in the reactive field of a guide robot, the force exerted on the worker by the guide is greater than that of another worker, thus allowing the guide robots to manipulate the worker swarm. These manipulations and other motions occur without collisions, as shown in Lemmas 1 and 3.

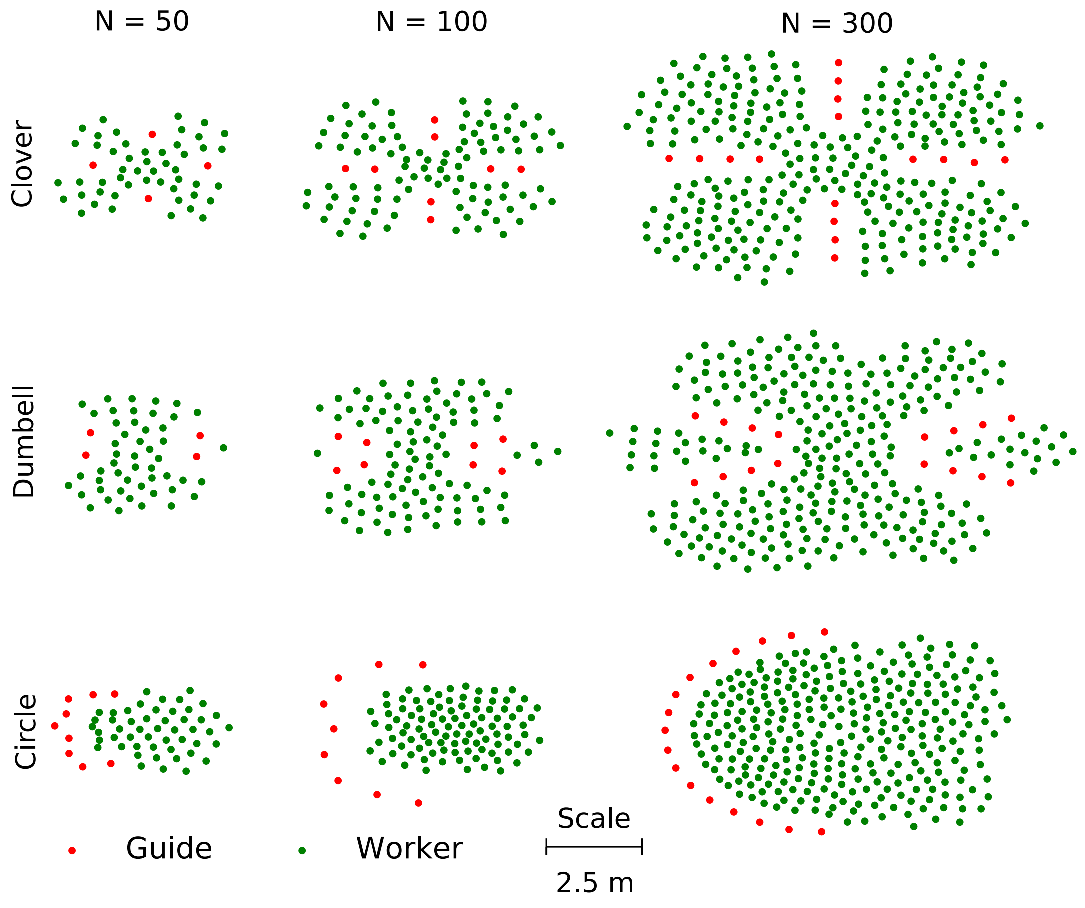


Figure 6.4 The range of shape configurations simulation used in the experiments, with varying swarm sizes. The snapshots are taken as the shaping mode is complete, before the guides transition to movement mode.

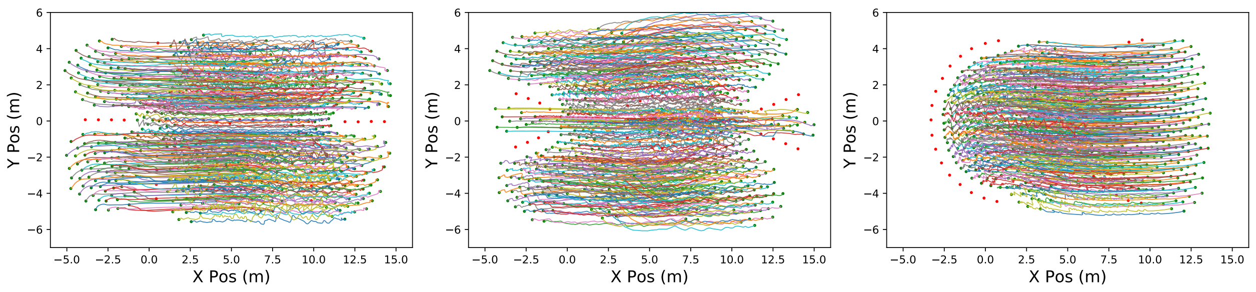


Figure 6.5 The different trajectories undertaken by robots in each shape: clover on the left, dumbbell in the center and circle on the right. Here we can already see the motion of the circle is more direct than the others, with the dumbell showing noticeable vertical and horizontal distortion en route to its destination.

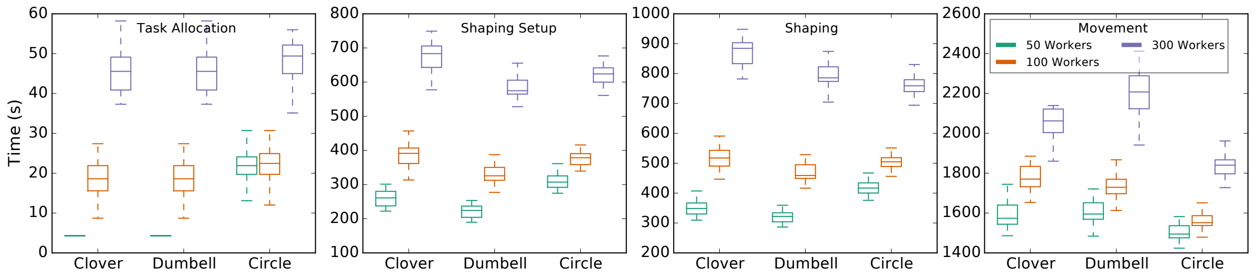


Figure 6.6 Time taken by the worker robots to transition from one state to another. We can see consistently longer and more varying times in the movement of clover and dumbell shapes. The task allocation, shaping setup, and shaping of clovers and dumbells also appear to be affected by increasing the swarm size.

## 6.5 Experiments

We evaluate our approach both in simulation and with real robots. We implemented our approach using Buzz [56] an extensible domain-specific programming language for robot swarms. We open-source our implementation and other analysis for use by the community (concealed for review). We used the Khepera IV robot model in simulation. Real experiments were conducted with custom, modified Khepera IV robots equipped with a Intel RealSense T265 [1] and an Nvidia Jetson TX1 (some robots in the worker class were using a Raspberry Pi 4) as their on-board computer. Robots in simulation used a situated communication [112] model and the real robots used batman-adv [2] a layer two ad-hoc networking protocol (IBSS) to communicate. Custom application layer software based on UDP used the underlying neighbors discovered in layer 2 by batman-adv to broadcast information. Situated communication in the real robots was realized by creating data packets pairs with [position,data]. The robots

[1]`https://www.intelrealsense.com/tracking-camera-t265/`
[2]`https://www.kernel.org/doc/html/v4.16/networking/batman-adv.html`

receiving a new communication broadcast used the positional information and computes a relative distance and angle to its neighbor.

**Simulation** Simulation experiments were performed using a physics-based simulator, ARGoS3 [96]. Simulation configuration used during our experiments were: number of worker robots $N \in \{50,\ 100,\ 300\}$ and three different shapes $Shape \in$ {Clover, Dumbell, Circle} (see fig. 6.4). The case with 50, 100 and 300 worker robots had 4, 8 and 16 guide robots respectively for both clover and dumbell shapes. The circle formation case used 9 guides for both 50 and 100, and 17 guides for the 300 worker case. A different number of dog robots for circle formation was used mainly because of the configuration involved; a lower number of robots do not provide the enclosure required to move the worker robots in this formation. Each experimental configuration was repeated 30 times with random robot initializations using a uniform distribution of parameters $a = \frac{\sqrt{(0.9155*N)/0.1}}{2} + 4, b = -a$. Experiments with 1000 worker robots case were performed only for the clover shape using 32 guides, and were repeated 30 times as well. Fig. 6.4 show a consistent shape for all three formations used in the experiments and plots the actual position of the robots at the end of shaping. As seen in the fig. 6.4 and fig. 6.5 the robots were able to form a consistent formation and move to a target placed at 10 meters from the start location for all three shapes.

The performance metric we used to study and the compare the various formations are time taken by guide state (as shown by fig. 6.6) and the distortion incurred during motion (as shown in fig. 6.7). Distortion metric $\Upsilon$ was computed using:

$$\Upsilon = \frac{1}{N} \sum_{i \in R_W} \sum_{j \in R_W, j \neq i} \left\| x_i(t_s) - x_j(t_s) \right\| - \left\| x_i(t_f) - x_j(t_f) \right\| \tag{6.12}$$

with $t_s$ being the time at shaping and $t_f$ being the time at reaching the target. The distortion metric is normalized with the largest value of all the configuration to obtain a range of $[0, 1]$. The purpose of the distortion metric is to quantify the final deformation of the formation as a result of motion. A value of 1 represents the worst distortion seen in the current formation and 0 is a perfect displacement with no distortion. The distortion metric can be also viewed as the relative change in position of a robot with respect to the worst case encountered in the experimental configuration.

As seen in fig. 6.6, the time taken for the simulations is grouped by Task Allocation for guide robots to select shaping targets, shaping setup to locate the worker flock and edge-follow to the target, shaping to create a desired formation with the worker and moving the desired formation to a target. Task allocation phase has a monotonic increase in time with number of robots and does not vary with configurations, since it is a function of number of robots

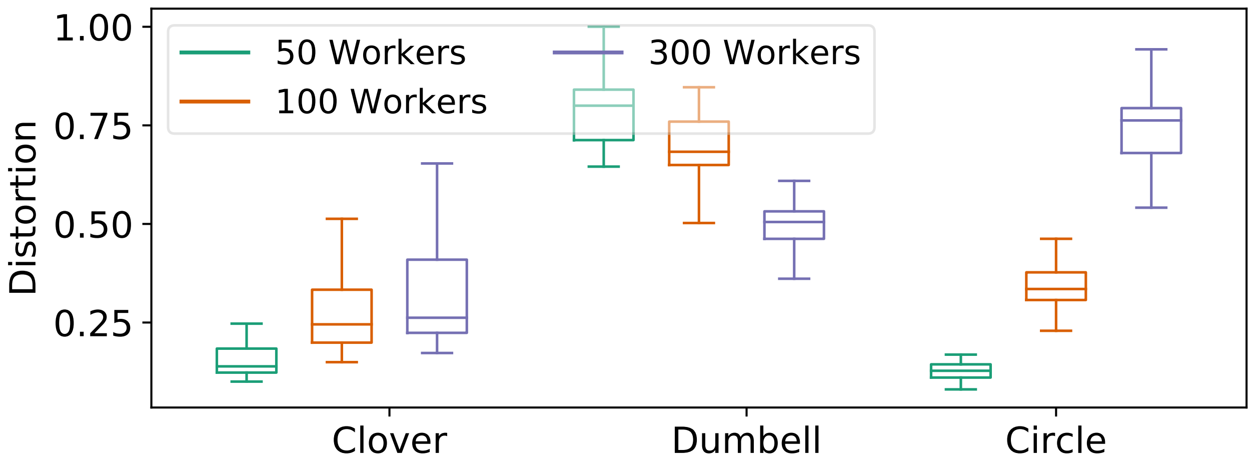


Figure 6.7 Distortion incurred by the swarm in different states, according to the metric defined in equation 6.12 and section $V$. Here we see how the dumbbell is particularly subject to distortion with smaller worker and guide swarms although reducing with higher sizes, whereas circle and clover shapes seem to increase in distortion with swarm size.

and available tasks. In particular, the mean time take was about 4s, 18s, 22s, 45s and 48s for 4, 8, 9, 16 and 17 guide robots respectively. Shaping setup and Shaping exhibit a similar patten because of these steps being configuration dependent, for instance, reaching a certain angular alignment through edge following requires a similar amount of time. Shaping setup on an average took 260s, 350s and 650s for 50, 100 and 300 worker cases in both clover and dumbell shape. In case of circle formation, shaping setup took 315s, 374s and 621s for the three different worker cases respectively. Movement on the other hand has its own dynamics for different shapes with various number of workers and hence exhibits different trends for different shapes. Clover shape with 50, 100 and 300 robots took $17171s$, $16101s$ and $15073s$ respectively. The circle shape took the least amount of time w.r.t other shapes because the robots required a lower number of corrective moves to maintain formation and spent more time moving towards the target.

The distortion metric shown in fig. 6.7 compare the distortions incurred from motion. The clover shape encountered the least distortion for all three cases (50, 100 and 300 workers), mainly because the guide robots are positioned well into the formation acting as pillars of the formation. There is a steady increase in distortion with more robots, which is expected due to the summation used in 6.12 of the distance error that accumulates with more robots. The overall average distortion for the clover shape stayed below 0.3. The dumbell shape on the other hand had the worst level of distortion due to the two ends without guides, which could swing freely and deformed significantly. Even with a lower number of worker robots, lower number of guide robots are unable to steadily hold the two sides of the dumbell in place. In particular, the dumbell had an average relative distortion of 0.78, 0.69 and 0.51 for 50, 100 and 300 robots. The circle had a steady increase in distortion, 0.25, 0.4 and 0.75 on average for 50, 100 and 300 robots.

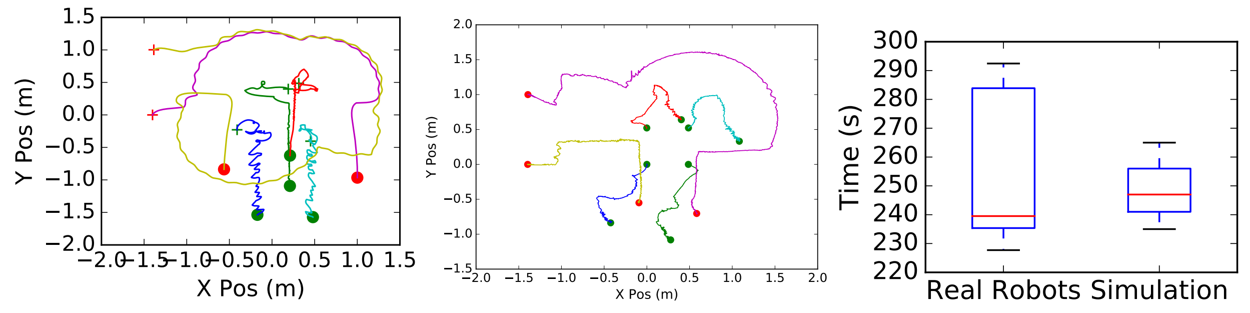


Figure 6.8 Trajectory for shaping and translating a swarm of 6 real robots. Left: simulation; middle: real robots; right: the time taken for the operation. Slightly different actuation and forces affecting the real robot dynamics make for a smoother set of trajectories simulation, albeit subject to minor drift.

**Real robots** We used 6 Khepera IV robots with 4 as workers and 2 guides. We used a simple square like shape to be formed by the robots. The robots were able to task allocate, preform the shaping setup to reach the provided shaping targets and move to a target of about 1m in formation. Fig. 6.8 left shows the trajectory of the robots for a similar shape in simulation, middle, the trajectory of the real robots and the right, shows the time taken to reach the target in formation. The robots took about 240s with real robots and 247s in simulation. The simulation results are comparable to the real robots.

## 6.6 Conclusions

We propose a hierarchical approach to control a swarm of robots that form shapes and translate in formation with two types of robots: a large group of simple robots (workers) and a small swarm of intelligent robots (guides). The process of decision making for formation and translation to targets is performed by the guides, and workers act as the muscle in forming shapes with no knowledge of the overall task. The interactions of the swarms are designed to be simple and local through virtual potential fields that depend on the relative position of the robots. This approach relies on the knowledge of the guide swarm, which coordinates to obtain the desired shape of the worker swarm, like smart particles. The interaction properties of the worker swarm are dynamically adapted by the guide swarm, leveraging its additional sensing and knowledge. The approach scales well with increasing number of robots forming similar shapes. We evaluated the approach on a small swarm of robots in a realistic setting that use visual inertial odometry to compute relative positions, with no additional external setup for position estimation. The prototype framework implemented in this work lays the foundation for hierarchically self-organizing swarms that could be one day be used for targeted drug delivery and cancer treatment using nanomedicine.

# CHAPTER 7 ARTICLE 4: COLLECTIVE TRANSPORT VIA SEQUENTIAL CAGING

**Preface:** This chapter presents an approach to collaborative transport that has the potential to be used in a hierarchical organization. In this work, we consider again worker robots and guide robots. The worker robots are the robots perceived to perform the collaborative transport with the directives from the high-level guide robots. We predominantly focus on the worker's ability to transport the object in this work. The worker robots incrementally reach and enclose an object with no continuous update of the object position and only the relative position of the robots. Once the robots cage the structure, they transport the object through coordination. The main distinction between this work and existing methods is its type of incremental caging and the ability to change direction during motion. Large-scale experiments conducted on this method evaluate the time properties of the approach. Hardware validation of this approach demonstrates its usability.

**Full Citation:** Vardharajan V. S., Soma K, Beltrame G, "Collective transport via sequential caging" *International Symposium Distributed Autonomous Robotic Systems*, 2021 Jun 1 (pp. 349-362). Springer, Cham.

**DOI:** `https://doi.org/10.1109/LRA.2020.3006793`

**Abstract:** We propose a decentralized algorithm to collaboratively transport arbitrarily shaped objects using a swarm of robots. Our approach starts with a task allocation phase that sequentially distributes locations around the object to be transported starting from a seed robot that makes first contact with the object. Our approach does not require previous knowledge of the shape of the object to ensure caging. To push the object to a goal location, we estimate the robots required to apply force on the object based on the angular difference between the target and the object. During transport, the robots follow a sequence of intermediate goal locations specifying the required pose of the object at that location. We evaluate our approach in a physics-based simulator with up to 100 robots, using three generic paths. Experiments using a group of KheperaIV robots demonstrate the effectiveness of our approach in a real setting.

## 7.1 Introduction

Several insect species exhibit an incredible level of coordination to lift and carry heavy objects to their nest, whether for building materials or food. Paratrechina longicornis can collectively transport heavy food from a source location to its nest purely through local interaction with neighboring ants [79]. These ants are capable of carrying an object of arbitrary shape and ten times heavier than their bodies by collaborating in an effective manner. Designing approaches to realize collaborative transport using a group of robots can find its application in warehouse management [187] and collaborative construction of structures [188]. In this work, we take inspiration from these natural insect species to design an approach to collaboratively transport heavy objects (i.e. that cannot be carried by a single robot) of arbitrary shape using a swarm of robots. The main challenges of this type of collective transport are: 1. the effective placement of robots around the transported object, 2. the effective application of force around the object to avoid tugs-of-war, and 3. adapting to the objects center-of-mass movement and the alignment of forces between neighbors. Our approach starts with a task allocation phase, where the robots are sequentially deployed around the object, a process known as caging [82]. Completing the task allocation phase, the robots transport the object towards a target location as shown in fig. 7.1.

Collaborative transport is a well-studied topic: some approaches use ground robots [188, 189, 83, 86] for cooperative manipulation either with explicit communication [190] or force based coordination [191], while others use quadcopters carrying a cable-suspended [157] or rigidly attached [192] object that is heavier than one robot's maximum payload.

Many of the approaches discussed earlier do not explicitly consider robot's interaction with other robots to continuously maintain cage formation, while including an obstacle avoidance mechanism within their control framework. This latter is particularly limiting when the perimeter of the pushed object is small, as it might prevent the robots to get close to the object to apply an effective force. In our approach, we design a sequential placement of robots to avoid this scenario and maintain the initial formation continuously while moving.

Approaches like [193] assume that the position and shape of the object is either continuously known or periodically updated using a central system. Measuring the position and shape of the object in a real world scenarios might be difficult and would limit the use of the transport system to some indoor applications. Furthermore, some approaches either assume that the robots are readily placed around the object to be manipulated [194] and design control strategies for the manipulation of the object. A few approaches provide emphasis on caging or design a control policy for a specific type of caging [81]. In this paper, we

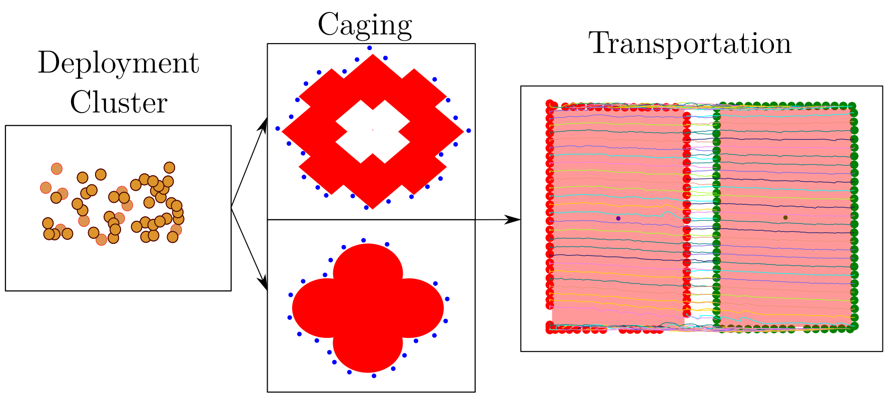


Figure 7.1 Illustration of the process of object transpiration, with the robots starting from a deployment cluster, caging the object and transporting the object.

design a complete system that allows the robots to start from a deployment cluster and take up positions around the object to be transported, in a way that is similar to [83]. Our approach periodically estimates the centroid of the object using only the relative positional information shared by neighboring robots, avoiding the need to have continuous external measurements. Our task allocation allows heterogeneous robots with varying capabilities (e.g. path planning), as well as provides a way to addressing robot failures [42]. We provide sufficient conditions for the convergence of our caging, and show that our approach terminates for convex objects.

## 7.2 Related Work

The concept of caging was first introduced by Rimon and Blake [80] for a finger gripper. Caging is a concept of trapping the object to be manipulated by a gripping actuator, in our work we use a group of robots to act as a single entity to grip the object using form closure as in [82]. A simple form of caging and leader-follower based strategies [195] are employed to push the object by sensing the resultant forces. The main constraint of these works is that the robots cannot follow paths with sharp turns.

Pereira et al. [81] introduce a new type of caging called object closure, in which the object to be transported is loosely caged until the configuration satisfies certain conditions in an imaginary *closure configuration space*. Each robot in the team has to estimate the orientation and position of the object to use this approach. Wan et al. [82] propose robust caging to minimize the number of robots to form closure using translation and rotation constraints. Wan et al. also extended their work for polygons, balls and cylinders to be transported on a slope [189]. This approach requires continuous positional updates from an external system and uses a central system to compute the minimum number of robots required.

An approach to caging L-shaped objects is proposed in [83]: the robots switch between different behavioral states to approach the object and achieve *potential caging*. This approach requires the robots to know certain properties of the object beforehand, such as the minimum and maximum diameter of the caged object. A caging strategy for polygonal convex objects is proposed in [196], the approach uses a sliding mode fuzzy controller to traverse predefined paths. A leader robot coordinates the transportation using the relative position of all the other robots.

Gerkey et. al. [84] propose a strategy for pushing an object by assigning "pushers" and "watchers". Watchers monitor the position of the object and other robots, while pushers perform the pushing task. The approach provides fault tolerance to robot failures and relies heavily on the performance of the watchers. Chen et al. [86] propose an approach in which the robots are placed around an object that occludes the transportation destination: the robots that do not see the destination are the ones that push the object. The intuition behind this placement is that the location that is most effective for pushing the object is the occluded region, i.e. the opposite side of the direction of movement. Our strategy is similar to [86], where we estimate the angular difference between the object and the goal using the proximity sensor, and only the robots that are below a threshold apply a force to the object. Our approach places robots all around the object to adjust and follow a sequence of changing target location, as opposed to only placing robots in the occluded region [86].

The work in [85] combines reinforcement learning and evolutionary algorithms to coordinate 3 types of agents to learn to push an object. "Vision robots" estimate the positions of the object and other robots, "evolutionary learning agents" generate plans for the "worker robots", and these latter execute the plan to push the object. Alkilabi et al. [197] use an evolutionary algorithm to tune a recurrent neural network controller that allows a group of e-puck robots to collaboratively transport a heavy object. The robots use an optical flow sensor to determine and achieve an alignment of forces. The authors demonstrate that the approach works well for different object sizes and shapes, however, the proper functioning of the algorithm relies heavily on the performance of the optical flow sensors.

## 7.3 Methodology

Fig. 7.2 shows the high level state machine used in our collaborative transport behavior. At initialization, the robots enter into a sequence of task allocation rounds allowing the robots to take up positions around the object to be transported with a desired inter-robot distance. Once caging is complete, the robots that are a part of the object cage agree on the desired object path and start pushing and rotating the object. The path is represented as a sequence of points, and all the robots in the cage use a barrier (i.e. they wait for consensus) to go through each intermediate point.

### 7.3.1 Problem Formulation

Let $C_o(t) \in \mathbb{R}^2$ be the centroid of an arbitrary shaped object at time $t$, $x_i(t) \in \mathbb{R}^2$ the position of robot $i$ at time $t$, and $X(t) = \{x_1(t), x_2(t), ..., x_n(t)\}$ the set containing the positions of the robots at time $t$. Let $S_o$ be a closed, convex set representing the perimeter of the object to be transported. Given a sequence of target locations $\mathbb{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ the task of the robots is to take up locations around the perimeter of the object $S_o$ with a desired inter-robot distance $I_d \in \mathbb{R}$ and drive the centroid of the object $C_o$ from a known initial state $C_o(0)$ to a final state $C_o(t_f)$ at some time $t_f$, passing through the target locations in $\mathbb{T}$.

We assume that the robots can perceive the goal in the environment and know an estimate of the initial centroid location $C_o(0)$ of the object to be transported. The mass of the object is assumed to be proportional to the size with a minimum density for the object. We also assume that the line connecting two subsequent targets $\tau_{x-1}$ and $\tau_x$ does not go through obstacles and the perimeter of the object $S_o$ to be transported is greater than $3 * I_d$. We consider a point mass model for the robots ($\dot{x}_i(t) = u_i$) and assume that the robots are fully controllable with $u_i$. The robots are assumed to be equipped with a range and bearing sensor to determine the relative positional information and communicate with neighboring robots within a small fixed communication range $d_C$. In our experiments, we used KheperaIV robots that are equipped with 8 proximity sensors at equal angles around the robot, and we assume similar capabilities in general.

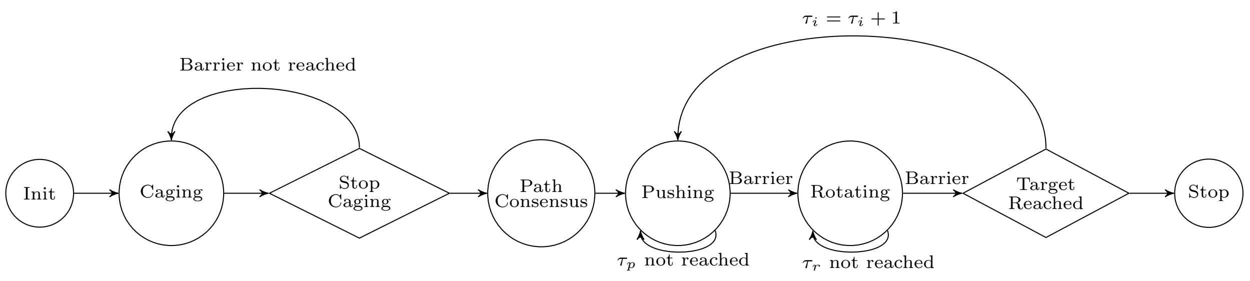
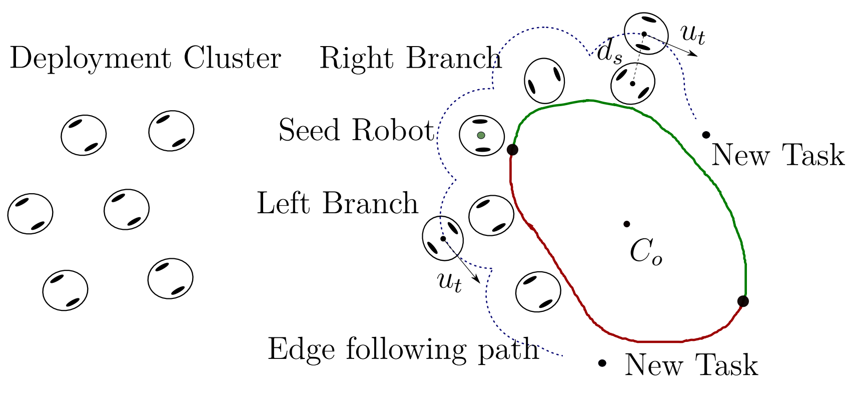


Figure 7.2 High level state diagram



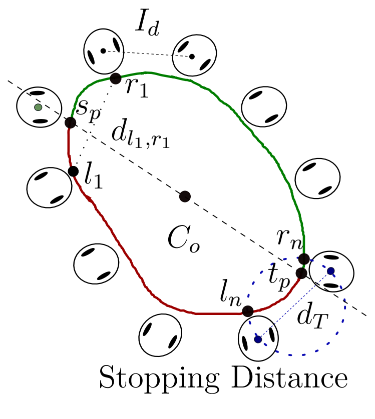


Figure 7.3 Illustration of the process of task allocation based caging (a) illustrate the process of edge following to reach the new target (b) the stopping condition to terminate caging.

### 7.3.2 Task Allocation based Caging

Consider a group of robots randomly distributed in a cluster and a known initial position of the object. The goal of the robots is to deploy to suitable location around the perimeter of the object $C_o$ to guarantee object closure while respecting the required inter-robot distance $I_d$.

The caging process starts with the allocation of the first task (an estimation of the centroid of the object) to a seed robot (closest robot to the object, elected via bidding) as shown in Figure 7.3 (a). The seed robot moves towards the center of the object until it detects the object with its on-board proximity sensors. As the seed robot touches the object, it creates two target locations (one to its left and one to its right, called *branches*). The robots bid for these new target locations and continue the process of spawning the new targets along their branch until the minimum distance between robots in the two branches is smaller than $d_T$.

Our task allocation algorithm performs the role of determining the appropriate target around the object for each robot, the *caging targets*. We consider a Single Allocation (SA) problem [149], where every robot is assigned a single task. The caging targets are sequentially available to the robots, i.e. a new target becomes available after a robot has reached its target. Note that these targets are considered to be approximate (created by establishing a local coordinate system like in [24]), hence they are refined by the robots using their proximity sensors and the position of the closest robot in the branch on reaching the assigned target. The approach of sequential caging is particularly appropriate for scenarios where the shape of the object is initially or continuously unknown, the robots sequentially assign robots to the closure and enclose the transported object.

We use a bidding algorithm [42] (described in the supplementary material [1]): the robots locally compute bids for a task and recover the lowest bid of the team from a distributed, shared tuple space [68]. The robots update the tuple space if the local bid is lower, with conflicts resolved using the procedure outlined in [42]). After a predetermined allocation time ($T_a$), the lowest bid in the tuple space is declared as the winner. $T_a$ has to be selected considering the communication topology and delays to avoid premature allocation as detailed in [42].

To reach the assigned target, the robots edge-follow the neighbors in their target branch. The control inputs ($u_i$) are generated using range and bearing information from neighbors: the robots find their closest neighbor and create a neighbor vector $x_n$ using the range and bearing information. The control inputs are then $u_i = (\perp x_n) + (||x_n|| - I_d) * x_n$; the first term makes sure the robot orbits the neighbor and the second term applies a pulling or pushing force to keep the robot at distance $I_d$. On reaching the target (detected using the proximity sensors), the robot measure the distance to the closest neighbor of the branch and apply a distance correction to keep the inter-robot caging distance to be $I_d$. The robot creates an obstacle vector using the proximity values: $x_o = \frac{\sum_{i \in P} p_i}{|P|}$, with $P = \{p_0, .., p_7\}, p_i \in \mathbb{R}^2$ being the set containing the individual proximity readings as vectors. The inter-robot distance correction control inputs are generated using: $u_i = \perp x_o$. When there are not enough robots to complete the caging, the robots can adapt by increasing $I_d$ and applying inter-robot distance correction control until the termination condition is met.

**Proposition 1** Consider two sets L and R denoting the left and right branch respectively, L contains all the attachment points of the left branch robots to the object, and R contains the attachment points on the right branch. The caging terminates if $\exists p \in L, q \in R$ such that $d_{pq} \leq d_T$ while $d_{lr} > d_T \quad \forall l \in L - \{p\} \wedge \forall r \in R - \{q\}$.

Referring to fig. 7.3(b), consider two closed, convex sets, $L_o = \{l_1, ..., l_n\}$ containing all the points on the curve $l_1 l_n$ and $R_o = \{r_1, ..., r_n\}$ containing all the points on the curve $r_1 r_n$. Set $L_o$ and $R_o$ are ordered and the distance between the two constitutive points satisfy $d(l_n, ln - 1) > d_T$.

### 7.3.3 Behaviors for pushing and rotating

Once two robots around the perimeter of the object satisfy the termination condition and a consensus is reached on the path, the robots initiate the target following routine. The path is represented as a sequence of desired object centroid target locations $\mathbb{T}$ and each entry

[1]https://mistlab.ca/papers/CollectiveTransport/

$\tau_i = (\tau_p, \tau_r)$, with $\tau_p \in \mathbb{R}^2$ and $\tau_r \in [0, 2\pi]$. $\tau_p$ is the local target location and $\tau_r$ is the desired object orientation along the z-axis (yaw) at that target location. The main intuition behind having these local targets is to use a geometric path planner. One of the robot in the swarm with the ability to compute a path to the user defined target $\tau_n$ compute the path and share it as a sequence of states(targets) using virtual stigmergy [68]. The robots sequentially traverse the targets in $\mathbb{T}$, on reaching a $\tau_p$, the robots rotate the object to $\tau_r$. Each robot in caging computes $u_{fp}$ as in equ. 7.1 to exert a forward force and push the object. Similarly, for rotating the object the robots apply $u_{fr}$ as in equ. 7.2.

$$u_{fp} = u_t + u_f + u_{cp}, \tag{7.1}$$

$$u_{fr} = u_r + u_f + u_{cr} \tag{7.2}$$

where, $u_t$ and $u_r$ are a force to move the object towards the target by pushing, and a torque to rotate the object to the desired angle, respectively. $u_f$, as shown in equ. 7.3, is the contribution that makes sure the robots stay in the same formation. $u_{cp}$ (equ. 7.4) and $u_{cr}$ (equ. 7.5) are contributions that ensure the robots stay in contact with the object during pushing and rotation.

**Maintaining Formation** The robot formation from the caging operation tends to get distorted as a result of its application of pushing force on the object to move it towards the target. The robots in the cage apply a force to stay in this formation throughout the transportation task: they store a set $N_f = \{(d_i, \theta_i) | d_i \leq k * I_d, \forall i \in N\}$ that contains the range and bearing measurements of their neighbors, with $k$ being a design parameter. The control input $u_f$ to maintain formation is:

$$u_f = \sum_{\forall i \in N_f} \frac{K_f(d_i - d_{cur})}{d_i} \begin{bmatrix} d_i \cos\theta_i - \cos\frac{\theta_i - \theta_{cur}}{\theta_i} \\ d_i \sin\theta_i - \sin\frac{\theta_i - \theta_{cur}}{\theta_i} \end{bmatrix} \tag{7.3}$$

The first term in the equ. 7.3 is the desired inter-agent distance correction, while the second term applies the desired orientation correction. This formulation is inspired from the commonly used edge potential to preserve a lattice structure among the robots [183]. We apply this edge potential among adjacent robots in a cage to preserve the formation and the desired inter-agent distance.

**Maintaining contact with the Object** The robots in the formation need to determine if they need to apply a force and stay in contact with the object. During pushing, the robots apply a control input to stay in contact with the object, determining its effectiveness in pushing as in equ. 7.4. The effectiveness of a robot's pushing depends on the position of the robot with respect to the object and the target.

The angle $\theta_p$ (a parameter) determines if the robot is an effective pusher: if the angle between the object $x_o$ and the target $\tau_p$ is greater or equal to $\theta_p$, pushing is considered effective, and the robots apply an input $u_{cp}$ to maintain contact. Similarly, the robots apply $u_{cr}$ to maintain contact during rotations:

$$u_{cp} = \begin{cases} [0,0]^T, & \angle(x_o,\tau_p) \geq \theta_p \\ \frac{K_{cp}x_o}{||x_o||}, & \angle(x_o,\tau_p) < \theta_p \end{cases} \tag{7.4}$$

$$u_{cr} = \frac{K_{cr}x_o}{||x_o||} \tag{7.5}$$

where $x_o$ is the proximity vector that determines the current object location in robots coordinate system, and $\theta_p$, $K_{cp}$ and $K_{cr}$ are design parameters. $\theta_p$ is a design parameter that defines the effective pushing perimeter around the object, as shown in fig. 7.4.

**Applying forces** The robots have to exert force in the right direction to move and rotate the object according to the targets in $\mathbb{T}$. The robots must apply force in the desired angular window around the perimeter of the object to avoid tugs-of-war. The control inputs $u_t$ and $u_r$ make sure the robots exert the force in the right direction:

$$u_t = \begin{cases} [0,0]^T, & ||\tau_{p_l} - x_i|| \leq d_{tol} \\ \frac{K_t[\tau_{p_l}-x_i]}{||\tau_{p_l}-x_i||}, & ||\tau_{p_l} - x_i|| > d_{tol} \end{cases} \tag{7.6}$$

$$u_r = \begin{cases} \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \frac{K_r(x_i-C_o)}{||x_i-C_o||}, & \angle(\tau_p,C_o) < \theta_r \\ [0,0]^T, & \angle(\tau_p,C_o) \geq \theta_r \end{cases} \tag{7.7}$$

where, $d_{tol}$ is a design parameter that defines the distance tolerance, $\tau_{p_l}$ is the local target computed by the robot using the centroid position and its position along the perimeter of the object. On reaching an intermediate target $\tau_i$ the robots share their approximate position with respect to a common coordinate system computed as in [24] in a distributed, shared tuple space [68] with all the other robots. The robots retrieve this positional information and compute the centroid of the object $C_o$, which is then used during the rotation of the object.

As in [86], we can prove that the object reaches the goal as $t \to \infty$, with the difference being that the robots exerting force are not based on the occluded perimeter of the object, but are

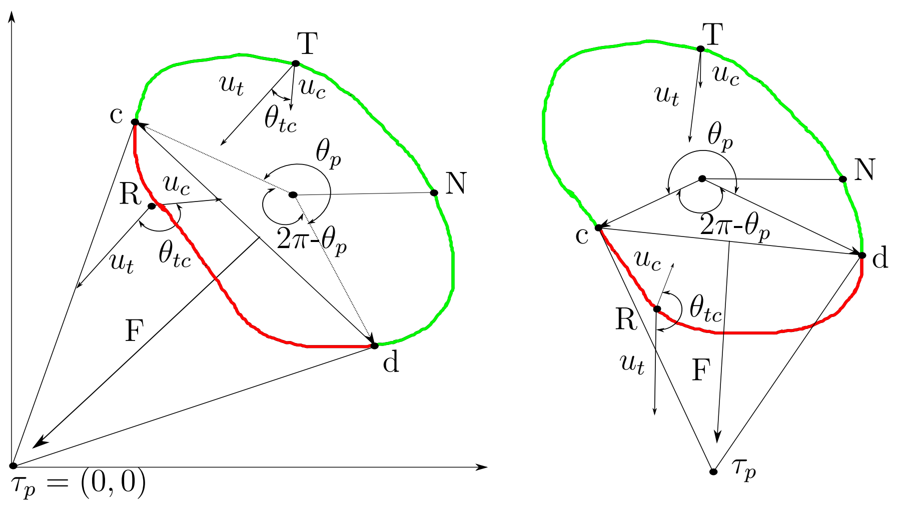


Figure 7.4 Illustration of the resultant force and the angle of effective robots, the effective pushing positions on the perimeter of the object is shown in green.

instead the robots satisfying $\angle(x_o, \tau_p) < \theta_p$.

**Theorem 1.** *The distance between the centroid of the object $C_o$ and the target location $\tau_p$ strictly decreases if the velocity of the transported object is governed by the translation dynamics equation of the object $\dot{v}_o = kF$. For $t \to \infty$, the center of the object $C_o$ reaches $\tau_p$, where $\dot{v}_o$ is the derivative of the object velocity and $kF$ is the fraction of the force that is transfered to the object from the robots.*

Fig. 7.4 shows the resultant F transferred from the robots to the object and the effective angular window (along the curve cTd) on the perimeter of the object to exert force. Consider all the robots along the curve cTd are applying a force using a control input determined by the unit vector $u_t$. The overall force transferred to the object is $F = (c_x - c_y) - (d_x - d_y)$, which is the tangent vector $(d - c)$ rotated by $(\frac{\pi}{2})$ [86].

Consider the squared distance between the target $\tau_p = [0, 0]^T$ and centroid $C_o$ at time t to be $d_g(t) = ||\tau_p - C_o||^2$, taking the time derivative gives $\dot{d}_g = 2k * C_o * F$, substituting F with the resultant force gives $\dot{d}_g = k * ((C_{o_y} c_x - C_{o_x} c_y) - (C_{o_y} d_x - C_{o_x} d_y))$. The distance $d_g(t) \geq 0, \forall t > 0$, when the center of the object lies outside the desired goal $\tau_p$ and since $C_o * F < 0$ is strictly decreasing because of the force applied by the robots, we get $\lim_{t \to \infty} \dot{d}_g = 0$. In other words, the center $C_o$ will eventually reach the goal $\tau_p$.

## 7.4 Experiments

We performed a set of experiments in a physics-based simulator (ARGoS3 [96]) with a KheperaIV robot model under various conditions to study the performance of our approach. We implemented our behavioral state machine for the robots using Buzz [56]. We set the number of robots $N_r \in \{25, 50, 100\}$ and adapt the size $S \in \{[2, 2], [3.6, 6], [7.2, 12]\}m$ and mass $M \in \{5.56, 30.024, 120.096\}kg$ of a cuboid object according to the number of robots. The mass of the object is calculated assuming a constant density hollow material. In another set of experiments, we used three irregular objects: cloud, box rotation, and clover. We set the design parameters of the algorithm to the values shown in Tab. 7.1. We choose the gain parameters for maintaining formation($k_f$), contact with object($k_{cp}$) and force application ($k_t$) based on several rounds of trail-and-error simulations. The tolerance parameters $d_{tol}$ and *Orient. tol.* are chosen to fit our non-holonomic robots: a large part of the error shown in fig. 7.7 is due to the non-holonomic nature of our robots. We evaluate the various performance metrics over three benchmark paths: a straight line, a zigzag, and straight line with two $\angle 90$ rotations (straight_rot in Fig. 7.5). All the paths consists of 9 waypoints (WPs) and straight_rot has its rotations at WP 3 and 6. Each experiment is repeated 30 times with random initial conditions.

**Results** We assess the performance of the algorithm observing the time taken to cage the object and push it along the benchmark paths, plotted in Fig. 7.6. The time to cage the object increases with the perimeter of the object: the median times to cage are 247 s, 779 s and 2753 s for 25, 50 and 100 robots, respectively transporting objects of size $\{2, 2\}$, $\{3.6, 6\}$, $\{7.2, 12\}$. The 3 irregular shapes took around 300 s to cage when using 30 robots. The time taken to push the object is approximately 100 s for the straight path (regardless of the object size) and about 160 s for the zigzag with 25 and 50 robots.

When using 100 robots for the straight line and the zigzag, the system was slightly faster, which could be explained by the higher cumulative force exerted by the robots. The time

Table 7.1 Experimental parameters

| Caging | |
|---|---|
| $d_s$ | 0.35m |
| $I_d$ | 0.45m |
| $d_{tol}$ | 0.05m |
| $d_T$ | $1.85 I_d$ |
| $K_t$ | 30 |
| Prox. thres. | 0.7 |

| Pushing | |
|---|---|
| $\theta_p$ | 115° |
| $K_{cp}$ | $40(< 115°)$, $20(\geq 115°)$ |
| $d_{tol}$ | 0.1m |
| $K_f$ | 40 |
| $K_t$ | 60 |
| Barrier | 90% |

| Rotating | |
|---|---|
| $K_{cr}$ | 450 |
| Orient. tol. | 5.72° |
| $K_f$ | 400 |
| $K_r$ | 600 |
| Barrier | 90% |

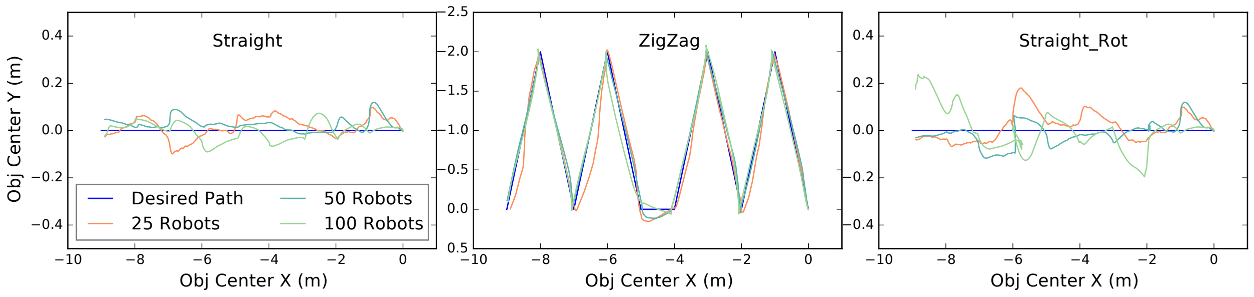


Figure 7.5 Trajectory taken by the centroid of the object vs the desired path in the three benchmarking paths.

taken following a straight path with rotations increases sub-linearly with the number of robots with median times being 135 s, 155 s and 200 s for 25, 50 and 100 robots, respectively.

Fig. 7.7 shows the centroid estimation error, position error and orientation error on each row from left to right. The centroid estimation error increases as the robots progress along the path, which can be explained by the distortion in formation as the robots progress towards the final target. The centroid estimation error for 100 robots is relatively large and shows some variability, which could be largely influenced by the communication topology during the centroid estimation process, as detailed in sec. 7.3. The object position error computed using the difference between the desired position and the ground truth position appears stable around 0.1m, which is within our design tolerance ($d_{tol}$). In the $straight_{rot}$ case with 100 robots, the position error drastically increases around the final WPs, likely due to the drift induced by the second rotation. The orientation error accumulates slowly for the other paths, likely because the pushing force applied towards the target induces a small torque. Without global positioning, the error accumulates at every rotation.

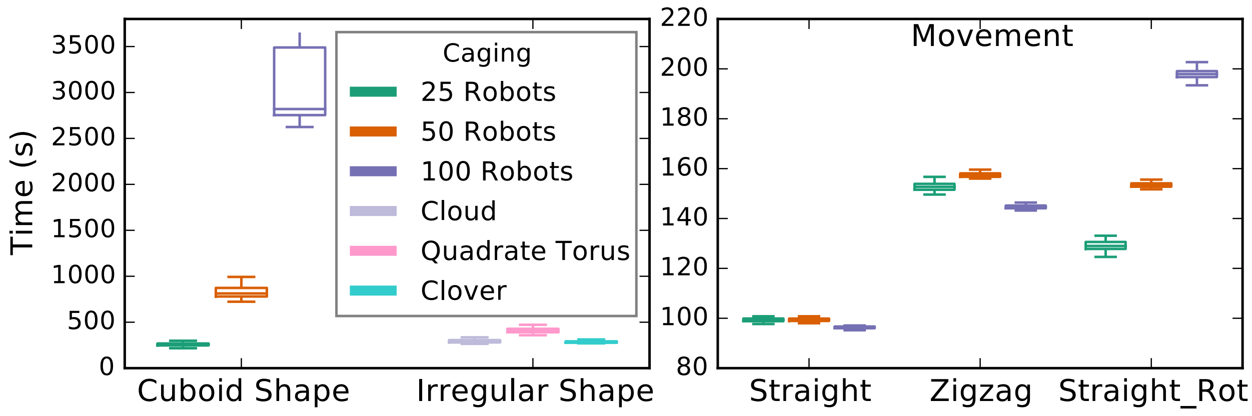


Figure 7.6 Time to complete caging (left) and push an object along 3 paths (right).

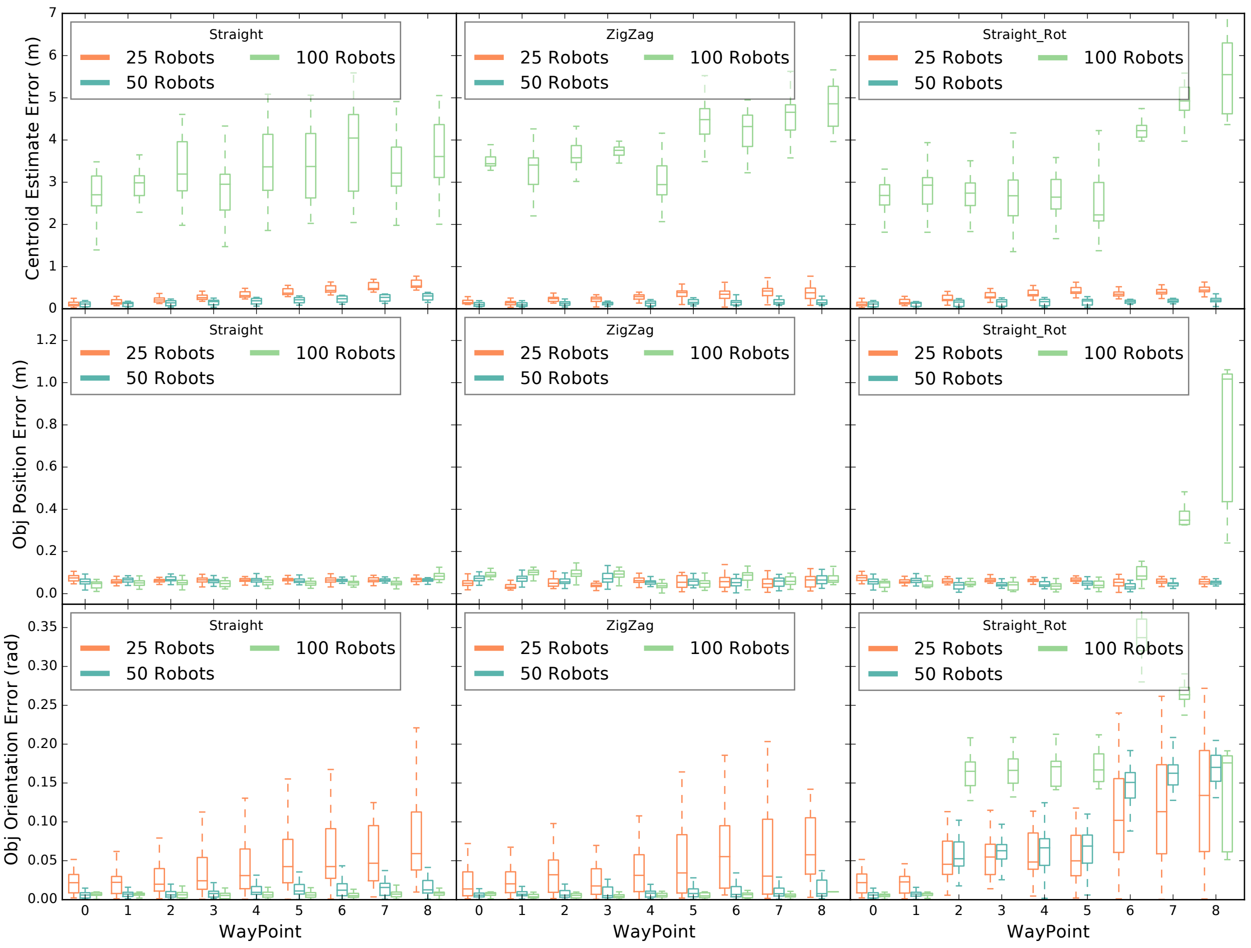


Figure 7.7 From top to bottom: the first plot shows the average centroid estimation error, the second shows the object centroid position error, and the third shows the object orientation error; from left to right, the figure shows the results for our 3 benchmark paths.

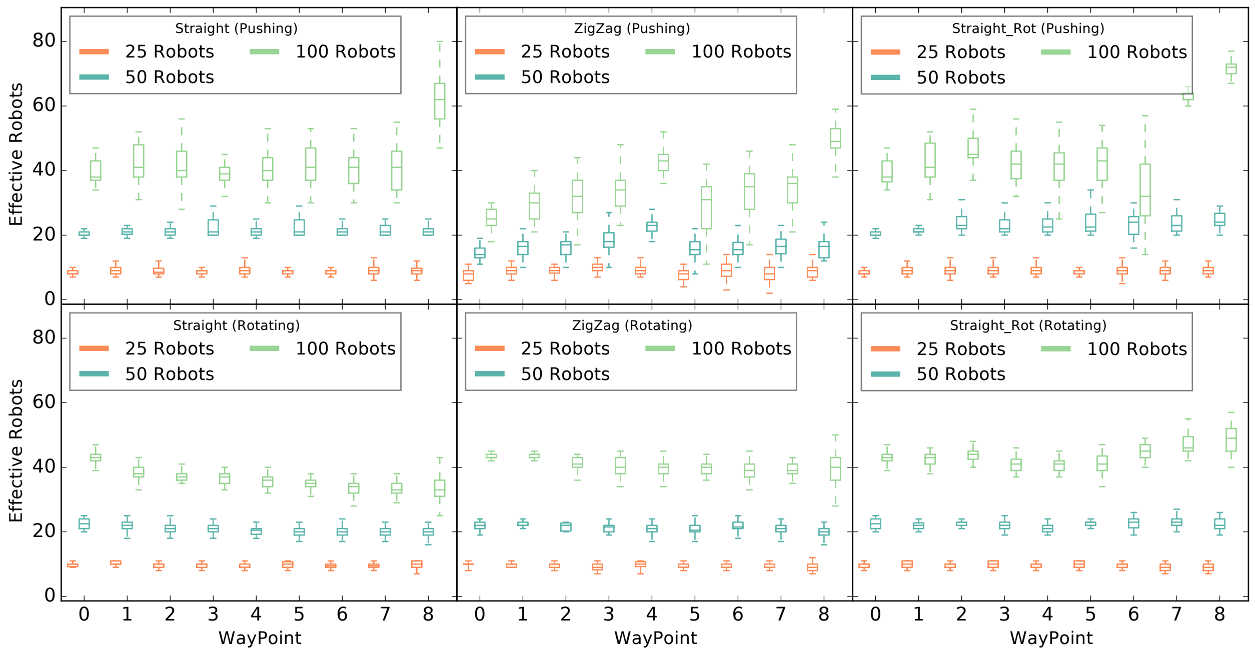


Figure 7.8 Number of robots that were effective in pushing and rotating at different waypoints of the three benchmark paths.

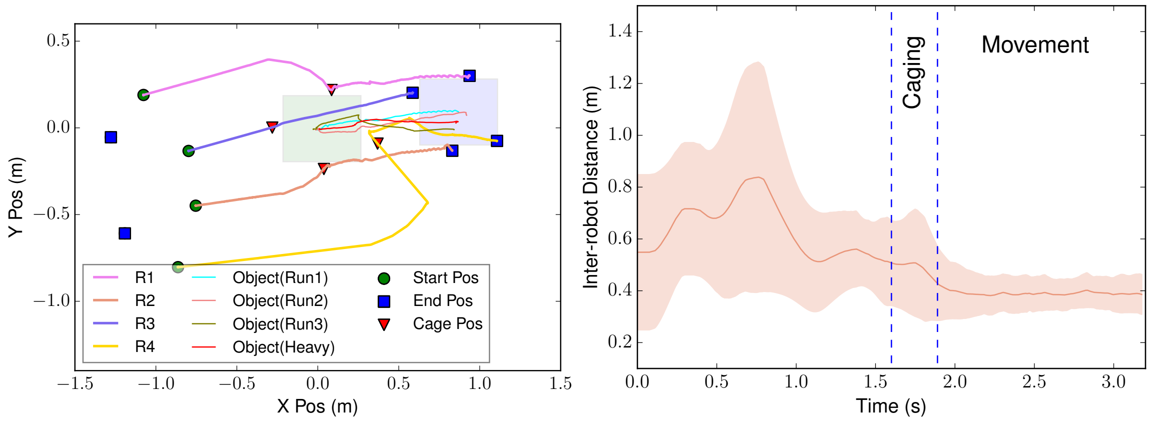


Figure 7.9 Trajectories taken by the robots (left) and the inter-robot distance between the two adjacent robot in the cage (right).

Fig. 7.8 shows the number of effective robots for pushing and rotating the transported object, computed using 7.4. The number of effective pushers appears to increase slowly as the robots progress towards the final target in all cases, which could be due to distortions of the caging formation. The number of effective rotators stays constant for most of the cases, but increases during rotations, due to the robots either getting closer to the corners or the mid point of the object. This could be caused by the large error in the estimation of the centroid resulting in a generation of biased control input to rotate the object.

**Robot Experiments** We perform a small set of experiments using a group of 6 KheperaIV robots. The robots use a hub to compute and transmit the range and bearing information from a motion capture system, for more details on the experimental setup, we refer the reader to [42]. We performed two sets of experiments with robots transporting a foam box of size (0.285, 0.435)m: without any payload, and with 4 kg of LiPo battery on the object. Fig. 7.9 shows the trajectories followed by the robots and the inter-robot distance during the 3 runs without any payload and one run with the payload. It can be observed that the robots were able to consistently reach the target (0,0.9) by following the 3 WPs placed at every 0.3m. The inter-robot distance between the two adjacent robots in a cage approximates well the desired $I_d = 0.45m$ at caging and it is maintained consistently during pushing in all runs with a maximum standard deviation of 0.1m.

## 7.5 Conclusions

We propose a decentralized algorithm to cage an arbitrary-shaped object and transport it along a desired path consisting of a set of object poses. The robots periodically estimate the centroid of the object based on the positional information shared by the robots caging the object, and use this information to transport the object. We study the performance of our algorithm using a large set of simulation experiments with up to 100 robots traversing 3 benchmark paths and a small set of experiments on KheperaIV robots. As future work, we plan to implement a path planner to provide the object path in a cluttered environment.

# CHAPTER 8 ARTICLE 5: OVER-THE-AIR UPDATES FOR ROBOTIC SWARMS

**Preface:** This chapter designs a tool-set for robust software deployment, release, and maintenance on robot swarms. The tool-set applies standard software engineering practices to deploy behavioral code and grant OTA software deployment infrastructure for use with behavioral code deployment in robot swarms. The tool generates patches to the deployed behavioral code on the robots and transmits them to the robots. The robots receiving a patch test the new patch and wait for a consensus among all the robots before switching to the latest release. The approach was evaluated in simulation and with a group of flying robots. The approach designed in this chapter has two types of triggers, the robots can send update patches to other robots, and an operator can send patches to the robots. The former is designed to apply updates to radically different workers' behaviors from the guide robots in hierarchically organized swarms. The operator can use the latter to perform updates on the robots and maintain a consistent version of the behavioral script on the robots.

**Full Citation:** Varadharajan VS, St-Onge D, Guß C, Beltrame G., "Over-the-air updates for robotic swarms" *IEEE software*, 2018 Jan 12;35(2):44-50.

**DOI:** `https://doi.org/10.1109/MS.2018.111095718`

**Abstract:** The currently available methods to deploy software to a group of robots, are rather primitive, and require physical access to the hardware. With the growing numbers of robotic devices introduced by automation and the Internet-of-Things, there is a novel interest in methods and tools to deploy new code to active sensor arrays and swarms of robots. This paper presents an Over-The-Air update toolset designed for robotic swarms to propagate code updates while in operation, without interrupting the mission. New update releases are generated as patches to the deployed code, and a consensus mechanism borrowed from swarm intelligence ensures the execution of a unique code version in the whole swarm at all times. Simulations were conducted with thousands of units to study the scalability and bandwidth consumption during the update process. Real deployment experiments were then performed on a small swarm of commercial quadcopters to demonstrate the effectiveness of the tool.

## 8.1 Introduction

Unmanned robotic teams are growing in popularity for many real world scenarios, such as search and rescue, surgical nanorobots, and space exploration. Despite the technological advancements that were introduced by these applications, there are still numerous problems that need to be addressed for robust multi-robot deployments. One of these challenges is continuous Over-The-Air (OTA) updates to the robot behavior during a mission. A deployment tool that can safely update the robots software OTA with minimal interruption time and without redeployment would be a real mission enabler [133]. Similarly, software deployment tools with OTA for sensor arrays (e.g. for traffic monitoring in smart cities, or use in mines and construction sites) can quickly enhance their adaptability to the environment.

**Common practices** A common method to update the software of a group of robots relies on a connection to a central node either broadcasting or unicasting the changes to each robot within the communication range. The Kilobot platform [24] uses such an approach. This method requires the robots to have access to a pre-deployed network infrastructure; a complex, sometimes even impossible, setup to achieve in real world missions such as space exploration or off-grid emergency response. It also requires to regroup the swarm, process the update and then redeploy it.

One possible way around this limitation could be to install a set of binary before deployment and select a binary in-mission as proposed in [109], [198] and [199]. Nevertheless, the robots still need to ensure consensus on the binary and new binaries cannot be installed in-mission.

**Contributions** This work presents the design of a deployment tool to update a swarm OTA, overcoming the above limitations. Our approach concatenates four techniques: 1. an optimized mechanism to achieve consensus within the swarm (avoiding variability and version conflicts); 2. a packet exchange protocol to relay updates (avoiding a need for pre-deployed network infrastructure); 3. a patch generation software to share only binary deltas (minimizing the bandwidth requirements and update time) and 4. a safe stand-by state during the update process (to avoid a perturbed state within the swarm).

To achieve consensus in a distributed network, our approach borrows concepts from swarm intelligence [136], and it reaches an agreement on the code version with a gossip based algorithm [68]. In this work, a new release is sent to an arbitrarily-chosen robot within the swarm, which in turn creates a patch and propagates the update to the rest of the swarm members, providing an incremental deployment [200] of new releases.

The main advantage, and contribution, of this work is that our approach scales well for large deployments, and can be applied in communication constrained and unpredictable environments. We provide an open-source implementation of our approach based on the Robot Operating System (ROS), as well as a robot-specific implementation for the Khepera IV robot. Our implementation was developed as an add-on to the Buzz Virtual Machine [56] (BVM), where Buzz is a programming language designed for heterogeneous robotic swarms. The terms script and code artifact used throughout the article correspond to Buzz scripts and Buzz bytecode, respectively.

## 8.2 Update Strategy

Fig. 8.1 shows the automaton view of the proposed OTA update process. All the robots in the swarm are initialized with a script artifact of version $C_v = 0$. A new release ($R_n$) of the code can be created from any robot within the swarm (by modifying the local script). A new release is then tested in two steps: 1. the initialization test and 2. the step test. Following successful tests, a new encrypted patch is created for the new release by comparing the differences (deltas) between the old and the new code artifact. The patch is created using the tool `bsdiff` ([1]) and the encryption is performed using a stream cypher called `salsa20` [201].

Every robot within the swarm has two triggers to start an update: from a local file system ($F_t$) and from a neighbor trigger ($N_t$).

**File system trigger** The update is triggered by modifications to the script within the local file system. Robots proceed to a standby state after a successful test. During this stand-by state, the robots are brought to an ideal environmental state to wait for their peers to reach consensus on the new update, defined within a script.

In case of a failed test, the update is dropped and the operator is notified of the test results. Moreover, since $F_t$ is the starting point of the update to be shared within the swarm, it is advisable to conduct extensive tests. Any robot of the swarm can start an update, but sufficient resources have to be available to compile, test and generate the patch in its embedded computer. For heterogeneous swarm, the operator is thus encouraged to start the update from a robot with powerful processing capabilities. However, if none is optimal, the operator can set a standard computer to generate it as long as it has the software infrastructure and access to the swarm network.

[1] `http://www.daemonology.net/bsdiff/`

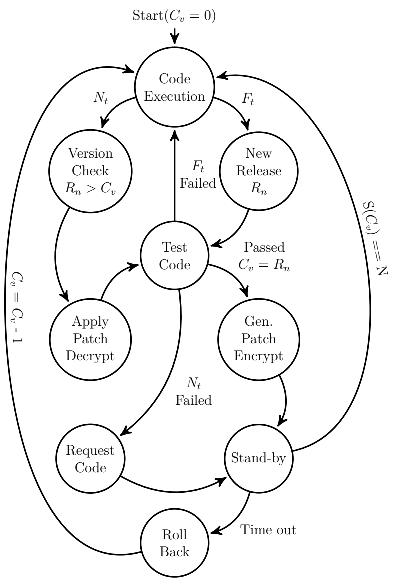


Figure 8.1 An automaton view of the Update protocol.

**Neighbor trigger** The update is trigged by receiving an update patch OTA from one of the peers in the swarm. The robots proceed with an update on reception of a new patch with consistent version number $C_v$. The received patch is: decrypted, tested in two stages as discussed earlier, applied to the current code artifact and a stand-by state is initiated.

If the tests fail, a rebroadcast request is sent to the neighbors. After a second attempt at testing the same patch fails or if any robot time-out while waiting for a patch rebroadcast, the swarm automatically roll back to the previous version.

## 8.3 Validation

Such a critical update system, to be used OTA on UAVs in flight, must be proven successful and stable over a range of assessable parameter set. A series of simulations were performed to validate:

- robustness to various topological distributions.
- scalability up to a thousand units.

To study the impact of various topological distribution on the update time, three topological classes with stationary robots were simulated: 1) `cluster`, 2) `scale free` and 3) `line`.

In a `cluster` topology, analogous to its name, the robots were spatially distributed in clusters and the clusters were densely connected. With the `scale free` distribution, the robots were again distributed in clusters and connections among the clusters were sparse. At last in `line` topology the robots were placed in a line, one behind the other and each robot is connected only to two of its neighbors. A detailed explanation of the topological configurations can be found in [68], from which these topologies were inspired.

**Simulation setup** The total simulation parametric set consists of $\langle$ N, topology, P, C$\rangle$, with N={10,100,1000} being the number of robots, P={0,0.25,0.75} being the packet drop probability and C={2,4,6,8,10,20} representing the patch size in kB. Each parametric choice was simulated 30 times for representative results. Simulations were performed using ARGoS multi-robot simulator ([2]), with a total sum of 4320 experiments excluding the 1000 robots case for `line` topology. This configuration was mainly excluded due to the fact that it consumed enormous time to simulate and the results can be extrapolated.

**Simulation results** Figure 8.2 plots the simulation time taken for 10, 100 and 1000 robot swarms to create, distribute and deploy a new patch of different code sizes OTA, following the three topological distributions.

The experimental results can be roughly summarized as follows: in one hand, the time required for updating a swarm increases linearly with the increase in patch size and the number of robots and; on the other hand, the update time increases exponential with the increase in packet drop. In particular, with `cluster` and `scale free` topology, the swarm updated in tens' of seconds for up to a thousand robots without packet drop. Whereas in `line` topology the update time varied from 10 seconds to over several minutes depending on the patch size and the number of robots.

Cluster and `scale free` topology consumed lower time to update the swarm in comparison with `line` topology. This phenomenon could be caused by the rich communication path within the swarm in both the topologies. These observations on different topologies led us to conclude that the time required to update a swarm, largely depends on the communication paths available within the swarm.

Due to space limitation only a part of the results are discussed, a more detailed experimental analysis and experimental scripts are available online in our supplementary material [3].

[2]`http://www.argos-sim/info`
[3] `www.mistlab.ca/papers/IEEESoftware/2017/`

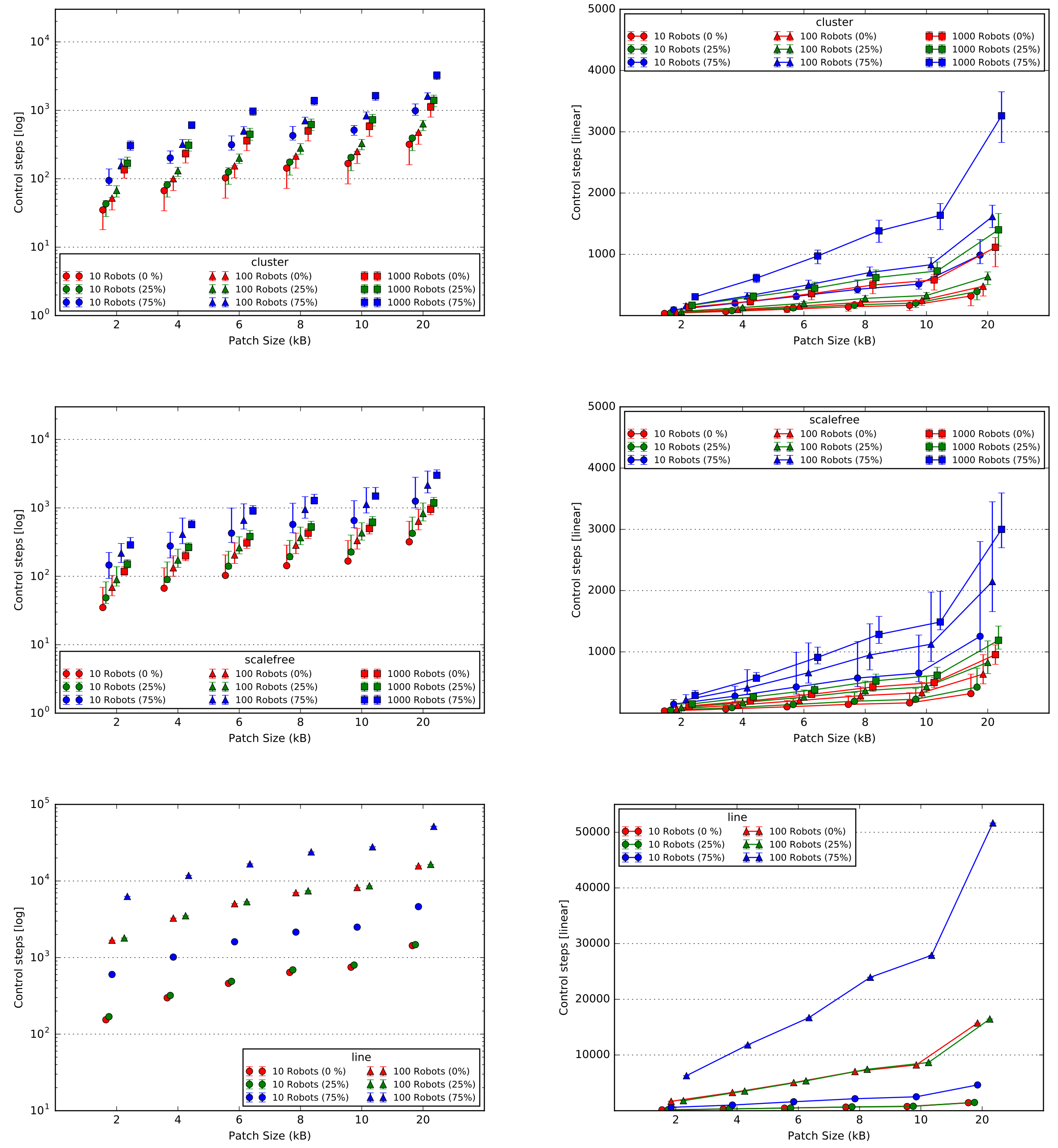


Figure 8.2 Figures top to bottom, on the left: the first figure shows cluster with y-axis in log scale, the second shows scale free topology with y-axis in log scale, and the third shows line with y-axis in log scale. On the right: cluster, scale-free and line topology in linear scale. Each plot show: min, mean and max simulation control steps required for 10, 100 and 1000 robot swarm to apply a new patch of different sizes with 0%, 25% and 75% packet drop. The markers within the plot are slightly offset for visual clarity.

### 8.3.1 Field Experiments

**Setup** To demonstrate the OTA update protocol, we designed an experimental scenario with four quadcopters: three DJI Matrice 100 and a 3DR solo. The DJI M100 were equipped with a NVidia TK1 companion computer and the 3DR Solos, with a Raspberry Pi 3, both hosting the ROS implementation of the update tool (within ROSBuzz [4]). The communication infrastructure between the robots were based on a mesh created by XBee transceivers. An artifact set of sizes {1,2,4,6} kB were selected to be test on these UAVs. The robots were placed at 2 meter distance during the patch size experiments. For statistical significance, these experiments were repeated thirty times.

**Results** Figure 8.3 left and figure 8.3 right shows the number of time steps required by the update initiator and recipient UAVs to update to a new release, over different distances and patch sizes respectively.

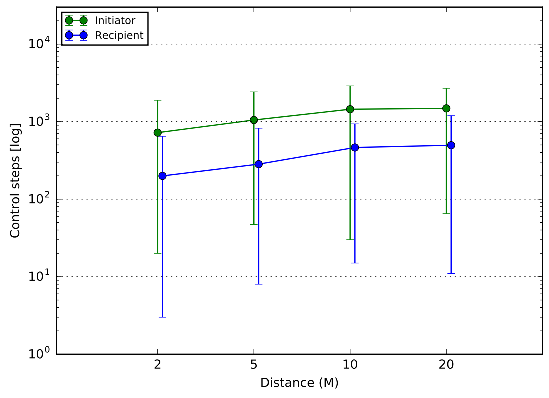
Within the field experimental results, a roughly linear relationship was observed in the update time growth with regard to the increase in inter-robot distances and patch sizes. Specifically, when the distances were below 10 meters, the update process took tens' of seconds and increased up to 280 seconds for distances over 10 meters. The 4 UAV group on an average consumed 10 to 100 seconds to update, to a patch of 1 to 6 kB in size.

Before the implementation of this approach within our experimental setup, a simple tunning of key gains within the script for 4 UAVs required up to 10 minutes for the following: to retrieve each UAV to a home location, manually connecting to the UAVs, update each of their scripts and resume the mission.

## 8.4 Conclusion

We presented an OTA update protocol for robotic swarms, providing a swift method to integrate new behaviors into a swarm with minimal human intervention and interruption time. This approach is a robust distributed method to integrate new behaviors after deployment by ensuring consensus on the updates among the swarm. The generation of patches within our approach minimizes the packet sizes and increases efficiency. Large set of simulation experiments with different topologies and code sizes prove its scalability and flexibility. Evaluation of this approach on a small swarm of commercial UAVs demonstrates the usability and advantages of the update protocol in mission scenarios. The update strategy and the open source tools developed in this work greatly increase the productivity of our team's field

[4]`https://github.com/MISTLab/ROSBuzz/`



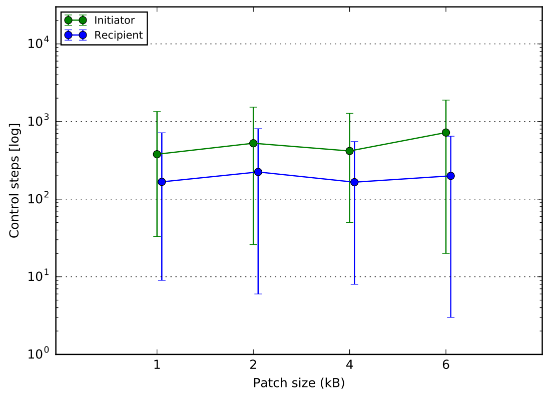


Figure 8.3 The left plot shows the results for varying the distance and on the right varying patch size. Both figures plot the min, mean and max number of control steps needed during thirty repetitions of the update experiments on the UAVs.

experiments and is believed to be a milestone in realizing swarm deployment on real mission scenarios, such as interplanetary explorations and emergency response with unmanned vehicles.

# CHAPTER 9 GENERAL DISCUSSION

This chapter discusses the results and the findings brought by the research presented in Chapters 4 to 8. The discussions presented in this chapter is organized in five dedicated sections that focus on each of the work packages introduced in Chapter 3.

## 9.1 Data sharing in robot swarms

The central aspect of maintaining situational awareness and effective knowledge representation of robots depends on their ability to store and manage data effectively. Current shared memory infrastructures available for robot swarms are limited to consensus mechanisms like virtual stigmergy [68]. In contrast, the sensor data collected by the robots used in perception and scene understanding are generally significant in size. The need for data storage mechanisms that can handle a large amount of data on groups of robots is the primary motivation for the work in WP1 described in Chapter 4.

The approach presented as a part of WP1 includes an open-sourced tool called SOUL. SOUL takes inspiration from DHT to provide a tuple space with key and blob on the programming front and takes inspiration from BitTorrents to split the blob into smaller chunks to be stored on robots with available storage. SOUL on the underlying implementation uses a specific data structure that maintains the meta-data on any given data blob. The meta-data is replicated across the robots to ensure uniform information on the data blob is present in the swarm. This replication of meta-data can sometimes be mistaken as redundant information, but this is essential to ensure uniform information is present across the swarm and provide the robots with the knowledge of the location of all the chunks of data associated with a blob.

Furthermore, SOUL has a short data turnaround time in writing and reading the data from this shared space. At the time of write, the turnaround time is caused by the robot performing the auction to distribute the chunks of a blob to robots with available storage space. Accordingly, the read turnaround time is caused by the robots waiting and receiving the data from the robots holding the data chunks of a blob. In subsection 4.4.1, we study this read and write data turnaround time with stationary and moving robots in various topological configurations with up to a thousand robots. The simulation experiments used a test image of size 1.02Kb, resulting in 1022 datagrams (aka. chunks). The time presented in subsection 4.4.1 is the turnaround time for writing and reading this test image from the shared space. The turnaround time was around 100 control steps for one to four robots injecting the blob simultaneously in a network of 10 robots. This time grows with more robots simultaneously sharing data; for example, the turnaround time was around 1000 control steps with ten robots simultaneously sharing the blob in a thousand robot network. The number of robots present in the robot network significantly influences the turnaround time than the number of robots simultaneously sharing the blob. The auctioning system that distributes the smaller chunks to robots might be the one that most contributes to the turnaround time.

Overall, SOUL enables the storage of a large amount of data by leveraging the storage capacity of all the robots in the swarm with a short turnaround time. SOUL enables robots in a swarm to collect and manage data effectively. Techniques like mission state storage and recovery used in this work paired with SOUL are novel techniques that can directly extend long-term autonomy for Arial robots with limited operation time. The benefits of applying SOUL to heterogeneous swarms become even more conspicuous since the swarm contains robots with varying sensing capabilities, and some can offer more storage while others produce more sensor data.

## 9.2 Connectivity preservation

Connectivity preservation plays a crucial role in ensuring coordination and task performance on a multi-robot system. Methods available to preserve connectivity are either computationally intensive or fallible to faults. The work proposed as a part of WP2 detailed in Chapter 5 proposes a light-weighted approach to maintaining connectivity. This approach can gracefully handle robot failures and self-heal broken communication links; this becomes more important since losing a few agents will not compromise the swarm's performance. Apart from failures, the measurements made by the robots are often very noisy. The sensor measurements noise is even more predominant if the robots are simple and use primitive sensors (e.g., proximity and ultrasound). Simpler robots are often used in swarm robots because of their cost-effectiveness and ability to deploy large numbers. The proposed connectivity approach can handle large amounts of sensor noise (white noise with up to std 0.6); refer to the online supplementary material [1] that presents sensor noise analysis. Its ability to handle sensor noise stems from a reactive approach that creates thresholds on the desired distance between robots in a communication link.

Alternative approaches [202] to measuring the relative position between the robots use the radio Received Signal Strength Indicator (RSSI) and preserve connectivity. RSSI measure-

[1]`https://mistlab.ca/papers/SwarmRelays/`

ments are also often noise-prone when there is no Line-Of-Sight communication. The main factors influencing the RSSI measurements are: path-loss, shadowing, and fading. Some approaches estimate these factors using underlying models that use environmental structure to predict path loss. With line-of-sight propagation (no obstacles), path loss can be approximated as a function of distance. Shadowing dominates environments with obstacles due to the multipath propagation of radio waves. The factors influencing shadowing and fading can be approximated to a certain extent and significantly vary depending on environmental characteristics [203]. The techniques that use models that compensate the RSSI values for the factors mentioned above have been found useful and more precise in enforcing the connectivity, for instance, during the efforts of the DARPA Subterranean Challenge [204]. Applying these methods to robots swarms is again challenging because of the computational demands of these methods. Even though these RSSI-based methods cannot be directly used on robot swarms, the connectivity preservation method proposed in this dissertation can apply to methods that use RSSI measurements.

Furthermore, our analysis of the connectivity preservation approach has made some interesting findings. This approach demonstrated robustness to up to 80% of consecutive robot failures. The robustness to failure indicates the main benefits of using a decentralized reactive approach, often seen in decentralized behaviors. The connectivity methods also demonstrate good scalability properties, another critical property seen in decentralized methods. The properties demonstrated by this method might allow its direct application in many use cases like planetary exploration and search and rescue.

## 9.3 Hierarchical swarms

Advancements in automation and robotics have enabled the deployment of robots on extraterrestrial planets. Consider the curiosity mars rover; it equips several autonomy modules for terrain analysis and navigation. On the other hand, swarm robotics remains undeveloped to realize real-world deployments. Hierarchies could allow application of advancements in robotics to robot swarms.

Having a hierarchical system is important for any application where only a few robots can exhibit a certain level of intelligence. In our case, the guides are the only robots that perceive the mission goals and can lead the workers. The workers are *smart particles* and can only sense and act locally, therefore missing the global view of their mission. Imagine a nanomedicine application: we have tiny robots with very limited intelligence that can be guided by larger more capable robots and “injected” into locations that are inaccessible by the larger robots due to their size. In summary, the hierarchy allows for a smaller number of *expensive* robots with a global view of the mission, while most of the workers are "cheaper" robots that can only sense and act locally, with limited computation. Even assuming that cost is the only metric (i.e. smarter robots can still do the job), in one of our examples, we show a swarm of 1000 "simple" robots guided by 32 "smarter" robots. Optimistically assuming that a cost of 1 for a simple robot and that a smarter robot costs 5, it means the cost of a hierarchical swarm is $1000 + 32 * 5 = 1160$ compared with a full smarter robot swarm $1000 * 5 = 5000$, or a 76.8% reduction in cost.

The hierarchal approach proposed in this dissertation demonstrated very good scalability properties. Scalability essentially means that the approach remains fundamentally usable when increasing the number of robots across several orders of magnitude, and it is not limited to toy examples with few robots. Our approach is completely decentralized, which means that it does not require additional memory or computing resources for workers or guides alike, independently of the number of robots. In addition, the stigmergic propagation has been proven to work with thousands of robots with several topologies [68]. This leaves the sequential approach to task allocation and shape formation as the main bottlenecks of our approach, and we show that the overall time taken follows a linear relationship with the number of robots. We ran tests across 3 orders of magnitude showing practically usable results, which makes us confident in the scalability of our approach for applications with thousands of robots.

## 9.4 Collaborative transport

The collaborative transport problem can be decomposed into two sub-problems, each requiring the robots to perform a specific task. The two main tasks solving the collaborative transport problem are: caging and coordinated transport. The first task deals with the caging, and the second performs the coordinated force application to transport the object. The process of caging is commonly used in collective transport, but the need for continuous object positional information makes existing methods hard to use in practice. Our approach instead uses the relative position of the robots to compute an estimated centroid for the object. The non-reliance on external positioning systems or complicated object position estimate modules makes this approach computationally less intensive and practical for use on simple robots. Another critical factor that influences the performance of the transportation task is determining the robots that will apply the force to the object. In general, the best location to apply force to a collaboratively transported object lies in the occluded circumference of the object and the goal. The proposed method allows only the robots in this occluded region to apply the force. The occluded region needs to be tracked by the robots since it keeps changing based on the direction of motion.

In our experimental evaluations, we used three types of test trajectories and varied the size of the objects, resulting in different weights. Since the simulator used a physics engine, the simulations could encode the weight information during the experimental evaluations. The findings indicate that large objects were transported faster and more precisely along the desired trajectory when the robots collaboratively pushed the object in a straight path. The decrease in time could be because we use a desired fixed distance between the robots in a cage, and larger objects have a bigger circumference, allowing more robots to be present to coordinate the object transport resulting in better control of the object. The potential application of this approach to a hierarchically organized swarm will enable the real-world use of this method. The envisioned approach to such a system might allow the guide robots on the higher level of the hierarchy to act as the supervisors, providing the workers transporting the object with essential directives to manipulate the object efficiently.

## 9.5 OTA behavioral updates

The behavioral update mechanism designed as a part of this dissertation operates using a state machine. The update state-machine can be triggered when a code modification is made locally or when a peer receives a patch. The former generally happens on a ground station and the latter on the robots. We consider the presence of a ground station where the human operator is monitoring the robots and deploying a behavioral patch. For the foreseeable future, human operators will remain indispensable to supervise and manage robot fleets because we are transitioning from systems that are generally already in use; technology gaps prevent us from performing all of the required functions autonomously; and particularly in visible, safety-critical applications, society's trust in decentralized technology will be earned gradually. However, integrating increasingly sophisticated AI techniques leads to increasingly opaque robot control programs. Furthermore, human supervisors' cognitive capacities are challenged (and eventually exceeded) as the size of autonomous fleets grows. The difficulty of ensuring operational performance is compounded when incoming information is scattered, delayed, asynchronous, or unreliable.

The patch generation system used in this work compress the patches, minimizing the amount of information required to be disseminated for an update. Initial experimental evaluations performed in this work suggest that larger patches impact the time to update. The influence of patch size on time is caused by the robots having to distribute larger gossip packets. Larger gossip packets create more traffic in the communication channels and increase the update time. Furthermore, the gossip-based patch propagation and consensus mechanism before switching to the new release make the system suitable for decentralized systems. The absence of proper infrastructure for consensus could create unpredictable behaviors since the robots could run various versions of the behavioral code, and their interactions would be unpredictable. Following the standard software engineering practice of building, testing, and deploying is essential in any software deployment tool and is closely adopted in this tool. The tool designed in the study considers all the above challenges and equips necessary mechanisms for use with a robots swarm, which makes it a dependable tool in our field deployment efforts.

# CHAPTER 10 CONCLUSIONS

This research started with the main aim of improving practical application of robot swarms in real-world scenarios. The six research objectives outlined in Chapter 1 were pursued through the five work packages presented in Chapter 3 and the main body of this dissertation corresponding to the articles (Chapters 4 to 8).

In particular, objective 1—design of a method to store data in robot swarms—was addressed in WPKG1 (Chapter 4); objective 2—develop fault-tolerant methods to enforce connectivity—in WPKG2 (Chapter 5); objective 3—design methods to hierarchically organize swarm—in WPKG 3 (Chapter 6); objective 4—exploit and design use case for hierarchical organization—in WPKG3 (Chapter 6); objective 5—implement behaviors that can leverage a hierarchical organization— in WPKG4 (Chapter 7) and finally, objective 6—Investigate and implement tools that change robot behaviors over time— in WPKG5 (Chapter 8).

The previous chapter presented the impact of the results in the field of swarm robotics, here some concluding remarks of these results are made:

1. SOUL provides a mechanism to store large blobs of data in robot swarms, leveraging the whole swarm's unused storage. The use of an auction-based algorithm results in a short data turnaround time during reading and writing operations. In its current form, SOUL demonstrates good scalability and robustness to failures. SOUL demonstrated the practical use of data storage systems on robot swarms through prototype missions.

2. The communication chain construction approach is a decentralized method that offers all the properties desired in a decentralized system: scalability, robustness to failure, and adaptivity. Swarm relays require the relative position observed by the robots to function and are robust to large amounts of measurement noise. The solutions to real-world problems can sometimes be simple and essentially improve the practicality of the solution.

3. The hierarchical approach took inspiration from biology and demonstrated its ability to perform a realistic mission. The hierarchical approach provides an attractive way to specify global goals to a swarm system. The implemented behavior on the lower-level robots uses a collection of simple mechanisms paired with sophisticated SLAM systems of the guides. Simulation comparisons with egalitarian swarm indicate that the hierarchical approach carries a significant potential for real-world task performance.

4. The particle swarm designed based on the hierarchical organization addressed a variant of the shepherding problem called formation-shepherding. The top-level robots made the whole swarm decision-making, and this approach demonstrated good scalability properties.

5. The collaborative transport approach proposed allows a swarm of robots to cage any arbitrarily shaped object and transport it along a prescribed trajectory. The robots internally determined the object's centroid based on the relative position between the neighbors. Experimental evaluations on simulation studied the time properties and the precision in following a prescribed trajectory. The real robot experiments validated the approach on the hardware.

6. The initial implementation of the software deployment tool performing OTA updates was designed following the standard practices used in software engineering. It generates a patch to pre-existing behavioral code and waits for consensus before switching to the new release. The approach demonstrated good scalability in simulation. Real-world use of the tool on robots demonstrated the tools practical application.

## 10.1 Future Work

Concerning future work, the natural next steps of the research presented in Chapters 4 to 8 would be to develop these approaches further to be more sophisticated. Some future directions of research are:

1. There are many possible extensions to SOUL: implementation of a version control system, mechanisms to provide data redundancy, and mechanisms for robots that allow reallocation of a datagram. These mechanisms are required in real-world missions because robots move, causing the cost of reconstruction to increase, and robot failures might cause data loss. Handling data often comes with the threat of data compromise by adversaries. Improving the security of these systems is another factor that requires consideration for real-world deployments.

2. The self-healing property of many methods in the literature is under-studied. More investigations on the self-healing aspect of the swarm relays could benefit some applications that require redundancy.

3. Based on the results obtained in this study, further investigations could understand the interplay between hierarchical approaches and self-organizing behaviors. A more

detailed study could be made with behaviors that have swarms that switch between hierarchical and egalitarians.

4. The smart particle swarms could be further extended to understand the advantages of better coordination between the guide robots. Other performance factors like the underlying costs involved in operating various combinations of robots at the hierarchy levels could be studied.

5. Further engineering investigations could be performed on the OTA tool designed in this work. A git-like system that maintains the version history could practically benefit real-world deployments.

## REFERENCES

[1] N. Majcherczyk, A. Jayabalan, G. Beltrame, and C. Pinciroli, “Decentralized connectivity-preserving deployment of large-scale robot swarms,” *arXiv preprint arXiv:1806.00150*, 2018.

[2] E. Bonabeau, G. Theraulaz, M. Dorigo, G. Theraulaz, D. d. R. D. F. Marco *et al.*, *Swarm intelligence: from natural to artificial systems.* Oxford university press, 1999, no. 1.

[3] M. Dorigo, M. Birattari, and T. Stutzle, “Ant colony optimization,” *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[4] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[5] L. Chang, C. Liao, W. Lin, L.-L. Chen, and X. Zheng, “A hybrid method based on differential evolution and continuous ant colony optimization and its application on wideband antenna design,” *Progress in electromagnetics research*, vol. 122, pp. 105–118, 2012.

[6] J. E. Bell and P. R. McMullen, “Ant colony optimization techniques for the vehicle routing problem,” *Advanced engineering informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[7] L.-N. Xing, Y.-W. Chen, P. Wang, Q.-S. Zhao, and J. Xiong, “A knowledge-based ant colony optimization for flexible job shop scheduling problems,” *Applied Soft Computing*, vol. 10, no. 3, pp. 888–896, 2010.

[8] M. Dorigo, G. Theraulaz, and V. Trianni, “Swarm robotics: Past, present, and future,” *Proceedings of the IEEE*, vol. 109, no. 7, pp. 1152–1165, 2021.

[9] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, and Others, “A novel concept for the study of heterogeneous robotic swarms,” *IEEE Robotics & Automation Magazine*, vol. 1070, no. 9932/13, 2013.

[10] T. Schmickl, R. Thenius, C. Moslinger, J. Timmis, A. Tyrrell, M. Read, J. Hilder, J. Halloy, A. Campo, C. Stefanini *et al.*, “Cocoro–the self-aware underwater swarm,” in *2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops.* IEEE, 2011, pp. 120–126.

[11] K. N. McGuire, C. de Wagter, K. Tuyls, H. J. Kappen, and G. C. de Croon, “Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment,” *Science Robotics*, vol. 4, no. 35, 2019.

[12] D. St-Onge, M. Kaufmann, J. Panerati, B. Ramtoula, Y. Cao, E. B. Coffey, and G. Beltrame, “Planetary exploration with robot teams: Implementing higher autonomy with swarm intelligence,” *IEEE Robotics & Automation Magazine*, vol. 27, no. 2, pp. 159–168, 2019.

[13] J. G. Mangelson, D. Dominic, R. M. Eustice, and R. Vasudevan, “Pairwise consistent measurement set maximization for robust multi-robot map merging,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2916–2923.

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

[15] N. Reid, “Literature Review : Purely Decentralized P2P File Sharing Systems and Usability,” Rhodes University, Grahamstown, Tech. Rep., 2015.

[16] J. Panerati, M. Minelli, C. Ghedini, L. Meyer, M. Kaufmann, L. Sabattini, and G. Beltrame, “Robust connectivity maintenance for fallible robots,” *Autonomous Robots*, pp. 1–19, 2018.

[17] L. Sabattini, N. Chopra, and C. Secchi, “On decentralized connectivity maintenance for mobile robotic systems,” in *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on.* IEEE, 2011, pp. 988–993.

[18] E. Stump, A. Jadbabaie, and V. Kumar, “Connectivity management in mobile robot teams,” in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1525–1530.

[19] J. Stephan, J. Fink, V. Kumar, and A. Ribeiro, “Concurrent Control of Mobility and Communication in Multirobot Systems,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1248–1254, 2017.

[20] Y. Kantaros, M. Guo, and M. M. Zavlanos, “Temporal Logic Task Planning and Intermittent Connectivity Control of Mobile Robot Networks,” *IEEE Transactions on Automatic Control*, vol. 64, no. 10, pp. 4105–4120, 2019.

[21] F. Klaesson, P. Nilsson, A. D. Ames, and R. M. Murray, “Intermittent connectivity for exploration in communication-constrained multi-agent systems,” in *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS).* IEEE, 2020, pp. 196–205.

[22] P. D. Hung, T. Q. Vinh, and T. D. Ngo, “Hierarchical distributed control for global network integrity preservation in multirobot systems,” *IEEE Transactions on Cybernetics*, pp. 1–14, 2019.

[23] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, “Optimized flocking of autonomous drones in confined environments,” *Science Robotics*, vol. 3, no. 20, pp. 1–14, 2018.

[24] M. Rubenstein, A. Cornejo, and R. Nagpal, “Programmable self-assembly in a thousand-robot swarm,” *Science*, vol. 345, no. 6198, pp. 795–799, 2014.

[25] F. Berlinger, M. Gauci, and R. Nagpal, “Implicit coordination for 3D underwater collective behaviors in a fish-inspired robot swarm,” *Science Robotics*, vol. 6, no. 50, 2021.

[26] A. Flack, M. Nagy, W. Fiedler, I. D. Couzin, and M. Wikelski, “From local collective behavior to global migratory patterns in white storks,” *Science*, vol. 360, no. 6391, pp. 911–914, 2018.

[27] L. Jiang, L. Giuggioli, A. Perna, R. Escobedo, V. Lecheval, C. Sire, Z. Han, and G. Theraulaz, “Identifying influential neighbors in animal flocking,” *PLoS computational biology*, vol. 13, no. 11, p. e1005822, 2017.

[28] M. Nagy, Z. Ákos, D. Biro, and T. Vicsek, “Hierarchical group dynamics in pigeon flocks,” *Nature*, vol. 464, no. 7290, pp. 890–893, 2010.

[29] T. D. Seeley, *Honeybee ecology.* Princeton University Press, 2014.

[30] L. Giuggioli, T. J. McKetterick, and M. Holderied, “Delayed response and biosonar perception explain movement coordination in trawling bats,” *PLoS computational biology*, vol. 11, no. 3, p. e1004089, 2015.

[31] R. O. Peterson, A. K. Jacobs, T. D. Drummer, L. D. Mech, and D. W. Smith, “Leadership behavior in relation to dominance and reproductive status in gray wolves, canis lupus,” *Canadian Journal of Zoology*, vol. 80, no. 8, pp. 1405–1412, 2002.

[32] R. Tron, J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, “Bearing-only formation control with auxiliary distance measurements, leaders, and collision avoidance,” in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 1806–1813.

[33] L. Consolini, F. Morbidi, D. Prattichizzo, and M. Tosques, “Leader–follower formation control of nonholonomic mobile robots with input constraints,” *Automatica*, vol. 44, no. 5, pp. 1343–1349, 2008.

[34] D. Panagou and V. Kumar, “Cooperative visibility maintenance for leader–follower formations in obstacle environments,” *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 831–844, 2014.

[35] M. Ji, A. Muhammad, and M. Egerstedt, “Leader-based multi-agent coordination: Controllability and optimal control,” in *2006 American Control Conference*. IEEE, 2006, pp. 6–pp.

[36] G. L. Mariottini, F. Morbidi, D. Prattichizzo, N. Vander Valk, N. Michael, G. Pappas, and K. Daniilidis, “Vision-based localization for leader–follower formation control,” *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1431–1438, 2009.

[37] D. Gu and Z. Wang, “Leader–follower flocking: Algorithms and experiments,” *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1211–1219, 2009.

[38] K. A. Farley, K. H. Williford, K. M. Stack, R. Bhartia, A. Chen, M. de la Torre, K. Hand, Y. Goreva, C. D. Herd, R. Hueso *et al.*, “Mars 2020 mission overview,” *Space Science Reviews*, vol. 216, no. 8, pp. 1–41, 2020.

[39] I. Slavkov, D. Carrillo-Zapata, N. Carranza, X. Diego, F. Jansson, J. A. Kaandorp, S. Hauert, and J. Sharpe, “Morphogenesis in Robot Swarms,” *Accepted for publication on Science Robotics. Available on Dec 19, 2018*, vol. 3, no. 24, pp. 1–17, 2018.

[40] V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, “Soul: data sharing for robot swarms,” *Autonomous Robots*, vol. 44, no. 3, pp. 377–394, 2020.

[41] V. S. Varadharajan, B. Adams, and G. Beltrame, “The unbroken telephone game: keeping swarms connected,” in *International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 2241–2243.

[42] V. S. Varadharajan, D. St-Onge, B. Adams, and G. Beltrame, “Swarm Relays: Distributed Self-Healing Ground-and-Air Connectivity Chains,” *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5347–5354, 2020.

[43] V. Varadharajan, S. Dyanatkar, and G. Beltrame, "Hierarchical control of smart particle swarms," *IEEE Transactions on Robotics*, 2022, under review.

[44] V. Varadharajan, K. Soma, and G. Beltrame, "Collective transport via sequential caging," in *International Symposium Distributed Autonomous Robotic Systems*, 2021, pp. 349–692.

[45] V. Varadharajan, D. St-Onge, C. Guss, and G. Beltrame, "Over-The-Air Updates for Robotic Swarms," *IEEE Software*, 2018.

[46] V. S. Varadharajan, D. St-Onge, I. Švogor, and G. Beltrame, "A software ecosystem for autonomous uav swarms," in *International Symposium on Aerial Robotics*, 2017.

[47] F. Wu, V. S. Varadharajan, and G. Beltrame, "Collision-aware task assignment for multi-robot systems," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2019, pp. 30–36.

[48] D. St-Onge, V. S. Varadharajan, I. Švogor, and G. Beltrame, "From Design to Deployment: Decentralized Coordination of Heterogeneous Robotic Teams," *Frontiers in Robotics and AI*, vol. 7, p. 51, 2020.

[49] M. Saboia Da Silva, L. Clark, V. Tangavelu, J. Edlund, K. Otsu, G. J. Correa, V. s. Varadharajan, A. Santamaria-Navarro, A. Bouman, S.-K. Kim, T. Vaquero, G. Beltrame, N. Napp, G. Pessin, and A.-a. Agha-mohammadi, "Achord: Communication-aware multi-robot coordination with intermittent connectivity," *IEEE Robotics and Automation Letters*, 2022, under review.

[50] C. R. Kube and H. Zhang, "Collective robotics: From social insects to robots," *Adaptive behavior*, vol. 2, no. 2, pp. 189–218, 1993.

[51] O. Holland and C. Melhuish, "Stigmergy, self-organization, and sorting in collective robotics," *Artificial life*, vol. 5, no. 2, pp. 173–202, 1999.

[52] F. Mondada, L. M. Gambardella, D. Floreano, S. Nolfi, J.-L. Deneuborg, and M. Dorigo, "The cooperation of swarm-bots: Physical interactions in collective robotics," *IEEE Robotics & Automation Magazine*, vol. 12, no. 2, pp. 21–28, 2005.

[53] J.-F. Boudet, J. Lintuvuori, C. Lacouture, T. Barois, A. Deblais, K. Xie, S. Cassagnere, B. Tregon, D. Brückner, J.-C. Baret *et al.*, "From collections of independent, mindless robots to flexible, mobile, and directional superstructures," *Science Robotics*, vol. 6, no. 56, p. eabd0272, 2021.

[54] S. Li, R. Batra, D. Brown, H.-D. Chang, N. Ranganathan, C. Hoberman, D. Rus, and H. Lipson, "Particle robotics based on statistical mechanics of loosely coupled components," *Nature*, vol. 567, no. 7748, pp. 361–365, 2019.

[55] J. Kuckling, A. Ligot, D. Bozhinoski, and M. Birattari, "Behavior trees as a control architecture in the automatic modular design of robot swarms," in *International Conference on Swarm Intelligence*. Springer, 2018, pp. 30–43.

[56] 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.

[57] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *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.

[58] J. Beal and J. Bachrach, "Infrastructure for engineered emergence on sensor/actuator networks," *IEEE Intelligent Systems*, vol. 21, no. 2, pp. 10–19, 2006.

[59] 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.

[60] R. O Grady, A. L. Christensen, and M. Dorigo, "Swarmorph: Morphogenesis with self-assembling robots," in *Morphogenetic Engineering*. Springer, 2012, pp. 27–60.

[61] K. H. Petersen, R. Nagpal, and J. K. Werfel, "Termes: An autonomous robotic system for three-dimensional collective construction," *Robotics: science and systems VII*, 2011.

[62] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[63] K. Dantu, B. Kate, J. Waterman, P. Bailis, and M. Welsh, "Programming micro-aerial vehicle swarms with karma," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, 2011, pp. 121–134.

[64] L. Mottola, M. Moretta, K. Whitehouse, and C. Ghezzi, "Team-level programming of drone sensor networks," in *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, 2014, pp. 177–190.

[65] M. Dorigo, G. Theraulaz, and V. Trianni, "Reflections on the future of swarm robotics," *Science Robotics*, vol. 5, no. 49, pp. 9–12, 2020.

[66] G. Urdaneta, G. Pierre, and M. V. Steen, “A survey of dht security techniques,” *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, pp. 1–49, 2011.

[67] M. Dorigo, E. Bonabeau, and G. Theraulaz, “Ant algorithms and stigmergy,” *Future Generation Computer Systems*, vol. 16, no. 8, pp. 851–871, 2000.

[68] 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*, ser. BICT’15, 2016, pp. 287–294.

[69] J. Benet, “Ipfs-content addressed, versioned, p2p file system,” *arXiv preprint arXiv:1407.3561*, 2014.

[70] N. Majcherczyk and C. Pinciroli, “Swarmmesh: A distributed data structure for cooperative multi-robot applications,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4059–4065.

[71] M. C. De Gennaro and A. Jadbabaie, “Decentralized control of connectivity for multi-agent systems,” in *Decision and Control, 2006 45th IEEE Conference on.* IEEE, 2006, pp. 3628–3633.

[72] P. D. Lorenzo and S. Barbarossa, “Distributed estimation and control of algebraic connectivity over random graphs,” *IEEE Transactions on Signal Processing*, vol. 62, no. 21, pp. 5615–5628, Nov 2014.

[73] L. Siligardi, J. Panerati, M. Kaufmann, M. Minelli, C. Ghedini, G. Beltrame, and L. Sabattini, “Robust area coverage with connectivity maintenance,” in *International Conference on Robotics and Automation*, 2019, pp. 2202–2208.

[74] M. Ji and M. Egerstedt, “Distributed coordination control of multiagent systems while preserving connectedness,” *IEEE Transactions on Robotics*, vol. 23, no. 4, pp. 693–703, 2007.

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

[76] M. Schuresko and J. Cortés, “Distributed tree rearrangements for reachability and robust connectivity,” in *International Workshop on Hybrid Systems: Computation and Control.* Springer, 2009, pp. 470–474.

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

[78] J. Panerati, L. Gianoli, C. Pinciroli, A. Shabah, G. Nicolescu, and G. Beltrame, “From swarms to stars: Task coverage in robot swarms with connectivity constraints,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7674–7681.

[79] A. Gelblum, I. Pinkoviezky, E. Fonio, A. Ghosh, N. Gov, and O. Feinerman, “Ant groups optimally amplify the effect of transiently informed individuals,” *Nature Communications*, vol. 6, 2015.

[80] E. Rimon and A. Blake, “Caging Planar Bodies by One-Parameter Two-Fingered Gripping Systems,” *The International Journal of Robotics Research*, vol. 18, no. 3, pp. 299–318, 1999.

[81] G. A. Pereira, M. F. Campos, and V. Kumar, “Decentralized algorithms for multi-robot manipulation via caging,” *International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 783–795, 2004.

[82] W. Wan, R. Fukui, M. Shimosaka, T. Sato, and Y. Kuniyoshi, “Cooperative manipulation with least number of robots via robust caging,” *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pp. 896–903, 2012.

[83] J. Fink, N. Michael, and V. Kumar, “Composition of vector fields for multi-robot manipulation via caging.” in *Robotics: Science and Systems*, vol. 3, 2007.

[84] B. P. Gerkey and M. J. Mataric, “Pusher-watcher: An approach to fault-tolerant tightly-coupled robot coordination,” in *International Conference on Robotics and Automation (ICRA)*, vol. 1, no. May. IEEE, 2002, pp. 464–469.

[85] Y. Wang and C. W. de Silva, “Cooperative transportation by multiple robots with machine learning,” in *2006 IEEE International Conference on Evolutionary Computation.* IEEE, 2006, pp. 3050–3056.

[86] J. Chen, M. Gauci, W. Li, A. Kolling, and R. Groß, “Occlusion-Based Cooperative Transport with a Swarm of Miniature Mobile Robots,” *IEEE Transactions on Robotics*, vol. 31, no. 2, pp. 307–321, 2015.

[87] V. Pilloni, M. Franceschelli, L. Atzori, and A. Giua, “Deployment of distributed applications in wireless sensor networks,” *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1828–1836, 2016.

[88] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks,” in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, ser. NSDI’04. Berkeley, CA, USA: USENIX Association, 2004, pp. 2–2.

[89] J. W. Hui and D. Culler, “The dynamic behavior of a data dissemination protocol for network programming at scale,” in *Proceedings of the 2nd international conference on Embedded networked sensor systems.* ACM, 2004, pp. 81–94.

[90] T. Stathopoulos, J. Heidemann, and D. Estrin, “A remote code update mechanism for wireless sensor networks,” DTIC Document, Tech. Rep., 2003.

[91] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz, and J. Kelner, “Mires: a publish/subscribe middleware for sensor networks,” *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, 2006.

[92] W. Dong, Y. Liu, C. Wang, X. Liu, C. Chen, and J. Bu, “Link quality aware code dissemination in wireless sensor networks,” in *Network Protocols (ICNP), 2011 19th IEEE International Conference on.* IEEE, 2011, pp. 89–98.

[93] J. Jeong, S. Kim, and A. Broad, “Network reprogramming,” *University of California at Berkeley, Berkeley, CA, USA*, 2003.

[94] S. S. Kulkarni and L. Wang, “Mnp: Multihop network reprogramming service for sensor networks,” in *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on.* IEEE, 2005, pp. 7–16.

[95] M. D. Krasniewski, R. K. Panta, S. Bagchi, C.-L. Yang, and W. J. Chappell, “Energy-efficient on-demand reprogramming of large-scale sensor networks,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 4, no. 1, p. 2, 2008.

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

[97] J. M. Soares, I. Navarro, and A. Martinoli, “The khepera iv mobile robot: performance evaluation, sensory data and software toolbox,” in *Robot 2015: second Iberian robotics conference*. Springer, 2016, pp. 767–781.

[98] M. Lliffe, “Drones in Humanitarian Action,” The Swiss Foundation for Mine Action, Geneva/Brussels, Tech. Rep., 2016.

[99] H. Surmann, N. Berninger, and R. Worst, “3D mapping for multi hybrid robot cooperation,” *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 626–633, 2017.

[100] A. S. Vempati, I. Gilitschenski, J. Nieto, P. Beardsley, and R. Siegwart, “Onboard real-time dense reconstruction of large-scale environments for UAV,” *IEEE International Conference on Intelligent Robots and Systems*, vol. 2017-Septe, pp. 3479–3486, 2017.

[101] D. St-Onge, V. S. Varadharajan, G. Li, I. Svogor, and G. Beltrame, “Ros and buzz: consensus-based behaviors for heterogeneous teams,” in *[submitted to] International Conference on Intelligent Robots and Systems*. IEEE, October 2018.

[102] C. Pinciroli and G. Beltrame, “Swarm-Oriented Programming of Distributed Robot Networks,” *Computer*, vol. 49, no. 12, pp. 32–41, 2016.

[103] D. H. Kelley and N. T. Ouellette, “Emergent dynamics of laboratory insect swarms,” *Scientific reports*, vol. 3, p. 1073, 2013.

[104] D.-H. Lee, “Resource-based task allocation for multi-robot systems,” *Robotics and Autonomous Systems*, vol. 103, pp. 151–161, 2018.

[105] C. Ghedini, C. H. C. Ribeiro, and L. Sabattini, “Improving the fault tolerance of multi-robot networks through a combined control law strategy,” in *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, Sept 2016, pp. 209–215.

[106] E. Chissungo, E. Blake, and H. Le, “Investigation into batman-adv protocol performance in an indoor mesh potato testbed,” in *2011 Third International Conference on Intelligent Networking and Collaborative Systems*. IEEE, 2011, pp. 8–13.

[107] R. Faludi, *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. " O’Reilly Media, Inc.", 2010.

[108] M. Y. Barange and A. K. Sapkal, “Review paper on implementation of multipath reactive routing protocol in manet,” in *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on.* IEEE, 2016, pp. 227–231.

[109] D. T. Davis, T. H. Chung, M. R. Clement, and M. A. Day, “Consensus-Based Data Sharing for Large-Scale Aerial Swarm Coordination in Lossy Communications Environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 3801–3808.

[110] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

[111] A. Howard, M. Matarić, and G. Sukhatme, “Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem,” in *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS).* New York: Springer, 2002, pp. 299–308.

[112] K. Støy, “Using situated communication in distributed autonomous mobile robots,” *Proceedings of the 7th Scandinavian Conference on Artificial Intelligence*, pp. 44–52, 2001.

[113] G. Li, D. St-Onge, C. Pinciroli, A. Gasparri, E. Garone, and G. Beltrame, “Decentralized progressive shape formation with robot swarms,” *Autonomous Robots*, vol. 43, no. 6, pp. 1505–1521, 2019.

[114] S. J. McDonald, M. B. Colton, C. K. Alder, and M. A. Goodrich, “Haptic Shape-Based Management of Robot Teams in Cordon and Patrol,” *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction - HRI ’17*, pp. 380–388, 2017.

[115] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[116] M. Shahriari, I. Svogor, D. St-Onge, and G. Beltrame, “Lightweight collision avoidance for resource-constrained robots,” in *Intelligent Robots and Systems (IROS), 2018 IEEE/RSJ International Conference on.* IEEE, 2018.

[117] S. K. Dhurandher, S. Singhal, S. Aggarwal, P. Pruthi, S. Misra, and I. Woungang, “A Swarm Intelligence-based P2P file sharing protocol using Bee Algorithm,” *2009*

*IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2009*, pp. 690–696, 2009.

[118] Q. H. Vu, M. Lupu, and B. C. Ooi, "Architecture of peer-to-peer systems," in *Peer-to-peer Computing.* Springer, 2010, pp. 11–37.

[119] P. Ganesan, K. Gummadi, and H. Garcia-Molina, "Canon in g major: designing dhts with hierarchical structure," in *Distributed computing systems, 2004. proceedings. 24th international conference on.* IEEE, 2004, pp. 263–272.

[120] P. Garbacki, A. Iosup, D. Epema, and M. van Steen, "2Fast : Collaborative Downloads in P2P Networks," *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pp. 23–30, 2006.

[121] G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-aware communication for decentralised multi-robot coordination," in *Proceedings of the International Conference on Robotics and Automation, Brisbane, Australia*, vol. 21, 2018.

[122] L. Brunet, H.-L. Choi, and J. P. How, "Consensus-based auction approaches for decentralized task assignment," in *AIAA Guidance, Navigation, and Control Conference*, no. August, 2008, pp. 1–24.

[123] W. Ren, R. Beard, and E. Atkins, "A survey of consensus problems in multi-agent coordination," in *Proceedings of the American Control Conference*, 2005, pp. 1859–1864.

[124] Y. Kuriki and T. Namerikawa, "Experimental Validation of Cooperative Formation Control with Collision Avoidance for a Multi-UAV System," in *Proceedings of the 6th International Conference on Automation, Robotics and Applications*, 2015, pp. 531–536.

[125] X. Wei, D. Fengyang, Z. Qingjie, Z. Bing, and S. Hongchang, "A New Fast Consensus Algorithm Applied in Rendezvous of," in *2015 27th Chinese Control and Decision Conference (CCDC)*, 2015, pp. 55–60.

[126] M. B. Dias, R. Zlot, N. Kalra, and A. Stentz, "Market-based multirobot coordination: A survey and analysis," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1257–1270, 2006.

[127] K. Zhang, E. G. Collins, D. Shi, X. Liu, and O. Chuy, "A Stochastic Clustering Auction (SCA) for centralized and distributed task allocation in multi-agent teams," *Distributed Autonomous Robotic Systems 8*, vol. 7, no. 2, pp. 345–354, 2009.

[128] T. Sandholm, T. Sandholm, S. Suri, and S. Suri, "Improved Algorithms for Optimal Winner Determination in Combinatorial Auctions and Generalizations," *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pp. 90–97, 2000.

[129] L. Vig and J. A. Adams, "Market-Based Multi-robotMulti-robot coalition formation," in *Distributed Autonomous Robotic Systems.* Springer, Tokyo, 2006, vol. 7, pp. 227–236.

[130] M. Otte, M. Kuhlman, and D. Sofge, "Multi-robot task allocation with auctions in harsh communication environments," in *Multi-Robot and Multi-Agent Systems (MRS), 2017 International Symposium on.* IEEE, 2017, pp. 32–39.

[131] S. D. Sen and J. A. Adams, "A decision network based framework for multiagent coalition formation," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 55–62.

[132] L. Bayindir, "A review of swarm robotics tasks," *Neurocomputing*, vol. 172, pp. 292–321, 2016.

[133] S. Brown and C. Sreenan, *Software Updating in Wireless Sensor Networks: A Survey and Lacunae*, 2013, vol. 2, no. 4.

[134] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein, "Growth codes: Maximizing sensor network data persistence," in *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 4. ACM, 2006, pp. 255–266.

[135] E. Laukkanen, J. Itkonen, and C. Lassenius, "Problems, causes and solutions when adopting continuous delivery: A systematic literature review," *Information and Software Technology*, vol. 82, pp. 55–79, 2017.

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

[137] M. Guo and M. M. Zavlanos, "Distributed data gathering with buffer constraints and intermittent communication," in *International Conference on Robotics and Automation.* IEEE, 2017, pp. 279–284.

[138] G. Hollinger and S. Singh, "Multi-robot coordination with periodic connectivity," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4457–4462, 2010.

[139] M. M. Zavlanos, A. Ribeiro, and G. J. Pappas, "Network integrity in mobile robotic networks," *IEEE Transactions on Automatic Control*, vol. 58, no. 1, pp. 3–18, 2013.

[140] R. K. Williams, A. Gasparri, and B. Krishnamachari, "Route swarm: Wireless network optimization through mobility," in *International Conference on Intelligent Robots and Systems*, 2014, pp. 3775–3781.

[141] J. Fink, A. Ribeiro, and V. Kumar, "Robust control of mobility and communications in autonomous robot teams," *IEEE Access*, vol. 1, pp. 290–309, 2013.

[142] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[143] M. M. Zavlanos and G. J. Pappas, "Distributed connectivity control of mobile networks," *Proceedings of the IEEE Conference on Decision and Control*, vol. 24, no. 6, pp. 3591–3596, 2007.

[144] Y. Marchukov and L. Montano, "Multi-robot coordination for connectivity recovery after unpredictable environment changes," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 446–451, 2019.

[145] A. A. Abbasi, K. Akkaya, and M. Younis, "A distributed connectivity restoration algorithm in wireless sensor and actor networks," in *32nd IEEE Conference on Local Computer Networks (LCN 2007)*. IEEE, 2007, pp. 496–503.

[146] K. Akkaya, F. Senel, A. Thimmapuram, and S. Uludag, "Distributed recovery from network partitioning in movable sensor/actor networks via controlled mobility," *IEEE Transactions on Computers*, vol. 59, no. 2, pp. 258–271, 2009.

[147] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013.

[148] T. Shima, S. J. Rasmussen, and P. Chandler, "UAV Team Decision and Control Using Efficient Collaborative Estimation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 129, no. 5, pp. 609–619, 04 2007.

[149] H. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[150] Y. Mostofi, A. Gonzalez-Ruiz, A. Gaffarkhah, and D. Li, “Characterization and modeling of wireless channels for networked robotic and control systems-a comprehensive overview,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2009, pp. 4849–4854.

[151] M. Caserta and S. Voß, “The robust multiple-choice multidimensional knapsack problem,” *Omega*, vol. 86, pp. 16–27, 2019.

[152] L. Zhang, Y. Kim, and D. Manocha, “A simple path non-existence algorithm using c-obstacle query for low dof robots,” *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, pp. 1–16, 2006.

[153] Y. Li, Z. Littlefield, and K. E. Bekris, “Sparse methods for efficient asymptotically optimal kinodynamic planning,” *Springer Tracts in Advanced Robotics*, vol. 107, pp. 263–282, 2015.

[154] I. A. Sucan, M. Moll, and L. E. Kavraki, “The open motion planning library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[155] J. Van den Berg, M. Lin, and D. Manocha, “Reciprocal velocity obstacles for real-time multi-agent navigation,” in *2008 IEEE International Conference on Robotics and Automation.* IEEE, 2008, pp. 1928–1935.

[156] N. Sturtevant, “Benchmarks for grid-based pathfinding,” *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012.

[157] R. Cotsakis, D. St-Onge, and G. Beltrame, “Decentralized collaborative transport of fabrics using micro-uavs,” in *2019 International Conference on Robotics and Automation (ICRA).* IEEE, 2019, pp. 7734–7740.

[158] G. Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield, B. J. Nelson, B. Scassellati, M. Taddeo, R. Taylor, M. Veloso, Z. L. Wang, and R. Wood, “The grand challenges of science robotics,” *Science Robotics*, vol. 3, no. 14, 2018.

[159] M. Dorigo, D. Floreano, L. M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy *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.

[160] A. Prorok, M. A. Hsieh, and V. Kumar, “The impact of diversity on optimal control policies for heterogeneous robot swarms,” *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 346–358, 2017.

[161] M. Scheutz, “Real-time hierarchical swarms for rapid adaptive multi-level pattern detection and tracking,” in *2007 IEEE Swarm Intelligence Symposium.* IEEE, 2007, pp. 234–241.

[162] H. Chen, Y. Zhu, K. Hu, and X. He, “Hierarchical swarm model: a new approach to optimization,” *Discrete Dynamics in Nature and Society*, vol. 2010, 2010.

[163] J. Hu, A. E. Turgut, T. Krajník, B. Lennox, and F. Arvin, “Occlusion-based coordination protocol design for autonomous robotic shepherding tasks,” *IEEE Transactions on Cognitive and Developmental Systems*, 2020.

[164] S. Razali, N. F. Shamsudin, M. Osman, Q. Meng, and S.-H. Yang, “Flock identification using connected components labeling for multi-robot shepherding,” in *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR).* IEEE, 2013, pp. 298–303.

[165] A. Özdemir, M. Gauci, and R. Groß, “Shepherding with robots that do not compute,” in *ECAL 2017, the Fourteenth European Conference on Artificial Life.* MIT Press, 2017, pp. 332–339.

[166] C. Huang, G. Yang, Q. Ha, J. Meng, and S. Wang, “Multifunctional “smart” particles engineered from live immunocytes: toward capture and release of cancer cells,” *Advanced Materials*, vol. 27, no. 2, pp. 310–313, 2015.

[167] R. P. Ramos, S. M. Oliveira, S. M. Vieira, and A. L. Christensen, “Evolving flocking in embodied agents based on local and global application of reynolds’ rules,” *Plos one*, vol. 14, no. 10, p. e0224376, 2019.

[168] B. T. Fine and D. A. Shell, “Unifying microscopic flocking motion models for virtual, robotic, and biological flock members,” *Autonomous Robots*, vol. 35, no. 2, pp. 195–219, 2013.

[169] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[170] F. Gökçe and E. Şahin, "To flock or not to flock: the pros and cons of flocking in long-range" migration" of mobile robot swarms," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2009, pp. 65–72.

[171] V. Gazi and K. M. Passino, "Stability of a one-dimensional discrete-time asynchronous swarm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 4, pp. 834–841, 2005.

[172] S. Hauert, S. Leven, M. Varga, F. Ruini, A. Cangelosi, J.-C. Zufferey, and D. Floreano, "Reynolds flocking in reality with fixed-wing robots: communication range vs. maximum turning rate," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, no. CONF. Ieee Service Center, 445 Hoes Lane, Po Box 1331, Piscataway, Nj 08855-1331 Usa, 2011.

[173] D. V. Dimarogonas, M. Egerstedt, and K. J. Kyriakopoulos, "A leader-based containment control strategy for multiple unicycles," in *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006, pp. 5968–5973.

[174] R. Tron, J. Thomas, G. Loianno, K. Daniilidis, and V. Kumar, "Bearing-only formation control with auxiliary distance measurements, leaders, and collision avoidance," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 1806–1813.

[175] S. Björkenstam, M. Ji, M. B. Egerstedt, and C. F. Martin, "Leader-based multi-agent coordination through hybrid optimal control." Georgia Institute of Technology, 2006.

[176] J. Wiech, V. A. Eremeyev, and I. Giorgio, "Virtual spring damper method for nonholonomic robotic swarm self-organization and leader following," 2018.

[177] M. Ji, G. Ferrari-Trecate, M. Egerstedt, and A. Buffa, "Containment control in mobile networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 8, pp. 1972–1975, 2008.

[178] H. Hoshi, I. Iimura, S. Nakayama, Y. Moriyama, and K. Ishibashi, "Robustness of herding algorithm with a single shepherd regarding agents' moving speeds," *Journal of Signal Processing*, vol. 22, no. 6, pp. 327–335, 2018.

[179] L. Chaimowicz and V. Kumar, "Aerial shepherds: Coordination among uavs and swarms of robots," in *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 243–252.

[180] D. Strömbom, R. P. Mann, A. M. Wilson, S. Hailes, A. J. Morton, D. J. Sumpter, and A. J. King, "Solving the shepherding problem: heuristics for herding autonomous, interacting agents," *Journal of the royal society interface*, vol. 11, no. 100, p. 20140719, 2014.

[181] N. K. Long, M. Garratt, K. Sammut, D. Sgarioto, and H. A. Abbass, "Shepherding autonomous goal-focused swarms in unknown environments using hilbert space-filling paths," *Shepherding UxVs for Human-Swarm Teaming: An Artificial Intelligence Approach to Unmanned X Vehicles*, pp. 31–50, 2021.

[182] M. Kubo, M. Tashiro, H. Sato, and A. Yamaguchi, "Herd guidance by multiple sheepdog agents with repulsive force," *Artificial Life and Robotics*, pp. 1–12, 2022.

[183] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks.* Princeton University Press, 2010, vol. 33.

[184] H. T. Zhang, C. Zhai, and Z. Chen, "A general alignment repulsion algorithm for flocking of multi-agent systems," *IEEE Transactions on Automatic Control*, vol. 56, no. 2, pp. 430–435, 2011.

[185] S. J. Stuart, A. B. Tutein, and J. A. Harrison, "A reactive potential for hydrocarbons with intermolecular interactions," *The Journal of chemical physics*, vol. 112, no. 14, pp. 6472–6486, 2000.

[186] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[187] A. Rosenfeld, A. Noa, O. Maksimov, and S. Kraus, "Human-multi-robot team collaboration for efficent warehouse operation," *Autonomous Robots and Multirobot Systems (ARMS)*, 2016.

[188] K. H. Petersen, N. Napp, R. Stuart-Smith, D. Rus, and M. Kovac, "A review of collective robotic construction," *Science Robotics*, vol. 4, no. 28, 2019.

[189] W. Wan, B. Shi, Z. Wang, and R. Fukui, "Multirobot Object Transport via Robust Caging," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 50, no. 1, pp. 270–280, 2020.

[190] A. Franchi, A. Petitti, and A. Rizzo, "Distributed Estimation of State and Parameters in Multiagent Cooperative Load Manipulation," *IEEE Transactions on Control of Network Systems*, vol. 6, no. 2, pp. 690–701, 2019.

[191] C. Gabellieri, M. Tognon, D. Sanalitro, L. Pallottino, and A. Franchi, "A study on force-based collaboration in swarms," *Swarm Intelligence*, vol. 14, no. 1, pp. 57–82, 2020.

[192] Z. Wang, S. Singh, M. Pavone, and M. Schwager, "Cooperative Object Transport in 3D with Multiple Quadrotors Using No Peer Communication," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1064–1071, 2018.

[193] R. S. M. B, D. G. Macharet, M. F. M. Campos, R. S. Melo, D. G. Macharet, and M. F. M. Campos, "Collaborative object transportation using heterogeneous robots," in *Robotics*. Springer, 2016, pp. 172–191.

[194] M. Rubenstein, A. Cabrera, J. Werfel, G. Habibi, J. McLurkin, and R. Nagpal, "Collective transport of complex objects by simple robots," *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, pp. 47–54, 2013.

[195] Z. Wang, Y. Hirata, and K. Kosuge, "Control a rigid caging formation for cooperative object transportation by multiple mobile robots," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 2. IEEE, 2004, pp. 1580–1585.

[196] Y. Dai, Y. Kim, S. Wee, D. Lee, and S. Lee, "Symmetric caging formation for convex polygonal object transportation by multiple mobile robots based on fuzzy sliding mode control," *ISA transactions*, vol. 60, pp. 321–332, 2016.

[197] M. H. M. Alkilabi, A. Narayan, and E. Tuci, "Cooperative object transport with a swarm of e-puck robots: robustness and scalability of evolved collective strategies," *Swarm intelligence*, vol. 11, no. 3-4, pp. 185–209, 2017.

[198] T. H. Chung, M. R. Clement, M. A. Day, K. D. Jones, D. Davis, and M. Jones, "Live-fly, large-scale field experimentation for large numbers of fixed-wing UAVs," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2016-June, pp. 1255–1262, 2016.

[199] T. H. Chung, K. D. Jones, M. A. Day, M. Jones, and M. Clement, "50 vs. 50 by 2015: Swarm vs. swarm uav live-fly competition at the naval postgraduate school," 2013.

[200] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[201] C. Paar and J. Pelzl, *Understanding cryptography: a textbook for students and practitioners.* Springer Science & Business Media, 2009.

[202] F. Amigoni, J. Banfi, and N. Basilico, "Multirobot exploration of communication-restricted environments: A survey," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 48–57, 2017.

[203] A. Ranjan, H. B. Sahu, and P. Misra, "Modeling and measurements for wireless communication networks in underground mine environments," *Measurement*, vol. 149, p. 106980, 2020.

[204] M. F. Ginting, K. Otsu, J. A. Edlund, J. Gao, and A.-A. Agha-Mohammadi, "Chord: Distributed data-sharing via hybrid ros 1 and 2 for multi-robot exploration of large-scale complex environments," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5064–5071, 2021.