Hybrid control in multi-robot systems and distributed computing

by

Aryo Jamshidpey

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Supervisors: Marco Dorigo and Paola Flocchini
Co-Supervisor: Nicola Santoro

The Ph.D program is a "cotutelle" program between the University of Ottawa and the Université Libre de Bruxelles. The Ph.D program is also a joint program with Carleton University, administered by Ottawa-Carleton Institute for Electrical and Computer Engineering

Abstract

Multi-agent systems (MAS) have been of interest to many researchers during the last decades. This thesis focuses on multi-robot systems (MRS) and programmable matter as two types of MAS. Regarding MRS, the focus is on the 'mergeable nervous system' (MNS) concept which allows the robots to connect to one another and establish a communication network through self-organization and then use the network to temporarily report sensing events and cede authority to a single robot in the system. Here, in a collective perception scenario, we experimentally evaluate the performance of an MNS-enabled approach and compare it with that of several decentralized benchmark approaches. We show that an MNS-enabled approach is high-performing, fault-tolerant, and scalable, so it is an appropriate approach for MRS. As a goal of the thesis, using an MNS-enabled approach, we present for the first time a comprehensive comparison of control architectures in multi-robot systems, which includes a comparison of accuracy, efficiency, speed, energy consumption, scalability, and fault tolerance. Our comparisons provide designers of multi-robot systems with a better understanding for selecting the best-performing control depending on the system's objectives. Additionally, as a separate goal, we design a high-level leader-based programmable matter, which can perform some basic primitive operations in a grid environment, and construct it using lower-level organisms. We design and implement deterministic algorithms for "curl" operation of this high-level matter, an instance of shape formation problem. We prove the correctness of the presented algorithms, analytically determine their complexity, and experimentally evaluate their performance.

Acknowledgements

I have been very fortunate to have great PhD advisors from whom I learnt many things. I would like to express my deep and sincere gratitude to my supervisors for their continuous support and invaluable guidance. It was impossible to develop the ideas that resulted in this dissertation without their guidance, support, and patience. It was a great pleasure for me to be their student.

Special thanks go to Prof. Paola Flocchini and Prof. Nicola Santoro for accepting me as their student at the University of Ottawa and teaching me the first notions and interesting concepts of distributed computing, besides their excellent guidance, support and encouragement over the years of my PhD studies. Also, I would like to express my sincere appreciation to them for providing me with the chance of pursuing my studies under the Cotutelle program with the Université Libre de Bruxelles, under the supervision of Prof. Marco Dorigo.

Huge thanks go to Prof. Marco Dorigo, for giving me the chance to do research under his supervision at the artificial intelligence research laboratory of the Université Libre de Bruxelles (IRIDIA). I am very grateful to him for his great guidance, support, and for considering me as a member of one of the most exciting research projects in the field of multi-robot systems. I am proud of being his student.

I also like to warmly thank Dr. Mary Katherine Heinrich for her amazing help, advices, the time she spent, and the meetings we had, which provided the chance of discussion about many parts of my research.

I would like to kindly thank the members of my PhD defense committee, Prof. Enrico Natalizio, Prof. Amiya Nayak, and Prof. Wei Shi for their time, incredible feedback, and valuable suggestions.

Last but not least, to my beloved parents, thanks for your unconditional love and support. During my life, you were always there for me, day and night, and never gave up on me, even for one moment. Obviously, without your support, I could not have undertaken this journey.

I am proud of having such kind, supportive, caring, and lovely parents. Furthermore, I am very thankful to my dear brothers, Arash and Armin, for all their kindness, help, and support. Special thanks go to them, for their wonderful suggestions and advice during my life. If it was not for their help and support, I would have missed many great opportunities during my life. Finally, I would like to acknowledge the rest of my family, my sisters-in-law, uncles, aunts, and my close friends, for their love and support.

Dedication

To my parents

Contents

Chapter 1

Introduction

Multi-agent systems and multi-robot systems are two important research domains that have been extensively studied during the past few decades. Multi-agent systems, which is a subfield of artificial intelligence, aims at providing principles of construction of systems consisting of multiple agents and mechanisms of coordination of the behaviors of the agents [117]. An agent is an entity which has been situated in an environment and has certain goals, knowledge, and actions[117]. Multi-agent systems are simply referred to systems consisting of multiple interacting agents capable of perceiving their local environment. Examples of such systems could be humans, computer programs, robots, and machines. If agents are robots, we refer to these systems as multi-robot systems. In such systems, multiple agents may need to communicate with each other in order to achieve a global task. However, this is not always true; sometimes, agents can achieve a global task without communicating with their peers. For example, consider a group of agents in which each constituent member directly receives commands from a central operator and sends back its feedback in response. To put it another way, there are two main types of control mechanisms for such systems: centralized and decentralized controls. In centralized control, the behavior of each constituent member of the system (i.e., robots or agents) and consequently the behaviour of the entire system is directly managed and controlled by a single central computational unit. A central computational unit is a unit that can perform complex computations for all constituent members of a system. Centralized approaches are often high performing and fast, but can suffer from single points of failure and poor scalability, due to limitations such as communication bottlenecks. On the other hand, in decentralized control, there is no such a controller unit, and agents/robots should autonomously accomplish their tasks through local interactions with each other or with the environment. Decentralized approaches are scalable and fault tolerant due to redundancy and parallelization; however, they are usually slow, and their performance is not as good as the centralized approaches. In this thesis, the focus is on two types of multi-agent systems: multi-robot systems and programmable matter. Both types

of systems are distributed, but some terminologies used to describe them are different. In multi-robot systems our focus is on a recently developed robotic concept which combines aspects of centralized and decentralized controls and benefits from both—called 'mergeable nervous system' (MNS) [84, 140]. On the other hand, in programmable matter, our focus is on introducing a novel high-level matter consisting of low-level particles which can be used in different applications and make programming of programmable matter easier, without sacrificing efficiency in runtime execution. Both an MNS-enabled system and the proposed high-level programmable matter are higher-level leader-based systems that function according to the principles of self-organization. Furthermore, both systems are hybrid, characterized by central coordination using solely local interactions between the robots. In the following, the motivations, goals, and contributions related to multi-robot systems are stated in section 1.1 while those associated with programmable matter are mentioned in section 1.2.

1.1 Multi-robot systems

In multi-robot systems, multiple robots with a wide range of capabilities collaborate with each other to accomplish their given tasks in the environment. Such systems can utilize either central coordination mechanisms or decentralized ones, depending on the scenario and application. In the following we briefly recall the characteristics of centralized and decentralized controls in multi-robot systems. In "centralized control", there exists a single central computational unit that can directly communicate with all the robots and manages and controls their behaviors. To this purpose, it is usually assumed that such a unit has an unlimited communication range and unlimited field of view to observe positions of the robots and communicate with them, no matter how far they are. In other words, this computational unit is a central coordinating entity (i.e., an entity responsible for coordination). On the other hand, "decentralized control" is characterized by local interactions between robots. More precisely, in decentralized control, there is no central computational unit, and robots do not have access to global information. Robots are autonomous (i.e., they act autonomously) and able to locally interact with each other or with their environment in order to perform their tasks.

Centralized approaches are often better than decentralized approaches in terms of accuracy, speed, and efficiency. However, they are poor candidates for simulating collective behaviours due to the presence of single points of failure and poor scalability and fault tolerance. To get the benefits of both centralized and decentralized controls, a well-designed combination of both can be used, but it is not easy to design such an approach.

The 'Mergeable nervous system' (MNS) [84] is a concept in which a robot team can successfully implement a degree of central coordination without sacrificing the power of de-

centralized control, based on the principles of self-organization. A robot team utilizing the MNS concept is called an MNS robot. MNS robot is the first self-assembling multi-robot system that exhibits sensorimotor coordination similar to that observed in monolithic robots. In the MNS concept, a group of ground robots, through a self-organizing process, assemble and physically connect to each other and establish a communication network. Then, they temporarily yield control of their sensors and actuators to a single *brain* robot. More specifically, the brain is responsible for centralized decision-making and controlling the behavior of the whole system. MNS robots can merge and form larger bodies with a single brain, split into separate bodies with independent brains, and self-heal by removing or replacing faulty robots. The MNS concept takes a step toward forming robot teams that can change their size, shape, and function in a self-organized way and getting the best of both centralized and decentralized controls.

Zhu et. al [140], extended the MNS concept to wireless communication instead of using physical connections as it was done in [84]. In their work, a group of ground robots and unmanned aerial vehicles (UAVs) establish a dynamic ad hoc network in a self-organized way and use this network to establish a formation. It is worth noting that the group's behaviors are managed by a UAV brain; the role of the brain is dynamically assigned through self-organization. The main focus of this thesis in the multi-robot systems domain is on this version of the MNS.

Since this approach can benefit from centralized and decentralized controls without suffering from single point of failure and is feasible in real world scenarios (i.e., it does not use any central computational unit, unlimited communication range, or unlimited field of view), it is very important to know whether it is a good alternative for centralized and decentralized approaches in multi-robot systems. In this thesis, we show that MNS-enabled approaches are high-performing, fault tolerant, and scalable, so they are appropriate approaches in multi-robot systems.

In this thesis, we first experimentally study the performance of an MNS-enabled approach in a collective task that is usually studied in fully decentralized systems. More specifically, we try to answer the following question: How effective an MNS-enabled approach is under different conditions in performing a collective task compared to the most common decentralized approaches?

To the best of our knowledge, there is no published attempt in distributed systems to comprehensively assess and compare centralized and decentralized control architectures with one formalized set of systematic metrics. As a consequence, it is not clear how good a control architecture could be in performing a given collective task under different conditions. Because of this gap, designers of such systems cannot guarantee what control is the top performer for

the objectives of the systems. The first goal of this thesis is to take a step towards filling this gap from experimental and analytical points of view for multi-robot systems. More specifically, we compare control architectures through comparing the performance of some approaches with different degrees of centralization, including the Mergeable nervous systems.

Although the results of this thesis are task-specific, they are an initial step towards better characterizing the assumed trade-offs between centralization and decentralization in multi-robot systems. Our comprehensive comparisons will provide designers of multi-robot systems with a better understanding for selecting the best performing control depending on the system's objectives.

In this thesis, we study collective perception and multi-robot coverage as two fundamental collective tasks in multi-robot systems. Collective perception is a fundamental collective task that is popular among the community of researchers that are interested in large-scale multi-robot systems. This task has a close relationship with some other basic collective tasks such as consensus achievement and object classification. A group of robots can reach a consensus decision if they can collectively perceive their environment. We consider a collective perception scenario in order to study the performance of an MNS-enabled approach, which combines aspects of centralized and decentralized controls, and to compare it with several state-of-the-art decentralized approaches. Multi-robot coverage is another collective task which has a direct impact on some other collective problems in which visiting all portions of workspace improves the efficiency of proposed approaches, such as multi-robot task allocation, consensus achievement, search and rescue, foraging, collective perception, and so on. More precisely, we comprehensively investigate the performance of some multi-robot coverage approaches which have different degree of centralization and compare them with each other with one formalized set of systematic metrics.

1.1.1 Collective Perception Problem

In multi-robot systems, collective perception can be viewed as a type of collective decision-making. The robot team must both collect information and converge on a shared understanding of that information. Collective perception is usually set up as a best-of-n decision-making problem [128]. In one approach, robots operate in a grid environment and sense the color of the cells, and collectively decide on the color that is prevailing [9, 10, 49, 114, 126]. In another approach, robots explore an unknown environment and collectively decide on one of the options distributed in the environment based on a specific criterion such as intensity of color or intensity of light [130]. In still another approach, robots collectively compare different zones, whose location is known to the robots, based on their quality that can be measured from every position in the zone [100, 101] and decide on a zone that has the highest quality.

Sometimes collective perception is studied in scenarios that are more complicated than a best-of-n decision-making. In one approach of this type, robots collectively classify objects or areas they discover during exploration according to a criterion (e.g., color) [71, 82, 86].

Another collective task that can be related to collective perception is Aggregation. Aggregation-based approaches, which are inspired by biological systems such as bees [129] and cockroach pools [20, 54, 55], are control strategies that aim at gathering a group of robots in a common region and forming a cluster in the environment [128]. In order to achieve this goal, the collective perception strategies can be used. We will discuss these approaches in more details in Chapter 2.

In the literature, in nearly all existing studies, the accuracy of a robot team's environmental perception with respect to a ground truth is not directly evaluated. In one recent exception, although accurate perception is not the task of a robot swarm, the swarm's perception of absolute density is still measured and compared to ground truth in order to fully report the swarm's behavior [73]. Although the targeted action in this paper is successful, in the reported results, we see that the swarm's perception of density diverges widely from the ground truth. In summary, the perceptual accuracy of robot teams requires more research. In the collective perception task studied in this thesis, the focus is on this gap. In our scenario, a group of robots must collectively explore an unknown environment and estimate the density of the blocks in the environment, based on their sensor readings. Except for the mentioned paper, in the literature, the results of the collective perception of swarms have not been directly compared to a ground value. In this thesis, we propose a collective perception scenario in which a group of robots must collectively explore an unknown environment and frequently estimate the density of the blocks in the environment, based on their sensor readings.

1.1.2 Multi-Robot Coverage Problem

Multi-robot coverage control targets the systematic observation of a physical area or terrain. Coverage control has been widely studied in sensor networks (e.g., [39, 133]), and has also been studied for search and exploration tasks with single robots and multi-robot systems. In the task of single robot coverage, the robot should gather information about the environment as efficiently as possible [53, 68]. The overall time for a coverage task can be decreased by using multiple robots, but multi-robot approaches require solutions to efficient coordination.

In centralized approaches, multi-robot coverage control is often approached as a path planning problem [5, 36, 138] making use of optimization or learning techniques. In other words, in such approaches, the motion of the robots is often centrally planned and coordinated, sometimes with prior knowledge of the size and shape of the environment. These approaches sometimes incorporate aspects of decentralized control. For instance, in [108],

decentralized path planning relies on reinforcement and imitation learning through a centralized planner. As another example, in [80], robots use decentralized control to initially spread out in the environment, and then use a centralized approach for online learning of a density function. Centralized approaches to coverage path planning have high performance, but are limited in terms of scalability and fault tolerance, due to a lack of redundancy that results in single points of failure and communication bottlenecks.

In decentralized approaches, solutions to spatial coordination include maintaining communication (e.g., via line-of-sight [102]), leaving markings during exploration (e.g., artificial pheromones [75]), or using beacons as guidance [2]. As two examples of works on communication maintenance we can refer the interested readers to [77, 115]. In [77], the authors investigate how a group of robots can maximize coverage and minimize communication overhead while keeping the connectivity. In [115], communication maintenance has been studied using a Voronoi tessellation approach to add fault tolerance. Efficiency is also a key challenge for decentralized approaches, as they are prone to redundancy. In [63], a large number of robots perform coverage by simple collision avoidance, but full coverage is not guaranteed and efficiency is low, as robots frequently revisit areas. In a similar approach that reduces repeated coverage [57], robots leave markings during exploration; in another, a pheromone-based approach is used to achieve coverage efficiency [112]. Similar to pheromone-based approaches, activated beacons are used in [2] to guide coverage in a swarm of UAVs. In general, decentralized approaches to coverage are typically scalable and fault-tolerant, but are slow and inefficient compared to centralized approaches [63].

As already mentioned, it is also possible to combine aspects of centralized and decentralized controls. As a successful example of this type of approaches, in [116] some UAVs temporarily park themselves in an environment occupied with obstacles. and play the role of beacons while the rest of the swarm explores the environment. The beacons form a connected network in order to guide the other UAVs where to visit and increase the coverage performance. This approach achieves a near to perfectly complete coverage, and also demonstrates scalability. However, repeated coverage and fault tolerance are not directly measured.

Ideally, when designing a multi-robot system, one would be able to guarantee that a selected control is the top performer for its objectives. Tools have been proposed in swarm robotics to assess general performance [59] and specific aspects of performance (e.g., [125]), and in self-organizing software and embedded systems to assess general performance [46, 47, 69] and the degree of self-organization [122]. Although these contributions progress towards general metrics for decentralization, they are not applicable to centralized multi-robot systems. Therefore, the choice to use centralized or decentralized control is often made intuitively and most of the time is discipline dependent (e.g., decentralized control in the swarm robotics community).

1.1.3 Contributions of the thesis: Multi-robot systems

In this section, we present the contributions related to multi-robot systems presented in this thesis.

MNS in collective perception. Here we study, in a collective perception scenario, the performance of an MNS-enabled approach (which combines some aspects of centralized and decentralized control architectures) compared to the most common decentralized approaches. As already mentioned, the perceptual accuracy of robot teams needs more study. In our collective perception scenario, we assess the accuracy of the robot team's perception with respect to a ground truth. More specifically, we implement an MNS-based approach enabling the robots to collectively perceive the density of randomly distributed blocks in an environment. To the best of our knowledge, our MNS-enabled approach is the first hierarchical hybrid approach to the collective perception task in the multi-robot systems domain. We test the perception performance of the MNS-enabled approach in both static and dynamic simulated environments. To evaluate whether increased centralization improves performance, we compare the accuracy of the MNS-enabled approach to that of the three state-of-the-art decentralized approaches to the collective perception problem. We also test whether the MNS maintains scalability and fault tolerance which are typical features of decentralization. In short, the results show that the MNS-enabled approach outperforms the other three decentralized approaches in terms of performance and is comparable with the decentralized approaches in terms of scalability and fault tolerance.

The details on this contribution can be found in Chapter 2. For more information on this, please see section 1.3.

Centralization vs. decentralization in multi-robot coverage. In this work, we present for the first time a comprehensive comparison of control structures in multi-robot systems, which includes comparison of speed, accuracy, efficiency, scalability, fault tolerance, and energy consumption. To put it another way, we take a step towards comprehensive comparison between centralized and decentralized control in multi-robot systems, for the task of uniform coverage. We experimentally and analytically compare the performance of four approaches—from fully decentralized to fully centralized—in a comprehensive way. Here, in our comparison, we use the MNS as an approach which can combine aspects of centralized and decentralized control. We run all the experiments in the multi-robot physics-based simulation environment ARGoS [99]. In comparing the ground robots performing the coverage task, we assess the speed and efficiency advantages of centralization, in terms of coverage completeness and coverage uniformity, and evaluate the scalability and fault tolerance ad-

vantages of decentralization which are typical features of self-organized systems. In summary, this comprehensive comparison provides designers with a better understating of different aspects of control mechanisms helping them to select an appropriate performer with respect to the objectives of the problem.

The details of this contribution can be found in Chapter 3. For more information on this, please see section 1.3.

1.2 Programmable Matter: Caterpillars

Programmable matter, a concept introduced by Toffoli and Margolus [121], has widely drawn attention of researchers and engineers in the past few decades. Programmable matter refers to a matter that has the ability of changing its physical properties such as color, shape, density, conductivity, in a programmable fashion, based on user inputs or autonomous sensing. In this field of research, matter consists of a very large number of small computational entities that have limited computational, communication and movement capabilities. These simple computational units are programmed to interact locally with their neighbors, move in the environment and establish and release bonds in a self-organized way, and collectively perform a task without the need for any central or external intervention. In this thesis, since such matters consist of computational entities that cooperatively accomplish their tasks, we consider them as multi-agent systems. In other words, a programmable matter is considered as a multi-agent system in which agents are simple computational entities that perform their given tasks through self-organization. There are many applications for such matters like smart materials, drug delivery, large scale robotics, autonomous monitoring and repair, and minimal invasive surgery.

Many theoretical models have been designed to study programmable matter, such as molecular programming [136], DNA self-assembly (e.g., [98, 104]), nature inspired insects and micro organisms (e.g., [41]), and metamorphic robots (e.g., [17, 131]). Among all existing models, the Geometric Amoebot [29] has gained wide attention in the recent years. In this amoeba-inspired model, a swarm of decentralized autonomous self-organizing entities (i.e., particles), called amoebots, operate on an infinite regular triangular grid graph. The particles have local information, have weak computational power, and can bond to their neighboring particles to form connected structures. The particles can move on the grid through expanding and contracting their body, simulating the behavior of amoeba. Moreover, in order to maintain the connectivity while moving in the environment, the particles use a property called handover. Based on this property, there are two possible scenarios: 1) a contracted particle can push its expanded neighboring particle through expanding its own body; 2) an expanded particle can pull its contracted neighboring particle through contracting

its own body. The geometric amoebot model has been widely used in designing algorithms for problems like leader election [12, 22, 32, 38, 45, 50, 56], coating [23, 30, 34], shape formation (called pattern formation in multi-robot systems) [21, 24, 31, 33, 37, 38, 51, 52, 91, 119], convex hull formation [25], and gathering (called aggregation in multi-robot systems) [15].

In the literature of programmable matter, the proposed algorithms under amoebot model are decentralized. In general, decentralized algorithms making use of leaders are very common in programmable matter, and centralized algorithms are completely ignored in this field. In a decentralized system, if a leader fails, the system does not stop working as a new leader can be elected through self-organization. In this thesis, the focus of the chapter related to programmable matter will be on a leader-based particle system.

Designing individual-level behaviors for a given collective mission is not a straightforward nor easy task. More specifically, in order to design a macro-level behavior, a phase of trial and error refinement of individual agent control rules is needed. As a solution to the mentioned existing challenge, higher-level organisms which are made of lower-level ones and capable of performing some powerful primitive operations can be designed to ease the design phase of collective behaviors. The second goal of this thesis is to design and implement such a higher-level system. Here, we design a higher-level leader-based programmable organism, called *Caterpillar*, which has a linear structure and is made of lower-level organisms amoebots. This self-organized system should not cross itself nor be broken and is capable of performing some primitive operations such as "flock" (i.e., follow the leader), "curl" (i.e., form a filled hexagon), "uncurl" (i.e., undo a filled hexagon), and "merge". Compared to lower level amoebots, this higher level system with such a set of basic primitive operations can be much easier programmed to perform complex collective tasks (e.g., exploration, coating etc.) in the environment.

1.2.1 Contributions of the thesis: Programmable matter

In this thesis, regarding programmable matter, as already stated, we introduce the caterpillar as a novel higher-level programmable matter, which is made by amoebots. One of these constituent particles is the leader of this system and can initiate and control the behavior of the matter. This higher-level organism has a set of primitive operations which will enable it to be easily programmed for performing complex task once all these operations have been implemented. The novelty of this system is in the ability of the caterpillar to keep its linear structure while performing different tasks in an environment (e.g., coating an object, forming a shape, flocking, transporting drugs in the body, and so on). In this thesis, we show how a primitive operation of this new type of matter can be implemented using the lower-levels organisms amoebots. More precisely, we design and analyze different amoebots

algorithms that implement curling as an example of primitive operations. When a caterpillar curls it forms a filled hexagon. So, the curl operation can be seen as an instance of shape formation, where particles, initially in a straight line, move to form a filled hexagon always maintaining the linear connection among them. The hexagon formation problem had never been investigated before under this constraint, and none of the algorithms in the literature can solve it in our setting. We design five deterministic algorithms for this task using Geometric Amoebot model, called `Caterpillar-1`, `Caterpillar-2`, `Caterpillar-3`, `Caterpillar-4`, and `Caterpillar-5`. Furthermore, we implement all these algorithms in simulation and evaluate their number of movements (i.e., complexity) both theoretically and empirically.

The details of this contribution can be found in Chapter 4. For more information on this, please see section 1.3.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows:

- In Chapter 2, an MNS-enabled approach is proposed for the task of collective perception. The performance of the approach in both simulated dynamic and static environments is assessed and compared with that of three state-of-the-art fully decentralized approaches. The scalability and fault tolerance of the approaches is also evaluated. An article on this chapter has been submitted to the Intelligent Computing journal.

- In Chapter 3, in order to evaluate the impact of (de)centralization on coverage control, we experimentally test the coverage performance of four approaches with different level of centralization including the MNS as an efficient hybrid approach. We also experimentally examine the fault tolerance and scalability of the MNS approach, and analytically compare these results to the fault tolerance and scalability of the other approaches. Finally, we assess the impact of adding UAVs by analyzing the performance results of the approaches according to the energy consumption metric. An article on this chapter has been submitted to the Swarm Intelligence journal.

- In Chapter 4, we introduce the *Caterpillar* as a novel higher-level leader-based programmable matter, which is made of lower-level particles. We design five deterministic algorithms under geometric amoebot model to implement the “curle” operation of this high level matter and calculate their complexity with respect to the total number of movements of the particles. Furthermore, we implement all the proposed algorithms in

a simulator and empirically evaluated their performance.

- In Chapter 5, we conclude this thesis with a summary of the main results, contributions, and some future directions.

Chapter 2

Collective perception accuracy in multi-robot systems using self-organized hierarchy

This chapter considers the problem of collective perception in multi-robot systems. Here, we study the performance of an MNS-enabled approach in a collective perception task and compare its perceptual accuracy, scalability, and fault tolerance to that of three state-of-the-art decentralized approaches.

Self-organization is a mechanism or process in which global spatial-temporal order emerges in an initially disordered system solely from local interactions of lower components and without using centralized control [44, 113]. Self-organized multi-robot systems are not necessarily as efficient as fully centralized ones, but they are more scalable and fault tolerant, due to redundancy and parallelization. Because centralized approaches suffer from single points of failure and a lack of scalability, they are poor candidates for many applications despite their higher performance. A well-designed composite of self-organized and centralized control might yield a hybrid approach that maintains the contrasting benefits of both. In multi-robot systems, centralized approaches are less practical than self-organized systems because of their critical drawbacks (i.e., single point of failure).

In this chapter, we study the performance of an MNS-enabled approach in a collective task. More precisely, we propose a new hierarchical approach to a collective perception task in which a team of robots collectively perceives the density of blocks which are randomly distributed in the environment. This is the first work on collective perception in which the robot team's perceptual accuracy of an absolute condition (i.e., the density of blocks) is directly assessed. Our approach is based on the concept of the remote mergeable nervous systems (MNS) [140]. To the best of our knowledge, this is the first hierarchical approach to a collective perception task in multi-robot systems in which some aspects of centralized and decentralized control are combined. We test the perceptual accuracy of our MNS-enabled approach in both static (i.e., time-invariant) and dynamic (i.e., time-varying) simulated environments. To confirm whether increased centralization improves performance, we compare the MNS-enabled approach to three fully self-organized approaches. We also test whether our MNS-enabled approach maintains scalability and fault tolerance, which are crucial features of self-organized systems.

The remainder of this chapter is organized as follows. Section 1 reviews related work. Section 2 describes our collective perception scenario, the approaches to this task, and the experiment setups that are used in simulation. In section 3, the results of the experiments are presented. In section 4, we discuss the results obtained in the experiments. Finally, in section 5, we present our conclusion and discuss possible future works.

2.1 Related Work

Collective perception is a process by which a robot swarm collaboratively perceives an environmental stimulus [71, 111, 118, 126]. This problem has a close relationship with collective decision-making and aggregation which are two popular collective tasks in multi-robot systems.

Existing studies involving collective perception are typically set up as a best-of-n decision-making problem, where a robot team compares multiple options according to a given criterion. For example, the following are several common setups for this problem: 1) robots sense colors in a grid environment and compare them according to the criterion of highest representation [9, 10, 48, 114, 126]; 2) robots compare discrete zones according to a criterion of *quality* that can be sensed from anywhere in the zone [100, 101, 124, 127]; 3) robots collectively decide on one of the existing options in the environment based on a criterion such as intensity of light after exploring the environment [130].

Collective perception can also enable the robots to gather in a specific location in the environment, which is unknown to them a priori. In the literature, there are just a few works on this application of collective sensing [86, 130].

Sometimes collective perception is studied in more complicated setups than those mentioned above. For example, object classification is one of these setups in which robots explore an unknown environment and collectively classify objects or areas they discover based on a criterion such as color [58, 71, 76, 82, 90, 95]. In object classification, robots deal repeatedly with best-of-n decision-making problems.

The accuracy of the robot team's perception is not assessed directly in the studies mentioned above. For instance, in decisions based on highest color representation, if a team of robots underestimates or overestimates true density, it might do so for each color somewhat consistently, and therefore could still accurately determine a color's relative dominance. In almost all existing works in the literature, the robot team's perceptual accuracy for an absolute condition is not measured in relation to ground truth. The one exception is [73], in which the swarm's perception of absolute density with respect to ground truth is measured. The results show that there is a noticeable difference between the swarm's perception of density and the ground truth. In short, collective perception of an absolute condition in robot teams requires more study.

2.1.1 Collective perception in collective decision-making scenarios

Collective perception has a close relationship with collective decision-making [128], which has been widely studied in multi-agent systems. In the following, we present a summary of each related work focusing on the approaches usually used as benchmarks and the most common scenarios.

One of the foundational approaches to collective decision-making is presented in [27], which is called DeGroot model. In this model, a social system is represented as a weighted graph, and agents are the vertices of the graph. Each agent has an opinion about a common

question of interest and influences its neighbors' opinions. The significance of the influence depends on the weight of the edges. At each step, each agent communicates with its neighbors and updates its opinion by taking a weighted average of its neighbors' opinions. Eventually, this simple process leads to a consensus. This rule is relevant to one of the decision models in this chapter.

In [18, 60], the Voter model is proposed as an effective model for consensus achievement. In the Voter decision model, agents are represented as nodes of a graph, and each agent interacts only with its direct neighbors in the graph. At random times, an agent's opinion is changed to that of a random neighbor. This process continues until consensus is achieved by the agents. In [124], the authors propose a collective decision-making approach based on the classic Voter model. In their scenario, there are two available sites in the environment, each with a different quality. A swarm of robots should achieve consensus on the site with higher quality.

In another work [127], majority rule is used in a similar decision-making scenario. Based on this approach, a robot changes its opinion to the one shared by the majority of its neighbors, instead of random selection.

In [126], the authors present three approaches to collective perception: DMMD, DMVD, and DC. In their scenario, a group of robots continuously explores the environment, estimates the frequency of certain features, and collectively perceives the most frequent feature. More specifically, the environment is a two-color grid where cells are painted in black and white, and the robots should perceive the frequency of the black and white cells. The DMMD strategy uses the majority rule model based on which each robot takes the opinion favored by the majority of its neighbors (including its own current opinion). The DMVD strategy uses the voter decision model, based on which each robot randomly selects a neighbor's opinion as its own opinion. In the DC strategy, robots adopt the opinion of a random neighbor only if the quality estimate of the neighbor is higher than the current quality estimate of the robot. In [9], novel benchmarks for collective perception are proposed. The authors test the approaches developed in [126] in environments with different tile patterns. The results of their work show that in addition to the ratio of the black and white tiles, the pattern has a large impact on the performance of collective perception approaches. In another work [114], a novel approach to collective perception, called Distributed Bayesian Hypothesis Testing (DBHT), is proposed. The presented approach is tested in a similar two-color environment, and its performance is compared with approaches developed in [126]. The results show that DBHT often outperforms the other three state-of-the-art approaches in determining the dominant color. The authors also illustrate that using a distributed perception approach along with a centralized opinion fusion strategy is easier than making use of a fully decentralized decision-making strategy. In [10], the authors show how isomorphic changes in the environment impact the

performance of state-of-the-art strategies in collective decision-making. They consider a collective perception scenario in which robots operate in a square grid painted in two colors similar to that used in [126]. The robots should explore the environment and achieve consensus on the prevailing color in the grid. The results show that through isomorphic transformations, complicated patterns in an environment can be mapped to simpler ones while keeping their original structure and improving the performance of the collective decision-making strategies. Although it is impossible to apply isomorphic transformation directly to the environment, the results give us a better understanding of designing collective decision-making approaches with higher performance in the future. In [49], the authors present a distributed Bayesian algorithm for the task of collective perception in a similar two-color grid environment. They define the speed-accuracy trade-off of the task as a multi-objective optimization problem. They illustrate that the accuracy of their approach to collective perception is guaranteed but at the cost of the time required for the decision.

In the literature, collective perception in dynamic environments has also been studied. For instance, in [100], the authors study the best-of-n decision-making problem in dynamic environments. In this research, a simulated robot swarm operates in an environment similar to the one presented in [124]. In the dynamic setup presented in this work, the site qualities are swapped at a given time. The robots should achieve consensus on the option with the highest quality. The authors show that the Voter model cannot guarantee consensus achievement on the best option after the swap. To tackle this problem, they introduce the notion of stubborn robots which are not allowed to change their opinion about the best quality. The authors show that the presence of this type of robot is enough to attain adaptability to dynamic environments. In a follow-up work [101], using the same scenario, the authors show that a relatively small number of stubborn robots is enough to achieve adaptability, but increasing the number of this type of robot leads to a reduction in the swarm's performance. The authors introduce a new adaptation mechanism called spontaneous opinion switching which can be used to achieve adaptability to dynamic environments. They also show that the system's adaptation to changes improves by increasing the swarm size, but this is independent of the density of the robots. The density of the robots only affects the adaptation if it is below a certain threshold.

There are also a few works related to collective decision-making and collective perception in which the decision collectively made by robots is reflected in a particular collective behavior. For example, in [90], the authors investigate how a robot swarm can collectively encode the density of black spots on the ground into flashing signals which are observable by all robots. The higher the density, the higher the frequency of the flashing signals emitted by the robots. In [95], a robotic swarm with limited communication uses a distributed artificial neural network to distinguish between spatial light patterns. The robots collectively sense light conditions in the environment and, depending on the pattern observed, perform

a particular collective behavior as a coordinated response.

There are also two works in the literature related to collective decision-making and collective perception, which either have a different scope or a different scenario from those reviewed above. In [118], the authors present a blockchain-based approach in a collective decision-making scenario which provides solutions for security issues caused by Byzantine robots. A robot swarm must explore an environment consisting of black and white tiles and reach a consensus on a color with higher percentage in presence of Byzantine robots. To this end, the robots should correctly perceive the percentage of each color. In [111], the authors propose two approaches to a collective perception task: Trophallaxis-inspired strategy and hop-count strategy. In their scenario, a swarm of robots collectively explores an environment, finds target areas, and recruits a cohort of robots to each target area proportional to its size.

2.1.2 Collective perception in aggregation scenarios

Another collective task that can be related to collective perception and collective decision-making is aggregation. Inspired by biological organisms such as bees [129] and cockroaches [20, 54, 55], aggregation aims at gathering a group of robots in a common region and forming a cluster in the environment [128]. Collective perception strategies can be used to achieve this goal. In other words, robots can collectively perceive their environment and gather in a region based on a specific criterion, such as intensity of color. In an aggregation task, the opinion of each robot can be implicitly represented by its position. In the literature, a few works on this problem are related to collective perception.

One example is the work presented in [130], which is an extension of the Beeclust algorithm [110]. The Beeclust algorithm is inspired by the aggregation behavior of young honeybees. In this algorithm, when a robot encounters another robot, it waits in place for a certain amount of time, depending on the illuminance of the area. The higher the illuminance, the longer the robot stays stationary. This simple behavior leads to aggregation in the brightest spot. In [130], the authors extend the Beeclust algorithm to adapt each robot's waiting times to dynamic swarm densities and light conditions. In their extension, through local communication, robots collaboratively perceive the brightest spot in the environment and aggregate there. At each step, each robot independently estimates illuminance and robot density while moving in the environment. The robots exchange this information when they are in proximity of each other and, by using these estimates, adjust their waiting times. The results of this work show a noticeable improvement in the performance of the Beeclust algorithm.

Another example is presented in [86], an aggregation-based approach to collective perception. In this approach, a group of robots explores an environment populated with good

and bad quality spots. Robots collectively perceive and destroy bad spots while maintaining those perceived as good. In order to destroy a spot perceived as bad, the robots start forming an aggregate inside the spot. When the size of the aggregate reaches a certain threshold, the robots collaboratively destroy the spot.

2.1.3 Collective perception in classification scenarios

Sometimes, collective perception is set up as a classification task: robots should collectively classify the objects they perceive in the environment. Each instance of classification can be considered a best-of-n decision-making problem. In one example, in [82], robots collectively classify objects. For each object the robots encounter during exploration of the environment, they collectively decide on the best class that matches it, making a series of best-of-n decisions. In [71], the problem of environmental edge detection, which is a classification problem, is studied using a swarm of homogeneous robots. In this scenario, robots should detect and isolate areas with a specific characteristic in an unknown environment by marking their edges. Robots explore the environment and share their observations on a specific environmental feature (i.e., the color of the ground) with their neighbors. Then, they collectively decide on a threshold they can use to detect feature edges. They then search the environment and mark the detected edges. In [58], the authors study a robot swarm involved in a cooperative hand gesture classification task. Each individual is equipped with a statistical gesture classifier which is used to generate an opinion about the sensed gesture. The robots exchange their opinions and achieve consensus on a given gesture using a distributed consensus protocol. Another classification approach that uses collective perception is presented in [76]. In this approach, each robot forms an opinion by observing a given object. Then, the robots exchange their local hypotheses and positions, and the information obtained by them is fused, enabling them to classify the objects cooperatively.

2.1.4 Collective perception of absolute conditions

In nearly all the works related to collective perception, the robot team's perceptual accuracy for absolute conditions has not been directly assessed. More specifically, they either deal with the perception of a relative condition (e.g., determining the color more represented in a two-color grid environment) or the completion of a targeted action using the information sensed by the robots. The results of these works cannot be directly used for the perception of absolute conditions (e.g., determining the exact percentage of the colors of a multiple-color grid environment). For example, in a scenario where robots should perceive the prevailing color of a two-color grid, if the robots underestimate or overestimate the percentage of colors, they still might be able to determine the dominant color correctly. In one exception,

perception of an absolute condition is evaluated as an auxiliary contribution [73]. In this work, a robot swarm should collectively perceive the density of blocks around a construction zone and keep a minimum density of blocks available to the robots for the construction task. The blocks that are directly used for construction are kept in a cache area, which is an area that surrounds the construction zone. In this scenario, the robots should frequently explore a foraging area and transport the blocks to the cache area while maintaining a minimum density of the blocks in the cache. This process is done by estimating the number of blocks in the cache and using them for construction. The results show that the swarm's perceptual accuracy substantially diverges from the ground truth. The authors also show that accurate perception is not required in their scenario to achieve the targeted construction goal. The perceptual accuracy of robot teams for absolute conditions requires more research.

2.2 The remote Mergeable Nervous Systems (MNS) concept

Mergeable Nervous Systems (MNS) [84] is an existing concept for establishing dynamic hierarchy among robots using self-organization and physical connections. The concept has been extended to cases with remote connections rather than physical ones and has been implemented in heterogeneous swarms of ground robots and unmanned aerial vehicles (UAVs) [140].

In the remote MNS concept, a heterogeneous swarm can self-organize into a dynamic ad-hoc wireless communication network with a hierarchical structure. More specifically, the topology of an MNS is a directed rooted tree that is formed and maintained using local communication. In order to make a communication link between two robots, a Recruit message is sent from one to the other. A link is successfully established when the receiver of the message replies with an Accept message. In such a case, the sender of the Recruit message becomes the parent of the receiver. A child temporarily cedes authority to its parent, which can track its motion and send it motion instructions. Using the communication links, a child also reports sensor information to its parent. The root of the tree has access to the information of all the members in the tree and sends control information to steer the behavior of the whole system. The robots in an MNS can self-reconfigure into a new communication network and control hierarchy on the fly, for instance, in case of task change or brain failure. Please refer to [140] for full implementation details about how the MNS functions and the self-organizing process by which an MNS is established, maintained, and modified.

2.3 Problem statement

We propose a self-organized hierarchy approach to collective perception, based on the MNS concept, and test it against fully decentralized collective perception in simulated experiments. In the **self-organized hierarchy approach** (HIER), a robot fulfilling the temporary role of "brain" forms one collective opinion on behalf of the group, using collective sensor information merged hierarchically by all robots. In the fully decentralized approaches, each robot partakes (either explicitly or implicitly) in a collective decision-making process to form its own opinion and reach decentralized consensus with its peers.

We test the HIER approach against the following three fully decentralized approaches, as benchmarks for comparison.

- **Voter decision model** (VOTE): each robot selects new opinions randomly, from among the current opinions of itself and its neighbors (based on [126]).
- **Mean decision model** (MEAN): each robot averages the opinions of itself and its neighbors (based on [126]).[1]
- **Stigmergy** (STIG): robots do not communicate explicitly, but leave cues for each other to observe in the environment (based on [62]).

In all approaches, simulated ground robots use short-range onboard sensing to detect some objects that are distributed randomly in an arena of unknown size. Their collective goal is to form an accurate opinion on the mean density of objects in the whole arena.

2.3.1 Experiment design

In this chapter, robots collect samples by detecting individual objects, and then infer the mean density of the arena using odometry and knowledge of their own sensor ranges.

Absolute object density $\lambda = b/a$ is defined as the number of objects b per unit area a. The true mean density λ in the environment is noted as λ^{true}. Robots are not given any information about the size and shape of the objects nor the size and shape of the arena. Robots only know the dimensions of their own fields of view and must use this knowledge to infer the number of objects per unit area.

To study the baseline performance of the tested methods, we construct stochastic fields $\mathcal{R}$ with relatively low spatial variability by distributing the features (the objects to be detected)

[1] To have the best possible performance as a benchmark, we use average opinion instead of majority opinion, as in [126], because our task is a continuous decision not a discrete decision.

uniformly randomly in a regular polygon area. Still, inference ψ is not negligible even in the case of optimal sampling, because the robots cannot detect the target value (i.e., density) directly, and must instead infer it from representative information (i.e., short-range boolean detection of objects).

We test the approaches under several time-invariant and time-varying $\mathcal{R}$. Each approach is assessed in terms of perceptual accuracy (i.e., the bias in the collective opinion, with respect to the true value), consistency of accuracy (i.e., the variance in the bias), and reaction time. The scalability and fault tolerance of each approach is also assessed in time-invariant $\mathcal{R}$, in terms of the change in accuracy under robot failures and group size variations.

2.4 Methods

The robots run two parallel processes.

- Process A: robot r **individually** counts detected objects and infers the density in its own field of view.
- Process B: robots influence the **collective** opinion of the group via either the inputs or outputs of Process A, and robot r forms its opinion on the absolute object density (λ) in the arena.

Robots use only local or indirect communication to influence the collective opinion of the group, so the motion routines and communication rules of each approach are presented in the descriptions of Process B.

Process A is the same in each approach (Hier, Vote, Mean, Stig), except for the tuning parameters. Process B is different in each approach.

2.4.1 Process A: Robots make individual interpretations

In all approaches, robot r counts sensed objects and infers the density in its (direct or indirect) field of view, outputting the value ϵ_r. The sensing and inference output ϵ_r is defined as the average number of objects per unit area in the robot's field of view in a given time window, calculated as follows for time t:

$$\epsilon_r = \frac{\sigma_t - \sigma_{t-k_t}}{v \cdot s \cdot k_t^{\texttt{max}}} \cdot P \ , \tag{2.1}$$

where σ_t is the cumulative number of objects seen from $t = 1$ to the current time t by robot r, k_t is the time window at time t, $k_t^{\texttt{max}}$ is the maximum time window, and the remaining

terms are tuning parameters: v is the view area of robot r, s is the average speed of robot r, and P is a parameter related to the motion routines of the different approaches. (For details on the tuning parameters, see Sec. 2.5.)

The time window k_t keeps track of the elapsed time since the robot started the mission, up to the maximum time window $k_t^{\texttt{max}}$, i.e.,

$$k_t = \begin{cases} t, & \text{if } t < k_t^{\texttt{max}} \\ k_t^{\texttt{max}}, & \text{otherwise} \end{cases}. \tag{2.2}$$

At each time step, a new value σ_t is saved to a circular buffer $\boldsymbol{\sigma}$ of length $k_t^{\texttt{max}}$ in the memory of robot r. In other words, after $\boldsymbol{\sigma}$ reaches $k_t^{\texttt{max}}$ elements (i.e., when $t \geq k_t^{\texttt{max}}$), then at every subsequent update of σ_t (at position $i = k_t^{\texttt{max}}$), each element at $i : i \in \{1, \ldots, k_t^{\texttt{max}} - 1\}$ is replaced by the element at $i + 1$. Buffer $\boldsymbol{\sigma}$ is defined as:

$$\boldsymbol{\sigma} = (\sigma_{t-k_t}, \sigma_{t-k_t+1}, \ldots, \sigma_t), \qquad \sigma_t = \sum_{i=1}^{t} b_i, \tag{2.3}$$

where b_t is the number of objects detected (directly or indirectly) at the current time step t by robot r. Note that, in order to calculate σ_t, only σ_{t-1} and the most recent b_t values are required; it is not necessary to maintain a memory of all previous b_t values.

In summary, at each time step t, Process A: (1) inputs b_t, the number of objects the robot detects, and (2) outputs ϵ_r, the inferred density in the robot's field of view during the time window.

2.4.2 Process B: Robots influence the collective opinion

In all approaches, all robots influence collective opinion through either the inputs or outputs of Process A. Also, each robot r uses the outputs of Process A to form its opinion λ_r^{app} of the apparent absolute density in the arena.

In the decision model approaches (VOTE, MEAN), robots modulate the input b_t individually and process the output ϵ_r collectively. In the other two approaches (STIG, HIER), robots do the opposite—they modulate the input b_t collectively and process the output ϵ_r individually. So, in all four approaches, the opinion λ_r^{app} is influenced by a collective process.

2.4.2.1 Decision model approaches (Vote, Mean)

We test two fully decentralized approaches that use a collective decision-making process to reach consensus about the apparent density in the arena. Both approaches use a basic stochastic motion routine and explicit communication among neighbors. The robots coordinate their opinions using either a voter decision model or a mean decision model.

Stochastic motion routine. The VOTE and MEAN approaches use a stochastic motion routine based on RANDOM BILLIARDS [19, 66]. Each robot moves forward at a constant velocity unless it detects the boundary line of the arena. A robot can detect a line's angle relative to its own heading by driving over the line, and can likewise detect the "inside" or "outside" of the arena by driving partially over the respective area. When a robot detects an arena boundary line, it turns away from the line in a random direction towards the "inside" of the arena. For a boundary line with detected angle θ_1 and the "inside" of the arena towards the direction $\theta_1 + \frac{\pi}{2}$, the robot turns to a random direction with uniform distribution $U(\theta_1, \theta_1 + \pi)$.

A robot pauses its RANDOM BILLIARDS motion and performs obstacle avoidance when it meets an object or another robot. Robots use line-of-sight sensing and communication to detect (and distinguish between) objects and robots that are within short-range radius ρ_1 and within $\pm\frac{\pi}{3}$ of the heading angle θ_h. When a robot sees either an object or a robot, it turns away from it at until it is no longer visible within $\theta_h \pm \frac{\pi}{3}$. If the detection was of an object, then the input of Process A is updated as $b_t \leftarrow b_t + 1$.

Voter decision model (Vote). In this approach, robots share their individual inference outputs ϵ_r (i.e., the outputs of Process A) using explicit communication. Each robot then uses a voter model process to form its opinion λ_r^{app} of absolute object density.

For explicit communication, each robot r_n maintains and shares a matrix $\boldsymbol{\epsilon}$ as follows:

$$\boldsymbol{\epsilon} = \begin{pmatrix} r_1 & \epsilon_{r_1} & t_{r_1} \\ r_2 & \epsilon_{r_2} & t_{r_2} \\ \vdots & \vdots & \vdots \\ r_m & \epsilon_{r_m} & t_{r_m} \end{pmatrix}, \tag{2.4}$$

where r_j is the robot ID, ϵ_{r_j} is the most up-to-date ϵ_r value known for robot r_j by robot r_n, and t_{r_j} is the time stamp of the known ϵ_{r_j}. At each time step, robot r_n updates its own ϵ_{r_n} and t_{r_n} according to Equation 2.1 and then sends its matrix $\boldsymbol{\epsilon}$ to its current neighbors. If a robot receives a matrix that contains a higher t_{r_j} than its current entry for that robot, it

updates its row for r_j accordingly. In this way, the $\boldsymbol{\epsilon}$ of robot r_n always contains the most up-to-date ϵ_r values for its peers that r_n has seen thus far.

At each time step, robot r decides λ_r^{app} by randomly selecting one ϵ_{r_j} entry from its matrix. In other words,

$$\lambda_r^{\text{app}} = \epsilon_{x\sim\ U(r_1,r_m)} \in \boldsymbol{\epsilon}, \tag{2.5}$$

such that opinion λ^{app} of absolute density is the result of a voter decision model.

Mean decision model (Mean). In this approach, robots also share their individual inference outputs ϵ_r using matrices $\boldsymbol{\epsilon}$ as defined in Eq. 2.4.

At each time step, robot r decides λ_r^{app} by averaging the ϵ_{r_j} entries in its matrix. In other words,

$$\lambda_r^{\text{app}} = \overline{\epsilon_{r_j}} \in \boldsymbol{\epsilon}, \tag{2.6}$$

such that opinion λ^{app} of absolute density is the result of a mean decision model.

2.4.2.2 Stigmergy approach (Stig)

In this approach, robots influence the collective opinion via the inputs of Process A—the number of objects detected (b_t)—not the outputs of Process A.

This approach uses stigmergic (indirect) communication, through artificial pheromones—i.e., cues left in the environment that are observable by the robots within a certain range. Robots use the pheromones deposited in the environment to reach a consensus about the apparent density in the arena.

When a robot detects an object within short-range radius ρ_1, it counts the object and deposits pheromone at the object location. Robots can detect and differentiate pheromone sources within long-range radius ρ_2, which allows them to count objects already found by their peers in a much larger vicinity than that in which they can detect objects directly. Each robot counts sensed pheromone sources the same as sensed objects when b_t (see Eq. 2.3 of Process A). Robots also understand that pheromones are cues that the immediate area has already been explored by another robot, and therefore turn away to search for new unexplored areas.

Each robot can sense a pheromone in any direction, and can sense the relative direction and distance of its source. When a robot senses a pheromone source within long-range radius $\rho_2 - \delta_2$ and in a direction close to that of its heading, it turns away from the source to a randomized direction, according to Algorithm 1.

Algorithm 1 Pheromone reaction

Input:
θ_X // Randomized angle, see Eq. 2.7
$\overrightarrow{s}$ // Vector from robot to pheromone source
$\overrightarrow{r}$ // Vector of robot heading
t_s // Time since last pheromone reaction

$\theta_{sh} \leftarrow$ angle between $\overrightarrow{s}$ and $\overrightarrow{h}$
while ($|\theta_{sh}| \leq |\theta_X|$ **and** ($t_s \leq 50$ or $t_s \geq 250$)) **do**
  robot turns heading in the SIGN(θ_x) direction
end while
move forward

The angle θ_x is selected randomly once every 200 time steps, with the following uniform distributions:

$$\theta_{x\sim} \begin{cases} U(\frac{\pi}{6}, \frac{\pi}{2}), & \text{if } \left\lfloor \frac{t}{200} \right\rfloor \text{ is even} \\ U(-\frac{\pi}{2}, -\frac{\pi}{6}), & \text{otherwise} \end{cases}. \tag{2.7}$$

If a robot is simultaneously within range of multiple pheromone sources, it logs them in a list with an arbitrary order, and reacts to the first pheromone source in its list for that time step. After a robot reacts to a given pheromone source, it continues moving forward until it encounters a different pheromone, an object, or a boundary line.

Recall that robots have already influenced the collective opinion via the inputs of Process A. Therefore, at each time step, robot r simply takes its own inference output ϵ_r as its opinion λ_r^{app} of absolute density. In other words,

$$\lambda_r^{\text{app}} = \epsilon_r. \tag{2.8}$$

2.4.2.3 Hierarchical approach (Hier)

In this approach, robots influence the collective opinion via the inputs (not the outputs) of Process A, and only the robot occupying the dynamic leadership position, the MNS-brain robot r, performs Process A.

The HIER approach is based on the existing MNS concept, which is a general framework for constructing and reconstructing self-organized hierarchy [84]. Under the MNS framework, robots can self-organize a dynamic ad-hoc control network in which robots temporarily and interchangeably occupy certain positions in a leadership hierarchy, including a MNS-brain position [65, 66, 139, 140]. In an MNS control network, each robot communicates only with its direct neighbors, to prevent the type of bottleneck that would occur at the communication hub

in a fully centralized system. According to task specifications and system constraints, sensor information can be merged as it is passed upstream, control information can be unmerged as it is passed downstream, and the balance of individual behaviors versus collective behaviors can be actively managed. This flexibility can be used to reduce or eliminate the potential for bottlenecks throughout the hierarchical network.

In this chapter, we use the MNS implementation of [140], in which camera-equipped UAVs[2] are responsible for sensing the relative positions and orientations of the ground robots for the purpose of keeping the robot formation together during sweeping. We use the sweeping technique of [65] and apply it to the task of collective perception.

In the motion routine of Process B of the HIER approach, robots self-organize into a roughly linear formation to sweep the environment (for details, see [65]. In the HIER approach, the MNS-brain robot r detects arena boundary lines and reacts to them using a deterministic process. The dimensions of the MNS's collective ground robot sensor range, calculated by the MNS based on the number of ground robots in the formation, are used as parameters in the deterministic process. The MNS sweeps the unknown environment in a zigzag-shaped path suited to its own dimensions, in such a way that the collective sensor ranges of the ground robots can cover the environment as optimally as possible.

As it sweeps the arena, the MNS-brain robot sends control information downstream, to maintain the formation(for full details, see [140]. Each robot in the MNS receives motion instructions from its parent that include the targeted relative linear velocity $\mathbf{v}$ and angular velocity $\boldsymbol{\omega}$, as well as the current orientation quaternion $\mathbf{q}_t$, following [140]. In order to send motion instructions, each parent in the MNS senses its child's displacement $\mathbf{d}_t$ and relative orientation $\mathbf{q}_t$, determines the new targeted values based on the most recent motion instructions it received from its own parent, and calculates motion instructions for its child as follows [140]:

$$\mathbf{v} = k_1 \left(\frac{\mathbf{d}_{t+1} - \mathbf{d}_t}{||\mathbf{d}_{t+1} - \mathbf{d}_t||} \right), \ \boldsymbol{\omega} = k_2 \cdot ||f(\mathbf{q}_{t+1}^{-1} \times \mathbf{q}_t)||, \tag{2.9}$$

where k_1 and k_2 are speed constants and function $f(x)$ converts a quaternion to a Euler angle.

In the HIER approach, when a ground robot detects an object within short-range radius ρ_1 and within $\pm\frac{\pi}{3}$ of the heading angle θ_h, it temporarily ignores the motion instructions received from its parent in order to circumvent the object in a predefined arc path until the the object is no longer within $\theta_h \pm \frac{\pi}{3}$. Complementarily, if the parent of a ground robot detects that its child is behind another ground robot within $\rho_1 + \delta_1$, then the parent will temporarily ask the child to stop moving, until the two ground robots are no longer within

[2]Note that UAVs in the HIER approach in this chapter cannot sense objects directly, so do not increase the total sensing range available in the approach. If we were using ground robots that were capable of sensing each others' relative positions and orientations, the UAVs could be removed, and this removal would not have an impact on the collective perception results.

$\rho_1 + \delta_1$ of each other. The child will accept the request as long as it detects an object within short-range radius ρ_1.

In the HIER approach, ground robots essentially act as temporary remote sensors of the MNS-brain robot r. Each child robot sends its sensor readings upstream to the MNS-brain robot r via its parent[3], which calculates one inference output ϵ_r for the whole MNS. Therefore, at each time step, the MNS-brain robot r takes its own inference output ϵ_r as its opinion λ_r^{app} of absolute density, i.e.,

$$\lambda_r^{\text{app}} = \epsilon_r. \tag{2.10}$$

2.5 Experiment setup

All the experiments of this chapter are conducted using the ARGoS multi-robot simulator [99] with robot models implemented using an existing plugin [3, 4]. The experiments are all conducted in a 6 x 6 m^2 environment occupied with blocks of size 5 x 5 x 5 cm^3. The blocks are positioned randomly in the environment, following a uniform distribution. The experiments begin when the robots start sweeping the arena and the duration of the experiments is 50000 simulation steps (i.e., 2000 s). The ground robot and UAV models used in simulation are based on the e-puck ground robot [89] and the S-drone quadcopter [94], respectively. Both UAV and ground robot models are represented by a cylinder with 2.5 cm radius. The UAV model is equipped with a downward facing camera, enabling it to sense the relative positions and orientations of the ground robots for the purpose of keeping the robot formation together during sweeping. The UAVs are unable to detect the blocks directly. The ground robot model is equipped with a ring of 12 proximity sensors with a range of 3 cm, and a unique tag that is attached on the top, enabling the UAV model to detect the ground robot. Furthermore, both UAV and ground robot models have the same average linear velocity of 7.5 cm/s.

Regarding the tuning parameters used in Eq. 2.1, in the MEAN and VOTE approaches, the view area v of robot r is based on the sensor range of onboard object sensing and in the STIG approach v is based on the sensor range of onboard pheromone sensing. In the HIER approach, v is based on the maximum bounds of the combined sensor ranges of all ground robots. Note that in all approaches, all ground robots have the same sensor range for detecting objects (short-range radius $\rho_1 = 3$ cm). The time window (i.e., $k_t^{\texttt{max}}$) used in the experiments has been set to 1000 time steps. This implies that in all four approaches a

[3]Implementation note: To reduce the per-step simulation time needed by CPU and GPU solvers, some of the upstream and downstream data transfers within the MNS are simulated as a single-step rather than multi-step process. In this chapter, the approaches are assessed according to the simulation steps, not the per-step time. The implementation strategy used negligibly impacts the number of simulation steps needed to collect and pass information (adds a maximum of 1 step), so it does not undermine the analysis of the reported results.

robot's memory of a sensing input lasts for 1000 time steps (40 s).

The likelihood that robots underestimate or overestimate the absolute object density is impacted by the degree of stochasticity in the motion strategy used by the approach. The approaches used in our comparison navigate the environment with varying degrees of stochasticity, so we consider a coefficient P which is tuned for each approach. To prevent a dependency on prior knowledge, the tuning phase is not adjusted to specific density conditions. In the HIER approach, the motion trajectory of the brain robot is deterministic, so $P = 1$. Regarding each decentralized approach, we have tuned P in a trial-and-error phase to get the best performance from Eq. 2.1 for each approach respectively. After the trial-and-error phase, we have set P as follows: for the VOTE approach $P = 0.48$, for the MEAN approach $P = 0.55$, and for the STIG approach $P = 1$. It is worth noting that Eq. 2.1 could be optimized by conducting a more extensive investigation of each of its parameters.

In all the experiments related to the HIER approach, the robots initially are positioned randomly in a 0.25 x 1.0 m^2 rectangular area close to the left southern corner, outside the boundaries. The MNS is formed on the boundary of the working space, and after completing the formation, it follows the zigzag path as already mentioned. In the experiments relevant to the decentralized approaches, the robots are initially distributed in the arena, following a uniform distribution. It is worth noting that, in the setups of the STIG approach, an artificial pheromone deposited by robot r at time step t evaporates after 20000 time steps. For more information about the details of the implementation of the approaches and the experiment setups, please see the open-source code repository [4].

The perception accuracy of all the approaches is evaluated in both static (time-invariant) and dynamic (time-varying) environments. Regarding the static environments, we consider three different numbers of blocks (100, 200, 300) and run 10 experiments per each (overall, 120 runs). The true densities λ^{true} in the case of 100 blocks, 200 blocks, and 300 blocks are 2.78 objects/m^2, 5.56 objects/m^2, and 8.33 objects/m^2, respectively. Regarding the dynamic environments, we test the accuracy of the mentioned approaches by considering two variable properties: amount of changes in block density and rate of changes in block density. More precisely, we consider four setups: 1) the number of blocks alternates between 50 blocks and 100 blocks (minor change) per each 1000 steps (fast mode); 2) the number of blocks alternates between 50 and 100 (minor change) per each 10000 steps (slow mode); 3) the number of blocks alternates between 50 blocks and 300 blocks (major change) per each 1000 steps (fast mode); 4) the number of blocks alternates between 50 and 300 (major change) per each 10000 steps (slow mode). It is worth mentioning that the true density for the environment with 50 blocks is $\lambda^{\text{true}} = 1.39$ objects/m^2. For each approach, we execute 10 runs per each of these four setups (in total, 160 experiment runs).

[4] https://github.com/BlueDiamond07/Collective_perception

We evaluate the scalability of the presented approaches in the default arena with 100 blocks (i.e., an arena with $\lambda^{\text{true}} = 2.78$ objects/m^2). The environments are time-invariant and we consider robot groups that contain either 4, 8, or 12 ground robots. We use a fixed arena size, and therefore the density of robots increases as the group scales. Higher robot density can sometimes cause detrimental interference in fully decentralized systems. However, our focus is not to test the limitations of the decentralized approaches but rather to test whether bottlenecks occur in the MNS-based approach. Therefore, we limit the tested group sizes to maintain a relatively low robot density in our given arena. By increasing the number of ground robots from 4 to 8 and from 8 to 12, we can roughly estimate the expected impact on moderately larger robot groups. These group sizes are also representative of those normally studied in multi-robot systems research. For each approach and each set of ground robots, we execute 10 runs (in total, 120 scalability runs).

Finally, we study the fault tolerance of the presented approaches when ground robots fail in the default arena with 100 blocks (i.e., an arena with $\lambda^{\text{true}} = 2.78$ objects/m^2). If a failure happens, the faulty robot cannot be replaced, and it keeps moving, communicating with other peers, and calculating its ϵ_r and opinion λ_r^{app} as normal, but it experiences sensor failure such that it cannot directly count objects. This means in Eq. 2.3 such a robot always adds $n = 0$ to $b_t = b_t + n$ as the number of objects detected when it detects an object with its proximity sensors (i.e., within short-range radius $\rho_1 = 3$ cm). Note that, in the Stig approach, this implies that a failed ground robot cannot deposit an artificial pheromone, but can still detect pheromones left by other ground robots. We examine the robustness of the approaches in the four following different setups where we use 8 ground robots in each approach: 1) none of the ground robots is faulty (the default static setup); 2) 2 out of 8 ground robots are faulty; 3) 4 out of 8 ground robots are faulty; and 4) 6 out of 8 ground robots are faulty. In all the fault tolerance experiments, the environment is time-invariant with 100 blocks randomly placed in. Per each approach and per each setup, we execute 10 experiment runs (in total, 160 runs).

2.6 Analysis approach

Recall that λ^{true} is the true global density of the environment and that each robot r generates the values ϵ_r (i.e., the density that robot r senses and infers in its own field of view) and λ_r^{app} (i.e., the opinion of robot r on the apparent absolute object density in the whole environment which is formed based on the collective influence of all robots). Note that, in the Stig and Hier approaches $\lambda_r^{\text{app}} = \epsilon_r$ at each time step.

2.6.1 Perception accuracy

We evaluate the perception accuracy of the approaches based on the bias of the robot opinions on apparent density λ^{app} with respect to true density λ^{true}. To this purpose, we consider mean squared error (MSE).

The cumulative bias is measured by the mean squared error of the opinion λ^{app} of robots r over time, calculated as:

$$\text{MSE}(\lambda_{rt}^{\text{app}}) = \frac{1}{nm} \sum_{r=1}^{n} \sum_{t=1}^{m} (\lambda_{t}^{\text{true}} - \lambda_{rt}^{\text{app}})^2. \tag{2.11}$$

The instantaneous bias is measured as follows:

$$\text{MSE}(\lambda_{r}^{\text{app}}) = \frac{1}{n} \sum_{r=1}^{n} (\lambda^{\text{true}} - \lambda_{r}^{\text{app}})^2. \tag{2.12}$$

2.7 Results

In this section, we present the results of the experiments. We first present the results of the experiments related to the static environments and then give the results related to the dynamic environments. Finally, we discuss the results of the scalability and fault tolerance experiments.

2.7.1 Perception accuracy in static environments

The results show that in static fields $\mathcal{R}$, the Hier approach has higher and more consistent accuracy, and displays faster reaction times than the decentralized benchmark approaches.

Figure 2.1 shows robot opinions on apparent density λ_r^{app} based on collective processes at all time steps (all robots in all runs), in the three mentioned static setups. As can be seen, increasing the number of blocks in the environment results in higher values for collective opinions.

Figure 2.2 and Table 2.1 show the instantaneous bias (Eq. 2.12) of each run and cumulative bias (Eq. 2.11) of all runs under time-invariant fields $\mathcal{R}$, respectively. From the plots and table, it is clear that the bias of the Hier approach in all the static setups is the least among all the tested approaches. Moreover, as can be seen, the spikes of this approach are minor and stable opinions are reached in less than 50 s. The Mean and Vote approaches are less accurate compared to the Hier approach, but they converge very quickly as well. In both

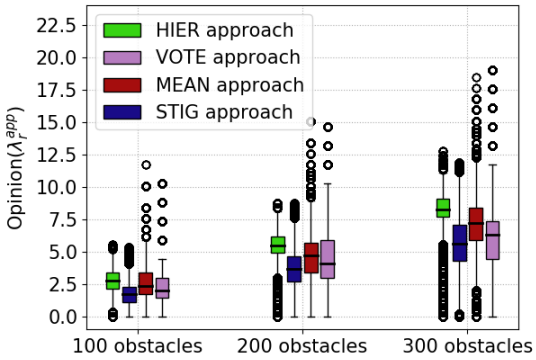


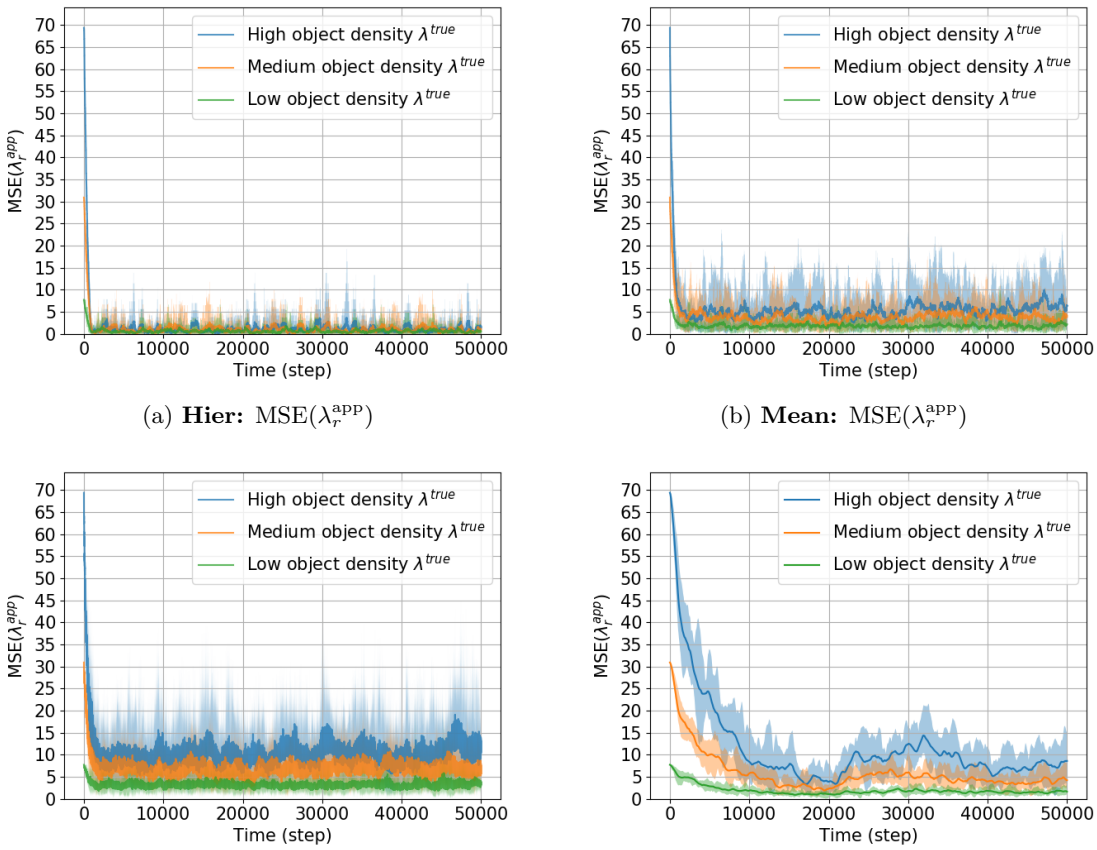
Figure 2.1: Opinions based on collective processes (λ_r^{app}) at all time steps (all robots in all runs), in static environments with three different true densities.

approaches the spikes are relatively high and last for less than 200 s. The MEAN approach is more accurate (please see Table 2.1) and more consistent (i.e., shows less variance in the bias) than the VOTE and STIG approaches. The VOTE and STIG approaches show almost the same accuracy and consistency but different convergence time. The STIG approach converges much more slowly compared to the VOTE approach and it displays more variance in the mean bias during the experiments in environments with higher λ^{true}. It is worth noting that for the STIG approach, the bias gradually decreases (i.e., the performance increases) from time step 0 to 20000 (i.e., the mean squared error decreases). Afterwards, the bias of this approach increases (i.e., the performance decreases) for a while and again it starts decreasing. These reductions in performance occur because of the pheromone evaporation rate in the STIG approach. As already mentioned, an artificial pheromone deposited at an object location lasts for 20000 time steps in all the experimental setups of the STIG approach.

As can be seen in both Figure 2.2 and Table 2.1, static environments with higher true densities λ^{true} are more challenging than those with lower λ^{true}. More specifically, for all the approaches, by increasing the number of obstacles, the accuracy of collective opinions decreases. The reason is that by increasing the number of blocks the inferences caused by blocks that should be avoided and circumvented are increased which results in lower performance. The amount of reduction in performance is extremely small in case of the HIER approach, and the largest amount of reduction belongs to the STIG approach.

2.7.2 Perception accuracy in dynamic environments

The same as the static environments, the results show that in dynamic fields $\mathcal{R}$, the HIER approach has higher and more consistent accuracy, and demonstrates faster reaction times than the decentralized approaches.



(a) **Hier:** MSE(λ_r^{app})

(b) **Mean:** MSE(λ_r^{app})

(c) **Vote:** MSE(λ_r^{app})

(d) **Stig:** MSE(λ_r^{app})

Figure 2.2: **Instantaneous bias under time-invariant fields $\mathcal{R}$.** Shaded line plots showing mean, minimum, and maximum of MSE(λ_r^{app}) of all runs, for three different true densities λ^{true} (objects/m^2): low density $\lambda^{\text{true}} = 2.78$, medium density $\lambda^{\text{true}} = 5.56$, and high density $\lambda^{\text{true}} = 8.33$.

Table 2.1: **Time-invariant field $\mathcal{R}$.** MSE(ϵ_{rt}) and MSE($\lambda_{rt}^{\text{app}}$) (i.e., cumulative bias) of all runs. In the STIG and HIER approaches, $\epsilon_{rt} = \lambda_{rt}^{\text{app}}$, therefore MSE($\epsilon_{rt}$) is redundant and is omitted from this table.

Control type	λ^{true} (objects/m^2)	MSE(ϵ_{rt})	MSE($\lambda_{rt}^{\text{app}}$)
HIER	2.78	-	0.6563
HIER	5.56	-	1.177
HIER	8.33	-	1.5154
VOTE	2.78	3.2392	3.2252
VOTE	5.56	7.0243	7.0108
VOTE	8.33	10.6300	10.6169
MEAN	2.78	3.6938	1.8346
MEAN	5.56	7.1599	3.9410
MEAN	8.33	9.7224	5.6553
STIG	2.78	-	1.8959
STIG	5.56	-	5.7643
STIG	8.33	-	11.5380

Figure 2.3 and Table 2.2 show the instantaneous bias (Eq. 2.12) of each run and cumulative bias (Eq. 2.11) of all runs under time-varying fields $\mathcal{R}$, respectively. As can be seen, the bias of the HIER approach in all the dynamic setups is the least among the tested approaches (see Table 2.2), and it is the only approach that consistently reaches very low bias after all changes in true density λ^{true}. From the table, among the decentralized approaches, the bias of the MEAN approach is smaller than that of the other decentralized approaches in all the dynamic setups. In the setup with slow, minor fluctuations, the STIG approach has a lower bias than the VOTE approach, while in the rest of the setups, the opposite is true.

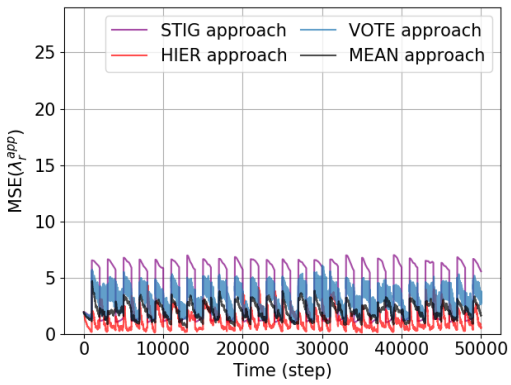
Under slow fluctuations (see Fig. 2.3b,d), after the true density immediately decreases (i.e., a shift to a lower density), each approach can successfully return to its lowest respective bias. During these periods of lowest respective bias, the accuracy of the HIER, MEAN, and STIG approaches is almost the same and higher than that of the VOTE approach. However, after the true density λ^{true} immediately increases (i.e., a shift to a higher density), the reaction of all the approaches is slower, and the bias is higher compared to when a shift to a lower true density λ^{true} occurs. In terms of reaction time under slow fluctuations, the reaction of the HIER approach after each time that the density changes is faster than other approaches. Among the decentralized approaches, the reaction of the MEAN approach is faster than the VOTE approach which in turn has faster reaction times than the STIG approach.

Under fast fluctuations, the performance of all the approaches worsens (see Fig. 2.3a,c), but still the HIER approach reacts much more faster compared to the other approaches. Under fast, minor fluctuations, the MEAN and VOTE approaches have relatively enough time to converge after a shift to higher or lower density λ^{true}, but they cannot converge under fast, major fluctuations. In case of fast, minor fluctuations, once the MEAN and VOTE approaches converge, their accuracy is the same as their accuracy under slow fluctuation setups. Finally, under fast fluctuations, the STIG approach does not have enough time to converge after a shift to a higher density λ^{true}, and it often reports lower values as robot opinions on apparent density λ_r^{app}; for this reason its bias is low when a shift to a lower density λ^{true} occurs.

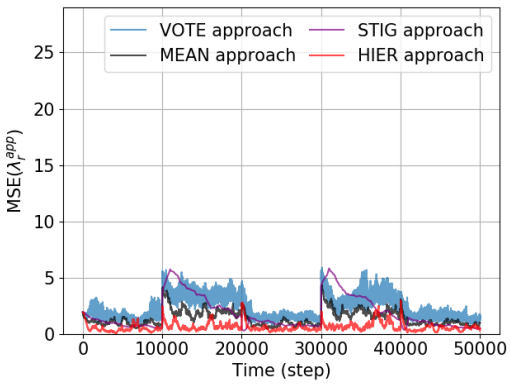
In general, in terms of reaction time, the HIER approach reacts much more quickly compared to the other approaches in all the setup of dynamic fields. The reaction time of the MEAN and VOTE approaches is fairly fast, but that of the STIG approach is very slow such that it cannot converge within 400 s (see Fig. 2.3d).

2.7.2.1 Summary of perception accuracy:

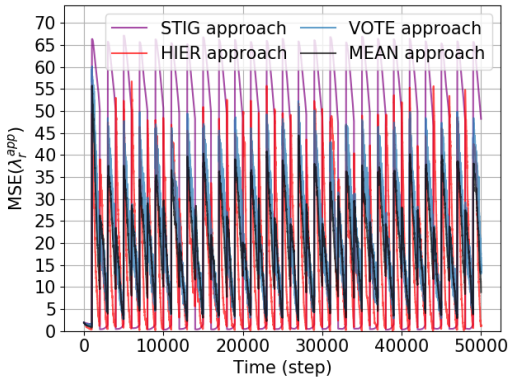
Overall, as has been discussed and can be seen in Table 2.1 and Table 2.2, the HIER approach demonstrates the minimum bias among the tested approaches in all the setups of time-invariant and time-varying fields. As can be seen in Fig. 2.2, under time-invariant



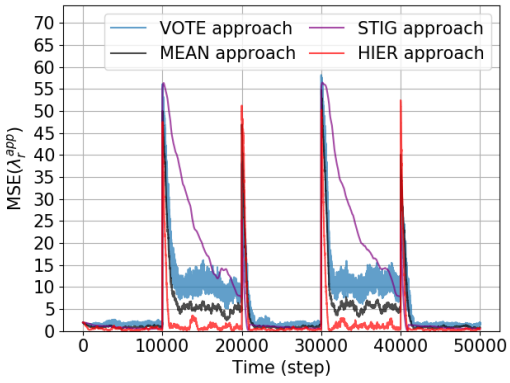
(a) **Fast, minor fluctuations:** Mean MSE(λ_r^{app}) under a field alternating every 40 s between densities that differ slightly: $\lambda^{\text{true}} = 1.39$ or 2.78 objects/m^2.



(b) **Slow, minor fluctuations:** Mean MSE(λ_r^{app}) under a field alternating every 400 s between densities that differ slightly: $\lambda^{\text{true}} = 1.39$ or 2.78 objects/m^2.



(c) **Fast, major fluctuations:** Mean MSE(λ_r^{app}) under a field alternating every 40 s between densities that differ substantially: $\lambda^{\text{true}} = 1.39$ or 8.33 objects/m^2.



(d) **Slow, major fluctuations:** Mean MSE(λ_r^{app}) under a field alternating every 400 s between densities that differ substantially: $\lambda^{\text{true}} = 1.39$ or 8.33 objects/m^2.

Figure 2.3: **Instantaneous bias under time-varying fields $\mathcal{R}$.** Mean MSE(λ_r^{app}) of all runs, for four spatiotemporal variations of fluctuating true density λ^{true}.

fields, the mean bias of the HIER approach is almost negligible. In some runs, the bias does spike occasionally, but it successfully recovers within 20–40 s. In general, compared to the decentralized approaches, the spikes of the HIER approach are less frequent and its largest spikes are minor. Similarly, in all the time-varying setups (see Fig. 2.3), the bias of the HIER approach is also noticeably less than that of the other tested approaches.

It is worth noting that regarding time-varying fields, immediately after some of the shifts to a lower or higher density, the bias of the HIER approach mostly spikes more aggressively than that of the other approaches. However, the HIER approach quickly recovers when these spikes occur, and usually, it reaches a smaller bias after recovery compared to the other approaches.

Table 2.2: **Time-varying field $\mathcal{R}$.** MSE(ϵ_{rt}) and MSE($\lambda_{rt}^{\text{app}}$) (i.e., cumulative bias) of all runs. In the STIG and HIER approaches, $\epsilon_{rt} = \lambda_{rt}^{\text{app}}$, therefore MSE($\epsilon_{rt}$) is redundant and is omitted from this table.

Control type	λ^{true} (objects/m^2)	speed of change	MSE(ϵ_{rt})	MSE($\lambda_{rt}^{\text{app}}$)
HIER	1.39, 2.78	40 s	-	1.2223
HIER	1.39, 2.78	400 s	-	0.6066
HIER	1.39, 8.33	40 s	-	17.5048
HIER	1.39, 8.33	400 s	-	2.0559
VOTE	1.39, 2.78	40 s	2.9333	3.0351
VOTE	1.39, 2.78	400 s	2.2762	2.3249
VOTE	1.39, 8.33	40 s	19.7566	22.2476
VOTE	1.39, 8.33	400 s	6.3037	6.9537
MEAN	1.39, 2.78	40 s	3.3630	2.0121
MEAN	1.39, 2.78	400 s	2.6962	1.3982
MEAN	1.39, 8.33	40 s	20.5309	19.5730
MEAN	1.39, 8.33	400 s	6.2645	4.5715
STIG	1.39, 2.78	40 s	-	3.7754
STIG	1.39, 2.78	400 s	-	1.8381
STIG	1.39, 8.33	40 s	-	29.8167
STIG	1.39, 8.33	400 s	-	10.8360

In summary, we can conclude that the HIER approach has higher and more consistent accuracy, and reacts accurately and more quickly compared to the three benchmark approaches.

2.7.3 Scalability

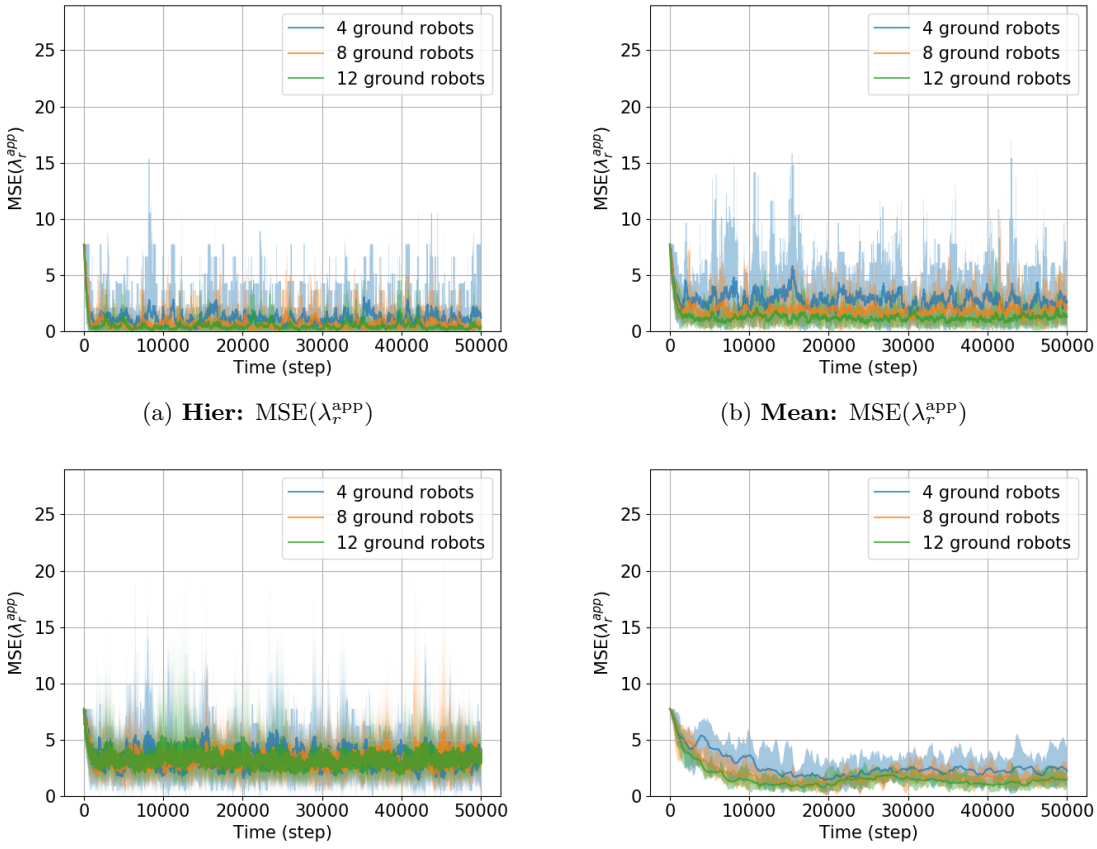
The results of the scalability experiments show that, in the tested group sizes, the bias of the HIER approach is lower than in the other approaches.

Figure 2.4 and Table 2.3 show the instantaneous bias (Eq. 2.12) of each run and cumulative bias (Eq. 2.11) of all runs in the scalability setups, respectively. It can be seen that by increasing the size of the robot team, the accuracy of none of the approaches decreases. This shows that the threshold at which robot-robot interference could have a negative impact on accuracy has not been reached, and none of the approaches demonstrates a communication bottleneck at the tested group sizes. Instead, in some of the approaches, the accuracy slightly decreases as the size of the robot team decreases.

The accuracy of the HIER, MEAN, and STIG approaches slightly decreases by using smaller group sizes. However, the VOTE approach demonstrates almost no difference in accuracy between group sizes. Although the difference in accuracy of the HIER, MEAN, and STIG approaches is relatively small, the MEAN approach shows a bigger difference between group sizes compared to the HIER and STIG approaches. Furthermore, the MEAN approach displays

larger and more frequent spikes in the smaller group sizes, which implies that it has much less consistent accuracy in the smaller group sizes than the other three approaches. According to Table 2.3, in all the setups, the HIER approach has a lower bias compared to the other approaches. In the setup with four robots, bias in the STIG approach is less than in the MEAN and VOTE approaches, while in the case of twelve robots, the MEAN approach shows a lower bias than the other two decentralized approaches. In general, all four approaches show good accuracy scalability as their accuracy does not decrease as the size of the robot team increases. The VOTE approach also demonstrates excellent resiliency to smaller group sizes as its accuracy stays the same in such cases. However, it is worth noting that the bias of the VOTE approach in all the tested group sizes is more than that of the HIER approach in its worst group size.

Overall, in all the tested group sizes, the HIER approach shows less bias than other approaches.



(a) **Hier:** MSE(λ_r^{app})

(b) **Mean:** MSE(λ_r^{app})

(c) **Vote:** MSE(λ_r^{app})

(d) **Stig:** MSE(λ_r^{app})

Figure 2.4: **Instantaneous bias when testing scalability.** Shaded line plots showing mean, minimum, and maximum of MSE(λ_r^{app}) of all runs, for three variations of group size: 4, 8, and 12 ground robots.

Table 2.3: **Scalability.** MSE(ϵ_{rt}) and MSE($\lambda_{rt}^{\text{app}}$) (i.e., cumulative bias) of all runs. In the STIG and HIER approaches, $\epsilon_{rt} = \lambda_{rt}^{\text{app}}$, therefore MSE($\epsilon_{rt}$) is redundant and is omitted from this table.

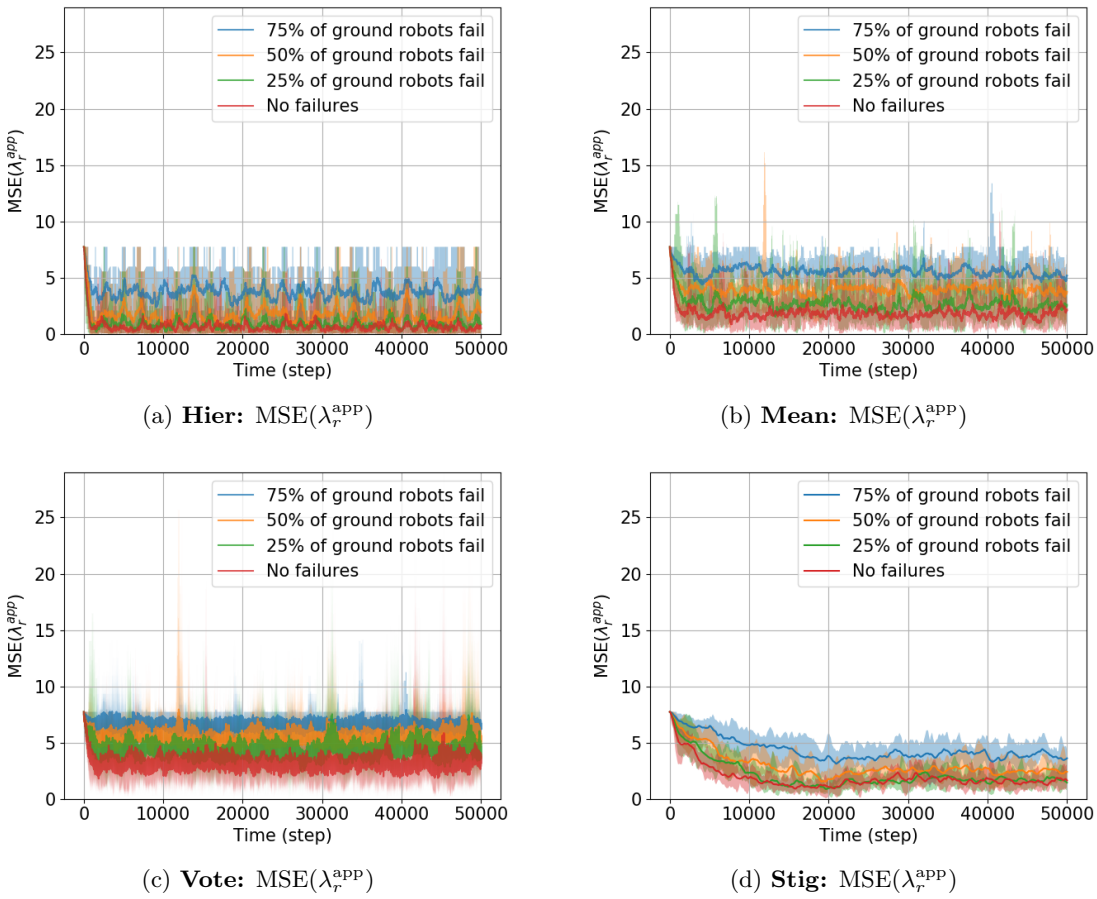
Control type	# of ground robots	MSE(ϵ_{rt})	MSE($\lambda_{rt}^{\text{app}}$)
HIER	4	-	1.1475
HIER	8	-	0.6563
HIER	12	-	0.4709
VOTE	4	3.3315	3.3373
VOTE	8	3.2392	3.2252
VOTE	12	3.2694	3.2414
MEAN	4	3.8103	2.8065
MEAN	8	3.6938	1.8346
MEAN	12	3.7828	1.2344
STIG	4	-	2.6160
STIG	8	-	1.8959
STIG	12	-	1.5867

2.7.4 Fault tolerance

The results of the fault tolerance experiments show that by increasing the percentage of robots that fail, the accuracy of all four approaches noticeably decreases. However, we expected this reduction as the fault tolerance setup is more challenging than the scalability setup. More specifically, in this setup, as already mentioned, failed robots keep moving in the arena and communicate with other peers but always record that they have directly detected zero objects, and this causes extra bias during the collective perception process.

Figure 2.5 and Table 2.4 show the instantaneous bias (Eq. 2.12) of each run and cumulative bias (Eq. 2.11) of all runs under time-varying fields $\mathcal{R}$, respectively. We can directly compare the results of some fault tolerance setups with those of the scalability setups. More specifically, the fault tolerance condition of 50% failure leaves the robot team with four failed robots and four functioning robots, which matches the scalability condition of four functioning robots. When we compare the results of these two setups, we see that the HIER and MEAN approaches have a slightly higher bias in the fault tolerance setup with 50% failure compared to that of the matching scalability variant. The VOTE approach demonstrates a more noticeable increase in bias. In contrast, the STIG approach shows no noticeable difference. By increasing the percentage of failed robots from 50% to 75%, the bias of all four approaches substantially decreases. Among all the approaches, the HIER approach shows the least bias under all failure rates. Furthermore, under 75% failure, HIER shows less consistency (noticeably higher spikes) than the STIG approach.

Overall, the STIG approach shows the slightest increase in bias as the rate of failure increases from 0% failure to 75% failure. The other approaches show roughly similar amounts



(a) **Hier:** MSE(λ_r^{app})

(b) **Mean:** MSE(λ_r^{app})

(c) **Vote:** MSE(λ_r^{app})

(d) **Stig:** MSE(λ_r^{app})

Figure 2.5: **Instantaneous bias when testing fault tolerance.** Shaded line plots showing mean, minimum, and maximum of MSE(λ_r^{app}) of all runs, for four variations of arbitrary failures: 0% (0 out of 8), 25% (2 out of 8), 50% (4 out of 8), and 75% (6 out of 8) ground robots failing.

of increase. However, under all failure rates, the HIER approach has a lower bias than the STIG approach. On the other hand, the MEAN and VOTE approaches show more bias than the other two approaches in all the fault tolerance setups. Therefore, the overall bias of the HIER approach is the lowest of all approaches in the fault tolerance setups.

In summary, from the results discussed above, the accuracy of all four approaches in the presence of faulty robots is roughly comparable. Furthermore, the HIER approach has the most negligible overall bias, while the STIG approach shows the most consistency in its tolerance.

Table 2.4: **Fault tolerance.** MSE(ϵ_{rt}) and MSE($\lambda_{rt}^{\text{app}}$) (i.e., cumulative bias) of all runs. In the STIG and HIER approaches, $\epsilon_{rt} = \lambda_{rt}^{\text{app}}$, therefore MSE($\epsilon_{rt}$) is redundant and is omitted from this table.

Control type	# of faulty robots	MSE(ϵ_{rt})	MSE($\lambda_{rt}^{\text{app}}$)
HIER	0	-	0.6563
HIER	2	-	0.8887
HIER	4	-	1.7077
HIER	6	-	3.7488
VOTE	0	3.2392	3.2252
VOTE	2	4.4482	4.4156
VOTE	4	5.4810	5.5005
VOTE	6	6.6035	6.6078
MEAN	0	3.6938	1.8346
MEAN	2	4.8110	2.6813
MEAN	4	5.6990	3.9270
MEAN	6	6.7214	5.5371
STIG	0	-	1.8959
STIG	2	-	2.1223
STIG	4	-	2.9657
STIG	6	-	4.3673

2.7.5 Summary

The results of the experiments can be summarized as follows: (i) the HIER approach shows higher accuracy, more consistency in accuracy, and faster reaction times than the other approaches; (ii) regarding scalability, the VOTE approach has the highest resiliency to group size changes compared to the other approaches while the HIER approach has the highest accuracy under all group sizes; (iii) regarding fault tolerance, the STIG approach demonstrates the least change in accuracy when the failure rate increases, but the HIER approach achieves the highest accuracy under all failure rates.

2.8 Discussion

In terms of performance in static and dynamic environments, the HIER approach outperforms the other approaches. More specifically, in the HIER approach, as the MNS-brain robot follows a zigzag path based on which the MNS can cover the environment as optimally as possible and produces the opinion on the absolute object density for the whole system based on all the ground robots' sensor readings, its perceptual accuracy is higher compared to the other approaches. Regarding the scalability and fault tolerance, the results show that the HIER approach can conduct collective perception without a significant increase in bias, similar to the decentralized approaches, and, in all the setups, the HIER approach has a lower mean bias than other approaches.

The positions of blocks in the experiments are uniformly random. Consequently, depending on the size of the local area the robots observe, which depends on the view area v and speed s of robots and the maximum time window $k_t^{\texttt{max}}$, the robots might underestimate or overestimate the object density in their local area. By increasing one of these parameters, the robots observe a more extensive area locally in a shorter time, allowing them to estimate closer to the true object density of the blocks in the environment.

According to the results of the experiments, among non-stigmergic decentralized approaches presented in this chapter, the MEAN approach outperforms the VOTE approach in terms of opinion accuracy in static and dynamic environments. The results also show that in terms of scalability, the MEAN approach reaches a higher accuracy under all group sizes than the VOTE approach, and they are roughly comparable in terms of the rate of change in accuracy. Furthermore, although the resilience of the VOTE approach to robot failures is higher than the MEAN approach, the MEAN approach has a lower bias in all the fault tolerance setups. All these show that the mean decision model is better for achieving higher perceptual accuracy than the voter model in the collective perception task presented in this chapter.

As depicted in Figure 2.2, in the STIG approach, the accuracy of opinions on apparent density gradually improves from step 0 to step 20000, but after this step, a reduction occurs. Then, after a while, the accuracy gradually increases until step 40000, when it undergoes another reduction. Thus, it is a repeated trend. As already mentioned, the duration of pheromone emission is 20000 steps. After this step, the pheromones placed at the beginning of the experiment evaporate one by one depending on the time step of pheromone placement, and this causes a reduction in opinion accuracy. However, after this reduction, because of discovering new blocks or rediscovering the previously detected blocks, the opinion accuracy once again grows steadily towards the true density for the next 20000 steps. This process is repeated over time.

In all the dynamic setups, in the STIG approach, the robots cannot perceive the changes fast enough. More specifically, it takes a while for the robots to correctly perceive the true density of the environment in the slow setups, and they can never correctly estimate the densities in the fast setups. The reason is behind the concept of pheromone placement and the dependency of this approach on that. When the amount of pheromone is low in the arena, the opinions on apparent density are closer to 0. However, as time passes, the number of pheromone resources increases, which results in more accurate estimates since the robots sense more pheromones around themselves per step, affecting their opinions. Therefore, in all the dynamic setups, when the density of blocks changes and the pheromones evaporate more frequently, the robots do not have enough time to discover most of the blocks such that their opinions become accurate. The situation gets worse in the case of fast changes.

Regarding the impact of faulty robots on the accuracy of collective opinions, the accuracy drops for all four approaches by increasing the number of faulty robots. However, the amount of this reduction is the least for the STIG approach. One of the reasons behind this behavior of the STIG approach is the pheromone evaporation rate used in the experiments. If we reduce the length of the activation duration, we expect that the rate of drop in accuracy will increase for this approach. Of course, in all the setups of fault tolerance, the accuracy of the HIER approach is higher than the other approaches.

In the scalability experiments, by increasing the number of ground robots, the accuracy of the HIER approach improves according to Table 2.3. The reason behind this is that by increasing the number of ground robots, the view area of the MNS becomes larger. As the objects' distribution is not entirely uniform in the environment, this bigger local area improves the accuracy of this approach. For example, if we increase the number of ground robots to 48, the view width of the MNS becomes 6 meters, equal to the length of the arena used in the experiments. In such a case, the MNS only needs one pass to sweep the entire arena. With an average speed of 7.5 cm/s, the MNS can sweep a lane with a length of 6 m in 2000 steps.

Although, according to the literature, the voter decision model is relatively accurate in discrete best-of-n decision-making scenarios (e.g., choosing one option among several available options), when dealing with high-variance samples of a continuous stochastic field, it has low accuracy. More specifically, in the latter case, the voter decision model only shuffles biased opinions among group members. Therefore, its performance is roughly the same as that of a single robot. As can be seen in the tables, the MSE(ϵ_{rt}) and MSE($\lambda_{rt}^{\text{app}}$) of the STIG approach are similar to each other, which confirms that the collective bias in robot opinions after the voter decision model process is similar to that in individual inferences.

Finally, tuning parameters are known to be challenging to design regardless of using simulated or real robots. In this chapter, parameter P is tuned during an initial manual testing phase, and it is still possible to be optimized in every approach. However, with this optimization, only the mean bias of the opinions improves, not the variance. It should also be acknowledged that optimization of P requires prior knowledge of the environment. For example, if we tuned P to a low-density environment for one approach, the accuracy would presumably improve only under low-density environments. In that case, the approach will not display its best potential performance in high-density environments.

2.8.1 Future work

Future work on collective perception should study the effects of various conditions on the performance of the approaches, such as 1) the shape of the arena (e.g., non-convex areas,

mazes); 2) different sizes and shapes of obstacles; and 3) features of the stochastic field (e.g., a field in which the density in some regions is significantly higher than other regions).

Performance optimization could also be investigated in collective perception. Online adaptation methods could be helpful in an unknown environment as the robot team can adapt to the environment and optimize performance. In practice, all the approaches can benefit from adaptation techniques, but it would be easier to design such techniques in the Hier approach than in the fully decentralized approaches.

Although it might be simple to design advanced behaviors such as online adaptivity in the MNS framework because of the aspects of centralization that are integrated into self-organization, we did not implement them in the Hier approach. The motivation of this study was to make a fair comparison by limiting the capabilities of the Hier approach to be more similar to those of the fully decentralized approaches. For example, in the fault tolerance setups, the online adaptivity could be easily used in the Hier approach to enable the parent robots to detect faulty robots and adjust the indirect field of view of the MNS and improve its performance. So, the addition of even more advanced behaviors such as self-awareness can be considered in the future.

Future work could also investigate other types of failures, such as motion control errors (e.g., robots get stuck in some regions of the arena), odometry errors (e.g., robots underestimate or overestimate the distance they have travelled so far), other errors (e.g., robots believe they are continuously detecting objects, robots produce random numbers as their inferences), or complete failure (i.e., robots cannot move, cannot communicate with their peers and cannot sense any object).

2.9 Conclusion

In this chapter, we tested the performance of an MNS-enabled approach called Hier in a collective perception task where a team of robots should collectively perceive the density of objects randomly distributed in an unknown environment. We empirically showed that the Hier approach is more accurate, more consistent, and faster than fully decentralized benchmark approaches. We also experimentally showed that the Hier approach does not suffer from scalability and fault tolerance disadvantages noticeably more than decentralized benchmark approaches. The results verify that the MNS-enabled approach can successfully combine aspects of centralized and decentralized approaches. In other words, it can improve accuracy and maintain scalability and fault tolerance features, compared to decentralized approaches.

Chapter 3

Centralization vs. decentralization in multi-robot coverage: Ground robots under UAV supervision

This chapter considers the problem of multi-robot coverage. In this chapter, we presented for the first time a comprehensive comparison of control structures in multi-robot systems, which includes comparison of speed, accuracy, efficiency, scalability, fault tolerance, and energy consumption.

3.1 Introduction

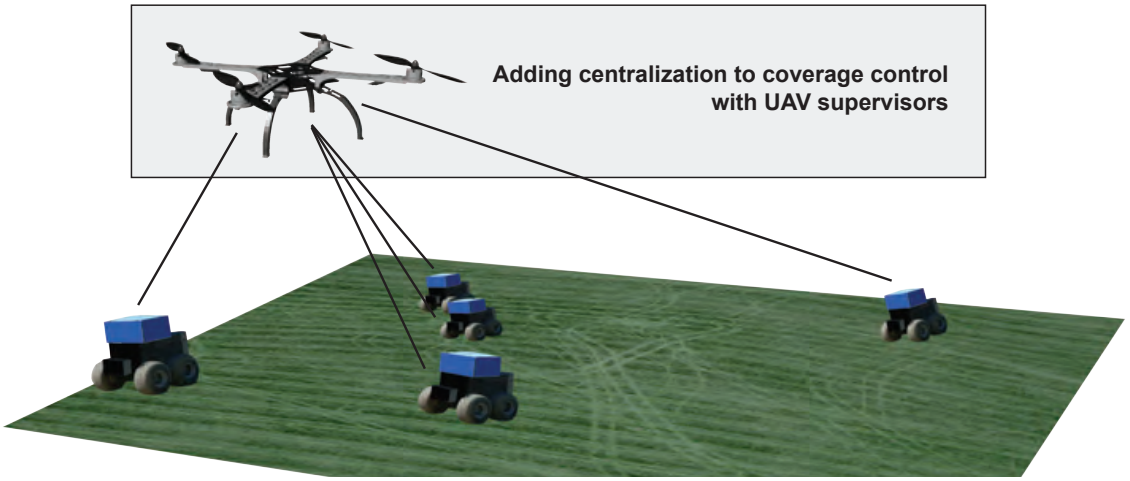
In multi-robot systems, a fundamental design decision is whether to use centralized control, decentralized control, or some combination of the two. Fully centralized control approaches are often high performing and efficient, but can suffer from single points of failure and poor scalability, due to limitations such as communication bottlenecks. Fully decentralized approaches do not have these drawbacks, due to features such as redundancy and parallelization, but may have lower speed and efficiency. The relative advantages and disadvantages of centralized and decentralized control are somehow well-accepted common knowledge, but the precise trade-offs involved when considering specific tasks have not been fully characterized. Some research has studied the relationship between controller structure and controller performance—e.g., in network topology [93] or controller architectures [67]—but these studies do not cover the full set of issues that can arise during robot deployment, such as fault tolerance of group performance after arbitrary robot failures, or physical interference when robots collide.

In this chapter, we take a step towards comprehensive comparison between centralized and decentralized control in multi-robot systems, by studying the performance of these configurations in a sweep coverage task, i.e., when an environment must be uniformly swept. We start by using a group of ground robots and then we compare it to the same group, but enlarged with unmanned aerial vehicle (UAV) supervisors, with progressively increased centralization.[1] In all methods, the sweep coverage task is completed exclusively by the ground robots. The UAVs are strictly supervisors; they observe the environment and send motion instructions to ground robots. To make a fair comparison between methods, we only change the sensing and communication capabilities of the UAV supervisors, not of the ground robots. In this chapter we consider multi-robot sweep coverage that would take place in large outdoor environments (e.g., search and rescue, environment monitoring, or precision agriculture), and setup our simulation experiments accordingly. We use UAV supervisors to provide the ground robots with a mobile hub for communication, control, and position tracking. A similar approach could also be applicable to small indoor environments, using external infrastructure such as cameras and local area networks to add varying degrees of centralization.

We consider (de)centralization in terms of two key categories: control structure and access to information. The control structure includes the topology of the communication network, the distribution of control and decision-making roles, and the processes by which the network is formed and the control roles are allocated. Access to information includes the availability of information about the environment (size, shape, and locations of boundaries) and each robot

[1] All the experiments discussed in this chapter have been run in simulation using the multi-robot physics-based simulation environment ARGoS [99].

(position and orientation) and the sensing process by which the information is obtained. We compare four example control approaches, ranging from fully reactive and decentralized to fully predetermined and centralized, and evaluate their performance in simulated experiments.



	Fully Decentralized (randomized)	**Hybrid Formation** (dynamic hierarchicy)	**Centralized Formation** (central broadcast)	**Predetermined** (*a priori* knowledge)
Speed	- - -	+++	+++	++++
Short-term accuracy	+	-	-	- -
Long-term accuracy	- -	+++	+++	++++
Efficiency	- - -	+	+	++
Unknown obstacles	-	++	+++	++++
Single point of failure	++++	++++	- - -	- - -
Communication bottleneck	++++	++++	- - -	- - -
Fault-tolerance	++	++	++	- -
Scalability of performance	++	++	++	++++
Large environments	+	- -	-	- - -
Energy expenditure	-	-	-	-

Figure 3.1: **Summary of the key findings of this chapter.** Relative advantages and disadvantages of adding centralization to coverage control, according to our experimental and results and analysis. Worse performance is demarcated by more minus signs (-) and better performance is demarcated by more plus signs (+). For instance, " - - - " indicates very poor performance, " + + + " indicates very good performance, " + + + + " indicates excellent performance, and " + " indicates performance that is mediocre but still relatively better than other approaches. The last two rows (large environments and energy consumption) depend on the use of UAVs.

We define perfect sweep coverage performance as the uniform and complete exploration of an environment, and therefore evaluate the performance of the four control approaches using two metrics: coverage uniformity and coverage completeness. We define a scalable system as one without a communication bottleneck, in which communication overhead increases linearly with the number of robots at a slow rate, interference between robots does not decrease task performance, and task performance increases linearly or better with the number of robots. We therefore evaluate scalability in terms of bottlenecks, inter-robot interference, communication, and the impact of scaling on coverage performance. We define a fault-tolerant system as one without a single point of failure, in which arbitrary robot failures do not disrupt connectivity

between the remaining robots, and do not cause task performance to decrease worse than linearly. We therefore evaluate fault tolerance in terms of connectivity and performance after robot failures.

We also evaluate the limitations associated with increasing centralization by adding mobile external infrastructure in the form of UAVs. When adding UAV supervisors, energy consumption is a key metric to assess their added value, because UAVs and ground vehicles usually have different rates of energy consumption, and therefore different operating times and travel distances. We compare the coverage performance of the four example control methods under the limitations of maximum operating time and maximum distance traveled (which is proportional to the amount of used energy). We also compare them under the limitation of a maximum energy consumption budget for the whole multi-robot system. We conduct this analysis according to the listed specifications of a number of real ground robots and UAVs that are available off-the-shelf and suitable for experimentation in multi-robot systems (see the Appendix).

One would expect more centralized and predetermined control to perform well in terms of speed, accuracy, and efficiency, and not so well in terms of fault tolerance and scalability. For more reactive and decentralized control, one would expect the opposite. We would therefore anticipate that well-designed hybrid control could potentially achieve better task performance than fully reactive and decentralized control, and better fault tolerance and scalability than fully predetermined and centralized control. One would also expect approaches without UAV supervisors to perform better in terms of energy consumption. However, our experimental results and analysis indicate that the relative advantages and disadvantages of (de)centralization are much more nuanced than the assumed simple trade-offs (see Fig. 3.1).

For instance, our results show that the accuracy gap between centralized and hybrid control is quite small, suggesting that features such as a central coordinating entity do not necessarily deliver a substantial performance boost. Our results further suggest that the somewhat better efficiency of the predetermined approach is due to its *a priori* knowledge of the environment, rather than its central coordinating entity. As another example, one might expect decentralized approaches to be better suited to cluttered environments. However, our results indicate that more centralized approaches perform quite well with randomized obstacles, outperforming fully decentralized control. In terms of scalability and fault tolerance, our results show that hybrid control can have similar advantages to fully decentralized control. Some of the existing hybrid approaches (e.g., using a dynamically elected leader) still suffer from setbacks such as a bottleneck when scaling. It is therefore interesting to note that in the hybrid approach tested in this chapter, communication scales linearly and no bottlenecks occur. Finally, in terms of energy consumption, one might expect the energy efficiency of UAV supervisors to be the most relevant limitation. However, our analysis indicates that the

limited operating time of UAVs is more restrictive than their energy efficiency, and inefficient ground robots pose more of a problem than inefficient UAVs. Overall, our results contradict some expectations and progress towards a better understanding of the trade-offs between centralization and decentralization in multi-robot systems.

3.2 Related work

Ideally, when designing a multi-robot system, one would be able to guarantee that a selected control approach is the top performer for its objectives. Tools have been proposed in swarm robotics to assess general performance [59] and specific aspects of performance (e.g., [125]), and in self-organizing software and embedded systems to assess general performance [46, 47, 69] and the degree of self-organization [122]. Although these contributions progress towards general metrics for decentralization, they are not applicable to centralized multi-robot systems. Therefore, the choice to use centralized or decentralized control is often made intuitively and most of the time is discipline dependent (e.g., decentralized control in the swarm robotics community).

In this chapter, we present for the first time a comprehensive comparison of control structures in multi-robot systems, which includes comparison of speed, accuracy, efficiency, scalability, fault tolerance, and energy consumption. As mentioned above, we are not aware of any published attempt to formalize comprehensive and systematic metrics to assess both centralized and decentralized control, which would enable their direct comparison.

Whether centralization or decentralization is better for a given application will evidently depend on the scenario. Decentralization can address limitations such as unavailability of global communication in search and rescue [5], single points of failure in object transport [123], and poor fault tolerance in task allocation [74]. However, downsides to decentralization are unavoidable—for instance, in collective decision making, accurate decisions tend to have long convergence times [125]. Recently proposed methods could enable a multi-robot system to switch between centralized and decentralized control on demand (e.g., mergeable nervous systems [84, 140]), which is crucial for the future of swarm robotics [42]. To be useful, autonomous reorganization would need to be based on a systematic understanding of the relationship between task performance and control structure.

In this chapter, we study the task of sweep coverage as an example case, and compare different control structures. This chapter builds upon our previous work, in which we proposed mergeable nervous systems (MNS) [84, 140] to be an appropriate hybrid control approach for multi-robot coverage [65].

3.2.1 Coverage control

Multi-robot sweep coverage targets the systematic, uniform observation of an environment. It is part of environment monitoring, and any other application in which more uniform observation would improve performance. For instance, in adaptive sensor networks, more efficient coverage in the exploration phase could improve the positions chosen by mobile sensors [80, 107, 115]. In robot swarms and multi-robot systems, more efficient observation of the environment could improve performance during, e.g., task allocation [64], collective decision making [118, 125], collective perception [111], search and rescue [11], or foraging [78]. In short, developing better solutions to coverage could improve performance in many other tasks.

3.2.1.1 Centralized approaches

When using a single robot, the goal of coverage control is to quickly and efficiently collect information that comprehensively represents an environment [53, 68], for instance by sweeping the environment using boustrophedon (i.e., "back-and-forth") motion [5, 8]. To speed up the process, multiple robots can be centrally organized into a formation such as a line [79] and sweep the environment together, or be assigned to different zones for single robots to sweep individually (e.g., [103, 109]). When robots sweep individually, the environment is often decomposed into zones using centralized control. Coverage is then executed using offline or online path planning, depending on whether the environment is decomposed *a priori* or not [5]. In offline decomposition, results can be improved through multi-robot path planning strategies, e.g., using particle swarm optimization [120], genetic algorithms [92], or graph-based heuristics [137]). In this chapter, for centralized and predetermined coverage, we use *a priori* decomposition and offline path planning.

In online decomposition, aspects of decentralization are often used, in combination with robots building a shared reference map of the environment [5]. The map is built either before coverage begins [88], or during coverage using broadcasting [57, 87]. Shared reference maps are also used without strict decomposition into zones, with maps updated by single-broadcast [83] or somewhat less efficiently by multi-broadcast [2]. All of the approaches using shared reference maps are mostly decentralized, but their centralized aspects are sufficient to hinder scalability, and perhaps also fault tolerance. In these hybrid approaches, efficient performance and system scalability are not necessarily balanced, and the trade-offs between centralization and decentralization are not studied directly.

3.2.1.2 Decentralized approaches

Fully decentralized coverage can be completed using independent randomized motion with simple obstacle and collision avoidance [61, 63, 85], but the approach is highly inefficient. Many studies have improved upon random walk strategies to increase overall coverage performance, e.g., by optimizing step lengths in Brownian motion and Lévy flight [97], improving the scalability of Lévy walk [72], or proposing a new style of random walk [96]. These approaches can achieve very high coverage completeness in unknown environments (up to 97% reported, [97]), if the available time is long enough and the swarm is large enough [63]. Randomized motion is also expected to eventually reach high coverage completeness in environments cluttered with obstacles [5]. However, inefficiency is very high, as robots often repeat coverage of areas already explored by their peers [61, 141]. In other words, there is a presumed trade-off in decentralized approaches between coverage efficiency and environment difficulty [5], which has not yet been precisely characterized. In this chapter, for basic decentralized coverage, we use simple diffusive motion based on random billiards [19]—where each agent has constant velocity until reflecting off a boundary in a random direction—combined with obstacle avoidance.

One fault-tolerant approach to improving coordination and efficiency in decentralized coverage is to leave ant-inspired artificial pheromones in the environment [75]. However, both randomized motion and pheromone-based approaches have shown poor scalability in terms of speedup [61, 75]. As the swarm scales, new robots provide diminishing returns—i.e., each added robot provides increasingly less speedup in overall coverage. In [75], adding a sixth robot to a swarm of five, for instance, barely reduces the total coverage time. Parameters of pheromone-based coverage approaches have been tuned for efficiency when using with Lévy flight [112] or a combination of Lévy flight and Brownian motion [35], but it is not clear if scalability in terms of speedup has been improved. Currently there are no studies directly comparing the coverage performance of pheromone-based approaches to simpler randomized motion approaches.

Some decentralized approaches include aspects of centralization by leaving robots in the environment to act as static beacons, either permanently before coverage begins [81], or temporarily during coverage [116]. In the approach of [116], some robots temporarily park themselves while the swarm explores as a group. In a 5 x 5 analysis grid in a 15 m x 15 m environment with obstacles, the approach achieves mean coverage completeness of 99.7%, and also demonstrate scalability. Repeated coverage and fault tolerance are not directly measured. The approach of [116] is comparable to the formation approaches in this chapter, in that some robots act as supervisors while others perform coverage by staying together in a group.

3.2.1.3 Formation approaches

If robots sweep the environment as a group, their coordination can be improved by using formation control strategies [79] to manage motion control and relative positions. Formation control is often cited as a good approach for coverage applications such as search and rescue [14, 79], but the coverage performance of the various strategies has not been directly compared. Of the common formation shape types cf. [14]), line formations are considered suitable for sweeping and mapping tasks [79]. In the two formation approaches considered in this chapter, we use a line formation. We offset the robot positions slightly, into a zigzag line (see Fig. 3.3, to better avoid inter-robot collisions during obstacle avoidance.

Formation control can be fully centralized, or can incorporate aspects of decentralized control, such as a potential field for local collision avoidance [79]. In centralized approaches, robots share a common reference via broadcast, for instance a predetermined leader [132] or a dynamically selected navigator and virtual leader [40]. In this chapter, we use basic leader-follower formation control with a predetermined leader for our centralized formation approach. Formation control can also be accomplished without centralized communication, by instead using a self-organized ad-hoc communication network [140]. In this chapter, for our hybrid formation approach, we use a self-organized network and hierarchical control, based on mergeable nervous systems [84, 140] and hierarchical frameworks [139].

3.3 Methods

We investigate the advantages and limitations associated with varying degrees of (de)centralization in a group of ground robots that may be supervised by one or more UAVs. We assess four control approaches spanning from reactive and fully decentralized control (i.e., all robots are acting autonomously) to predetermined and fully centralized control (i.e., all robots are given fully predetermined instructions by a central entity). In this section, we describe the methods for our experiments. First, we define the sweep coverage task. Then, we present the implementation details for the four coverage approaches: fully decentralized, hybrid formation, centralized formation, and predetermined. Finally, we describe the details of the simulation setup.

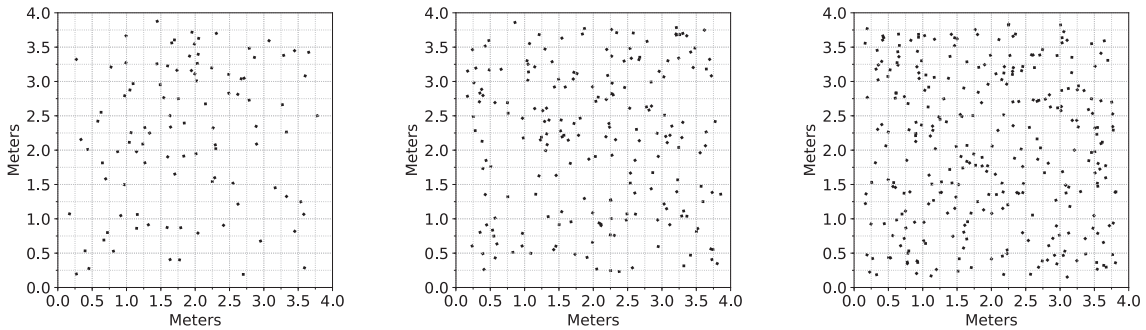
3.3.1 The sweep coverage task:

We define the sweep coverage task as uniform and complete exploration of the environment. If an environment is partitioned, the ground robots should collectively visit all portions and should spend equal time visiting each portion. In this work, the experiment arena is a 4×4 m^2

enclosed square, overlaid with a 16×16 grid. For complete coverage, each of the 256 grid cells must be visited by the ground robots. In our sweep coverage scenario, we have the following restrictions:

- The ground robots used are not equipped with camera, so they cannot see their surrounding environment.
- The ground robots do not know their position in the environment.
- The number of grid cells of the environment is very bigger than the size of robot team—e.g., we use 9 ground robots in the default experiment setup while the environment consists of 256 grid cells.
- The UAVs cannot localize themselves nor can they detect the obstacles with their camera.

The difficulty of this task is increased by adding small $4 \times 4 \times 2$ cm^3 obstacles to the environment (see Fig. 3.2). In each run of an experiment, the obstacle positions and orientations are defined randomly with uniform distribution. We test arenas with the following three obstacle difficulties: 100 obstacles for low difficulty (i.e., 1% of the environment surface occupied), 200 obstacles for medium difficulty, and 300 obstacles for high difficulty.



(a) 100 obstacles (b) 200 obstacles (c) 300 obstacles

Figure 3.2: **Obstacle size, density, and distribution in the simulated environment.** 2D plots of the environment, including the analysis grid and examples of randomized obstacle distributions: (a) low difficulty, 100 obstacles; (b) medium difficulty, 200 obstacles; (c) high difficulty, 300 obstacles. The obstacles, arena, and analysis grid are shown at their correct relative sizes.

3.3.2 Coverage methods

We compare the following four types of coverage control, from most decentralized to most centralized (see Fig. 3.3).

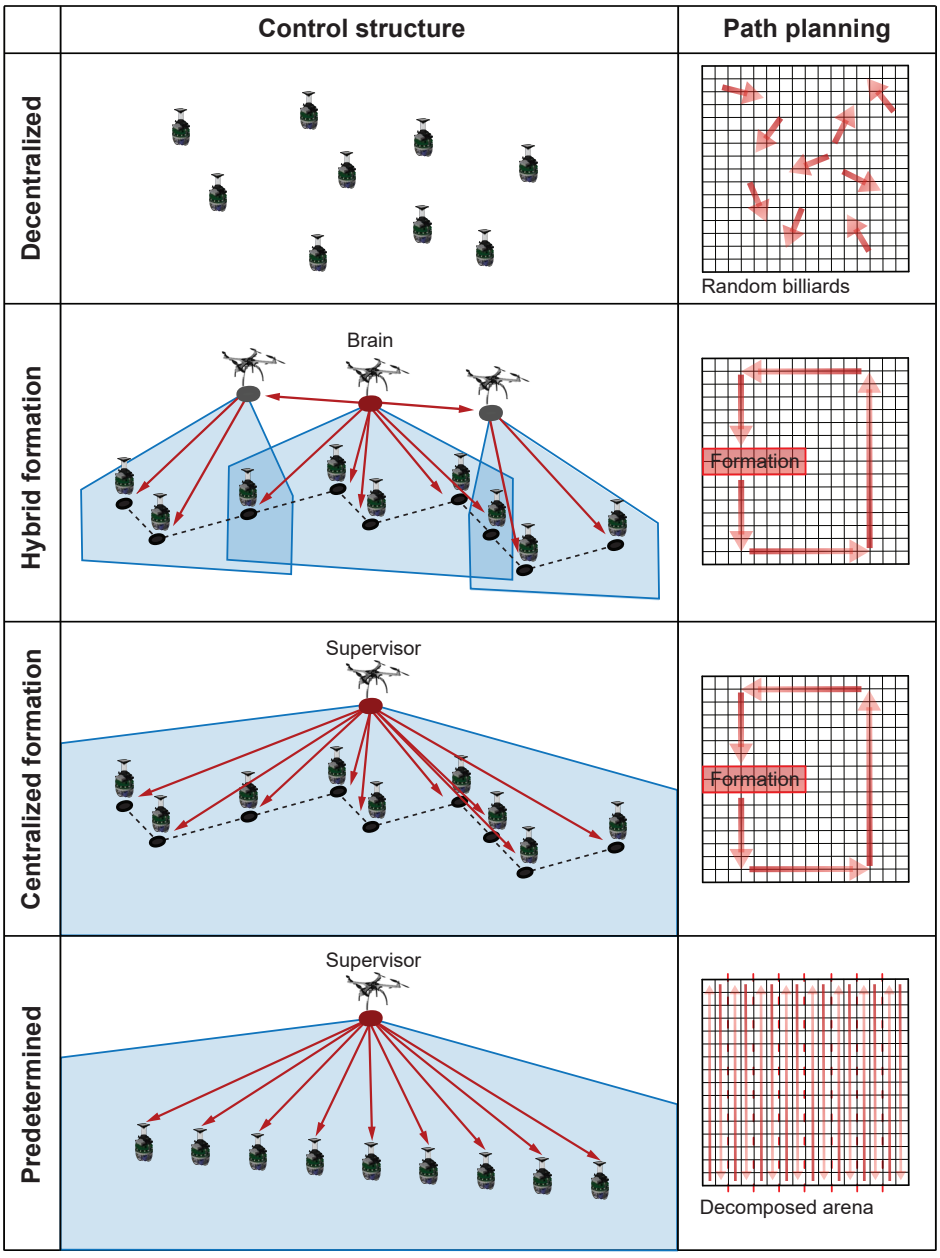


Figure 3.3: The four control approaches. **Decentralized:** Ground robots explore the environment independently via randomized motion control, without any UAV supervisors. **Hybrid formation:** UAVs give motion instructions to ground robots in their limited fields of view, using a self-organized hierarchical communication network. **Centralized formation:** One UAV with unlimited field of view gives motion instructions to all ground robots using a predetermined centralized communication network. **Predetermined:** After the environment is decomposed and paths are planned offline, one UAV gives ground robots the predetermined motion instructions, using a predetermined centralized communication network.

1. **Decentralized.** There are no UAV supervisors. The ground robots are fully independent; there is no explicit communication and each robot controls itself. The robots can locally detect objects in their environment. The motion control strategy is basic

diffusion with obstacle avoidance.

2. **Hybrid formation control using mergeable nervous systems.** UAVs are limited to local onboard sensing, through which they can observe the positions and orientations of ground robots and of environment boundaries, within a certain range. UAVs have a limited communication range and send motion control instructions to ground robots using a self-organized ad-hoc communication network (for details, see the mergeable nervous systems (MNS) approach described below). Motion control is based on a formation shape designed for sweeping.

3. **Centralized formation control.** The single UAV has unlimited access to all ground robots' positions and orientations and can sense the environment boundaries. The UAV has an unlimited communication range and directly sends motion control instructions to all ground robots over a predetermined star-graph communication network. Motion control is based on the same formation shape as in the hybrid formation approach.

4. **Predetermined control based on** *a priori* **knowledge.** The UAV calculates instructions for the ground robots before the task begins, based on full *a priori* knowledge of the environment and the robots. The UAV has the same unlimited communication range and predetermined network topology as in the centralized formation approach. For motion control, the UAV decomposes the environment into zones and calculates motion trajectories offline, and then gives each ground robot its predetermined motion trajectory to sweep its predetermined zone.

In all approaches, the ground robots execute reactive obstacle avoidance using onboard sensing. The details of ground robot and UAV control for the four approaches are given below.

3.3.2.1 Decentralized

We have chosen a random walk as the representative of fully decentralized control because of the constraints of our sweep coverage scenario. More specifically, existing decentralized approaches in which robots park themselves permanently in the environment are not applicable to our coverage scenario, because we study sweep coverage. Similarly, robots parking themselves temporarily would worsen performance, because we study sweep coverage that should be uniform. Furthermore, as in our setup, ground robots cannot self-localize or distinguish each other from obstacles, we cannot use existing approaches in which robots make formations or networks through line-of-sight communication, or directly communicate their location to their peers to avoid repeated coverage. The ground robots are capable of implicit communication, so pheromone-based approaches could be applicable to our scenario.

However, artificial pheromones are difficult to implement in practice, so we chose not to use them. Therefore, random walk strategies are the best options in the literature for uniform sweep coverage in our setup. It has been shown that random billiards, which is equivalent to ballistic motion, is very better than Brownian motion, correlated random walk, Levy walk, and Levy taxis which have been reported as fully random, specialized in coverage task and/or specialized in exploration task in the literature (refer to ballistic motion results in [70]). In ballistic motion the same as random billiards, the robots move forward with a constant speed and randomly change their direction when they detect an obstacle. In our comparison in this chapter, for fully decentralized coverage, we use motion control based on random billiards [19] and reactive obstacle avoidance. Each ground robot independently follows a constant velocity unless it reflects off a boundary in a random direction, or unless it turns to avoid an obstacle. The robots' onboard sensing can distinguish a boundary from an obstacle or robot. When avoiding an obstacle, a robot turns in place until the obstacle is no longer near its heading. The pseudocode for boundary reflection and obstacle avoidance is given in Algorithm 2. The starting positions and headings of the robots are defined randomly with uniform distribution.

Algorithm 2 Fully decentralized: ground robot.

```
if an object is detected within 60° of the heading then
    if the object is an environment boundary, not an obstacle then
        turn to a random direction facing away from the boundary;
    else
        if the obstacle is located in the left side then
            turn right with an average angular velocity of 1.36 rad/s;
        else
            turn left with an average angular velocity of 1.36 rad/s;
        end if
    end if
else
    move forward with an average linear velocity of 6.8 cm/s;
end if
```

3.3.2.2 Hybrid formation control using Mergeable Nervous Systems (MNS)

As already mentioned, in a previous work, we proposed 'mergeable nervous systems' (MNS) [84, 140] to be an appropriate hybrid formation control approach for multi-robot coverage [65]. Based on the results of that work, we have chosen the MNS as the representative of hybrid formation control in our comparison in this chapter. In MNS formation control, a heterogeneous swarm can self-organize into a dynamic ad-hoc communication network with a hierarchical structure (for full implementation details, see [140]). Using this hierarchy, robots report sensing events and cede their autonomy to a temporary 'brain' robot (see Fig. 3.3). The brain robot determines its own motion trajectory. As the brain moves, it acts as a motion reference for its children (i.e., the brain acts as a reference coordinate frame used to calculate

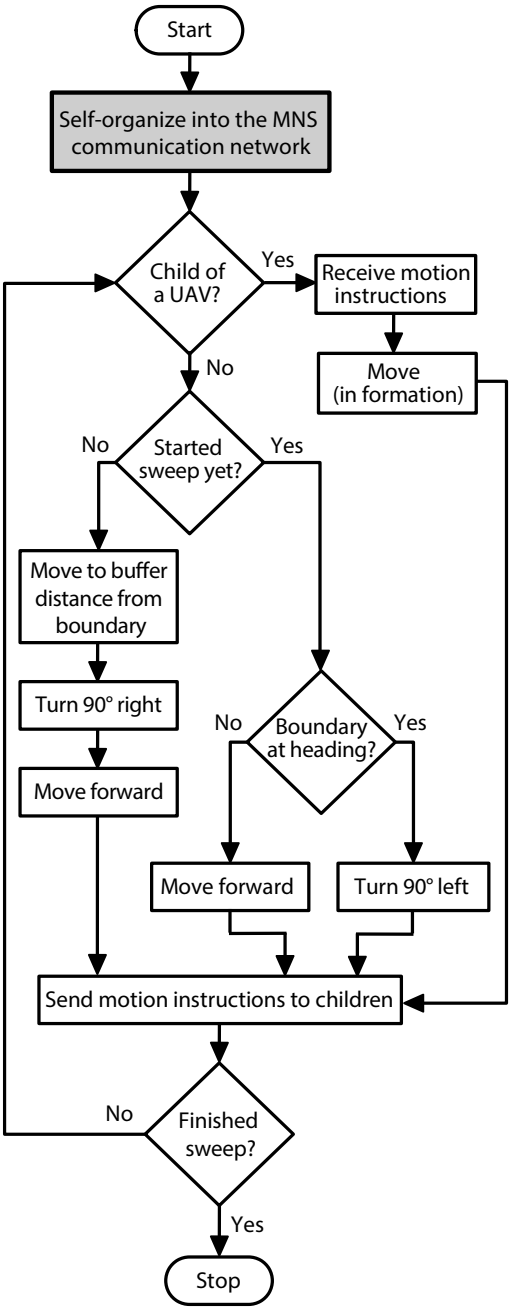
the motion instructions sent to its children), which in turn act as motion references for their children. The robots in an MNS can self-reconfigure into a new communication network and control hierarchy on the fly, for instance, in case of task change or brain failure [140].

Algorithm 3 Hybrid formation (with MNS) and centralized formation methods: ground robot.

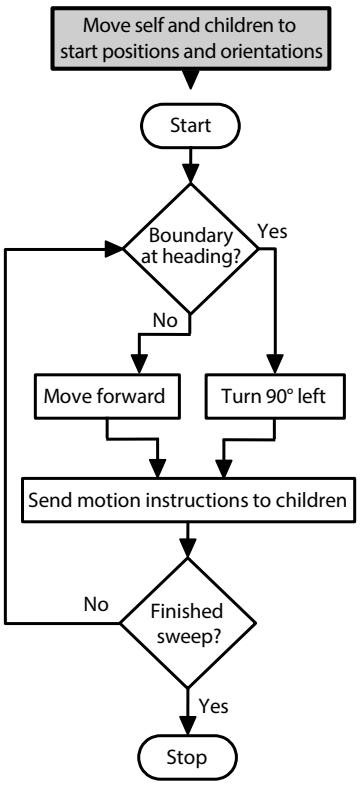
```
if an obstacle is detected within 60° of the heading then
    if the obstacle is located in the left side then
        turn right with an average angular velocity of 1.36 rad/s;
    else
        turn left with an average angular velocity of 1.36 rad/s;
    end if
else
    follow motion instructions received from a UAV;
end if
```

In our MNS formation control setup, there are some targets guiding the self-organization of the network. The target communication network topology is a caterpillar tree—i.e., a tree in which all inner nodes are on one central path, to which each leaf node is connected. Ground robots try to become leaf nodes in the communication network. They complete the coverage task and perform independent obstacle avoidance. The pseudocode for the ground robots is given in Algorithm 3. UAV supervisors try to become inner nodes or the brain node and act as motion references for their children. The UAVs direct their ground robot children into a line-like formation, which is ideal for sweeping an environment. The target line formation has a slight zigzag, to reduce inter-robot collisions when ground robots perform obstacle avoidance. The UAV that becomes the brain (see [140]) uses a reactive perimeter-following behavior to sweep the environment counterclockwise, turning when it encounters a boundary. A behavior flowchart for all UAVs in the MNS hybrid formation is given in Figure 3.4a.

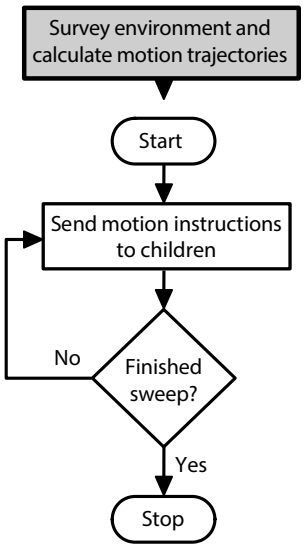
In each experiment run, the starting positions and headings of the ground robots and UAVs are defined randomly with uniform distribution. When the experiment begins, the robots self-organize into an MNS, according to the approach defined in [140]. Once the MNS is formed, the brain begins its reactive sweeping behavior. The brain UAV moves at a constant speed, irrespective of the speed of the other robots. The formation is maintained because the other robots adjust their motion according to the motion of their parents. The perimeter-following behavior of the brain is deterministic, therefore each sweep of the environment by the robots takes the same amount of time. Given the dimensions of the line formation and the arena, this simple counter-clockwise path is sufficient to enable full coverage.



(a) **Hybrid formation**: UAVs (both the brain UAV and non-brain UAVs).



(b) **Centralized formation**: UAV (supervisor).



(c) **Predetermined control**: UAV (supervisor).

Figure 3.4: Flowcharts for UAV behaviors in the MNS hybrid formation, centralized formation, and predetermined methods.

3.3.2.3 Centralized formation control

In order to conduct a fair comparison between hybrid and centralized controls, we have considered a centralized formation approach for the sweep coverage task which shares as much similarities as possible with the MNS approach. More specifically, for centralized formation control, we use a basic leader–follower approach [132], in which all robots follow one global leader that broadcasts information. At the start of each experiment run, the starting positions and headings of the ground robots are defined randomly with uniform distribution. At initiation, they are already connected to a UAV hub in a star (i.e., spoke and hub) communication network. The UAV acts as a single coordinating entity, providing all ground robots with motion instructions. The ground robots perform independent obstacle avoidance (see pseudocode in Algorithm 3), but otherwise follow the UAV's instructions. At initiation, the UAV immediately directs the ground robots into the target formation—a line with a slight zigzag, identical to the MNS formation—as shown in Fig. 3.3. Then, the UAV follows the same reactive perimeter-following behavior used by the brain in the MNS formation, sweeping the environment counterclockwise. A behavior flowchart for the UAV is given in Figure 3.4b.

3.3.2.4 Predetermined control based on *a priori* knowledge

We have considered a centralized sweep coverage approach with a priori knowledge of the environment as an achievable bound of the performance for the sweep coverage task. By definition, such an approach has more information about the environment, so it is expected to perform better. For this approach, we decompose the environment offline into zones for robots to sweep individually (e.g., [103, 109]). As in the centralized formation approach, ground robots receive instructions from one UAV over a predetermined central communication network. Unlike in the centralized formation approach, the UAV does not use reactive control to update ground robots' motion instructions in real time—rather, the UAV uses *a priori* knowledge to decompose the environment and calculate predetermined motion trajectories for the ground robots before they begin the task. The motion trajectories for the ground robots are based on boustrophedon (i.e., "back-and-forth") motion (cf. [8]) to sweep their zones. Based on this trajectory, the ground robots are also given a predetermined path to circumnavigate obstacles (see pseudocode in Algorithm 4). The UAV supervisor in this approach has access to all information about the environment (shape, size, and boundaries) and robots (number of robots, positions, orientations), and at initialization is connected to all ground robots as the hub of a star network (the same as the centralized formation approach). The UAV decomposes the environment into lanes that are suitable for the number of ground robots, boustrophedon motion style, and the dimensions of the environment and the analysis grid (see Fig. 3.3). To sweep every cell of the analysis grid in its lane, a ground robot

only needs to change direction once. At initialization, the starting positions and headings of the ground robots are defined randomly with uniform distribution. The UAV, which hovers in a stationary position during the experiment, immediately gives the ground robots predetermined trajectories to drive to the starting positions of their lanes and sweep them.

Algorithm 4 Predetermined control: ground robot.

```
if an obstacle is detected within 60° of the heading then
    follow a predetermined counterclockwise half-circle trajectory around the obstacle;
else
    follow motion instructions received from the UAV;
end if
```

3.3.3 Simulation setup

The experiments are conducted in the ARGoS simulator [99], with robot models implemented using a plugin for prototyping new robots [3, 4]. All four approaches complete the coverage task with nine ground robots. In addition, the predetermined control and centralized formation control approaches each have one UAV supervisor, whereas the MNS hybrid formation control approach has three UAV supervisors, one of which is the MNS 'brain' (see Fig. 3.3). The simulated ground robot is based on the e-puck ground robot [89]. It uses differential wheeled drive, with an average speed of 6.8 cm/sec, and is equipped with a ring of 12 short-range proximity sensors. The ground robots can detect obstacles, other ground robots, and the boundaries of the environment at a distance of 5 cm. The simulated UAV, which is based on the S-drone quadcopter [94], is a quadrotor restricted to a maximum speed of 7.4 cm/sec (such that the ground robots are able to follow the UAVs at their average speed), and is equipped with a downward-facing camera. Obstacles cannot be detected by the UAVs, only by the ground robots. Both simulated ground robots and UAVs are represented by simple 2.5 cm radius cylinders.

Fiducial markers encoding unique IDs sit atop the ground robots. The UAVs detect these markers, tracking the relative positions and orientations of the ground robots in order to give them motion instructions. UAVs can establish a communication link with ground robots that lie within their field of view. In centralized formation control and predetermined control, the UAV can view the full arena and communicate with all ground robots. In MNS hybrid formation control, the UAVs fly at a constant altitude and each UAV can view ground robots in an approx. 1.5×1.75 m^2 rectangular ground area (for details, see [140]). Three UAVs are required in order to keep all ground robots in their collective view when in a line formation. In the MNS, two UAVs can establish a communication link when they can see the same ground robot (for details, see [140]).

At the start of each experiment, the $4 \times 4 \times 2$ cm^3 obstacles are distributed randomly

throughout the 4 m^2 arena, with a 15 cm buffer to the outer boundary and a 15 cm buffer between separated obstacles. The buffers ensure that it is always possible for the ground robots to sweep the environment without becoming stuck. Also at the start, the robots are randomly distributed in a 1.0×1.25 m^2 rectangular area against the perimeter of the arena.

For the MNS hybrid formation, centralized formation, and predetermined control approaches, we define a *round* as one complete sweep of the environment. These experiments are terminated at the completion of the round. A round takes the same amount of time in the hybrid and centralized formation approaches, because the reactive controller that the UAVs use for perimeter following is deterministic. The decentralized approach does not perform a comparable sweep, therefore we define its round end to be at the same time step as the round ends of the hybrid and centralized approaches, to enable direct comparisons between the approaches. In the decentralized approach experiments, we continue to collect data after the round end, terminating the experiments at time step 11000.

3.4 Results

To assess the impact of (de)centralization on coverage control, we run experiments that test the coverage performance of all four approaches: fully decentralized, MNS hybrid formation control, centralized formation control, and predetermined control. Existing coverage approaches with high performance incorporate some degree of centralization, and most have not been tested for fault tolerance and scalability. We run experiments testing the fault tolerance and scalability of the MNS hybrid formation control approach, and conduct analysis to compare these results to the fault tolerance and scalability of the other approaches.

To assess the costs and limitations associated with adding UAV supervisors, we analyze the performance results of the four approaches according to energy consumption. We base our analysis on the energy consumption and efficiency of state-of-the-art mobile robots and drones. (See the Appendix for tables reporting the specifications of eleven autonomous ground vehicles and five UAVs currently available off-the-shelf.)

3.4.1 Performance

In high-performing and efficient multi-robot coverage, robots cover a high percentage of the environment with a low rate of repeated coverage. We assess the performance of the four approaches in terms of coverage completeness (i.e., the percentage of grid cells visited) and coverage uniformity (i.e., the variability of time that robots collectively spend visiting each cell). We also assess the speed of the approaches, defined as the ratio of coverage

completeness to the time required to reach that completeness. We complete 50 runs for each control approach, in each of the three obstacle setups (100, 200, and 300 obstacles), for a total of 600 runs.

3.4.1.1 Coverage completeness

At the end of one round, the predetermined control approach outperforms all other approaches in terms of coverage completeness, for all obstacle difficulties (see Table 3.1 and Fig. 3.6). More in general, the higher the degree of centralization, the higher the coverage completeness. The predetermined control approach outperforms the centralized formation and hybrid formation approaches by a relatively small margin, compared to the decentralized approach. For example, in the setup with 100 obstacles, the centralized formation and hybrid formation approaches only omit an additional 1.1% and 2.5% of the arena, respectively, while the fully decentralized approach misses an additional 18.2%.

As the number of obstacles increases, the difference in average coverage completeness between the predetermined control approach and the other approaches grows larger (see Table 3.2). All four approaches are impeded by obstacles, and all four suffer a reduction in average coverage completeness as the number of obstacles increases. The lower the degree of centralization, the greater the suffered reduction.

The predetermined control approach is faster at achieving higher coverage completeness than the centralized formation and hybrid formation approaches (see Fig. 3.5). However, as the number of obstacles increases, the speed advantage of the predetermined control approach decreases. In the setup with 100 obstacles, the predetermined control approach is approximately 1000 time steps faster (see Fig. 3.5-a), but in the setup with 300 obstacles, it is only approx. 280 time steps faster (see Fig. 3.5-b). If the decentralized approach continues past the completion of one round, it eventually reaches a coverage completeness comparable to that of the other approaches (see Fig. 3.5), but it takes more than twice the amount of time to reach that completeness. So, the fully decentralized approach is the worst approach in terms of speed.

3.4.1.2 Coverage uniformity

We define coverage uniformity as a measure of the variability of cell visits. The best performing coverage approach would be one without any repeated coverage—there would be no variability in cell visits, so uniformity would be 0. For each run, $v_i \in \mathbf{v}$ is defined as the total time spent by all robots in cell i. A median of a vector of sample points is the closest point to all the sample points. In order to calculate the uniformity, we first calculate the sum of

Table 3.1: Average coverage completeness (%) and coverage uniformity (p) achieved in one *round* (50 runs each). The best completeness would be the highest percentage (i.e., 100%) and the best uniformity would be the lowest variability (i.e., $p = 0$, see Eq. 3.1).

	Decentralized	**Hybrid formation**	**Centralized formation**	**Predetermined**
100 obstacles	Completeness : 81.3% Uniformity (p) : 9.56	Comp. : 97.0% Unif. : 8.29	Comp. : 98.4% Unif. : 7.93	Comp. : 99.5% Unif. : 4.95
200 obstacles	Completeness : 75.5% Uniformity (p) : 9.99	Comp. : 95.7% Unif. : 8.47	Comp. : 97.4% Unif. : 8.12	Comp. : 99.1% Unif. : 6.17
300 obstacles	Completeness : 70.1% Uniformity (p) : 10.23	Comp. : 93.1% Unif. : 8.73	Comp. : 95.8% Unif. : 8.37	Comp. : 98.6% Unif. : 6.87
Performance differences compared to the **Predetermined** approach				
100 obstacles	Completeness : -18.2% Uniformity (p) : 4.61	Comp. : -2.5% Unif. : 3.34	Comp. : -1.1% Unif. : 2.98	
200 obstacles	Completeness : -23.6% Uniformity (p) : 3.82	Comp. : -3.4% Unif. : 2.30	Comp. : -1.7% Unif. : 1.95	
300 obstacles	Completeness : -28.5% Uniformity (p) : 3.36	Comp. : -5.5% Unif. : 1.86	Comp. : -2.8% Unif. : 1.50	

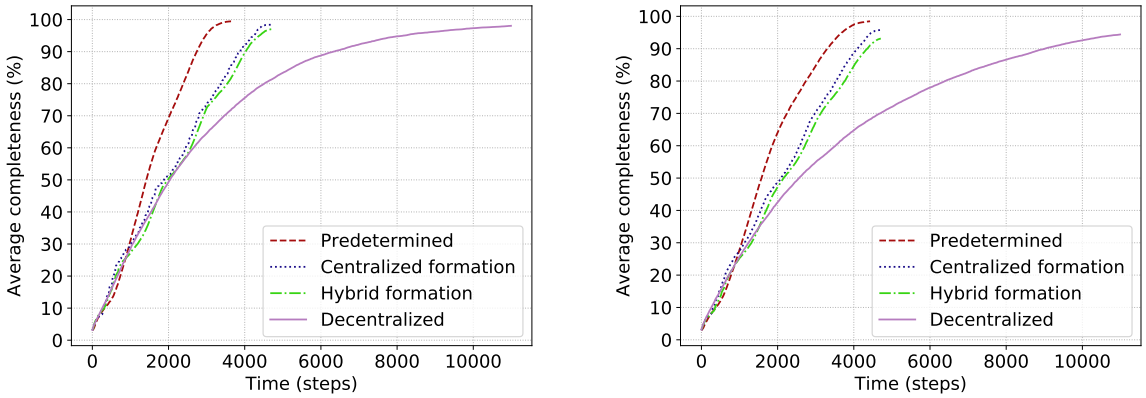
Table 3.2: Average change in coverage completeness and coverage uniformity achieved during one *round* (50 runs each), when increasing the number of obstacles by 100.

	Decentralized	**Hybrid formation**	**Centralized formation**	**Predetermined**
Change in **Completeness**	- 5.6%	- 2.0%	- 1.3%	- 0.5%
Change in **Uniformity (p)**	+ 0.34	+ 0.22	+ 0.22	+ 0.96

the square root of the distance between each element of $\mathbf{v}$ (i.e., v_i for i = 1, $\cdots$, 256) and the median of $\mathbf{v}$ to find out how far $\mathbf{v}$ is from its nearest uniform status (i.e., where v_i for i = 1, $\cdots$, 256, are set to the median of $\mathbf{v}$). We use the square root of the distances to scale down the uniformity values. Then, we divide the result by number of grid cells (i.e., 256) to have a smaller absolute value representing the uniformity, without changing the order of the uniformity values for different runs. The following equation gives the coverage uniformity p described above:

$$p = \frac{\sum_{i=1}^{c} \sqrt{|v_i - M(\mathbf{v})|}}{c}, \tag{3.1}$$

where $M(\mathbf{v})$ is the median of $\mathbf{v}$, and c is the number of cells. The smaller the value of p, the more uniformity between cells; the most uniform case is $p = 0$.



(a) Low difficulty: 100 obstacles

(b) High difficulty: 300 obstacles

Figure 3.5: Average coverage completeness of each approach over time (50 runs each), according to environment difficulty (i.e., number of obstacles). The predetermined control, centralized formation, and hybrid formation experiments end at the completion of one *round*. The decentralized experiments end at 11000 time steps.

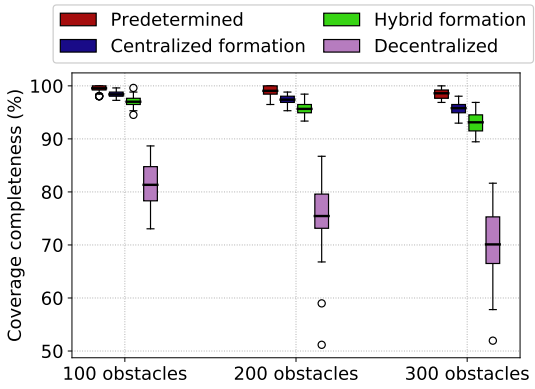
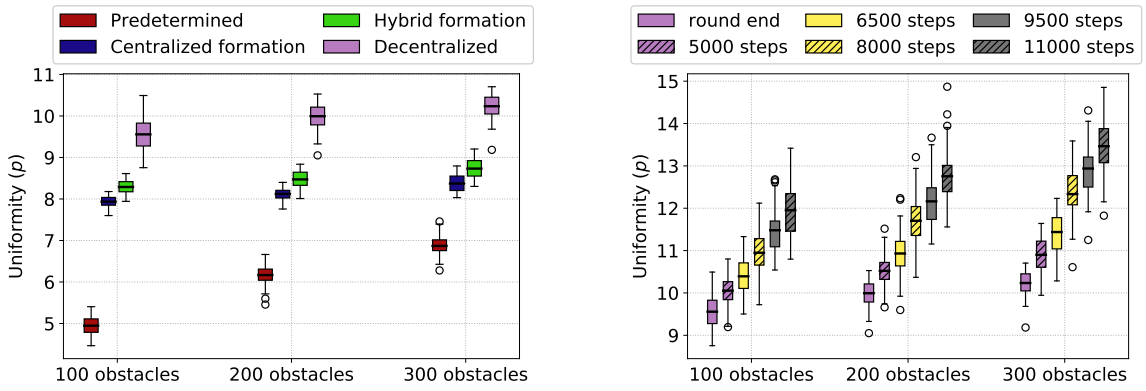


Figure 3.6: Coverage completeness achieved by each approach during one *round* (50 runs each).

At the end of one round, the predetermined control approach has a better coverage uniformity (p) than all other approaches, for all obstacle difficulties (see Table 3.1). The higher the degree of centralization, the better the coverage uniformity. However, as the obstacle difficulty increases, the coverage uniformity of the predetermined control approach worsens at a faster rate than the other approaches—approximately twice as fast as the fully decentralized approach and four times as fast as the formation approaches (see Table 3.2 and Fig. 3.7-a). In the 300 obstacle setup, the coverage uniformity of the centralized and hybrid formation approaches are worse than the predetermined control approach by margins of only 1.5 and 1.66, respectively (see Table 3.1).

Similar to coverage completeness, the fully decentralized approach has substantially worse uniformity than the other approaches (see Fig. 3.7-a). However, unlike coverage completeness,



(a) All approaches

(b) Decentralized approach, over time

Figure 3.7: Coverage uniformity of each approach (50 runs each), according to environment difficulty (i.e., number of obstacles). The lowest p value is the most uniform case, and therefore the most efficient. (a) Uniformity of all approaches at the end of one *round*. (b) Uniformity of the decentralized approach over time (where the earliest time step is the end of one *round*).

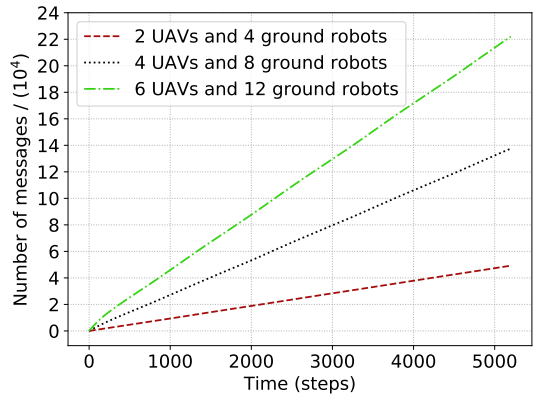
the uniformity of the decentralized approach does not improve if it continues past the end of a round (see Fig. 3.7-b). For instance, in the 300 obstacle setup, if the fully decentralized approach continues long enough to achieve coverage completeness that is comparable to the other approaches, its uniformity becomes worse by an additional margin of approximately 3.4.

In all approaches, at the end of one round, there is low variability of coverage uniformity between runs (see Fig. 3.7-a). The variability in the formation approaches is slightly lower than in the other two. However, after round completion, the variability of uniformity in the fully decentralized approach grows substantially (see Fig. 3.7-b).

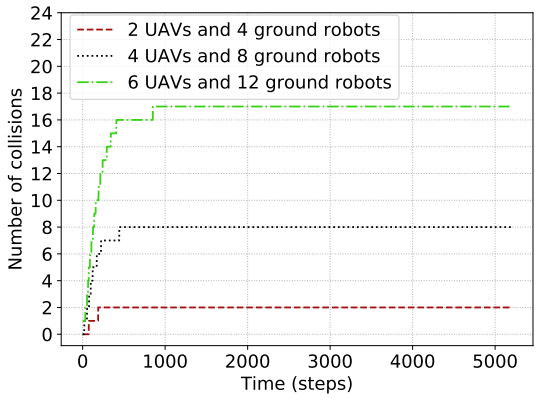
3.4.2 Fault tolerance

We assess the fault tolerance of coverage control in terms of the impact that robot failures have on coverage completeness. For the MNS hybrid formation control approach, we test this experimentally. For the other approaches, we conduct analysis to assess the impact of failures.

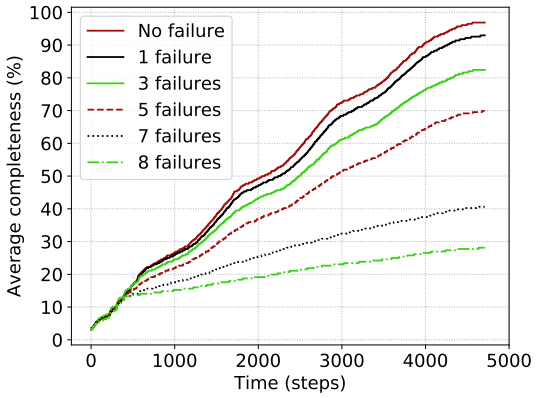
A common vulnerability in multi-robot systems is the presence of single points of failure. The fully decentralized approach is not susceptible to this problem, because it does not have a central coordinating entity. By contrast, a failure of the UAV in either the predetermined or centralized formation approaches results in a system-wide failure—neither approach can autonomously replace the failed UAV in order to recover. The MNS hybrid formation approach



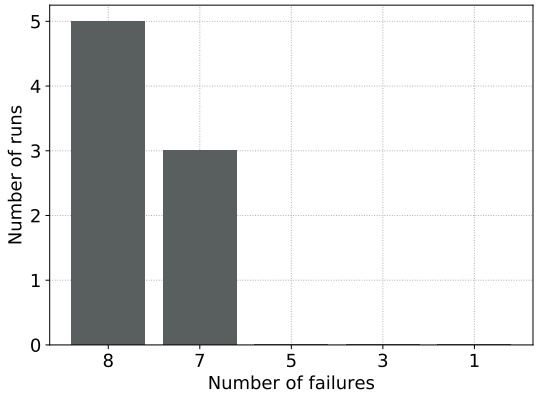
(a) Number of messages exchanged



(b) Number of robot–robot collisions



(c) Average coverage completeness



(d) Number of runs with disconnections (in 10 runs)

Figure 3.8: **Fault tolerance and scalability in the hybrid formation control approach.** (a) Scalability according to number of messages exchanged over time. (b) Scalability according to number of inter-robot collisions over time. (c) Average overage completeness over time, with varying number of ground robots. (d) Fault tolerance in terms of the number of runs with a permanent disconnection between the brain UAV and another UAV, according to number of ground robot failures.

also has a central coordinating entity—however, if the robot fulfilling this role fails, then another robot can substitute it on the fly, because the role is defined dynamically through self-organization. Previous work has already shown the fault tolerance of MNS formation control in terms of replacing a failed robot (even the brain) and reconfiguring the formation accordingly [140]. In short, the predetermined control and centralized formation control approaches suffer from a single point of failure, while the fully decentralized and MNS hybrid formation control approaches do not.

Another key possibility is failure of robots that are responsible for the task and that cannot be replaced during runtime. Here, we test the impact on coverage completeness when failures of ground robots reduce the size of the system. In the MNS hybrid formation control approach, performance depends in part on maintaining the ad-hoc communication network.

Table 3.3: Fault tolerance of performance: comparative analysis between the four methods. Estimated average coverage completeness achieved at the end of one round, if failure occurs near the start of the round (at time step 400).

	Estimate for **Decentralized**	**Hybrid formation**	Estimate for **Centralized formation**	Estimate for **Fully centralized**
No failure	Total : 82.0% Per robot : 9.1%	T. : 96.9% P. : 10.8%	T. : 98.4% P. : 10.9%	T. : 99.6% P. : 11.1%
3 failures	Total : 65% Per robot : 11%	T. : 82.1% P. : 13.7%	T. : 83% P. : 14%	T. : 77% P. : 13%
5 failures	Total : 52% Per robot : 14%	T. : 69.5% P. : 17.4%	T. : 71% P. : 18%	T. : 55% P. : 14%
7 failures	Total : 36% Per robot : 18%	T. : 41.0% P. : 20.5%	T. : 46% P. : 23%	T. : 32% P. : 16%

Table 3.4: Scalability of performance: comparative analysis between the four methods. Estimated performance speedup from the addition of one ground robot, in terms of the average coverage completeness per 1000 time steps achieved during one round.

	Estimate for **Decentralized**	**Hybrid formation**	Estimate for **Centralized formation**	Estimate for **Fully centralized**
4^{th} robot	+ 1.8% (per 1000 steps)	+ 2.7%	+ 2.8%	+ 3.6%
6^{th} robot	+ 1.5% (per 1000 steps)	+ 1.4%	+ 1.5%	+ 3.6%
8^{th} robot	+ 1.3% (per 1000 steps)	+ 1.3%	+ 1.3%	+ 3.6%

We therefore evaluate the impact of failures on both coverage completeness and connectivity. We run the fault tolerance experiments in the setup with 100 obstacles and allow the MNS hybrid formation control approach to complete one round. At time step 400, we impose failure on either one, three, five, seven, or eight out of the nine total ground robots. We complete 10 runs for each number of failing robots.

The MNS hybrid formation results show that the performance drop per robot failure is fairly consistent for 1, 3, or 5 failures—coverage completeness drops at a rate of approximately 5% per robot failure (see Fig. 3.8-c). When there are more than 5 failures, the performance drop per robot failure approximately doubles (see Fig. 3.8-c). This pattern of performance occurs because the brain UAV is capable of maintaining connectivity with its children UAVs whenever there are 5 or fewer failures (i.e., 56% or less of ground robots fail), but sometimes experiences disconnections when there are more failures (see Fig. 3.8-d). When there are 7 or 8 failures, the brain UAV might additionally lose contact with some of the remaining ground robots, causing the observed increase in performance drop.

We compare the MNS hybrid formation results to the other approaches using analysis (see Table 3.3), based on the assumption that no approach is permitted to adapt its coverage strategy during runtime (e.g., the formation approaches cannot adapt their path planning and the predetermined control approach cannot adapt its environment decomposition). As

Table 3.5: Scalability of communication: comparative analysis between the four methods. Maximum messages per time step for the robot with the greatest communication load.

	Decentralized	**Hybrid formation**	**Centralized formation**	**Fully centralized**
Maximum messages	0	10	1 per ground robot	1 per ground robot

in the MNS hybrid formation approach, the fully decentralized and centralized formation approaches both include some inter-robot redundancy and interference, so we calculate the impact of failures based on the results of the MNS approach, taking into account the respective differences in coverage uniformity.[2] By contrast, robots are confined to their own lanes in the predetermined control approach, so any variability in uniformity is due to self-redundant coverage. We calculate the impact of failures by simply subtracting the coverage achieved by the failed robots after time step 400. The predetermined control approach experiences the largest drop in overall coverage completeness due to robot failures (see Table 3.3), because the remaining robots have no opportunity to cover the zones assigned to the failed robots. In the other approaches, overall performance drops more slowly with fewer failures and more quickly with more failures. This change appears in the decentralized approach because robots have a higher chance of redundancy in a larger group, and it appears in the formation approaches because the redundancy between two robots depends on their positions in the formation. The decentralized approach experiences smaller performance drops overall, because it has more inter-robot redundancy. However, the decentralized approach never outperforms the formation approaches at the end of a round (see Table 3.3), because in the decentralized approach a robot has more self-redundant coverage. The centralized formation approach maintains a slight advantage over the MNS hybrid formation, up to 5 failures. With more failures, the centralized formation approach more substantially outperforms the MNS hybrid formation approach (see Table 3.3), because at these failure rates the MNS formation may lose contact with the remaining ground robots.

3.4.3 Scalability

We assess scalability in terms of the number of messages exchanged between the robots, the number of inter-robot collisions, and the performance speedup achieved by the addition of one robot. We test the MNS hybrid formation approach experimentally and conduct analysis to compare to the other approaches.

We assess the performance speedup associated with adding one ground robot, regardless

[2]For each failure rate, we scale the difference in performance recorded in the MNS hybrid formation experiments for fault tolerance, according to the difference between the MNS hybrid formation approach and the compared approach in terms of average absolute deviation, based on p.

of the number of UAV supervisors. For the MNS hybrid formation approach, we base our analysis on the data obtained in the fault tolerance experiments (i.e., in the 100 obstacle setup) after time step 400. Similar to the fault tolerance analysis, for the fully decentralized and centralized formation approaches we calculate the performance speedup based on the results of the MNS approach, taking into account the respective differences in coverage uniformity.[3] We calculate performance speedup for the predetermined control approach based on the assumption that before runtime the supervisor UAV calculates the environment decomposition according to the number of ground robots. In the predetermined control approach, the speedup in coverage completeness per time step is the same for each additional robot (see Table 3.4). In the predetermined control approach, the overall coverage completeness does not increase as the system scales—rather, the time required to complete a round decreases. In the other approaches, the speedup provided by one additional robot gets smaller as the system grows in size (see Table 3.4). Overall, the speedups in the fully decentralized approach are the lowest, because its inter-robot redundancy is highest and because its overall coverage completeness at the end of a round is lowest.

To test the scalability of communication and inter-robot collisions in the MNS hybrid formation approach, we run experiments without obstacles in the environment, with the following three heterogeneous system sizes: 1) two UAVs and four ground robots, 2) four UAVs and eight ground robots, and 3) six UAVs and twelve ground robots (see Chapter 2, section 2.5 for explanation of group sizes). We complete 10 runs per system size. For each system size, we keep the same type of communication topology and formation shape (i.e., caterpillar tree communication network, and linear formation with a slight zigzag).

In the MNS hybrid formation approach, the total number of messages exchanged is expected to scale linearly, because the robots communicate only with those they are connected to in the network. The experimental results confirm this expectation (see Fig. 3.8-a): at time step 5 000, the total messages passed is approximately 5×10^4 in the 6-robot system, 13.5×10^4 in the 12-robot system, and 22×10^4 in the 18-robot system. In the MNS hybrid formation approach, in our implementation, the maximum number of messages passed by one robot in one time step is 10, no matter the size of the system. Just as in the fully decentralized approach, no communication bottleneck will appear in larger systems in the MNS formation approach (see Table 3.5). By contrast, in the centralized formation and predetermined control approaches, the maximum number of messages passed by one robot in one time step is dependent on the number of ground robots (i.e., the supervisor UAV passes 1 message per ground robot). Therefore, the scalability limit of these approaches depends on the maximum communication load of the UAV.

[3]For each number of ground robots, we scale the difference in performance recorded after time step 400 in the MNS hybrid formation experiments for fault tolerance, according to the difference between the MNS hybrid formation approach and the compared approach in terms of average absolute deviation, based on p.

In all four approaches, inter-robot collisions can occur due to obstacles in the environment. Here, we only assess interference between robots in terms of collisions caused by the control approach. In the MNS formation, it is expected that inter-robot collisions will only occur during the formation establishment phase, which is confirmed by the experiment results (see Fig. 3.8-b). In the fully decentralized approach, inter-robot collisions are not considered to be interference, because changes in direction help the robots cover the environment. In both the centralized formation and predetermined control approaches, robots start the experiment by moving to their target positions before proceeding with coverage. However, in these approaches, it is possible for the supervisor UAV to calculate paths for all robots that prevent collision.

3.5 Analysis of energy consumption

When UAV supervisors are added to a group of ground robots, any improvements in performance need to be assessed in the context of the new limitations they introduce to the system. We identify two primary limitations introduced by the addition of UAV supervisors:

1. **reduced operating time** for the system during a single battery charge, due to the quicker energy exhaustion of the UAVs, and
2. **increased total energy consumption** if UAVs are added to the system, and the number of ground robots remains unchanged.

Under these limitations, we determine the highest performing approach in terms of coverage completeness. We also compare the performance of the MNS hybrid formation and decentralized approaches even if they are not the highest performers, because they are more comparable in terms of scalability and fault tolerance, in that they do not have a single point of failure or a communication bottleneck.

First, we analyze coverage completeness under the constraint of energy exhaustion. Although the state of the art includes a few robots that charge themselves continuously (e.g., using solar power), the majority require their batteries to be recharged separately from operation. We consider the coverage completeness that would be possible before energy exhaustion, with a single charge of both ground robots and UAVs (cf. off-the-shelf robots listed in the Appendix). Second, we analyze coverage completeness under the constraint of total energy consumption. UAVs and ground robots can have substantially different rates of energy consumption, so we consider the coverage completeness that would be possible for certain energy budgets and energy consumption ratios (cf. off-the-shelf robots listed in the Appendix).

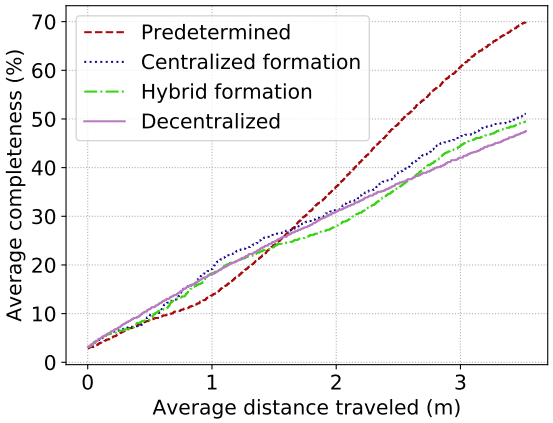


Figure 3.9: Average coverage completeness, according to the average distance that ground robots have collectively traveled in one square meter of the environment (all obstacle setups, all runs). The relative performance rankings do not change after 3 m traveled per sqm.

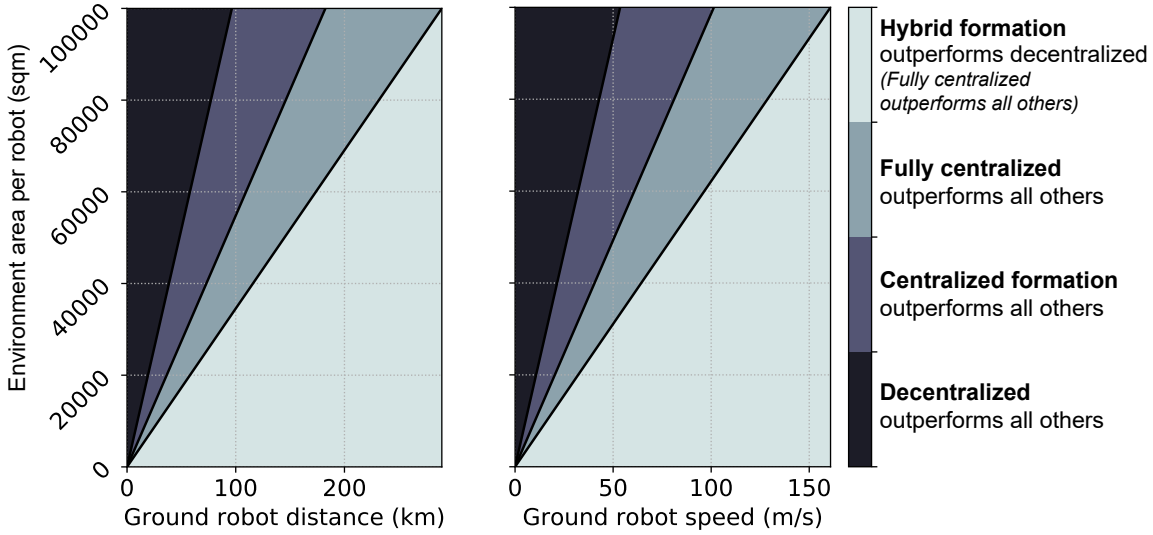
We base our analysis calculations on the coverage completeness results obtained in Sec. 3.4.1.1 (i.e., from experiments with nine ground robots in a square 4×4 m^2 arena with obstacles, with three UAVs for the MNS hybrid formation approach, and one UAV for both the centralized formation and predetermined control approaches).

3.5.1 Energy exhaustion

We analyze the coverage completeness achieved by each method under the constraint of energy exhaustion (i.e., limited operation). Although the experiment results for coverage completeness show that the predetermined control approach is the highest performing overall, the other methods outperform it in the initial 1 000 time steps (see Fig. 3.5). We analyze the robot, UAV, and environment constraints under which a system would be limited to those initial performance results.

To analyze the constraint of robot distance traveled, Fig. 3.9 gives the average coverage completeness according to the total meters traveled by all ground robots, in one square meter of the environment. Fig. 3.9 shows that the decentralized method is the top performer until 0.87 m traveled per sqm, then the centralized formation method is the top performer until 1.64 m traveled per sqm, after which the predetermined control approach is the top performer. Fig. 3.9 also shows that the decentralized method outperforms the MNS method until 2.61 m traveled per sqm.

The point at which a robot's energy is exhausted during a coverage task will depend on the power and efficiency of the robots, and on the size of the environment. These conditions can vary greatly, so it is relevant to better characterize performance according to energy exhaustion. To assess the limit imposed by ground robot exhaustion, we calculate the average



(a) Limited by ground robot exhaustion (b) Limited by UAV exhaustion

Figure 3.10: **Performance shifts at energy exhaustion thresholds.** Analysis of relative coverage completeness achieved by each approach when limited by one battery charge for (a) ground robots and (b) UAVs. The shaded areas indicate relative performance at the given environment size, according to ground robot limitations of speed and distance. From darkest to lightest, the shaded areas indicate: 1) the fully decentralized approach is the top performer; 2) the centralized formation control approach is the top performer; 3) the predetermined control approach is the top performer; and 4) the predetermined control approach is still the top performer, but the MNS hybrid formation control approach outperforms the fully decentralized approach. Results also given in Table 3.6.

coverage completeness depending on the maximum distance a ground robot can travel in one charge. To assess the limit imposed by typical UAV exhaustion (i.e., 30 minutes in one charge), we also calculate the average coverage completeness depending on the maximum speed of the ground robot. Because the maximum speed of the UAV will always be higher, it is not a limitation. We consider environment sizes of up to 100 000 square meters per ground robot. Such an environment is large enough to assess the performance shifts for ground robots with a maximum travel distance from 1.5 km to 100 km per robot per charge and a maximum speed from 0.1 m/s to 50 m/s (under the condition of 30 minutes of UAV flight time). These ranges are representative of ground robots available off-the-shelf (and the flight time is representative of the average UAV), as listed in the Appendix.

In general, the analysis results indicate that more centralization performs better if maximum distances or maximum speeds are higher, while more decentralization performs better if operating areas are larger (see Fig. 3.10). For indoor environments, even in the most limited robots available off-the-shelf (see the Appendix), the environment size in which the decentralized approach would be the best performer is somewhat large (see Table 3.6). For example, for e-puck2 or Thymio II robots designed for indoor research, for a group of nine

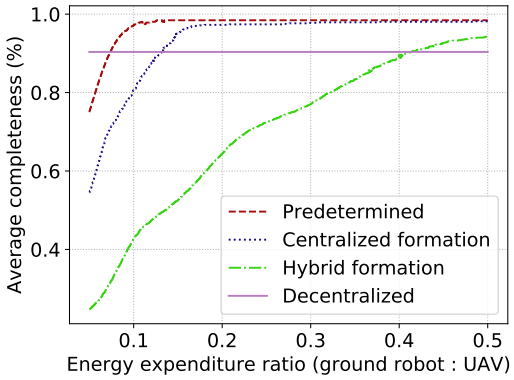
Table 3.6: Analysis of relative coverage completeness achieved by each approach in one battery charge. Environment sizes that result in a certain relative performance, according to maximum ground robot distances per charge, as currently available in off-the-shelf robots, or according to maximum ground robot speeds, as currently available in off-the-shelf robots. Results also given in Fig. 3.10.

Ground robot **distance** per charge, or max. **speed**	Area per robot (sqm)			
	Top performer: **Decentralized**	Top performer: **Centralized formation**	Top performer: **Fully centralized**	**Hybrid form.** outperforms decentralized
2 km	> 2,000	1,000–2,000	< 1,000	< 800
10 km	> 10,000	6,000–10,000	< 6,000	< 4,000
25 km	> 30,000	15,000–30,000	< 15,000	< 10,000
50 km	> 60,000	30,000–60,000	< 30,000	< 20,000
0.15 m/s	> 300	150–300	< 150	< 100
0.5 m/s	> 1,000	500–1,000	< 500	< 350
1.0 m/s	> 2,000	1,000–2,000	< 1,000	< 700
10.0 m/s	> 20,000	10,000–20,000	< 10,000	< 7,000

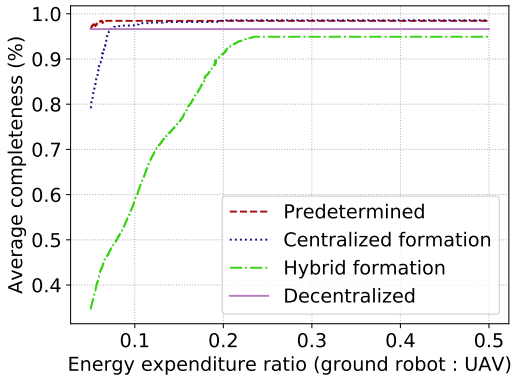
ground robots, the decentralized approach would only be the top performer in environments larger than 50×50 m^2. However, for applications such as outdoor search and rescue, the environment could easily be large enough to make the decentralized approach the top performer (see Table 3.6). For example, for a group of nine Boston Dynamics Spot robots, the decentralized approach will be the top performer in any environment larger than 135×135 m^2.

3.5.2 Energy consumption: ratio of ground robots to UAVs

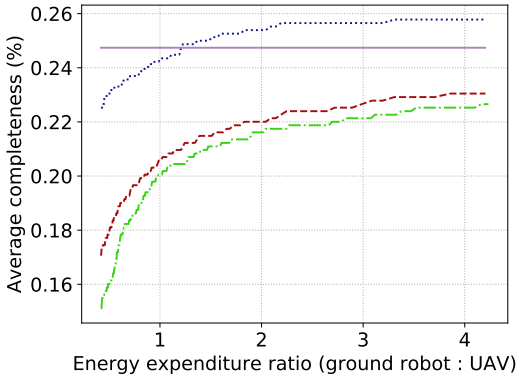
In some coverage applications, battery life could be increased or autonomous recharging could be implemented. The energy efficiency of ground robots might also be significantly different from the efficiency of UAVs. To characterize the impact of energy efficiency on performance, we analyze the coverage completeness achieved by each approach under an overall energy budget, regardless of battery life. Basing our analysis on the results obtained in Sec. 3.4.1.1, we assign the system a total energy consumption budget of either 200 or 300 kilojoules. The budget is shared among all ground robots and UAVs used in that method (i.e., nine ground robots for all methods, three UAVs for MNS hybrid formation, one UAV for both centralized formation and predetermined control, and no UAVs for decentralized). We consider ground robots with energy consumption rates from 3 joules/s to 70 j/s, and UAVs with energy consumption rates from 6 j/s to 60 j/s. These ranges represent robots available off-the-shelf (see the Appendix).



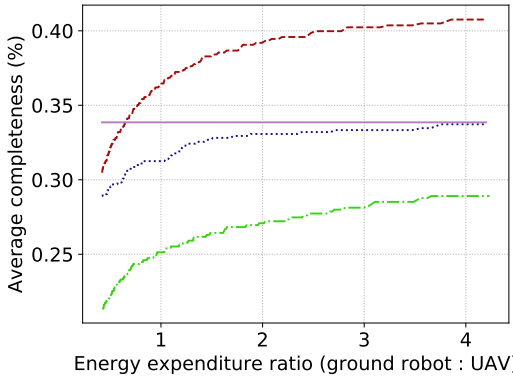
(a) **3 j/s** ground robot consumption; **200 kj** budget



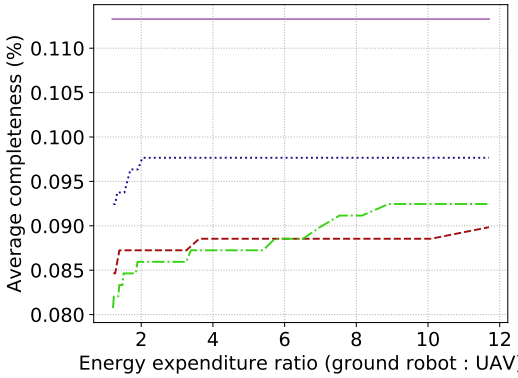
(b) **3 j/s** ground robot consumption; **300 kj** budget



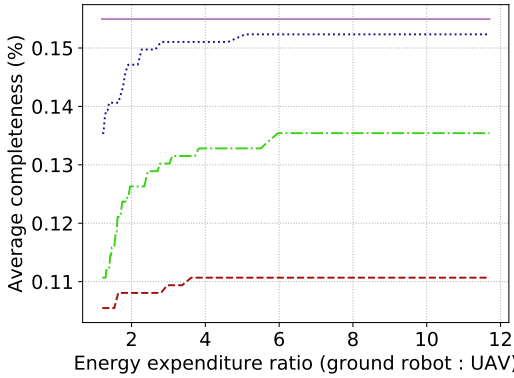
(c) **25 j/s** ground robot consumption; **200 kj** budget



(d) **25 j/s** ground robot consumption; **300 kj** budget



(e) **70 j/s** ground robot consumption; **200 kj** budget



(f) **70 j/s** ground robot consumption; **300 kj** budget

Figure 3.11: Average coverage completeness achieved under the constraint of an overall energy budget (kj), according to the relative energy consumption rates (j/s) of ground robots and UAVs.

Under a constrained energy budget, in general, more centralization tends to perform better if the ground robots are more efficient, and more decentralization tends to perform

better if the ground robots are less efficient (see Fig. 3.11). In most of the analyzed energy consumption scenarios, the decentralized approach is the top performer if the UAV has its highest possible energy consumption rate (i.e., 60 j/s). All the approaches that have UAV supervisors perform better if the UAVs are more efficient, but the MNS hybrid formation approach benefits the most from efficient UAVs, because it has more of them. As the energy budget increases, the performance of all approaches improves (see Fig. 3.11). If the budget is unlimited, the performance rankings of the four approaches will always be those seen at the end of a round in Sec. 3.4.1.1, regardless of the energy consumption ratio between ground robots and UAVs.

3.6 Discussion

It is generally expected that fully centralized and predetermined control approaches will have the highest possible performance, in terms of speed, efficiency, and accuracy. Decentralized approaches are often preferred only if the environment or other conditions cannot be known *a priori* (cf. [5, 79]). As expected, our results show that the predetermined control approach outperforms all other approaches, in terms of both coverage completeness and coverage uniformity (see Table 3.1). The performance gap between the predetermined control and fully decentralized approach is quite large—up to a 28.5% gap for coverage completeness. Also, as one would expect, the performance gap between the predetermined control and centralized formation control approaches is relatively small. However, perhaps more surprisingly, the performance gap between the predetermined control and hybrid formation control approaches is also relatively small—as small as 2.5%, and not more than 5.5%. Centralized and predetermined control approaches are sometimes criticized for being impractical to implement. Our results suggest that, beyond being impractical, some features commonly used in various centralized approaches—e.g., having unlimited *a priori* knowledge, unlimited communication ranges, and a central coordinating entity with unlimited field of view—may not always deliver as substantial a performance benefit as might be expected.

Decentralized approaches are expected to be much less efficient than other approaches and to decrease in efficiency with longer running times, because of inherent redundancy. This expectation is confirmed by our experimental results in coverage uniformity (see Fig. 3.7). Existing decentralized approaches often add more robots to speedup coverage completeness, but then repeated coverage also worsens [75]. Existing hybrid approaches that achieve high coverage completeness also suffer from high repeated coverage (e.g., [116], compared to their fully centralized counterparts. In our experimental results with the MNS hybrid formation approach, coverage uniformity is noticeably worse than in the predetermined control approach, but nearly the same as in the centralized formation control approach. This suggests that *a*

priori knowledge of the environment, rather than broadcasting from a central coordinating entity, provides the benefit of decreased repeated coverage.

As obstacle difficulty increases, it causes more disruption for all approaches. The MNS hybrid formation approach also has the added risk that its brain could lose contact with robots if blockage by obstacles is significant. Although the MNS formation did not lose any robots in our performance experiments (i.e., not fault tolerance experiments), it still might lose robots in more challenging environments than those tested. This risk could be alleviated by upgrading the behavior of the brain UAV to be responsive to its ground robots, waiting for them if they temporarily get disconnected.

Existing literature suggests that although decentralized approaches are less efficient in open environments, they may be better suited to environments cluttered with unknown objects (cf. [5]). However, in the tested obstacles difficulties, our results suggest that the predetermined control approach is the most resilient to obstacles in terms of coverage completeness (see Table 3.2). If the performance reduction rates for 100–300 obstacles are consistent at higher obstacle densities, then only the more centralized approaches would be tenable for coverage completeness. For example, if there were 1000 obstacles, the predetermined control approach would achieve coverage completeness of 94.2% in one round, while the fully decentralized approach would achieve only 32.5%. By contrast, at higher obstacle densities, the predetermined control approach's starting advantage in terms of coverage uniformity would disappear (see Table 3.2). If there were 1000 obstacles, the MNS hybrid formation control approach would achieve a coverage uniformity of 10.7, while the predetermined control approach would achieve the much worse uniformity of 14.03. Similarly, the predetermined control approach's speed advantage over the formation approaches begins to disappear at higher obstacle densities. In summary, our results suggest that there is not necessarily a simple trade-off between coverage efficiency and resilience to environment difficulty. Each of the trends suggested by the current results could be further investigated, in a variety of more challenging environments.

3.6.1 Fault tolerance and scalability

It is evident that the predetermined control and centralized formation control approaches suffer from a single point of failure and a communication bottleneck when scaling. As would be expected, our analysis also suggests that the predetermined control approach suffers much more from robot failures than the other approaches (see Table 3.3). Perhaps less expected, our analysis suggests that the MNS hybrid and centralized formation control approaches do not suffer much more from robot failures than the decentralized approach. These analysis results should be investigated further, in experimental setups with a variety of failure types.

Many existing hybrid approaches use a dynamically elected leader to increase fault tolerance and flexibility in an otherwise centralized approach (e.g., [40]), but these approaches still suffer from a bottleneck when scaling. It might be expected that the MNS hybrid formation control approach would suffer from a similar bottleneck. However, our experimental results indicate that communication in the MNS hybrid formation approach scales linearly (see Fig. 3.8), and that the communication load on the MNS brain does not increase when scaling.

3.6.2 Energy consumption

In later time steps (e.g., after the first 1000 time steps in the main experiments, see Fig. 3.5), the relative performance rankings of the four methods remain consistent—in terms of coverage completeness, the ranking is: 1st predetermined, 2nd centralized formation, 3rd MNS hybrid formation, and 4th fully decentralized (see Fig. 3.5). However, these performance rankings are not representative of earlier time steps. Although the coverage completeness in these earlier steps is low (e.g., 10% or 40%), our experiments use a high-resolution analysis grid, where adjacent grid points are 25 cm apart. This matches a real-world application in which, for instance, ground robots can sense a targeted condition within a 12.5 cm radius. In scenarios where a ground robot's sensing range is much higher, then a low coverage completeness in our experiments may be sufficient. For example, with a ground robot sensing range of 50 cm radius, intuitively we would posit that a coverage completeness of 25% could be sufficient for a monitoring application. This should be further investigated experimentally, using different grid resolutions in a discrete environment or using different sensor ranges. Similarly, all the analysis results in Sec. 3.5 need to be further investigated to be confirmed as representative—for example, by running experiments in larger arenas, up to 100 000 m^2. Such experiments would involve implementing new behaviors in some of the approaches. For instance, in a larger environment, the formation approaches would need to use a different path planning strategy such as boustrophedon motion [5, 8].

In general, our energy analysis results indicate that overall operation time is a highly relevant constraint. In large environments or with tight energy budgets, fully decentralized control may be the only feasible approach. In order to achieve high coverage completeness in real applications, our results indicate that the development of autonomous recharging is essential. Autonomous recharging will likely be easier to implement in hybrid or predetermined control approaches, because localization and positioning is more challenging in fully decentralized approaches. From among our tested approaches, the MNS hybrid formation control approach may be the most suitable for autonomous recharging, because it is already able to replace robots on the fly [140].

Future work could also investigate the implications of monetary costs. If many robots are added, the area per robot can be kept smaller, even in larger environments. However, in outdoor environments, suitable robots are expensive and the monetary cost of adding ground robots can be very high (see the Appendix). The cost of adding UAV supervisors is low in comparison, so adding UAV supervisors may sometimes be a more effective way to increase coverage completeness, depending on task conditions.

3.7 Conclusion

We have tested four approaches to multi-robot coverage with varying degrees of (de)centralization, and compared them using experiments and analysis. The predetermined control approach unsurprisingly achieves the highest performance. However, the performance of the hybrid approach is much closer to that of the centralized approaches than might be expected. Our results also indicate that the hybrid approach displays scalability and fault tolerance similar to the fully decentralized approach. In terms of resilience to challenging environments, our results indicate that more centralized approaches might perform better than expected in messy environments with randomized obstacles, outperforming the fully decentralized approach. In terms of the possible limitations associated with UAV supervisors, we find that the limited operating time of UAVs is more restrictive than their energy efficiency. Perhaps surprisingly, we also find that inefficient ground robots pose more of a problem than inefficient UAVs for the centralized approaches, and that in order for any approach to achieve coverage completeness in large outdoor environments, autonomous recharging will be necessary. These results provide a better understanding of the task conditions and constraints that are relevant when selecting the degree of (de)centralization for multi-robot coverage control. Although our ad-hoc comparisons are task-specific, they are an initial step towards better characterizing the assumed trade-offs between centralization and decentralization in multi-robot systems.

Chapter 4

Caterpillars

The focus of this chapter is on programmable matter. More specifically, we propose a novel higher-level leader-based programmable organism/matter, made of lower level organisms, that can be easily programmed to perform complex tasks (e.g., exploration, gathering etc.), and design one of its primitive operations. This programmable matter, which is called *Caterpillar*, can be considered a hybrid system as it is characterized by centralized coordination solely using local interactions between the robots.

4.1 Introduction

4.1.1 Framework

Programmable matter, introduced by Toffoli and Margolus [121], refers to a matter consisting of simple computational entities—called particles—that can change its physical properties (e.g., color, shape, density) in a programmable fashion. In other words, it refers to systems consisting of a very large number of simple computational units with limited computational, movement, and communication capabilities, called particles, that are programmed to move in the environment, interact locally with their neighbors and establish and release bonds in a self-organized way. The constituent particles of such a matter can collectively achieve their desired goals through self-organization. There are many applications for such systems in minimal invasive surgery, drug delivery, smart materials, large scale robotics and etc.

There are two classes of programmable matter systems, called active systems and passive systems. In passive programmable matter systems, particles either do not have any intelligence to control their behaviors or have a limited computational power but are incapable of controlling their movements. In the former case, they can only move and bond based on chemical interactions with environment or according to their structural properties. DNA computing (e.g., [1, 13, 16, 28, 134]) and tile self-assembly systems (e.g., [43, 98, 135]) are two examples of this kind of systems. On the other hand, in active programmable matter systems, particles can control their behaviors autonomously, so they can cooperatively accomplish a specific task. Example of research on such systems are theoretical mobile robots (e.g., [21, 51, 52, 91, 119]) and swarm robotics (e.g., [7, 105, 106]).

Many theoretical models have been proposed for programmable matter in nature inspired synthetic insects and micro-organisms (e.g., [41]), DNA self-assembly (e.g., [98, 104]), and metamorphic robots (e.g., [17, 131]). Geometric Amoebot [29] is one of the most recent models to study such self-organizing particle systems (SOPS), which is based on the behavior of amoeba. This model has been widely studied in applications such as coating [23, 30, 34], leader election [12, 22, 32, 38, 50, 56], convex hull formation [25], gathering (called aggregation in multi-robot systems) [15], and shape formation (called pattern formation in multi-robot systems). The latter one is prototypical for systems of self-organizing entities.

4.1.2 The problem

Designing individual-/micro-level algorithms for programmable matter that result in the desired swarm-/macro-level behavior is a very challenging task. More specifically, it is not easy

to program low level organisms (i.e., particles) of a self-organizing particle system to cooperatively perform a collective task while maintaining their connectivity. In order to make the design and implementation phase of programmable matter algorithms easier, higher level programmable organisms with higher level primitive operational capabilities can be developed and used. It is evident that one of the big advantages of considering higher level programmable matter would be programming simplicity.

In this thesis, we design a novel higher level leader-based programmable organism and start its implementation in the geometric Amoebot model, which is made of lower level organisms called particles,

In this higher level programmable organism, which is called *caterpillar*, one of the particles, the "head", plays the role of leadership and can initiate and control the behavior of the organism. This higher level organism has a continuous linear, or "string" structure, that should not be broken nor cross itself, and can perform some basic primitive operations such as "flock" (i.e., follow the head), "curl" (i.e., form a hexagon), "uncurl" (i.e., undo a hexagon), and "merge". Using this set of basic operations for caterpillars, will make it much easier to program them to perform complex tasks (e.g., exploration, gathering).

In this Chapter, we focus on the *curl* operation of this higher level organism and on the problem of implementing such an operation in the geometric Amoebot model. By "curling" we mean the operation of forming in the grid a filled hexagon shape (we shall call "hexagonal ball") starting from a line shape.

4.1.3 Contributions

In this Chapter, we design the *curl* operation of our higher level organism, implementing it in the geometric Amoebot model, thus solving the corresponding shape formation problem.

More precisely, we design and implement five deterministic algorithms, called `Caterpillar-1`, `Caterpillar-2`, `Caterpillar-3`, `Caterpillar-4`, and `Caterpillar-5`, for the new organism (i.e., Caterpillar) to curl and form a hexagonal ball. Each of these algorithms form a hexagonal ball based on different movement strategies, e.g., through winding around a specific particle, through zigzag motion led by a leader particle and pulling the rest of the particles.

We analyze the algorithms, prove their correctness, and establish tight bounds on their complexity with respect to the total number of movements of the particles.

In order to evaluate the performance of the algorithms empirically, we implement and run

experiments, for each algorithm, in the AmoebotSim simulator [1], and compare their results.

The remainder of this chapter is structured as follows. Section 2 reviews the related work. Section 3 describes the terminologies and model used in this chapter and formally defines the problem. Section 4 describes the five curl algorithms we designed and presents their correctness and complexity. Section 5 presents and discusses the results of the simulations conducted in the AmoebotSim simulator. Section 6 draws our conclusion.

4.2 Related Work

In this chapter, we design the curl operation of a caterpillar system. This problem is equivalent to a constrained hexagon formation problem by programmable particles in the geometric Amoebot model. Hexagon formation is an instance of *shape formation*, which is a prototypical problem for systems of self-organizing entities.

In this section, we discuss the related works on the shape formation problem, focusing on the geometric Amoebot model, as well as on the leader election problem in that model which is relevant to our study.

4.2.1 Shape formation

Shape formation is one of the fundamental collective tasks that has been intensively investigated in active systems such as autonomous mobile robots from theoretical point of view (e.g., [21, 51, 52, 91, 119]) and swarm robotics (e.g., [7, 105, 106]). There are two categories for this problem [24]: general shape formation and basic shape formation. In the basic shape formation, shapes such as line, triangle, or hexagon are constructed usually based on a follow-the-leader type protocol while in the general case a much broader range of shapes can be constructed using a much more complex algorithm.

The first study of shape formation under the geometric Amoebot model has been proposed in [31]. Based on this work, a swarm of particles can construct simple shapes like hexagon and triangle cooperatively. The presented algorithm has a complexity of $O(n)$ in terms of number of rounds (time complexity). Also, line formation as another sample problem in basic shape formation was first studied in [32]. In [31, 32], runtime bounds for the basic shape formation algorithms are proven in terms of number of particle movements (called work). The authors show that the worst-case number of movements to solve any of the three shape formation algorithms is $\Omega(n^2)$. On the other hand, in general shape formation, we can mention the

[1] https://amoebotsim.readthedocs.io/en/latest/

algorithm proposed in [33]. In that work, a much broader class of shapes can be built based on a complex algorithm. The important thing in this work is that, a well-constructed configuration (a triangle; not necessarily a complete one) is needed to be taken into account as an initial configuration of the particles. In [38], the authors provided a characterization of which shapes can be deterministically formed, starting from an unknown simply connected initial configuration of n particles. They also designed a universal shape formation algorithm in which a system of particles performs at most $O(n^2)$ moves in at most $O(n^2)$ rounds. They proved that such a system can form a shape similar to a given target shape of size m, starting from any simply connected configuration if n is larger than m and the given target shape is unbreakably k-symmetric if the initial configuration is. In [37], the authors investigated shape formation problem and showed that the particles can simulate a powerful Turing-complete entity that is able to move on the gird while computing. They proposed an algorithm that allows particles to form more general and abstract connected shapes (including circles, spirals, and fractal shapes of non-integer dimension) than the previous algorithms that allow particles to only form shapes made of arrangement of segment and triangles. Excluding very sparse pathological shapes, they proved that almost all feasible non-connected target shapes can be formed. Their results also provide a complete characterization of the connected shapes that a simply connected set of particles can form.

The curl operation implemented in this chapter can be seen as an instance of shape formation, where particles, initially in a straight line, move to form a filled hexagon always maintaining the linear connection among them. The hexagon formation problem had never been investigated before under this constraint, and none of the algorithms in the literature can solve it in our setting.

4.2.2 Leader election

Leader election is one of the basic problems in systems of programmable particles. There are two main strategies to elect a unique leader: in one strategy, particles use randomization to break symmetries [22, 32]; in another strategy, strong scheduling (i.e., sequential scheduling) is assumed, which means only one particle is activated at each time [50, 56]. Regarding the latter case, it has been shown that without particle movements, there might still be a special case in which electing a unique leader deterministically is impossible [12]. However, it is always impossible to deterministically elect a unique leader under weak scheduler, where activation of particles is non-atomic [45].

The first work in leader election using the geometric Amoebot model was presented in [32]. The authors assumed a common chirality for the particles and proposed a randomized algorithm to achieve leader election. The presented algorithm requires $O(L_{max})$ rounds to elect a

leader, where L_{max} is the length of the largest boundary in the shape. In another work [22], the authors improved the results of [32] by presenting a randomized algorithm that elect a unique leader in $O(L)$ rounds and terminates in $O(L + D)$ rounds, where L is the length of the outer boundary of the shape, and D is the diameter of the shape. In that work, a common chirality for the particles is still assumed. In [38], the authors proposed a deterministic algorithm for leader election problem for hole-free shapes of particles without using common chirality for the particles. In that work, multiple leaders could be elected in some situations, and the algorithm takes $O(n)$ rounds. In [12], a deterministic leader election algorithm is presented based on which multiple leaders could be elected in some cases. The presented algorithm works even in presence of holes. The algorithm elects a leader in $O(n^2)$ rounds. In [56], the authors presented a deterministic leader election algorithm for connected shapes of particles which assumes common chirality and an initial shape with no holes. According to their algorithm, a unique leader can be elected through assigning a k-local identifier to each particle (i.e., assigning a unique identifier to each of two particles that are at distance at most k with respect to each other). Their algorithm takes $O(r + mtree)$ rounds to terminate, where r and mtree are two specific terms to their paper. It can be shown that r+mtree is $\Omega(D)$. In [50], the authors presented a deterministic algorithm for the leader election problem which uses movement of the particles as a mechanism to break symmetry and elect a leader. Their algorithm elects a unique leader in $O(\log_{out} n^2)$ rounds even if common chirality is not assumed, and it also works for shapes with holes. Finally, in [45], authors proposed the first linear time algorithm for electing a unique leader deterministically with assuming chirality and strong scheduler. By assuming that those particles residing on a boundary of the matter know if that boundary faces a hole or the outer face, the run time of the algorithm is $O(D_A)$ rounds, where D_A is the diameter of the shape, including the holes. The authors also proved that without this assumption, the run time of the algorithm is $O(L_{out} + D)$ rounds, where L_{out} is the length of the outer boundary of the shape.

4.3 Model

In this section, we first present the formal model and terminologies used in the rest of the chapter, and then we formally describe the problem studied in this chapter. In the following, the geometric Amoebot model and subsequently, the caterpillar framework are described in details.

4.3.1 Amoebots

The geometric *Amoebot* model [26, 29], which is inspired by the behavior of amoeba, is one of the most popular models used to study programmable matter from the distributed computing perspective. In this model, programmable matter is viewed as a swarm of autonomous simple mobile computational entities, called *particles*, that can be programmed to perform collective tasks on a triangular tessellation of the plane.

Let $G_{eqt} = (V_{eqt}, E_{eqt})$ be the infinite regular triangular grid graph (see Fig. 4.1), where the set of nodes V_{eqt} represents the all the positions that can be occupied by particles, and the set E_{eqt} of edges describes the topological (i.e., neighbouring) structure of the graph. At any time, a node in V_{eqt} can be occupied by at most one particle, and a particle can only move from a node to a neighbouring node.

Each entity is provided with a constant-size local memory and has the computational power of a finite state machine. Each particle has a local (i.e., private) consistent sense of orientation of the grid G_{eqt}; based on this orientation, the ports (i.e., the edges) of any node are viewed by the particle as labeled {East, North-East, North-West, West, South-West, South-East} = {0, 1, 2, 3, 4, 5}. Different particles might have different sense of orientation.

Particles located on adjacent nodes are referred to as *neighbors* and can directly communicate with each other by exchanging information. More precisely, any pair of neighbors has a bounded shared memory that can be read and written by both of them and is used for exchanging information between them; the shared memory can be accessed by a particle using its edge (port) label corresponding to that connection.

Particles move by expanding and contracting their body, resembling the behavior of amoeba [6]. Each particle has two modes: *contracted* and *expanded*. When a particle is in *contracted* mode, it occupies only one node and can expand to one of its unoccupied adjacent nodes. When a particle is in expanded mode, it occupies two adjacent nodes and can contract to one of them. A *movement* is defined as an expansion followed by a contraction. For an expanded particle, the part of the particle on the node where it last expanded is called the *head* of the particle, and the other is called its *tail*. For a contracted particle, its head and tail coincide.

Each particle knows whether it is contracted or expanded, and its neighbors have access to this information (through the shared memories). The sense of orientation of a particle is consistent throughout all movements.

Another important operation is called *handover* which helps particles to keep their connectivity while moving. There are two types of handover, *push* and *pull*. (1) a contracted

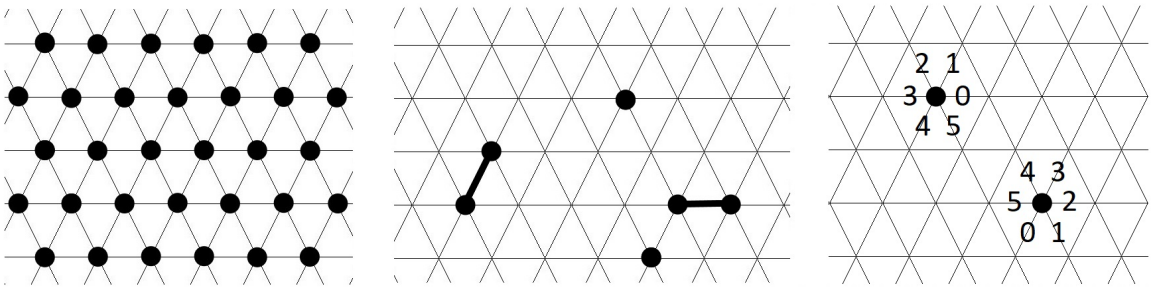


Figure 4.1: The left part of the figure shows a section of the infinite regular triangular graph where a set of contracted particles (i.e., particles that occupy a single node) have occupied the nodes of the graph. The middle part of the figure illustrates 4 particles (a particle occupying a single node of the graph is shown by a filled circle, and a particle occupying two nodes is shown by two filled circles connected by an edge). The right part of the figure illustrates two contracted particles with different orientations.

particle p can *push* an expanded neighbour q; as a result of this operation, p expands to the neighbouring node previously occupied by q, and q contracts to its other occupied node. 2) an expanded particle p can *pull* a contracted neighbour q; as a result of this operation, p contracts to its occupied node further from q, and q expands to the neighbouring node previously occupied by q (see Fig. 4.2).

The particles are anonymous (i.e., they do not have a unique ID) and they execute the same program. Time is divided into synchronous rounds. In each round, the particles activated in that round execute their program. The choice of which particles are activated in a round is under the control of an adversarial scheduler, which however is fair; that is, it activates every particle infinitely often. Such a scheduler is known in distributed computing as *Semi-Synchronous* ($\mathcal{S}synch$). Particular instances of this scheduler are the *Fully-Synchronous* ($\mathcal{F}synch$) one, where all particles are activated at every round, and the *Sequential* ($\mathcal{S}eq$) one, where only one particle is activated in each round.

To perform an operation, a particle must be active; however, in the handover operation, the involved neighbouring particle (being pushed or pulled) does not need to be active. In case of concurrency conflicts, several outcome are possible. In particular, If two or more particles expand to the same node at the same time, an occurrence we shall call collision, either none of them or only an arbitrary one of them will succeed.

4.3.2 Caterpillars

A caterpillar is a novel high-level programmable organism/matter that operates in an infinite triangular mesh, like amoebots.

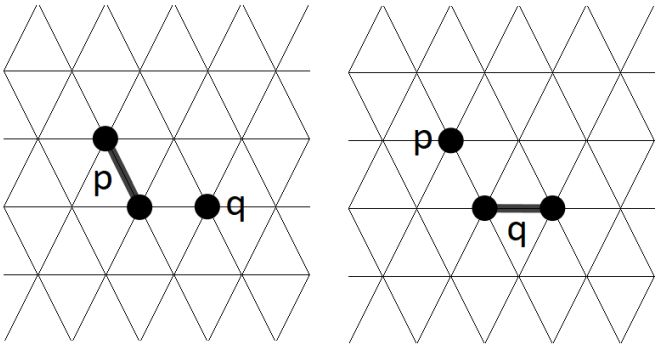


Figure 4.2: Two particles performing a handover operation. The left part illustrates an expanded particle p and a contracted particle q before performing the handover operation. The right part shows the configuration of the particles after executing the handover.

This new organism has a linear structure and is made of lower level amoebots. More precisely, a caterpillar $\mathcal{C}$ is a bonded linear chain $< p_1, ..., p_n >$ of $n > 1$ amoebot particles, in which every particle p_i $(1 < i < n)$ is bonded to its two neighbours p_{i-1} and p_{i+1}, and the extreme particle p_1 (resp., p_n) is bonded to p_2 (resp., p_{n-1}). The integrity of a caterpillar requires that the bonds should never be broken during the operations of the caterpillar in the grid; in other words, consecutive particles in the chain should never be disconnected. We shall refer to this constraint as the *bonding property* of caterpillars.

The two end particles of the chain play important roles. One plays the role of *leader*, coordinating the operations of the caterpillar; the other plays the role of *tail*, helping in controlling the execution of the operations. In a caterpillar, if needed, the leader particle can transfer its role to the tail particle, resulting into the two particles switching roles.

A caterpillar can perform a set of basic primitive operations such as *curl* (i.e., form a compact hexagon), *uncurl* (i.e., transform a hexagon into a line), and *flock* (i.e., move following the leader). This in turn allows it to perform more complex operations such as *coat* (i.e., uniformly surround an object) and *search* (i.e., explore the space). A caterpillar can be programmed to perform any combination of those operations (the program).

Initially, a caterpillar $\mathcal{C}$ forms a straight line segment where all particles are contracted, with the *leader* at one end, the *tail* at the other, and all other particle are in state *internal*. Although each particle has its own local orientation of the grid (defined in the amoebot part of the model), the leader can enforce its orientation; hence, without loss of generality, it is assumed that all the particles of a caterpillar have a common orientation. By definition, each particle always knows through which port(s) it is bonded to its neighbor(s).

To ensure the bonding property (i.e., that no disconnection occurs), an expanded particle

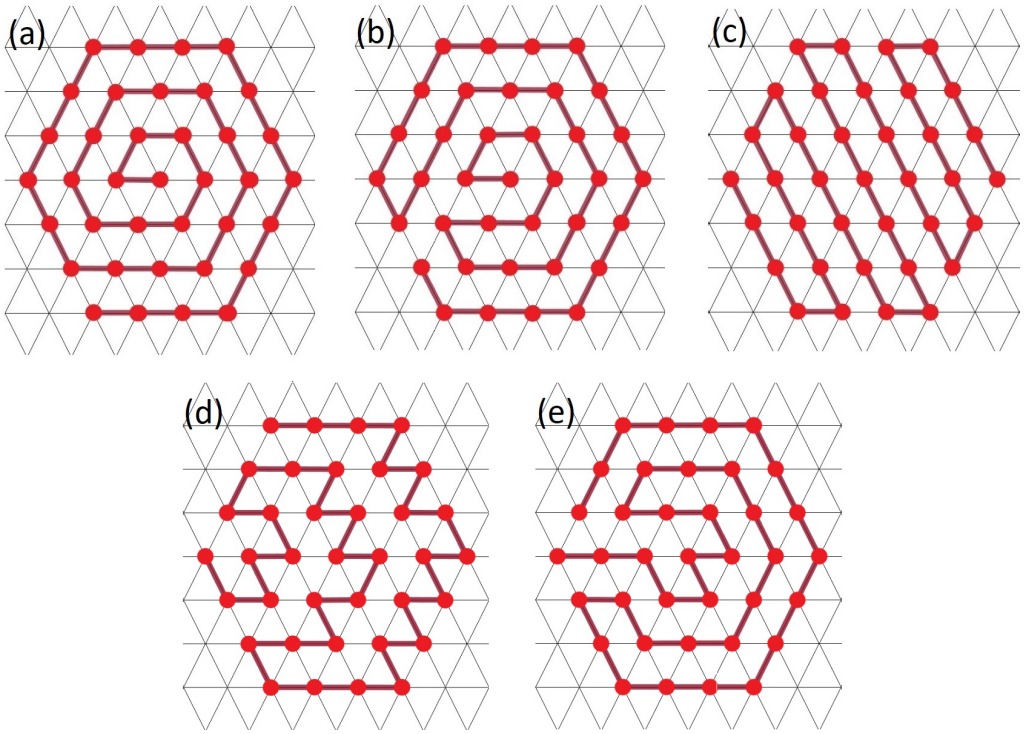


Figure 4.3: The complete hexagonal balls $HEX(4)$ formed by Algorithms `Caterpillar-1` (a), `Caterpillar-2` (b), `Caterpillar-3` (c), `Caterpillar-4` (d), and `Caterpillar-5` (e).

p (other than the tail) can contract only if its follower q in the line is contracted, and p will do so by pulling q. In other words, every movement is a pull handover operation.

As in the amoebot model, when we say that a particle p sends a message to a neighbor q, it means that p writes the message into the shared memory of q; when we say that a particle p writes the message on one of its incident ports, it means that p writes the message into the shared memory at the corresponding incident port. This transmission mechanism is used e.g. for communication between the leader and the tail.

4.3.3 Problem

In this thesis, the objective is to design and implement the *curl* operation of a caterpillar, using the principles of self-organization, in such a way that the bonding property of the caterpillar is always maintained.

When a caterpillar curls, it forms a hexagonal "ball" which may be either complete or incomplete, depending on the number n of particles of the caterpillar.

A complete hexagonal ball $HEX(h)$ of *dimension* $h \geq 1$, is a hexagon composed of

h layers. Its *size* $\sigma(h)$ (i.e., the number of nodes forming it) is defined by $\sigma(0) = 1$ and $\sigma(h > 0) = 1 + 3(h + 1)h$; its *perimeter* $\pi(h)$ (i.e., the number of its nodes on the exterior) is defined by $\pi(0) = 1$ and $\pi(h > 0) = 6h$. Clearly, if $n = \sigma(h)$, then $h = O(\sqrt{n})$.

We say that a caterpillar $\mathcal{C}$ of $n = \sigma(h)$ particles form a *complete* hexagonal ball $HEX(h)$ if all nodes of $HEX(h)$ contain a particle.

We say that a caterpillar $\mathcal{C}$ of n particles with $\sigma(h-1) < n < \sigma(h)$ forms an *incomplete* ball $HEX(h)$ if all nodes of $HEX(h-1)$ contain a particle.

In other words, the problem we study, that of implementing the curl operation, is the problem of transforming the shape of $\mathcal{C}$ from an initial straight line into that of a hexagonal ball. We address and solve this problem under the most commonly used adversarial scheduler for programmable particles, the sequential one.

The cost of this problem will be measured in terms of of the total number of movements performed by the particles to form a hexagonal ball, where each movement consists of an expansion and a contraction.

4.4 Algorithms

In this section, we design and present five different solution algorithms for the problem of forming a hexagonal ball, called `Caterpillar-1`, `Caterpillar-2`, `Caterpillar-3`, `Caterpillar-4`, and `Caterpillar-5`. Each algorithm fills the ball with a unique pattern. For each algorithm, we prove its correctness, and analyze its complexity.

We assume, without loss of generality, that the caterpillar $\mathcal{C}$ is initially on the line East-West, with the leader on the East-most node occupied by the chain. We also assume, to simplify the presentation, that the caterpillar can form a complete hexagonal ball; i.e., $n = \sigma(h)$.

4.4.1 Algorithm `Caterpillar-1`

In this section, we first briefly present the overall behaviour of Algorithm `Caterpillar-1`, and describe the details of the algorithm including the behaviour of the particles in each state. We then prove its correctness and analyze its complexity.

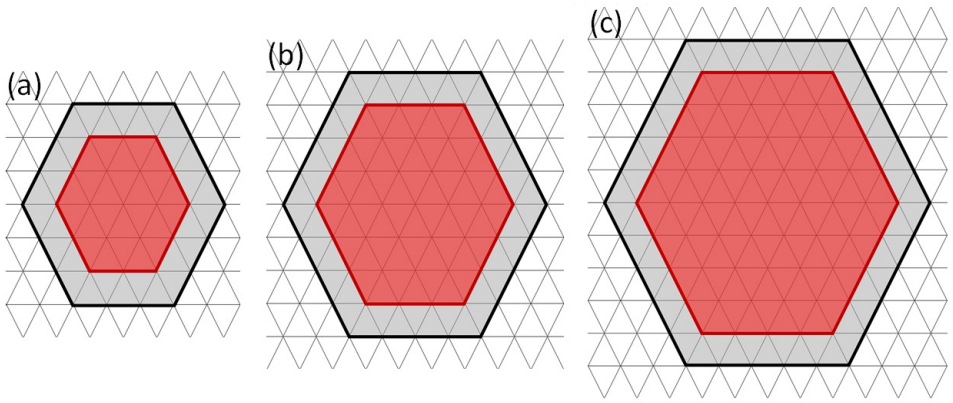


Figure 4.4: In Algorithms `Caterpillar-1` and `Caterpillar-5`, the hexagonal ball $HEX(i+1)$ is formed by adding a new layer all around $HEX(i)$

4.4.1.1 Informal description

In Algorithm `Caterpillar-1` a particle can be in one of the five following states: *follower*, *leader*, *root*, *retired*, and *terminated*.

In Algorithm `Caterpillar-1`, constituent particles of a caterpillar construct a hexagonal ball in a spiral shape, through winding around the head of the caterpillar while maintaining the bonding property throughout the process. In other words, in order to create the shape, particles move in a spiral through clockwise rotations, with the result being a spiral-shape hexagonal ball (see Fig. 4.3-a). Fig. 4.4 shows a general view of how the hexagonal ball grows using Algorithm `Caterpillar-1`.

This overall behavior is achieved through local rules that are described in the following sections.

4.4.1.2 Detailed description of the ROTATION procedure

The rotation process is obtained by appropriate movements of the particles, which coordinate their actions by local communication through the shared memories located on the nodes. Figure 4.5 shows an example of this process, where a line segment of 4 particles including a *root* and 3 *followers* rotates 60°. The *root* particle starts the rotation algorithm (see Algorithm 6) by writing $Next_i$ (where $i = \{0, 1, 2, 3\}$ is an indication of the layer that is currently under development) on the port corresponding to the target direction, which is the direction along which the *root's* preceding line segment of particles should be placed (Fig. 4.5-a). We recall that in all algorithms, layer 0 corresponds to the central node of the hexagonal ball, and layers' indices are considered modulo 4. After indicating the target direction, the *root* sends

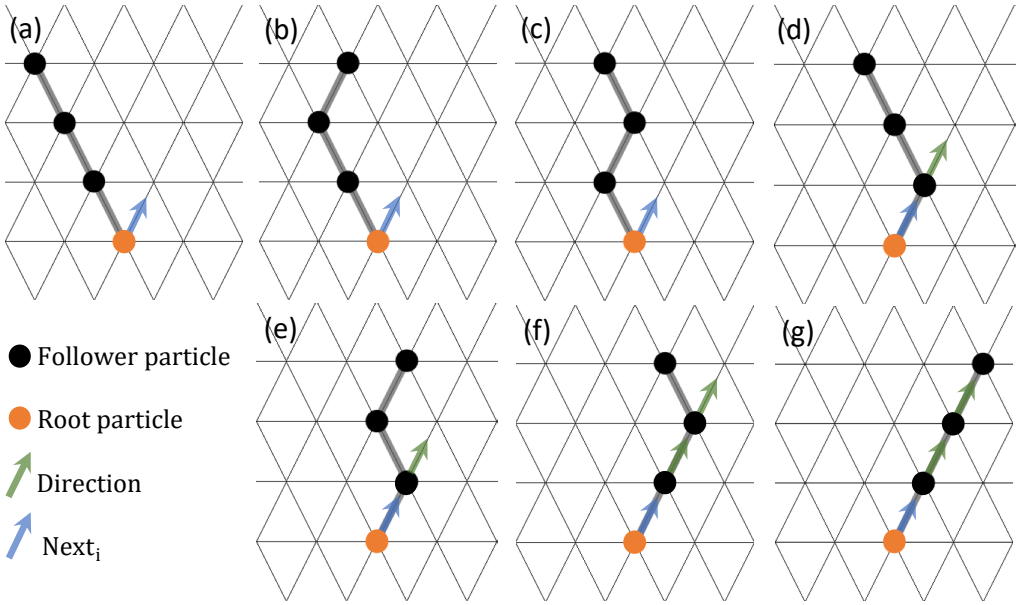


Figure 4.5: Algorithm `Caterpillar`-1: the rotation process. A line segment of 4 particles including the *root* (the orange particle) and 3 *followers* (the black particles) rotates 60°.

a *Rotate* message to its preceding bonded neighbor. A *Rotate* message is always forwarded toward the *tail* particle. Once the *tail* receives it, it moves one step clockwise around its following bonded neighbor (see Fig. 4.5-b, Fig. 4.5-e, and Fig. 4.5-g). Afterwards, if it is pointed by neither $Next_i$ nor *Direction*— which is a copy of target direction showing that the rotation is independent of the layer number (see Fig. 4.5-d as an example of *Direction* illustrated by green arrow)—it sends *Ack* message to its following bonded neighbor to indicate that the rotation is going forward but has not yet been completed (see Fig. 4.5-b and Fig. 4.5-e). Otherwise (Fig. 4.5-g), it sends *Completed* to its following bonded neighbor in the line indicating that the rotation has been successfully completed. Similarly, a *follower* receiving *Ack* understands that the rotation has not yet completed and moves one step clockwise around its following bonded neighbor, then it checks if it is pointed by either $Next_i$ or *Direction* (see Fig. 4.5-c, Fig. 4.5-d, and Fig. 4.5-f). If after the movement, the *follower* is not pointed by one of them (Fig. 4.5-c), it forwards *Ack* to its following bonded neighbor. Otherwise (see Fig. 4.5-d and Fig. 4.5-f), as opposed to the *tail*, it writes *Direction* on the port that is opposite to the port marked with $Next_i$ or *Direction* and then sends *Rotate* to its preceding bonded neighbor. If a *follower* receives *Completed* (Fig. 4.5-g), it forwards the message to its following bonded neighbor. If the *root* particle receives *Ack* (Fig. 4.5-b), it sends back a *Rotate* message in response. Finally, when the *root* particle receives *Completed* (Fig. 4.5-g), it realizes that the rotation process has been completed and the line segment of the *followers* (the line segment of the preceding particles) has been placed in the direction pointed by $Next_i$ and terminates the rotation process.

It is easy to see that:

Lemma 1. *The* ROTATION *procedure applied to a line segment, started by one extreme of the line (the* root *particle) results in a clock-wise rotation of the line segment of 60° from the* root*.*

4.4.1.3 Detailed description of the Algorithm

In this section, we present the details of Algorithm `Caterpillar-1`. Before getting into the details of the local rules, we introduce all the messages used by particles in this algorithm as follows:

- *RootActivation*: A message sent by either the *leader* or a *root* particle to its preceding bonded *follower* neighbor turning its state from *follower* to *root*.
- $FirstNode_p$, where p = 0,1,2,3 is an indication of the next layer that should be developed: This message is written by the *leader* or by the last *root* particle of layer $p-1$ on its West port and it is used for pointing to the first node of the next layer of the target shape that needs to be filled by its bonded neighbor. As the *leader* is the central node of the hexagonal ball (i.e., layer 0) in Algorithm `Caterpillar-1`, it writes $FirstNode_1$ on its West port.
- $Next_i$, where i = 0,1,2,3 is an indication of the current layer that should be developed: This message is written by a *root* particle on one of its ports and used for pointing to the next node of the current layer (i.e., layer i) that should be filled by the next *root* particle.

Figure 4.6 shows how a hexagonal ball can be formed using Algorithm `Caterpillar-1` (see Algorithm 5). In this algorithm, the *leader* particle starts the formation process by writing $FirstNode_1$ on its West port pointing to the first node of layer 1 that should be filled by its bonded neighbor. Then, it sends *RootActivation* to its preceding bonded neighbor which is in *follower* state. The receiver of this message becomes *root* and then points to the two nodes that are adjacent to both itself and the *leader* by writing $Next_1$ on the corresponding ports (i.e., North-East and South-East; see Fig. 4.6-a). In this way, the *root* shows both nodes pointed by $Next_1$ are eligible to be filled as the next node of layer 1. After that, the line segment of its preceding *follower* particles is rotated clockwise of 120° while maintaining the bonding property, following the ROTATION procedure. This results in placing the line segment of the *follower* particles along the North-East direction, which is pointed by $Next_1$ (Fig. 4.6-b). After completing the rotation process, the *root* particle sends a *RootActivation* message to its preceding bonded particle, which is a *follower*, and becomes *retired* (Fig. 4.6-b). From this step onward, the algorithm follows the following general rules:

– A *follower* particle that receives a *RootActivation* message always turns into *root*.

– A *root* particle is pointed by either $FirstNode_i$, one $Next_i$, or two $Next_i$, where i is the current layer number. The *root* follows one of the following behaviors based on the pointer type:

1. If it is pointed by $FirstNode_i$ (Fig. 4.6-t), it writes $Next_i$ on the North-West and South-East ports indicating that both nodes are eligible to be filled as the next node of layer i. Then, while maintaining the bonding property, the line segment of the *follower* particles (its preceding particles) is rotated clockwise following the ROTATION procedure until it is placed along North-West direction pointed by $Next_i$, which is a rotation of 60° (as an example of this case, see Fig. 4.6-t and 4.6-g).
2. If a *root* is pointed by only one $Next_i$, it searches for an adjacent node that is also adjacent to a *retired* particle from the previous layer (e.g., layer 4 if i = 0, layer 2 if i = 3). Then, it writes $Next_i$ on the corresponding port. Afterwards, it checks if the preceding bonded neighbor is pointed by $Next_i$ or not. In the former case it does not do anything (see Fig. 4.6-h, 4.6-j, 4.6-k, 4.6-l, 4.6-m, and 4.6-n). However, in the latter case (e.g., Fig. 4.6-b, 4.6-c, 4.6-d, 4.6-e, 4.6-g, 4.6-i), while keeping the bonding property, the line segment of its preceding particles is rotated clockwise following the ROTATION procedure until it is placed in the direction pointed by $Next_i$, which is a rotation of 60° (as an example of this case, see Fig. 4.6-c and 4.6-d).
3. If a *root* is pointed by two $Next_i$, it means it is the last particle in the layer that is currently under development. In this case, it writes $FirstNode_{(i+1 \ mod \ 4)}$ on its West port, pointing to the first node of the next layer that should be filled (Fig. 4.6-f).

– A *root* particle, after performing the actions associated to the pointer type as mentioned above, forwards *RootActivation* message to its preceding bonded *follower* (if there is any) and becomes *retired*.

Once the *tail* particle becomes *retired*, it sends $Termination$ to its bonded *retired* neighbor and becomes *terminated*. A *retired* particle that receives this message, forwards it to its following bonded particle, which is *retired* itself, and turns into *terminated*. When the *leader* particle receives this message, it becomes *terminated*, and the algorithm terminates. In this way, the caterpillar forms a spiral-shape hexagonal ball.

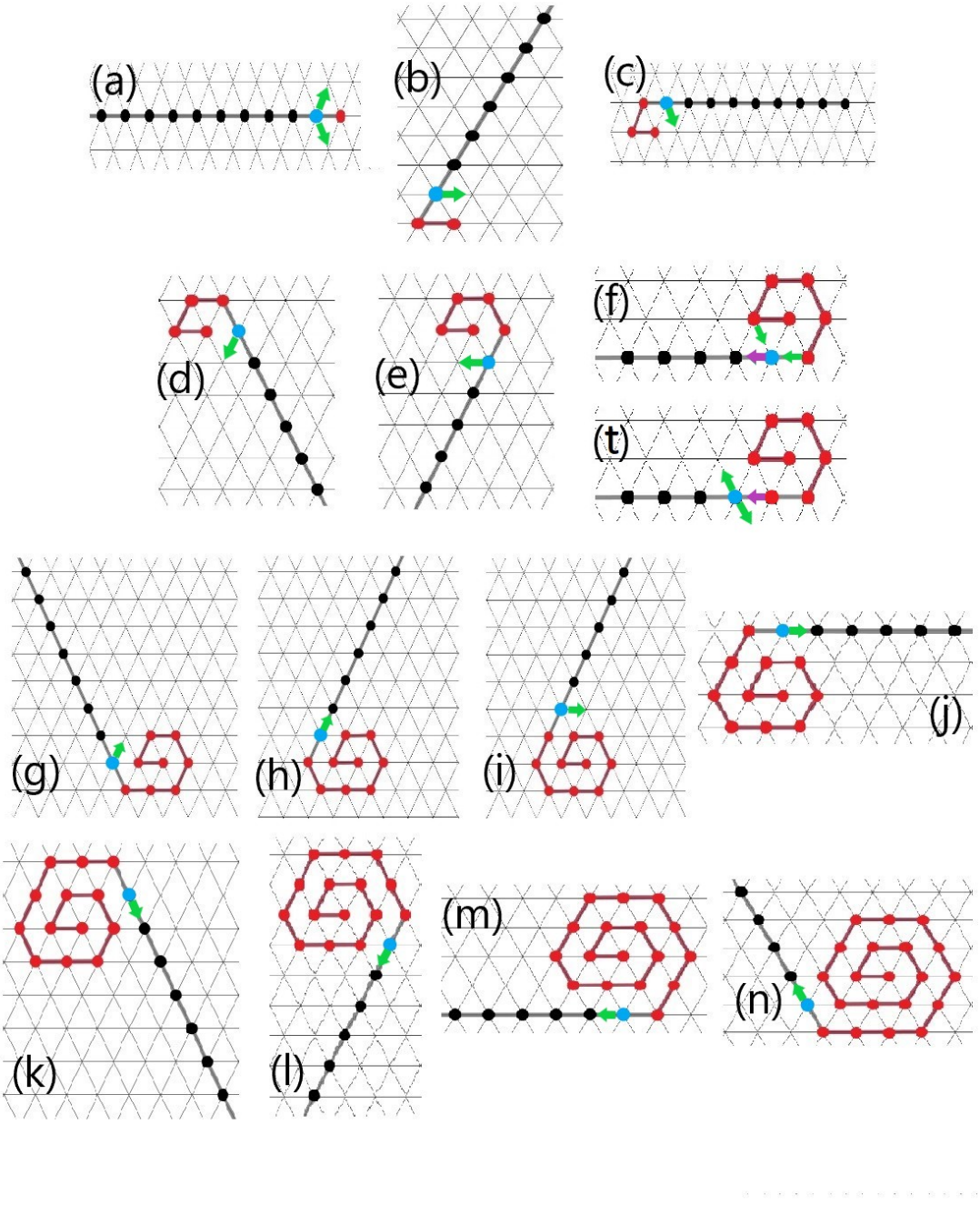


Figure 4.6: The *leader* particle and *retired* particles are shown in red. The *root* particle is indicated in blue. And the *follower* particles are shown in black. The purple arrows show $FirstNode_i$, and the green ones show $Next_i$. Each sub-figure shows the state of the system at the beginning of a step of execution, before performing any movement in that step.

Algorithm 5 `Caterpillar-1` - States an to initialize the rotation process Actions

Leader:

- Write $FirstNode_1$ on port 3
- Send RootActivation-0 to the preceding bonded neighbor

if (receive Termination) **then**
- Become *terminated*

end if

Root:

if (located on a node pointed by $FirstNode_1$) **then**
- Write $Next_1$ on port 1 and 5
- Call Rotate-Clockwise (see algorithm 6)

else if (located on a node pointed by $FirstNode_i$, where $i > 1$) **then**
- Write $Next_i$ on port 2 and 5
- Call Rotate-Clockwise (see algorithm 6)

else if (located on a node pointed by only one $Next_i$) **then**
- Search for an unoccupied adjacent node shared with a non-bonded *retired*
- Write $Next_i$ on the corresponding port

if (the preceding bonded neighbor is not pointed by $Next_i$) **then**
- Call Rotate-Clockwise (see algorithm 6)

end if

else ▷ If (located on a node pointed by two $Next_i$)
- Write $(FirstNode_{(i+1 \ mod \ 4)})$ on port 3

end if

- Send/Forward RootActivation to the preceding bonded neighbor if there is any
- Become *retired*

Follower:

if (receive RootActivation) **then**
- Become *root*

end if

Retired:

- Write Done on all ports

if (you are the *tail* particle) **then**
- Send Termination to the *retired* neighbor
- Become *terminated*

end if

if (receive Termination) **then**
- Forward Termination to the following bonded neighbor
- Become *terminated*

end if

Terminated:

- Do nothing

Algorithm 6 Rotate-Clockwise; Rotating the line segment of *follower* particles in a clockwise direction until it is placed in the direction pointed by $Next_i$.

Root:

- Send Rotate to next *follower* particle to start the rotation procedure

if (receive Ack) **then**
- Send Rotate to the preceding bonded neighbor

end if

if (receive Completed) **then**
- Return()

end if

Follower:

if (receive Rotate) **then**
if (there is a preceding bonded neighbor) **then**
- Forward Rotate to the preceding bonded neighbor

else
- Move one step clockwise around the following bonded neighbor

if (pointed by $Next_i$ or *Direction*) **then**
- Send Completed to the following bonded neighbor

else
- Send Ack to the following bonded neighbor

end if
end if
end if

if (receive Ack) **then**
- Move one step clockwise around the following bonded neighbor

if (pointed by $Next_i$ or *Direction*) **then**
- Write *Direction* on the port opposite to the port marked with $Next_i$ or *Direction*
- Send Rotate to the preceding bonded neighbor

else
- Forward Ack to the following bonded neighbor

end if
end if

if (receive Completed) **then**
- Forward Completed to the following bonded neighbor

end if

4.4.1.4 Correctness

In this section, we prove the correctness of Algorithm `Caterpillar-1`. solves the problem of forming a hexagonal ball. Let us assume that the number of particles allows to form exactly a complete hexagonal ball.

Theorem 1. *Algorithm* `Caterpillar-1` *forms a (complete) hexagonal ball.*

Proof. The algorithm proceeds layer by layer, completely filling all the nodes of each layer. We prove the theorem by induction on the number of layers of the ball.

Base Case. We show that $HEX(1)$ is fully formed and, when this happens, the particle occupying the South-Western corner is the *root*. According to Lemma 1, if the rotation procedure is applied around the *root*, the *root's* preceding line segment of particles is correctly rotated by 60° clockwise. So, after six rotations, $HEX(1)$ is fully formed (see Figure 4.6), and the *root* that initiated the last rotation (i.e., the particle located on the South-Eastern corner of $HEX(1)$) passes its rule to its preceding bonded neighbor (i.e., the particle located on the South-Western corner of $HEX(1)$) by sending a *RootActivation* message.

Induction step. Let us assume that the hexagonal ball is fully formed up to level i (i.e., $HEX(i)$ is formed), with the *root* being the South-Western corner, and consider the next layer $i+1$.

Algorithm Caterpillar-1 starts the formation of layer $i+1$ with the South-Western side of $HEX(i+1)$. To this purpose, the *root*, which is located on the South-Western corner of $HEX(i)$, writes $FirstNode_{(i+1 \ mod \ 4)}$ on its West port pointing to the first node that should be filled in layer $i+1$. Because of the last rotation performed in layer i, the mentioned pointed node has already been filled by the preceding bonded neighbor of the *root*. Next, the *root* passes the *root* rule to its preceding bonded neighbor though sending a *RootActivation* message. Afterwards, the new *root* initiates the rotation procedure. As already mentioned, based on Lemma 1, if the rotation procedure is applied around the *root*, there will be a 60° clockwise rotation of the *root's* preceding line segment of particles correctly performed. Therefore, after applying a rotation around the *root*, all the nodes of the South-Western side of $HEX(i+1)$, excluding the South-Western corner, are filled such that the bonding property is maintained. After this rotation, the *root* rule is passed along the particles occupying the South-Western side of $HEX(i+1)$ one by one, until the particle located on the west-most corner of $HEX(i+1)$ becomes *root*. Afterwards, the new *root* initiates the second rotation in order to form the North-Western side of $HEX(i+1)$. After completing the rotation, the North-Western side of $HEX(i+1)$ is fully formed, and the *root* rule is passed along that side until the particle located on the North-Western corner of $HEX(i+1)$ becomes *root*. This process is repeated for four more times (i.e., in total six times) to fully form

layer $i+1$. More specifically, the third rotation, fourth rotation, fifth rotation, and sixth rotation result in forming the top (i.e., Northern) side of $HEX(i+1)$, North-Eastern side of $HEX(i+1)$, South-Eastern side of $HEX(i+1)$, and bottom (i.e., Southern) side of $HEX(i+1)$, respectively. After performing the sixth rotation (which is applied around the *root* located on the South-Eastern corner of $HEX(i+1)$), all the nodes of the bottom side of $HEX(i+1)$, including the South-Western corner of $HEX(i+1)$, are occupied by particles, and $HEX(i+1)$ is successfully completed. Finally, the *root* rule is passed from the particle occupying the South-Eastern corner of $HEX(i+1)$ to the one occupying the South-Western corner of $HEX(i+1)$, particle by particle. □

4.4.1.5 Complexity

In this section, we calculate the complexity of Algorithm `Caterpillar-1`.

Theorem 2. *The total number of movements in Algorithm* `Caterpillar-1` *is* $\Theta(n^{2.5})$.

Proof. Algorithm `Caterpillar-1` consists of a sequence of 60° clockwise *line rotations*; in each iteration, the rotation is initiated by the particle, called *root*, located on the extreme node of the line (the other being the *tail*), and involves all the other particles of the line, called *followers*.

After each rotation, the root and some other particles of the line become integrated in the already formed part of the hexagon, a new particle becomes the new root, and starts the rotation of the remaining line.

Let L_j denote the length of the line *after* the completion of $HEX(j-1)$; then $L_j = n - \sigma(j-1)$.

A rotation is achieved by the (coordinated) movement of each particle of the line, from its current location to its destination, along the diagonal indicated by the clockwise rotation angle. Thus, the movements performed by the $i-th$ element of a line of length L, where the root has $i=0$, is exactly i; i.e., the 60° clockwise rotation of a line of length L requires precisely

$$R(L) = \sum_{0 \le i \le L-1} i = \frac{(L-1)(L)}{2}.$$

Initially, the head of the caterpillar forms the 0-th layer of the hexagon and no longer moves; its only neighbour becomes the root of the rest of the line and starts the process constructing the perimeter of $HEX(1)$ with the remaining $L_1 = n-1$ elements. This process consists of six consecutive rotations. The first two rotations are initialized by the bonded

neighbour of the leader. After the first rotation, the length of the line remains unchanged. After each of the five following rotations, the length of the line decreases by 1, and the follower neighbouring the root becomes the new root. Thus the total number $M(L_1)$ of movements for this process is

$$M(L_1) = R(n-1) + \sum_{i=1}^{5} R(n-i) = \frac{(n-2)(n-1)}{2} + \sum_{i=1}^{5} \frac{(n-i-1)(n-i)}{2}$$

Once $HEX(1)$ has been constructed, each of the following layers is obtained through a *cycle* of six consecutive rotations. In the cycle to form $HEX(j)$ ($j \geq 2$), after the first rotation, the length of the line decreases by $j-1$ units: the particle of the line at distance $j-1$ from the root (if any) becomes the new root; after each of the following five rotations, the length of the line decreases by j units: the particle of the line at distance j from the root (if any) becomes the new root.

In other words, for $j > 1$, the total number $M(L_j)$ of movements in the cycle of rotations constructing $HEX(j)$ starting from $HEX(j-1)$ is precisely

$$M(L_j) = R(L_j) + R(L_j - (j-1)) + \sum_{i=1}^{4} R(L_j - (j-1) - ij) =$$

$$= \frac{1}{2}((L_j - 1)L_j + (L_j - j)(L_j - j + 1) + \sum_{i=1}^{4}(L_j - j - ij)(L_j - j - ij + 1)).$$

The total number $TM(n,h)$ of movements performed by the algorithm to construct $HEX(h)$ when $n = \sigma(h)$ is precisely

$$TM(n,h) = M(L_1) + \sum_{j=2}^{h} M(L_j).$$

We can bound $M(n,h)$ by bounding $M(L_j)$. To bound it from above, it suffices to consider the cost of the most expensive rotation (the first) in each set of six and to bound it from below, we consider the least expensive (the last). In other words:

$$6R(L_j - 5j) < M(L_j) < 6R(L_j)$$

Let us first calculate the upper bound $6R(L_j)$. Recall that $\sigma(h > 0) = 1 + 3(h+1)h$ and $L_j = n - \sigma(j-1)$; thus, by substitution, $L_j = n - 1 - 3j(j-1) = n - 1 - 3j^2 + 3j$.

$$6R(L_j) = 3L_j(L_j - 1) = 3(n - 1 - 3j^2 + 3j)(n - 1 - 3j^2 + 3j - 1) =$$
$$= 3(n^2 - 3n + 2 + 9j^4 - 18j^3 + (18 - 6n)j^2 - (9 - 6n)j)$$

In total, for the completion of all the hexagons up to $HEX(h)$, we then have an upper bound on the number of movements $TM(n, h)$:

$$TM(n,h) < M(L_1) + 3\sum_{j=2}^{h}(n^2 - 3n + 2) + 3\sum_{j=2}^{h}(9j^4 - 18j^3 + (18 - 6n)j^2 - (9 - 6n)j)$$

The overall complexity is then $O(n^{2.5})$.

To see how tight this bound is, we now examine the lower bound on $M(L_j)$ provided by $6R(L_j - 5j)$.

$$6R(L_j - 5j) = 3(L_j - 5j)(L_j - 5j - 1) = 3(n - 1 - 3j^2 - 2j)(n - 1 - 3j^2 - 2j - 1) =$$
$$= 3(n^2 - 3n + 2 + 9j^4 + 12j^3 + (13 - 6n)j^2 + (6 - 4n)j)$$

This means that, for the completion of all the hexagons up to $HEX(h)$, the total number of movements is bounded below by:

$$TM(n,h) > M(L_1) + 3\sum_{j=2}^{h}(n^2 - 3n + 2) + 3\sum_{j=2}^{h}(9j^4 + 12j^3 + (13 - 6n)j^2 + (6 - 4n)j)$$

Also these summations are $O(n^{2.5})$, which means that both the lower and the upper bound are of the same order of magnitude; thus, the total number of movements of Algorithm `Caterpillar`-1 is $\Theta(n^{2.5})$, completing the proof of the theorem when $n = \sigma(h)$.

Observe that, in the case $\sigma(h) < n < \sigma(h + 1)$, after the creation of $HEX(h)$, there is still a line of $b = n - \sigma(h) < 6(h + 1)$ particles that will be rotated to position themselves on the $h + 1$th layer of the hexagon, without however completing $HEX(h + 1)$. The cost of these rotations is however order of magnitudes smaller than the overall complexity required to reach $HEX(h)$. Hence, if the cost is of these movements are omitted from the calculations

of the lower and upper bounds, the claimed results still hold. □

4.4.2 Algorithm `Caterpillar-2`

In this section, we first briefly present the overall behaviour of Algorithm `Caterpillar-2`, and we describe the details of the algorithm including the behaviour of the particles in each state. We then prove its correctness and analyze its complexity.

4.4.2.1 Informal Description

The idea behind Algorithm `Caterpillar-2` is to form a hexagonal ball in a way that each odd layer and its following even layer are formed in parallel. This algorithm results in forming a maze-shaped hexagonal ball (see Fig. 4.3-b). Starting from the bonded neighbour of the *leader* (which is triggered by the *leader*), the particles develop the hexagon through moving clockwise around their following neighbour in their activation turn until finding and occupying the next unoccupied node in the layer that is under construction. Each particle, during its movement, pulls its preceding line segment of particles which results in developing the hexagon two layers by two layers with maintaining the bonding property. In other words, once an odd layer (e.g., layer 1) is completed by the particles, the next even layer (e.g., layer 2) has been formed in parallel as well because of the pulling actions. This overall behavior is achieved through local rules that are described in the following section. Fig. 4.4 shows how a hexagonal ball is developed using Algorithm `Caterpillar-2`.

4.4.2.2 Detailed Description

In this section, we present the details of Algorithm `Caterpillar-2` (see Algorithm 7). In Algorithm `Caterpillar-2`, a particle can be in one of the five following states: *follower*, *leader*, *root*, *retired*, and *terminated*. Before getting into the details of the local rules, we introduce all the messages used by particles in this algorithm as follows:

- *RootActivation*: A message sent by either the *leader* or a *root* particle to its preceding bonded *follower* neighbor turning its state from *follower* to *root*.
- $FirstNode_p$, where p = 0,1,2,3 is an indication of the next layer that should be developed: This message is written by the *leader* or the last *root* particle of layer $p-1$ on its West port and it is used for pointing to the first node of the next layer of the target shape that needs to be filled by its bonded neighbor. As the *leader* is the central node

of the hexagonal ball (i.e., layer 0) in Algorithm `Caterpillar-2`, it writes $FirstNode_1$ on its West port.

– $Next_i$, where i = 0,1,2,3 is an indication of the current layer that should be developed: This message is written by a *root* particle on one of its ports and used for pointing to the next node of the current layer (i.e., layer i) that should be filled by the next *root* particle.

Figure 4.7 shows how a hexagonal ball can be formed using Algorithm `Caterpillar-2`. In this algorithm, the *leader* particle (the blue particle in Fig. 4.7) starts the formation process by writing $FirstNode_1$ on its West port, pointing to the first node of layer 1 that should be filled by its bonded neighbor (see Fig. 4.7-a). Then, it sends *RootActivation* to its preceding bonded neighbor which is in *follower* state. From this step onward, the algorithm follows the following general rules:

– A *follower* particle (the black particles in Fig. 4.7) that receives a *RootActivation* message, always turns into *root* (the orange particles in Fig. 4.7).

– A *root* particle follows one of the behaviors listed below based on the mentioned conditions, then forwards the *RootActivation* message to its preceding bonded *follower*, if there is any, and becomes *retired* (the red particles in Fig. 4.7):

1. If it is located on a node pointed by $FirstNode_i$, and the current layer is odd (i.e., $i = 2k + 1$), it writes $Next_i$ on the North-East and South-East ports indicating that both nodes are eligible to be filled as the next node of layer i. As two examples of this case, see Figs. 4.7-b and 4.7-p which show the $Next_i$ messages when the corresponding *root* has turned into
2. If it is located on a node pointed by $FirstNode_i$, and the current layer is even (i.e., i = 2k), it writes $Next_i$ on the North-West and South-East ports indicating that both nodes are eligible to be filled next in layer i. As an example of this case, see Fig. 4.7-n which shows the $Next_i$ messages when the corresponding *root* has turned into *retired*.
3. If a *root* is pointed by only one $Next_i$ (e.g., Fig. 4.7-e), it writes $Next_i$ on the port corresponding to an adjacent node that is not occupied by its following bonded neighbor, and is also adjacent to a *retired* particle from the previous layer (e.g., Fig. 4.7-f which shows the $Next_i$ message when the corresponding *root* has turned into *retired*).
4. If a *root* is pointed by two $Next_i$ (e.g., Fig. 4.7-k), it means it is the last particle in the layer that is currently under development. In this case, it writes $FirstNode_{(i+1 \ mod \ 4)}$ on its West port, pointing to the first node of the next layer

that should be filled (e.g., Fig. 4.7-l which shows the $Next_i$ message when the corresponding *root* has turned into *retired*).

5. Otherwise, the *root* moves around its following bonded neighbor, which is *retired*, in a clockwise way (while maintaining its bonding property) to find a node that is pointed by either $FirstNode_i$, one $Next_i$ or two $Next_i$ (e.g., Figs. 4.7-b, 4.7-d, 4.7-j, and 4.7-l which show the required clockwise motion with a curved arrow).

Once the *tail* particle becomes *retired*, it sends $Termination$ to its bonded *retired* neighbor and becomes *terminated*. A *retired* particle that receives this message, forwards it to its following bonded particle, which is *retired* itself, and turns into *terminated*. When the *leader* particle receives this message, it becomes *terminated*, and the algorithm terminates.

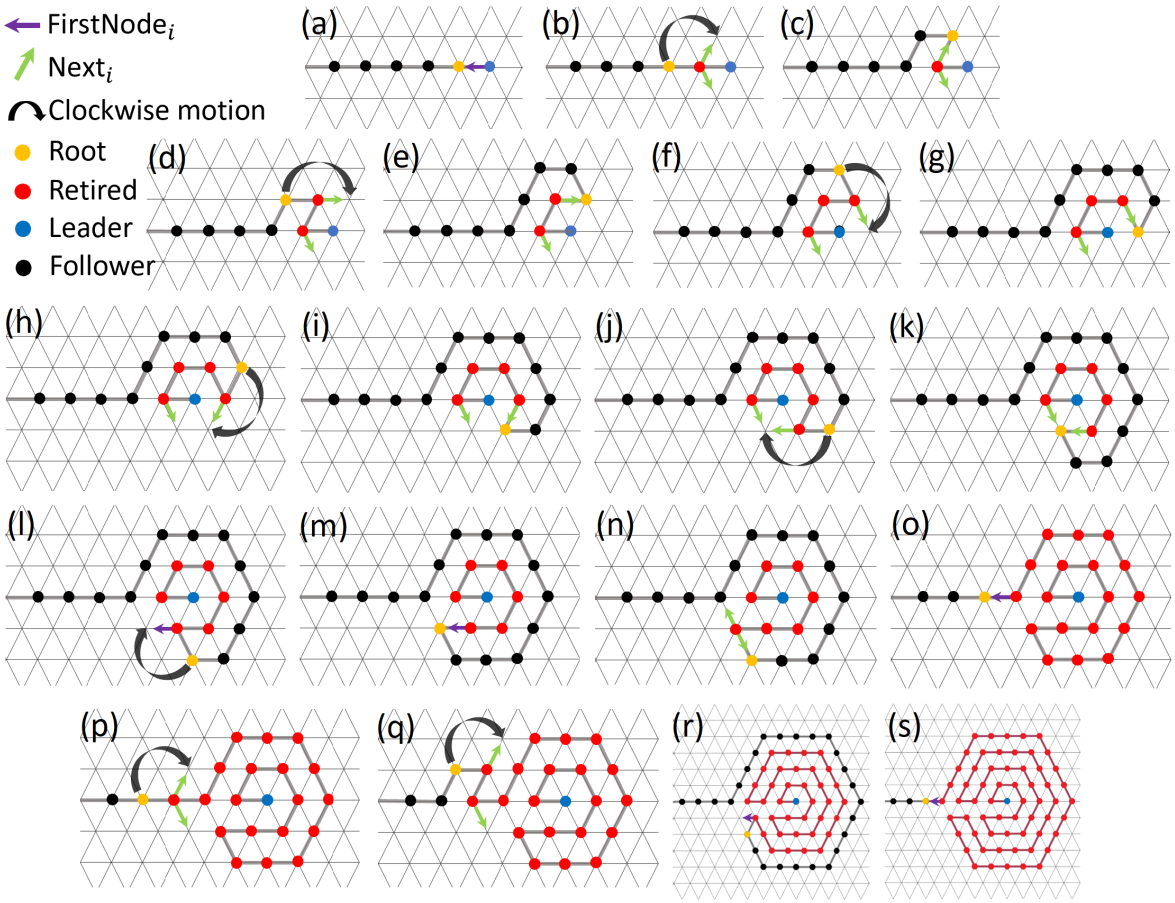


Figure 4.7: Algorithm `Caterpillar-2`. The *leader* particle is shown in blue. The *retired* particles, *root* particles, and *follower* particles are shown in red, yellow, and black, respectively. $FirstNode_i$ messages and $Next_i$ messages have been shown with purple arrows and green arrows, respectively. The clockwise motions have been illustrated with black curved arrows.

Algorithm 7 `Caterpillar-2` - States and Actions

Leader:

- Write $FirstNode_1$ on port 3
- Send *RootActivation* to the preceding bonded neighbor

if (receive Termination) **then**

- Become *terminated*

end if

Root:

if (not located on a node pointed by $FirstNode_i$ or $Next_i$) **then**

- Move around the following bonded neighbor in a clockwise manner while keep connectivity, to find an unoccupied node pointed by $FirstNode_i$ or $Next_i$

else

if (located on a node pointed by $FirstNode_i$, where i is odd) **then**

- Write $Next_i$ on ports 1 and 5

else if (located on a node pointed by $FirstNode_i$, where i is even) **then**

- Write $Next_i$ on ports 2 and 5

else if (located on a node pointed by two $Next_i$) **then**

- Write $(FirstNode_{(i+1 \ mod \ 4)})$ on port 3

else if (located on a node pointed by only one $Next_i$) **then**

- Search for an adjacent node that is also adjacent to a *retired* particle from the previous layer and is not occupied by your following bonded neighbor
- Write $Next_i$ on the corresponding port

end if

- Forward RootActivation to the preceding bonded neighbor, if there is any
- Become *retired*

end if

Follower:

if (receive RootActivation) **then**

- Become *root*

end if

Retired:

- Write Done on all ports

if you are the *tail* particle **then**

- Send Termination to the following bonded neighbor
- Become *terminated*

end if

if (receive Termination) **then**

- Forward Termination to the following bonded neighbor
- Become *terminated*

end if

Terminated:

- Do nothing

4.4.2.3 Correctness

In this section, we show that Algorithm `Caterpillar-2` solves the problem of forming a complete hexagonal ball $HEX(h)$, where $n = \sigma(h)$.

Theorem 3. *Algorithm* `Caterpillar-2` *forms a (complete) hexagonal ball.*

Proof. We prove the theorem by induction on the number of layers.

Base Case. It is easy to see by inspection of the algorithm's rules that $HEX(1)$ and $HEX(2)$ are formed by the algorithm (see also Figure 4.7 (a) to (o) to follow all the steps).

Induction step. Let us assume that $HEX(2i)$ is fully formed for some $i \geq 1$ with the *root* being the west neighbour of the west-most corner. We now show that, if there are enough particles still outside $HEX(2i)$ (i.e., if $h \geq 2i + 2$), the algorithm constructs the next two layers $2i + 1$ and $2i + 2$ in the same operation, and the *root* is the west neighbour of the west-most corner. Otherwise (i.e., if $h = 2i + 1$), the algorithm forms $HEX(h)$ completing its task.

To form layer $2i + 1$, all $6(2i + 1)$ nodes coating layer $2i$ should be filled by the particles. According to the algorithm, the first node that is filled in layer $2i+1$ is the western neighboring node of the west-most corner of $HEX(2i)$ which is already pointed by a $FirstNode$ message and occupied by the current *root*. The *root* initiates the process of forming layer $2i + 1$ and $2i + 2$ by pointing to two adjacent nodes that are also adjacent to its following bonded neighbor (i.e., neighboring nodes located on the North-East and South-East). Then, it makes its preceding bonded neighbor *root* (through forwarding the *RootActivation* message) and becomes *retired*. Based on Algorithm `Caterpillar-2`, all layers of the target shape are formed in a clockwise order, so the next node of layer $2i + 1$ that is filled is the one located on the North-East of the new *root's* following bonded neighbor (i.e., the node located on the North-East of the previous *root* which has been pointed by it). The new *root* (the particle that received the *RootActivation*) moves clockwise around its following bonded neighbor while keeping its connectivity with that particle. The preceding line segment of particles, which are *followers*, are also pulled by this particle .

When the *root* finds the pointed node, it occupies it and then searches for an unoccupied adjacent node that is also adjacent to a *retired* particle from the previous layer. It is clear that such a node is located on layer $2i+1$. Afterwards, the *root* points to that node indicating that it is the next node that should be filled, forwards the *RootActivation* to its preceding bonded neighbor, and becomes *retired*. This process is repeated and it results in filling all the $6(2i + 1)$ nodes of layer $2i + 1$ in a clockwise order while maintaining the bonding property. Once the node located on the South-East (port 5) of the first *root* of this layer (i.e., the *root*

initiated the process of forming layer $2i+1$ by pointing to two of its adjacent nodes) is filled, layer $2i+1$ is completed, and $HEX(2i+1)$ is formed. Thus, if $h = 2i+1$, the algorithm has formed $HEX(h)$ completing its task.

If, instead, $h \geq 2i+2$, the operation continues. Based on the algorithm, the South-Eastern neighbor of the particle occupying the node located on the west-most corner of $HEX(2i+1)$ is pointed by two of its *retired* neighbors. The *root* occupying this node points to the node that is located on its West as the first node of layer $2i+2$ that should be filled by its preceding bonded neighbor (by writing a $FirstNode$ message on the corresponding port). Then, it forwards the *RootActivation* to its preceding bonded neighbor and becomes *retired*. The bonded neighbor receiving this message becomes *root* and moves clockwise around its following bonded neighbor and occupies the pointed node. At this moment, all the nodes of layer $2i+2$ have been already occupied by particles, and $HEX(2i+2)$ has been formed. □

4.4.2.4 Complexity

In this section, we calculate the complexity of Algorithm `Caterpillar-2`.

Theorem 4. *The total number of movements in Algorithm* `Caterpillar-2` *is* $\Theta(n^2)$.

Proof. Starting from $i = 1$, the algorithm constructs the perimeter of $HEX(2i-1)$ and $HEX(2i)$ at the same time (completing the former before the latter).

Let $p(x) \in \{1, ..., n\}$ denote the initial position particle x on the line, with the "head" in the first position. During the execution of the algorithm, every particle x with $p(x) > 1$ moves from its initial position $p(x)$ to its final destination in $HEX(h)$. According to the algorithm, the final destination of particle x where

$$\sum_{1 \leq i \leq j} (\pi(2i-1) + \pi(2i)) < p(x) \leq \sum_{1 \leq i \leq j+1} (\pi(2i-1) + \pi(2i))$$

that is

$$12j^2 + 6j < p(x) \leq 12j^2 + 30j + 18$$

is on the perimeter of $HEX(2j+1)$ or $HEX(2j+2)$.

Notice that, according to the algorithm, x moves in the direction of the line until it reaches the boundary of $HEX(2j+2)$, thus performing $M_1(x) < 12j^2 + 30j + 18$ moves. It then traverses the boundary of $HEX(2i+2)$ until it reaches its final destination on the perimeter of either $HEX(2j+1)$ or $HEX(2j+2)$, performing $M_2(x) \leq \pi(2j+2) = 12j + 12$ moves.

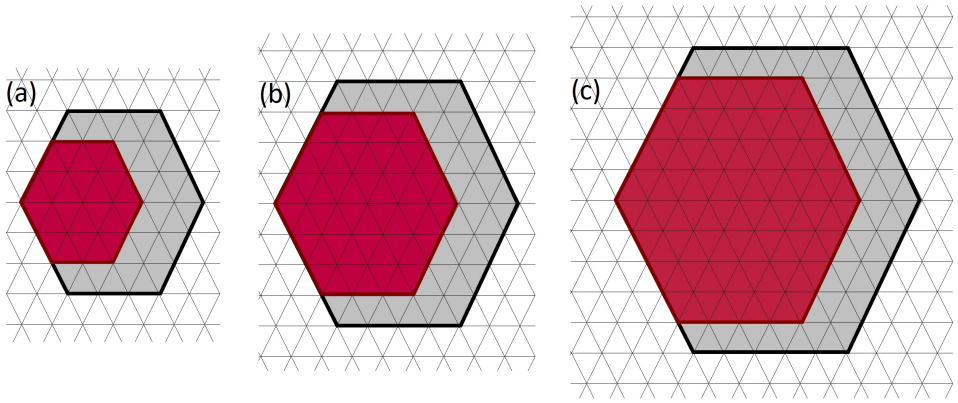


Figure 4.8: A general view of how the hexagonal ball $HEX(i+1)$ is formed by expanding $HEX(i)$ in Algorithms `Caterpillar`-3 and `Caterpillar`-4.

Since $M_1(x)+M_2(x) = O(j^2) = O(n)$, it follows that the total number of moves performed by the particle is $\sum_x (M_1(x) + M_2(x)) = O(n^2)$.

This order of complexity of the algorithm is tight; in fact, it is not difficult to see that the moves performed by the particles just to reach the boundary of their final destinations on $HEX(h)$ is quadratic; i.e., $\sum_x M_1(x) = \Omega(n^2)$. □

4.4.3 Algorithm `Caterpillar-3`

In this section, we first briefly present the overall behaviour of Algorithm `Caterpillar`-3, and we describe the details of the algorithm including the behaviour of the particles in each state. We then prove its correctness and analyze its complexity.

4.4.3.1 Informal Description

In Algorithm `Caterpillar`-3, the pattern formed by the caterpillar inside the hexagonal ball $HEX(h)$ is an alternance of parallel diagonal chords (along the NW-SE axis), filling the hexagon in a "zig-zag" manner (see Fig. 4.3-c).

The hexagonal ball $HEX(h)$ is constructed layer by layer, with the expansion from $HEX(i)$ to $HEX(i+1)$ occuring in the grid as shown in Figure 4.8.

Once $HEX(i)$ is formed, $i < h = \sigma^{-1}(n)$, $HEX(i+1)$ is constructed in two stages. In the first stage, the North and South borders of $HEX(i)$ are sequentially expanded and filled through handover movements controlled by some of the particles located on layer i. In the second stage, the still unoccupied location of $HEX(i+1)$ are filled through a "zig-zag"

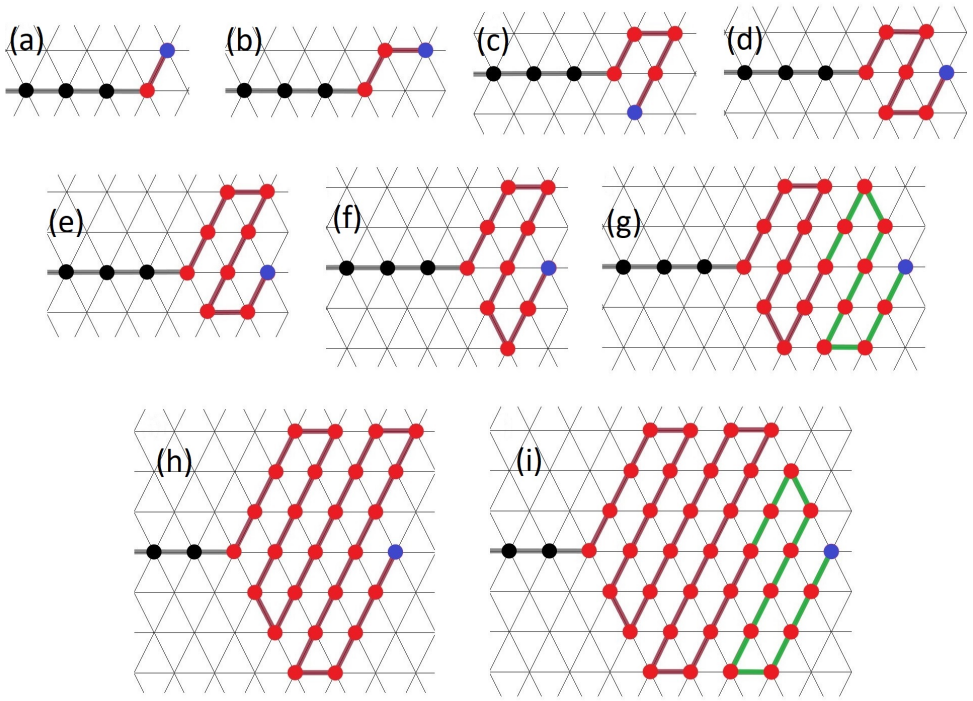


Figure 4.9: Algorithm `Caterpillar`-3. The *leader* particle is shown in blue. The particles occupying a node in the hexagonal ball are shown in red. Other particles are shown in black. The green links illustrates the Zig-Zag motion process performed by the *leader*.

handover movement of the entire chain controlled by the leader. See Figure 4.9.

4.4.3.2 Detailed Description

In Algorithm `Caterpillar`-3, a particle can be in one of the four following states: *follower*, *leader*, *semi-retired*, and *terminated*.

The first layer of the hexagonal ball is formed via the *leader's* movements. To this end, the *leader* first moves one step along North-East; then, it moves one step along East; next, it moves two steps in direction of South-West; then, it moves one step along East; finally, it moves one step along North-East (Figs. 4.9-a to 4.9-d shows this process).

The process of formation of other layers is done in two stages. In the first stage, the process of formation of other layers is based on five primitive rules, which are performed by *non-leader* particles under some circumstances. In the second stage, a zig-zag motion process is performed by the *leader*. After completing each layer (including layer 1), the *leader* sends a *Round_Anymore*? message to its preceding bonded neighbor to announce the completion of the layer to other particles and to initiate the process of forming the next layer. This message

is forwarded particle by particle toward the *tail* of the caterpillar and turns the receivers into *semi-retired*. If a *follower* particle that has its following bonded neighbor on its East and either it is a *tail* particle or it has a bonded *follower* neighbor on its West receives it, it sends back *Ack* in response. This message is forwarded toward the *leader*. When a *semi-retired* particle receives this message, it first checks if it should initiate one of the following primitive rules based on its current position in the hexagonal ball. If none of the rules is needed to be initiated by the particle, it forwards the message to the following bonded neighbor, which is *semi-retired*, and then becomes *follower*.

In the first stage, the primitive rules are as follows:

Rule A (Algorithm 8) This type is indicated in Fig. 4.10.a. If a *semi-retired* particle which has a bonded *follower* neighbor on its South-West and a bonded *semi-retired* one on its East receives a *Ack* message (the red particle in Fig. 4.10.a.1), it moves one step in direction of North-East and then moves one step in direction of East (see Fig. 4.10.a.2). Next, it forwards the *Ack* message to the following *semi-retired* neighbor and becomes *follower*.

Rule B (Algorithm 9) Fig. 4.10.d shows this rule. If a corner *semi-retired* particle that has a bonded *follower* neighbor on its South-West and a bonded *semi-retired* one on its South-East (the red particle in Fig. 4.10.d.1) receives *Ack*, it moves one step in direction of East (see Fig. 4.10.d.2), then sends $P10$ to its preceding bonded neighbor, which is a *follower* particle (the green particle in Fig. 4.10.d.2). When the *follower* receives this message, it moves one step in direction of North-East and then moves one step in direction of East (see Fig. 4.10.d-3). Next, it informs the sender of $P10$ (the red particle in Fig. 4.10.d.3) message that the mission is completed by sending *Completed* message to it. Finally, once the sender of $P10$ receives the *Completed* message, it forwards the *Ack* message to the following bonded neighbor, which is a *semi-retired* particle, and becomes *follower*.

Rule C (Algorithm 10) This type of movement is shown in Fig. 4.10.e. If a *semi-retired* particle that is not located on a corner node of the formed pattern (which is called non-corner *semi-retired* particle) and has a bonded *follower* neighbor on its South-West and a bonded *semi-retired* one on its South-East (the red particle in Fig. 4.10.e.1), receives *Ack*, it moves one step in direction of East (see Fig. 4.10.e.2). Then, it sends $P10P1$ message to its preceding bonded neighbor which is a *follower* particle (the green particle in Fig. 4.10.e.2). This *follower* moves one step in direction of North-East and then moves one step in direction of East (see Fig. 4.10.e.3), as soon as receiving such a message. Next, it sends $P1$ message to its preceding bonded neighbor which is *follower* (the blue particle in Fig. 4.10.e.4). The *follower* receiving the message moves one step in direction of North-East (see Fig. 4.10.e.4) and then informs

the sender of $P1$ that the mission is completed by sending a *Completed* message to it. This message is forwarded to the sender of $P10P1$ (the red particle in Fig. 4.10.e.4). Finally, when the sender of $P10P1$ receives *Completed* as response, it forwards the *Ack* message to the following bonded neighbor, which is *semi-retired*, and then becomes *follower*.

Rule D (Algorithm 11) Fig. 4.10.c illustrates this rule. If a corner *semi-retired* particle that has a bonded *follower* neighbor on its North-East and a bonded *semi-retired* one on its East (the red particle in Fig. 4.10.c.1), receives *Ack*, it moves one step in direction of South-East (see Fig. 4.10.c.2). Then, it forwards the *Ack* message to the following bonded *semi-retired* particle and becomes *follower*.

Rule E (Algorithm 12) Fig. 4.10.b shows this type of movement. If a non-corner *semi-retired* particle that has a bonded *follower* neighbor on its North-East and a bonded *semi-retired* one on its East (the red particle in Fig. 4.10.b.1), receives a *Ack* message, it moves one step in direction of South-East (see Fig. 4.10.b.2) and then sends $P4$ message to its preceding bonded neighbor, which is *follower* (the green particle in Fig. 4.10.b.2). The *follower* receiving this message moves one step in direction of South-West (see Fig. 4.10.b.3) and then sends back *Completed*. The receiver of the *Completed* message (the red particle in Fig. 4.10.b.3) forwards the *Ack* message to its following bonded neighbor, which is a *semi-retired* particle and then becomes *follower*.

Algorithm 8 *RuleA*: States and Actions

Semi-retired:

- Move one step along port 1; Move one step along port 0; Forward the *Ack* to the following bonded neighbor; Become *follower*

Algorithm 9 *RuleB*: States and Actions

Semi-retired:

- Move one step along port 0; Send $P10$ to the preceding bonded neighbor

if (receive *Completed*) **then**

- Forward the *Ack* to the following bonded neighbor; Become *follower*

end if

Follower:

if (receive $P10$) **then**

- Move one step along port 1; Move one step along port 0; Send *Completed* to the sender in response

end if

if (receive *Completed*) **then**

- Forward it to the following bonded neighbor

end if

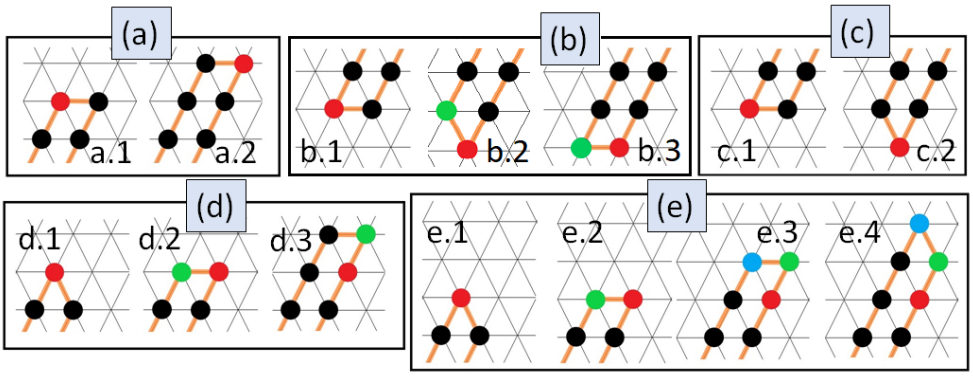


Figure 4.10: Algorithm `Caterpillar-3`: the primitive rules. The shape of the line segment of particles before and after the execution of each rule (i.e., the extension process) has been shown using the orange edges which illustrate the bonds. The *semi-retired* particle that starts the rule has been shown in red. In case of those rules in which the preceding *follower* particle of a *semi-retired* particle needs to move based on the message received from that particle, the mentioned follower particle has been shown in green. In case of those rules in which the preceding *follower* particle of a *follower* particle needs to move based on the message received from that particle, the mentioned preceding *follower* particle that received the message and moves has been shown in green. The rest of the particles have been shown in black regardless of their state.

Algorithm 10 *RuleC*: States and Actions

Semi-retired:

- Move one step along port 0; Send $P10P1$ to your preceding bonded neighbor

if (receive *Completed*) **then**

- Forward the *Ack* to the following bonded neighbor; Become *follower*

end if

Follower:

if (receive $P10P1$) **then**

- Move one step along port 1; Move one step along port 0; Send $P1$

end if

if (receive $P1$) **then**

- Move one step along port 1; Send *Completed* to the sender in response

end if

if (receive *Completed*) **then**

- Forward it to the following bonded neighbor

end if

Algorithm 11 *RuleD*: States and Actions

Semi-retired:

- Move one step along port 5; Forward the *Ack* to the following bonded neighbor; Become *follower*

Algorithm 12 *RuleE*: States and Actions

Semi-retired:

- Move one step along port 5; Send $P4$ to the preceding bonded neighbor
if (receive *Completed*) **then**
  - Forward the *Ack* to the following bonded neighbor; Become *follower*
end if

Follower:

if (receive $P4$) **then**
  - Move one step along port 4; Send *Completed* to the sender in response
end if

Algorithm 13 Zig-Zag motion process

Leader:

- Move two steps along port 1; Move one step in direction of port 5; Keep moving in direction of port 4 as long as there is a neighbor on your West which itself has a neighbor on its South-West; Move one step toward port 0; Keep moving in direction of port 1 as long as you have a neighbor on your North-West which itself has neighbor on its North-East;

The second stage starts when the *leader* receives a *Ack* message. At this point, the leader performs the zig-zag motion process. Based on this process, the *leader* moves two steps along North-East. Then, it moves one step in direction of South-East and keeps moving in direction of South-West as long as it has one neighbor on its West which itself has a neighbor on its South-West. Afterwards, it moves one step toward East and keeps moving in direction of North-East as long as it has a neighbor on its North-West which itself has a neighbor on its North-East (see Figs. 4.9-e to 4.9-d and Figs. 4.9-e to 4.9-d as two examples; the green links show this motion process). After these consecutive movements, the current layer is completely formed, and this process is repeated for all layers.

As the first movement of the *leader* is along North-East, each particle that enters the area of the hexagonal ball because of being pulled by its following bonded neighbor, has its following bonded neighbor on its North-East. Therefore, once the *tail* particle's bonded neighbor is located on its North-East, it understands that it has entered to area of the hexagonal ball because of being pulled by its following bonded neighbor and sends a *Termination* message to its bonded neighbor. This message is forwarded toward the *leader* and makes all particles terminated one by one. In this way, all the particles terminate, and the algorithm halts. It is worth mentioning that in Algorithm `Caterpillar`-3, a particle can easily check if it is a corner particle or not. In fact, a particle is a corner if it has no neighbor (bonded or non-bonded) in either of the following sets of directions: (a) East and North-West (ports 0 and 2), or (b) East and South-West (ports 0 and 4), or (c) West and North-East (ports 3 and 1), or (d) West and South-East (ports 3 and 5), or (e) North-West and South-West (ports 2

and 4), or (f) North-East and South-East (ports 1 and 5). If none of the conditions above is verified, the particle is a non-corner particle.

4.4.3.3 Correctness

In this section, we show that Algorithm `Caterpillar-3` solves the problem of forming a hexagonal ball. Let us assume that the number of particle allows to form exactly a complete hexagonal ball.

Theorem 5. *Algorithm* `Caterpillar-3` *correctly terminates.*

Proof. We prove the theorem by induction on the number of layers. For the basis of the induction, observe that $HEX(1)$ is trivially formed by the *leader* performing the following sequence of movements: one step along North-East; one step along East; two steps along South-West; one step along East; and finally, one step along North-East (see figure 4.9 (a) to (d)). at the end of this operation, the *leader* is the east-most corner.

Let $j > 1$; observe that the sequence of rules followed by the algorithm to form $HEX(j)$ from $HEX(j-1)$ is different depending on whether j is even or odd; hence, we will consider the two cases separately, starting with $j = 2i$ even.

Case 1: Assume that the hexagonal ball is fully formed up to level $2i-1$, $i > 0$, with the *leader* at the east-most corner of $HEX(2i-1)$, and consider the next layer $2i$.

By construction, the following sequence of rules is performed in order (from left to right) by associated particles, followed by the zig-zag motion process of the *leader* to form layer $2i$: $A^i D(CE)^{i-1}$

In order to form $HEX(2i)$, Algorithm `Caterpillar-3` extends $HEX(2i-1)$ from west to east as depicted in figure 4.8. As shown in the figure, the caterpillar extends the shape from the top (i.e., North) side and bottom (i.e., South) side of $HEX(2i-1)$ for one level while it extends the shape from North-Eastern side and South-Eastern side of $HEX(2i-1)$ for two levels. More specifically, the caterpillar fills $2i+1$ nodes above the North side of $HEX(2i-1)$ to extend it for 1 level in the North direction; fills $2i+1$ nodes below the lower side of $HEX(2i-1)$ to extend it for 1 level in the South direction; and fills $3(2i)+(2i-2)$ to extend $HEX(2i-1)$ for two levels in the North-East and South-East directions.

As already mentioned, the rules are executed according to the *Ack* message forwarded from East to West of the Caterpillar (i.e., in the direction of head) and such a message only is forwarded when either no rule is triggered or after completion of a triggered rule. In this way, the process of forming the next layer is sequential. Moreover, as all the chords of

Algorithm 14 `Caterpillar-3` - States and Actions

Leader:

if (Initialization == 1) **then** ▷ Forming the first layer
- Move one step along port 1; Move one step along port 0; Move two steps along port 4; Move one step along port 0; Move one step along port 1
- Set Initialization to 0

else if (receive *Ack*) **then** ▷ Forming layer i, where i is bigger than 1
- Call Zig-Zag motion process (see algorithm 13)

end if

- Send *Round_Anymore*? to the preceding bonded neighbor

if (receive *Termination*) **then**
- Become *terminated*

end if

Semi-retired:

if (receive *Ack*) **then**
if (you have a bonded *follower* neighbor on your South-West and a bonded *semi-retired* one on your East) **then**
- Call Rule A procedure (see algorithm 8)

else if (you are a corner *semi-retired* particle & have a bonded *follower* neighbor on your South-West and a bonded *semi-retired* one on your South-East) **then**
- Call Rule B procedure (see algorithm 9)

else if (you are a corner *semi-retired* particle & have a bonded *follower* neighbor on your North-East and a bonded *semi-retired* one on your East) **then**
- Call Rule D procedure (see algorithm 11)

else if (you are a non-corner *semi-retired* particle & have a bonded *follower* neighbor on your South-West and a bonded *semi-retired* one on your South-East) **then**
- Call Rule C procedure (see algorithm 10)

else if (you are a non-corner *semi-retired* particle & have a bonded *follower* neighbor on your North-East and a bonded *semi-retired* one on your East) **then**
- Call Rule E procedure (see algorithm 12)

else
- Forward the *Ack* to the following bonded neighbor; Become *follower*

end if
end if
if (receive *Termination*) **then**
- Forward the *Termination* to the following bonded neighbor; Become *terminated*

end if

Algorithm 14 `Caterpillar-3` - States and Actions (continued)

Follower:

if (you are *tail*, and your bonded neighbor is on your North-East) **then**
- Send *Termination* to the following bonded neighbor
- Become *terminated*

end if
if (receive *Round_Anymore*?) **then**
if ((you are *tail*, or you have a bonded *follower* neighbor on your West) & (you have a bonded neighbor on your East)) **then**
- Send *Ack* to the bonded *semi-retired* neighbor

else if (there is a preceding bonded neighbor) **then**
- Forward the *Round_Anymore*? to the preceding bonded neighbor
- Become *semi-retired*

end if
end if

Terminated:

- Do nothing

the hexagonal ball from North-West to South-East are filled with a zig-zag pattern of the caterpillar, the bends of the caterpillar are located across its North, South-West, South, and North-Eastern sides. All 6 primitive rules are sequentially executed by some particles located on the aforementioned sides of $HEX(2i-1)$.

The first side of $HEX(2i)$ that is formed is the North-Western side. This is done simply by performing rule A, according to which, the particle occupying the North-Western corner of $HEX(2i-1)$ moves to its North-Eastern adjacent node and then moves one step in the East direction. By performing this rule, two nodes of the upper side of the shape are occupied, including the North-Western corner of $HEX(2i)$.

After performing rule A, based on the *Ack* message forwarded toward the West of the caterpillar, rule A is executed for $i-1$ more times resulting in filling $2i$ nodes of the upper side of $HEX(2i)$ in total. After performing the last rule A, the *Ack* message is forwarded toward the South-Western corner of $HEX(2i-1)$, through the particles occupying the longest chord of the hexagonal ball. When the particle located on the South-Western corner of $HEX(2i-1)$ receives the *Ack* message, it performs rule D, according to which, it moves one step toward its South-East and forms the South-Western corner of $HEX(2i)$. In this way, the South-Western side of $HEX(2i)$ is completed. Then, the message is forward along a chord in the direction of North-East until it is received by a particle occupying a node of the North-Eastern side of $HEX(2i-1)$. Such a node performs rule C which results in filling 4 nodes in the North-East area of $HEX(2i)$ including the North-Western corner. By filling the North-Western corner, the North side of $HEX(2i)$ is completed.

Next, the message is forward along another chord of the hexagonal ball until it is received

by a particle occupying a node in bottom side of $HEX(2i-1)$. Such a node executes rule E according to which the particle moves one step towards south-West and then moves one step towards East. Then the message is forwarder through another chord until a particle occupying a node in the North-Eastern side of $HEX(2i-1)$ receives it. The aforementioned pair of rules, i.e., C and E are executed in the same order (because of the zigzag pattern of the caterpillar along the chords of the hexagonal ball) for i-1 times in total. In this way all the nodes of the bottom side of $HEX(2i)$ except for two of them, and all nodes of the North-Eastern side of $HEX(2i)$ except for three of them are filled by particles.

Afterwards, the *Ack* message is forwarded until the *leader* receives it. By receiving the message, the leader executes the zigzag motion process, according to which, the leader moves two steps along North-East resulting in filling one of the three remaining nodes on the North-Eastern side of $HEX(2i)$, and then moves to the South-East resulting in filling another node of the North-Eastern side of $HEX(2i)$. Next, the leader moves toward its South-West until it reaches to a node of the bottom side of $HEX(2i)$. In this way, the leader forms a chord connecting the North-Western side to the South side of $HEX(2i)$. Then, the leader moves one step to East resulting in filling the South-Eastern corner of $HEX(2i)$. At this point, the South-Eastern side of $HEX(2i)$ is completed. The leader keeps moving along its North-East until it reaches the final unoccupied node of the North-Eastern side of $HEX(2i)$ which is the east-most corner of $HEX(2i)$. In this way, the leader forms the South-Eastern side of $HEX(2i)$ and this means the $HEX(2i)$ is completely formed.

As a consequence, since $HEX(1)$ can be formed, it follows that also $HEX(2)$ can be formed with the *leader* at the east-most corner of $HEX(2)$.

Case 2: Assume that the hexagonal ball is fully formed up to level $2i$, $i > 0$, with the *leader* at the east-most corner of $HEX(2i)$, and consider the next layer $2i+1$. In this case, the following sequence of rules is performed in order (from left to right) by associated particles, followed by the zig-zag motion process of the *leader* to form layer $2i+1$: $A^iBE(CE)^{i-1}$ This case can be proved in a similar way as the proof related to the extension from $HEX(2i-1)$ to $HEX(2i)$ (i.e., the proof discussed above). □

As an example, Figure 4.11 shows how a hexagonal ball has been completely extended from layer 3 (see Fig. 4.11-a) to layer 4 (see Fig. 4.11-g) by performing the following sequence of rules, followed by the zig-zag motion process: $A^2D(CE)$. Figure 4.12 shows how by performing ABE, followed by the zig-zag motion process, $HEX(2)$ (see Fig. 4.12-a) is extended to $HEX(3)$ by the caterpillar (see Fig. 4.12-e). As another example, Figure 4.12-f shows the situation in which a hexagonal ball has been completely extended from $HEX(4)$ to $HEX(5)$ by performing the following sequence of rules, followed by the *leader's* zig-zag motion process: $A^2BE(CE)$.

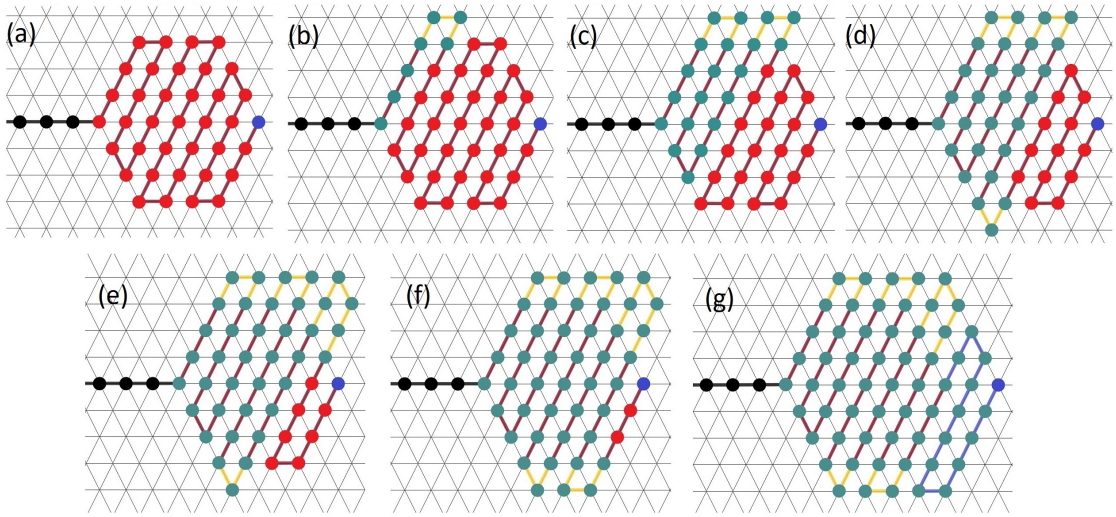


Figure 4.11: Algorithm `Caterpillar-3`: An example of construction of an even layer (i.e., layer 4) and the sequence of primitive rules used (the primitive rules have been illustrated with yellow links). The blue links illustrates the Zig-Zag motion process performed by the *leader* (i.e., the blue particle).

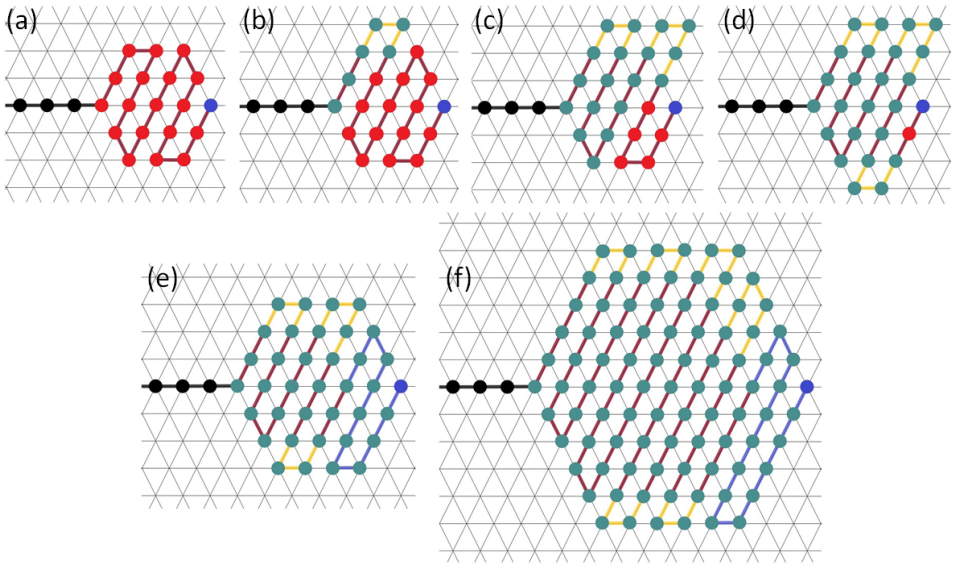


Figure 4.12: Algorithm `Caterpillar-3`: An example of construction of an odd layer (i.e., layer 3) and the sequence of primitive rules used (the primitive rules have been illustrated with yellow links). The blue links illustrates the Zig-Zag motion process performed by the *leader* (i.e., the blue particle).

4.4.3.4 Complexity

In this section, we calculate the complexity of Algorithm `Caterpillar-3`.

Theorem 6. *The total number of movements of Algorithm* `Caterpillar-3` *is* $\Theta(n^2)$.

Proof. First observe when a Zig-Zag motion process is performed, all n particles make the same number of movements.

On the other hand, when any of the operations shown in Figure 4.10 is performed, none of the particles ahead of the particle shown in red in Figure 4.10 (a.1)-(e.1) performs any movement. This means that a particle that is not involved in any of these operations performs a total of $n-1$ movements. This is for example the case of the last k particles in the line, where $k = 2h + 5$ is the number of empty cells that are occupied during the last Zig-Zag motion process to complete $HEX(h)$.

In other words, the total number of moves $T(n)$ performed by all the particles is $(n - k)(n-1) < T(n) < n(n-1)$, proving the theorem. □

4.4.4 Algorithm `Caterpillar-4`

In this section, we first briefly present the overall behaviour of Algorithm `Caterpillar`-4, and we describe the details of the algorithm including the behaviour of the particles in each state. We then prove its correctness and analyze its complexity.

4.4.4.1 Informal Description

In Algorithm `Caterpillar-4`, a particle can be in one of the three following states: *follower*, *leader*, and *terminated*. The idea behind Algorithm `Caterpillar-4` is to let a caterpillar form each layer of the hexagonal ball only following the movements of the *leader*. More specifically, the *leader* forms the hexagonal ball layer by layer by moving in a boustrophedon path from West to East and pulling the rest of the caterpillar behind (see Fig. 4.13). The length of the sub-path that should be traversed to form layer $k+1$ is equal to the length of the sub-path that should be traversed to form layer k plus six. The details of the *leader's* behaviour is described in the following section. Fig. 4.8 shows a general view of how a hexagonal ball is developed using Algorithm `Caterpillar`-4.

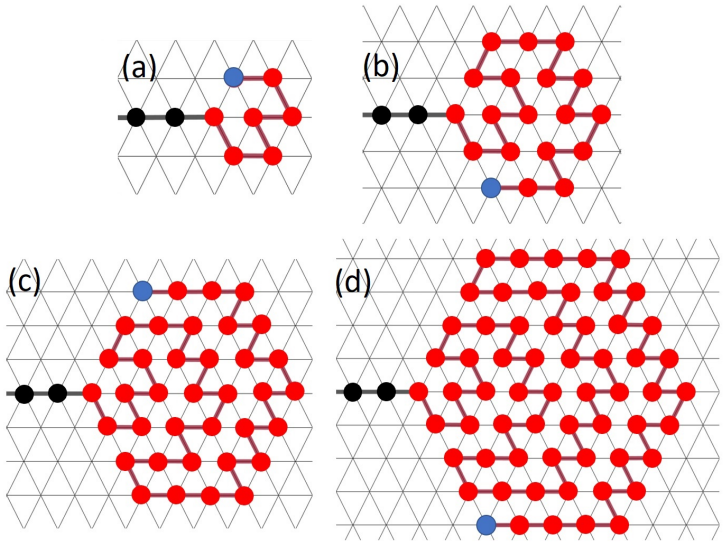


Figure 4.13: Algorithm `Caterpillar`-4. The *leader* particle is shown in blue. The particles occupying a node in the hexagonal ball are shown in red. Other particles are shown in black.

4.4.4.2 Detailed Description

Based on Algorithm `Caterpillar`-4 (see Algorithm 15), a hexagonal ball is developed layer by layer toward East, as can be seen in Fig. 4.8. As already mentioned, all layers of the hexagonal ball are formed through the movement strategy of the *leader*. The *leader* follows two different sets of movement rules to form odd layers and even layers. Since in our implementation of Algorithm `Caterpillar`-4 the first movement of the *leader* is performed along its South-East port, once a *follower* enters the area of the target shape, its following bonded neighbor is located on its South-East. Therefore, once the *tail* particle's bonded neighbor is located on its South-East, it understands that it has entered the area of the hexagonal ball because of being pulled by its following bonded neighbor, and sends a *Termination* message. This message is forwarded toward the *leader* and makes all particles terminated one by one. In this way, all the particles terminate, and the algorithm halts. However, if the *leader* does not receive a *Termination* message, it extends the shape based on the following set of rules (i.e., a set of conditions associated with layer number and their corresponding movement actions):

- If the layer that should be completed is odd:
  - **Rule A**: To start the formation of the layer, the *leader* takes one step along South-East (port 5).
  - **Rule B**: If the *leader's* preceding bonded neighbor is located on its North-West (port 2), it moves one step along its East (port 0).

- **Rule C**: If the *leader's* preceding bonded neighbor is located on its West (port 3), it first checks if the adjacent node that is accessible through its North-West port (port 2) is unoccupied. If the mentioned node is unoccupied, the *leader* moves to it. Otherwise, the *leader* takes one step along its East (port 0).
- **Rule D**: If the *leader's* preceding bonded neighbor is located on its South-East (port 5), it first checks if the adjacent node that is accessible through its West port (port 3) is unoccupied. If the mentioned node is unoccupied, the *leader* moves to it. Otherwise, the *leader* moves one step along its East (port 0).
- **Rule E**: If the *leader's* preceding bonded neighbor is located on its East (port 0), it first checks if the adjacent node that is accessible through its West port (port 3) is unoccupied. If the mentioned node is occupied, the *leader* takes one step along its North-East (port 1). Otherwise, as long as the leader has a neighbor on its South-West (port 4) which itself has a neighbor on its West, the *leader* takes one step along its West. If at a point, there is no such a neighbor, the *leader* considers that the current odd layer has been successfully completed. To this end, the *leader* sets the variable *Layer* to "EVEN".
- **Rule F**: If the *leader's* preceding bonded neighbor is located on its South-West (port 4), it moves one step toward its West (port 3).

– If the layer that should be completed is even:

- **Rule G**: To start the formation of the layer, the *leader* takes one step along North-East (port 1).
- **Rule H**: If the *leader's* preceding bonded neighbor is located on its South-West (port 4), it moves one step along its East (port 0).
- **Rule I**: If the *leader's* preceding bonded neighbor is located on its West (port 3), it first checks if the adjacent node that is accessible through its South-West port (port 4) is unoccupied. If the mentioned node is unoccupied, the *leader* moves to it. Otherwise, the *leader* takes one step along its East (port 0).
- **Rule J**: If the *leader's* preceding bonded neighbor is located on its North-East (port 1), it first checks if the adjacent node that is accessible through its West port (port 3) is unoccupied. If the mentioned node is unoccupied, the *leader* moves to it. Otherwise, the *leader* moves one step along its East (port 0).
- **Rule K**: If the *leader's* preceding bonded neighbor is located on its East (port 0), it first checks if the adjacent node that is accessible through its West port (port 3) is unoccupied. If the mentioned node is occupied, the *leader* takes one step along its South-East (port 5). Otherwise, as long as the leader has a neighbor on its North-West (port 2) which itself has a neighbor on its West, the *leader* takes one

step along its West. If at a point, there is no such a neighbor, the *leader* considers that the current even layer has been successfully completed. To this end, the *leader* sets the variable *Layer* to "ODD".

– **Rule L**: If the *leader's* preceding bonded neighbor is located on its North-West (port 2), it moves one step toward its West (port 3).

4.4.4.3 Correctness

In this section, we show that Algorithm `Caterpillar-4` solves the problem of forming a hexagonal ball. Let us assume that the number of particle allows to form exactly a complete hexagonal ball.

Theorem 7. *Algorithm* `Caterpillar-4` *forms a (complete) hexagonal ball.*

Proof. We prove the theorem by induction on the number of layers. For the basis of the induction, observe that $HEX(1)$ is trivially formed by the *leader* performing the following sequence of rules: $ABCDCD$.

Let $j > 1$; observe that the sequence of rules followed by the algorithm to form $HEX(j)$ from $HEX(j-1)$ is different depending on whether j is even or odd; hence, we will consider the two cases separately. starting with $j = 2i$ even.

Case 1: Let $HEX(2i-1)$ be fully formed, and consider the next layer $2i$.

By construction, the following sequence of rules is performed in order (from left to right) by the *leader* particle to form layer $2i$: $GHI^{2i-1}(IJ)^{2i+1}(KL)^{2i-1}K^{2i-1}$

In order to form $HEX(2i)$, Algorithm `Caterpillar`-4 extends $HEX(2i-1)$ from west to east as depicted in figure 4.8. As shown in the figure, the caterpillar extends the shape from the top (i.e., North) side and bottom (i.e., South) side of $HEX(2i-1)$ for one level while it extends the shape from North-Eastern side and South-Eastern side of $HEX(2i-1)$ for two levels.

The first side of $HEX(2i)$, formed by the movement strategy of the leader, is the North-Western side. This is done simply by performing rule G, according to which, the leader moves to its North-Eastern adjacent node.

After performing this rule, the leader is located on the North-Western corner of the $HEX(2i)$. Next, it moves for $2i$ steps in the East direction to form the Northern side of $HEX(2i)$. These $2i$ movements include one movement following rule H (as when the leader

Algorithm 15 `Caterpillar-4` - States and Actions

Leader:

- Initialize variable ***Layer*** to ODD ▷ the first layer that should be formed is layer 1 (an ODD layer)
if (receive *Termination*) **then**
- Become *terminated*
end if
if (*Layer* is EVEN) **then**
- Start the formation of the layer by taking one step along port 1
then keep moving according to the following conditions
if (bonded to the preceding bonded neighbor through port 4) **then**
- Take one step along port 0
else if (bonded to the preceding bonded neighbor through port 3) **then**
if (the adjacent node accessible through port 4 is unoccupied) **then**
- Take one step toward port 4
else
- Take one step toward port 0
end if
else if (bonded to the preceding bonded neighbor through port 1) **then**
if (the adjacent node accessible through port 3 is unoccupied) **then**
- Take one step toward port 3
else
- Take one step toward port 0
end if
else if (bonded to the preceding bonded neighbor through port 0) **then**
if the adjacent node accessible through port 3 is unoccupied and the adjacent node accessible through port 2 is occupied) **then**
if (the neighbor that is accessible through port 2 has a neighbor on its port 3) **then**
- Take one step toward port 3
else
- Set Layer to ODD
end if
else
- Take one step toward port 5
end if
else if (bonded to the preceding bonded neighbor through port 2) **then**
- Take one step toward port 3
end if
end if
if (*Layer* is ODD) **then**
- Start the formation of the layer by taking one step along port 5
if (bonded to the preceding bonded neighbor through port 2) **then**
- Take one step along port 0

Algorithm 15 `Caterpillar-4` - States and Actions (continued)

Leader (continued):

else if (bonded to the preceding bonded neighbor through port 3) **then**
if (the adjacent node accessible through port 2 is unoccupied) **then**
- Take one step toward port 2

else
- Take one step toward port 0

end if
else if (bonded to the preceding bonded neighbor through port 5) **then**
if (the adjacent node accessible through port 3 is unoccupied) **then**
- Take one step toward port 3

else
- Take one step toward port 0

end if
else if (bonded to the preceding bonded neighbor through port 0) **then**
if (it is the first layer) **then**
- Set Layer to EVEN

else if (the adjacent node accessible through port 3 is unoccupied and the adjacent node accessible through port 4 is occupied) **then**
if (the neighbor that is accessible through port 4 has a neighbor on its port 3) **then**
- Take one step toward port 3

else
- Set Layer to EVEN

end if
else
- Take one step toward port 1

end if
else if (bonded to the preceding bonded neighbor through port 4) **then**
- Take one step toward port 3

end if
end if

Follower:

if (you are *tail*, and your bonded neighbor is on your South-East) **then**
- Send *Termination* to the following bonded neighbor
- Become *terminated*

end if

Terminated:

- Do nothing

is on North-Western corner, its preceding bonded neighbor of the leader is located on its South-West, rule H is triggered, according to which, the leader takes one step along the East port), followed by $2i-1$ consecutive movements following rule I (as the preceding bonded neighbor of the leader is located on its West after each time it moves East, rule I is triggered, according to which, as the adjacent node accessible through South-West port is occupied by the particles of layer $2i-i$, the leader takes one step along the East port). After performing these $2i-1$ movements, the leader is located on the North-Eastern corner of $HEX(2i)$ and it starts forming the North-Eastern side of $HEX(2i)$.

As already mentioned, according to Algorithm `Caterpillar`-4, in order to form this side of $HEX(2i)$, $HEX(2i-1)$ is extended in the direction of its North-Eastern side for two levels. To do so, the leader performs $2(2i)$ movements based on which it alternates between moving one step in the direction of South West following rule I and moving one step in the East direction following rule J. More specifically, when the leader is on the North-Eastern corner of $HEX(2i)$, it needs to fill its South-Western and South-Eastern adjacent nodes. So, it starts with the South-Western adjacent node following rule I (as the preceding bonded neighbor of the leader is located on its West, rule I is triggered, according to which, as the adjacent node accessible through the South-West port is unoccupied, the leader takes one step along the South-West port) and then performs rule J in order to fill the other mentioned node (as after performing rule I, the preceding bonded neighbor of the leader is located on its North-East, rule J is triggered, according to which, as the adjacent node accessible through the West port is occupied, the leader takes one step along the East port). To fully form the North-Eastern side of $HEX(2i)$, the leader repeats this pair of rules (i.e., I and J) for $2i-1$ more times in the same order (i.e., in total $2i$ times) which results in placing the leader on the Eastern corner of $HEX(2i)$ while its preceding bonded neighbor is located on its West. In this way, the North-Eastern side of $HEX(2i-1)$ is extended for two levels and the North-Eastern side of $HEX(2i)$ is formed.

Then, the leader forms the South-Eastern side by extending the South-Eastern side of the $HEX(2i-1)$ for two levels. To start forming this side, the leader moves one step along its South-East port following rule I and then moves one step in the direction of West following rule J. After performing the last rule J, the leader performs $2(2i-1)$ movements, alternating between moving in the South-West direction and moving in the East direction. More specifically, after performing rule J, the leader needs to fill its South-Western and South-Eastern adjacent nodes. So, it starts with the South-Eastern adjacent node following rule K (as the preceding bonded neighbor of the leader is located on its East, rule K is triggered, according to which, as the adjacent node accessible through the West port is occupied, the leader takes one step South-East) and then performs rule L in order to fill the other mentioned node (as after performing rule K, the preceding bonded neighbor of the leader is located on its North-West, rule L is triggered, according to which, the leader takes one step along the

West port). To fully form the South-Eastern side of $HEX(2i)$, the leader performs this pair of rules (i.e., K and L) for $2i-2$ more times in the same order (i.e., in total $2i-1$ times) which results in placing the leader's preceding bonded neighbor on the South-Eastern corner of $HEX(2i)$ while the leader is located on its West of the corner. In this way, the South-Eastern side of $HEX(2i-1)$ is extended for two levels and the South-Eastern side of $HEX(2i)$ is formed.

Afterwards, the leader starts forming the Southern side of $HEX(2i)$ by moving West for $2i-1$ steps. To this end, the leader for $2i-1$ consecutive steps keeps moving West following rule K (as the preceding bonded neighbor of the leader is located on its East, rule K is triggered, according to which, as the adjacent node accessible through the West port is unoccupied, the leader takes one step West). After performing these sequence of rules, the leader is located on the South-Western corner of $HEX(2i)$ and this means the South-Western side of $HEX(2i)$ is also completed and $HEX(2i)$ is successfully formed. As a consequence, since $HEX(1)$ is formed, the algorithm forms also $HEX(2)$.

Case 2: Let $HEX(2i)$ be fully formed, and consider the next layer $2i+1$.

The following rules are performed in order (from left to right) by the *leader* particle to form layer $2i+1$: $ABC^{2i}(CD)^{2i+2}(EF)^{2i}E^{2i}$

As mentioned before, Algorithm `Caterpillar`-4 extends $HEX(2i)$ from West to East as depicted in figure 4.8 to form $HEX(2i+1)$. As can be seen, the caterpillar extends the shape from the top (i.e., North) and bottom (i.e., South) sides for one level while it extends the shape from North-Eastern and South-Eastern sides of $HEX(21-1)$ for two levels.

The first side of $HEX(2i+1)$ that is formed by the movements of the leader is the South-Western side. This is done simply by performing rule A, according to which, the leader moves to its South-Eastern adjacent node.

After performing this rule, the leader is located on the South-Western corner of $HEX(2i+1)$. Next, it moves for $2i+1$ steps in the East direction to form the Southern side of $HEX(2i+1)$. These $2i+1$ movements include one movement based on rule B (as when the leader is on the South-Western corner, its preceding bonded neighbor is located on its North-West, rule B is triggered, according to which, the leader takes one step along the East port), followed by $2i$ consecutive movements based on rule C (as the preceding bonded neighbor of the leader is located on its West after each time it moves along East, rule C is triggered, according to which, as the adjacent node accessible through North-West port is occupied by the particles of layer 2i, the leader takes one step along the East port). After performing these $2i$ movements, the leader is located on the South-Eastern corner of $HEX(2i+1)$ and starts forming the South-Eastern side of $HEX(2i+1)$.

In order to form this side of $HEX(2i+1)$, $HEX(2i)$ is extended in the direction of its South-Eastern side for two levels. To this end, the leader performs $2(2i+1)$ movements, in which it alternates between moving in the North-West direction and moving in the East direction. More specifically, when the leader is on the South-Eastern corner of $HEX(2i+1)$, it needs to fill its North-Western and North-Eastern adjacent nodes. So, it starts with the North-Western adjacent node following rule C (as the preceding bonded neighbor of the leader is located on its West, rule C is triggered, according to which, as the adjacent node accessible through the North-West port is unoccupied, the leader takes one step along the North-West port) and then performs rule D in order to fill the other mentioned node (as after performing rule C, the preceding bonded neighbor of the leader is located on its South-East, rule D is triggered, according to which, as the adjacent node accessible through West port is occupied, the leader takes one step along East port). In order to fully form the South-Eastern side of the $HEX(2i)$, the leader repeats this pair of rules (i.e., C and D) for $2i$ more times in the same order (i.e., in total $2i+1$ times) which results in placing the leader on the Eastern corner of $HEX(2i+1)$ while its preceding bonded neighbor is located on its West. In this way, the South-Eastern side of $HEX(2i)$ is extended for two levels and the North-Eastern side of $HEX(2i+1)$ is formed.

Next, the leader forms the North-Eastern side by extending the North-Eastern side of $HEX(2i)$ for two levels. To start forming this side, the leader moves one step along its North-West port following rule C and then moves one step in the direction of West following rule D. Then, the leader performs $2(2i)$ movements based on which it alternates between moving one step in the direction of North-East following rule E and moving one step in the direction of West following rule F. More specifically, after performing rule D, the leader needs to fill its North-Western and North-Eastern adjacent nodes. So, it starts with the North-Eastern adjacent node following rule E (as the preceding bonded neighbor of the leader is located on its East, rule E is triggered, according to which, as the adjacent node accessible through the West port is occupied, the leader takes one step along the North-East) and then performs rule F in order to fill the other mentioned node (as after performing rule E, the preceding bonded neighbor of the leader is located on its South-West, rule F is triggered, according to which the leader takes one step along the West port). In order to fully form the North-Eastern side of $HEX(2i+1)$, the leader performs this pair of rules (i.e., E and F) for $2i-1$ more times in the same order (i.e., in total $2i$ times) which results in placing the leader's preceding bonded neighbor on the North-Eastern corner of $HEX(2i+1)$ while the leader is located on the West of the corner. In this way, the North-Eastern side of $HEX(2i)$ is extended for two levels and the North-Eastern side of $HEX(2i+1)$ is formed.

Afterwards, the leader starts forming the Northern side of $HEX(2i+1)$ by moving toward West for $2i+1$ steps. To do so, the leader first following rule F moves one step along port West (as the preceding bonded neighbor of the leader is on its South-West when it is on the

North-Eastern corner of the $HEX(2i+1)$, rule F is triggered, according to which the leader takes one step West and then for $2i$ consecutive steps it keeps moving along West following rule E (as the preceding bonded neighbor of the leader is located on its East after performing rule F, rule E is triggered, according to which, as the adjacent node accessible through the West port is unoccupied, the leader takes one step West). After performing these sequence of rules, the leader is located on the North-Western corner of $HEX(2i+1)$ and this means the North-Western side of $HEX(2i+1)$ is also completed and $HEX(2i+1)$ is successfully formed. This completes the proof of the Theorem. □

An example can be seen in Figure 4.14-b, where a hexagonal ball has been completely extended from layer 3 to layer 4 by the *leader* particle by performing the following sequence of rules: $GHI^3(IJ)^5(KL)^3K^3$. As another example, Figure 4.14-d shows the situation in which a hexagonal ball has been completely extended from layer 5 to layer 6 by performing the following sequence of rules: $GHI^5(IJ)^7(KL)^5K^5$.

Figure 4.14-a shows how a hexagonal ball has been completely extended from layer 2 to layer 3 by the *leader* particle through performing the following sequence of rules: $ABC^2(CD)^4(EF)^2E^2$. As another example, Figure 4.14-c shows the situation in which a hexagonal ball has been completely extended from layer 4 to layer 5 by the *leader* particle through performing the following sequence of rules: $ABC^4(CD)^6(EF)^4E^4$.

4.4.4.4 Complexity

In this section, we calculate the complexity of Algorithm `Caterpillar-4`.

Theorem 8. *The total number of movements in Algorithm* `Caterpillar-4` *is* $\Theta(n^2)$.

Proof. The *leader* moves starting from its original position 1 through all the positions of $HEX(h)$ once and only once, and then stops. That is, it performs $\sigma(h) - 1 = n - 1$ moves. Every particle x, whose initial position is $1 < p(x) \leq n$ first performs $p(x) - 1$ movements to reach position 1; it then follows the path travelled by the leader; it finally stops after $n - p(x) + 2$ moves, with all the $p(x) - 1$ particles ahead of it already stopped. That is, each particle x, including the leader, performs $n - 1$ moves, proving the theorem. □

4.4.5 Algorithm `Caterpillar-5`

In this section, we first briefly present the overall behaviour of Algorithm `Caterpillar-5`, and we describe the details of the algorithm including the behaviour of the particles in each state. We then prove its correctness and analyze its complexity.

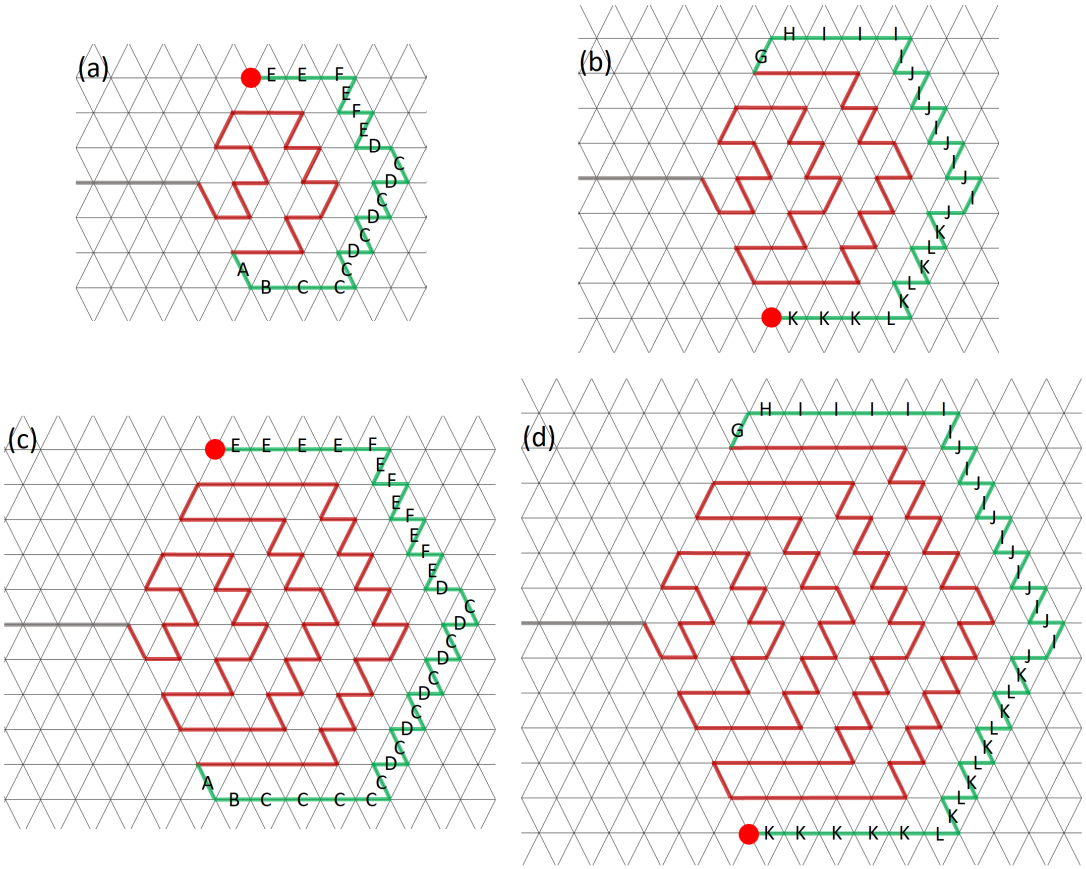


Figure 4.14: Algorithm `Caterpillar`-4: Some examples of construction of odd and even layers and the sequence of rules used. The red particle is the *leader*. The other particles are not shown in the figure for simplicity. The path that the *leader* traversed to form the latest layer has been illustrated in green. The red area is the area of the hexagon made by the *leader's* movements. The black line shows the line of particles that have not yet occupied any node in the area of the target shape.

4.4.5.1 Informal Description

In Algorithm `Caterpillar-5`, a particle can be in one of the three following states: *follower*, *leader*, and *terminated*.

The idea behind Algorithm `Caterpillar-5` is to let a caterpillar form each layer of the hexagonal ball through the movement strategy of the *leader*. In this algorithm, the *leader* forms the hexagonal ball layer by layer by moving along the border of the previous complete layer and pulling the rest of the caterpillar, like in Algorithm `Caterpillar`-4. The difference, with respect to Algorithm `Caterpillar`-4, is the shape of the movement followed by the leader, which performs a series of movements to form layer $i + 1$ by coating layer i (see Fig. 4.15).

The details of the *leader's* behaviour are described in the following section. Fig. 4.4 shows a general view of how a hexagonal ball is developed using Algorithm `Caterpillar`-5.

4.4.5.2 Detailed Description

As already noted, in Algorithm `Caterpillar-5`, all layers of the hexagonal ball are formed through the movement strategy of the *leader*. The *leader* follows two different simple sets of movement rules to form odd layers and even layers. According to this algorithm, the *tail* particle realizes that it has entered the area of the target shape once its bonded neighbor has one neighbor on its South-East and one neighbor on its North-East. When the *tail* particle enters the area of the target shape, it sends a *Termination* message to its bonded neighbor. This message is forwarded toward the *leader* and makes all particles terminated one by one. In this way, all the particles terminate, and the algorithm halts. However, if the *leader* does not receive a *Termination* message, it extends the shape based on the following set of rules (i.e., a set of conditions associated with layer number and their corresponding movement actions):

- The *leader* initializes the formation process by forming the first layer (layer 1) in the following way: it first moves one step along South-East (port 5), then one step along East (port 0), then one step along North-West (port 2), then one step along East (port 0), then one step along North-East (port 2), and finally one step along West (port 3).
- If the first layer has already been completed:
  - If the layer that should be completed is even:
    - **Rule A**: the *leader* takes one step along West (port 3), and then one step along North-East (port 1). The purple lines in figure 4.15-c show the path the

leader traverses based on this rule.

* **Rule B**: the *leader* keeps moving along the border of the previous layer in a clockwise manner to coat the previous layer entirely. The *leader* realizes that the layer has been completely coated when it has three non-bonded *retired* neighbours on its East, North-East, and North-West. The blue lines in figure 4.15-e show the path the *leader* traverses based on this rule.

– If the layer that should be completed is odd:

* **Rule C**: the *leader* takes one step along West (port 3), and then one step along South-East (port 5). The orange lines in figure 4.15-h show the path the *leader* traverses based on this rule.
* **Rule D**: the *leader* keeps moving along the border of the previous layer, in a counter-clockwise manner to coat the previous layer entirely. The *leader* realizes that the layer has been completely coated when it has three non-bonded *retired* neighbours on its East, South-East, and South-West. The green lines in figure 4.15-j show the path the *leader* traverses based on this rule.

4.4.5.3 Correctness

In this section, we show that Algorithm `Caterpillar-5` solves the problem of forming a hexagonal ball. Let us assume that the number of particle allows to form exactly a complete hexagonal ball.

Theorem 9. *Algorithm* `Caterpillar-5` *forms a (complete) hexagonal ball.*

Proof. We prove the theorem by induction on the number of layers. For the basis of the induction, by careful examination of the algorithm, it is easy to see that $HEX(1)$ is formed by the *leader* performing the following sequence of movements: one step along South-East; one step along East; one step along North-West; one step along East; one step along North-East; and finally, one step along West (see Figure 4.16-a). After this, it is easy to see that the *leader* is located on the North-Eastern adjacent node of the west-most corner.

Let $j > 1$; observe that the sequence of rules followed by the algorithm to form $HEX(j)$ from $HEX(j-1)$ is different depending on whether j is even or odd; hence, we will consider the two cases separately, starting with $j = 2i$ even.

Case 1: Let $HEX(2i-1)$ be fully formed, and consider the next layer $2i$.

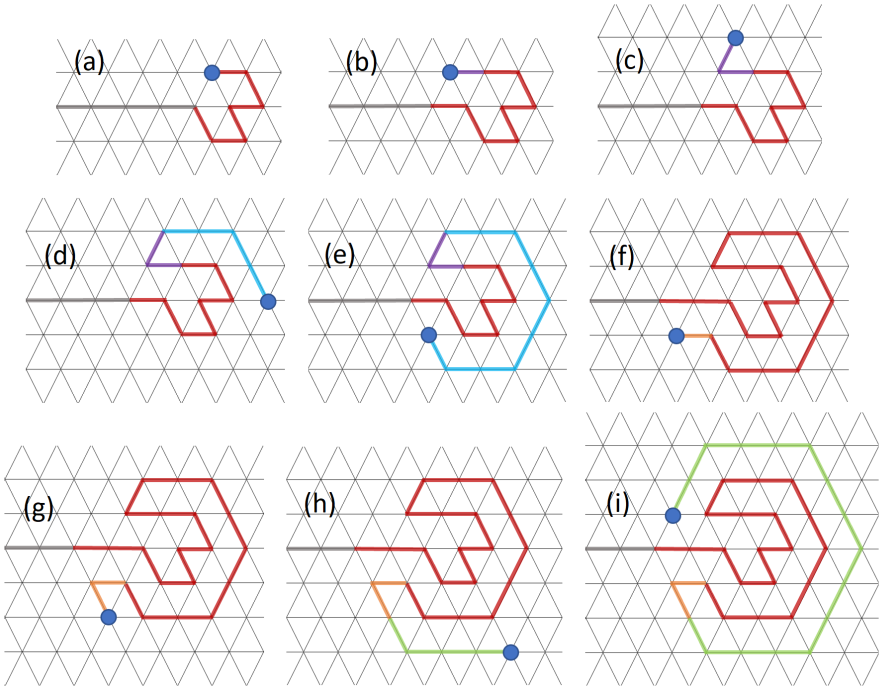


Figure 4.15: Algorithm `Caterpillar-5`. The blue particle is the *leader*. The other particles have not been shown in the figure for simplicity. The purple lines and orange lines show rules A and C, respectively. The light blue lines and green lines show rules B and D, respectively. The red area is the area of the hexagon made by the *leader's* movements. The black line shows the line of particles that have not yet occupied any node in the area of the target shape.

By construction, the sequence of rules A, B is performed in order (from left to right) by the *leader* particle to form layer $2i$.

In order to form $HEX(2i)$, Algorithm `Caterpillar-5` extends $HEX(2i-1)$ by filling all $6(2i)$ nodes coating layer $2i-1$ as depicted in figure 4.4. The *leader* manages the shape extension by moving around the perimeter of $HEX(2i-1)$ and pulling its preceding line segment of particles.

The *leader* starts by performing rule A, according to which, the *leader* moves one step along West, and then it moves one step along North-East. In this way, the *leader's* position is adjusted to have at least one non-bonded neighbor occupying a node of layer $2i-1$ (i.e., its East and South-East neighbors). Afterwards, the *leader* performs rule B. According to this rule, the *leader* keeps moving clockwise around the perimeter of $HEX(2i-1)$ until it fully coats that layer. The *leader* is informed of this when it occupies a node with three non-bonded neighbors. In such a case, two of those non-bonded neighbors are located in layer $2i-1$, and the third is a follower particle located in the straight part of the caterpillar. This follower neighbor, which is located on the *leader's* North-West, represents the west-most corner of

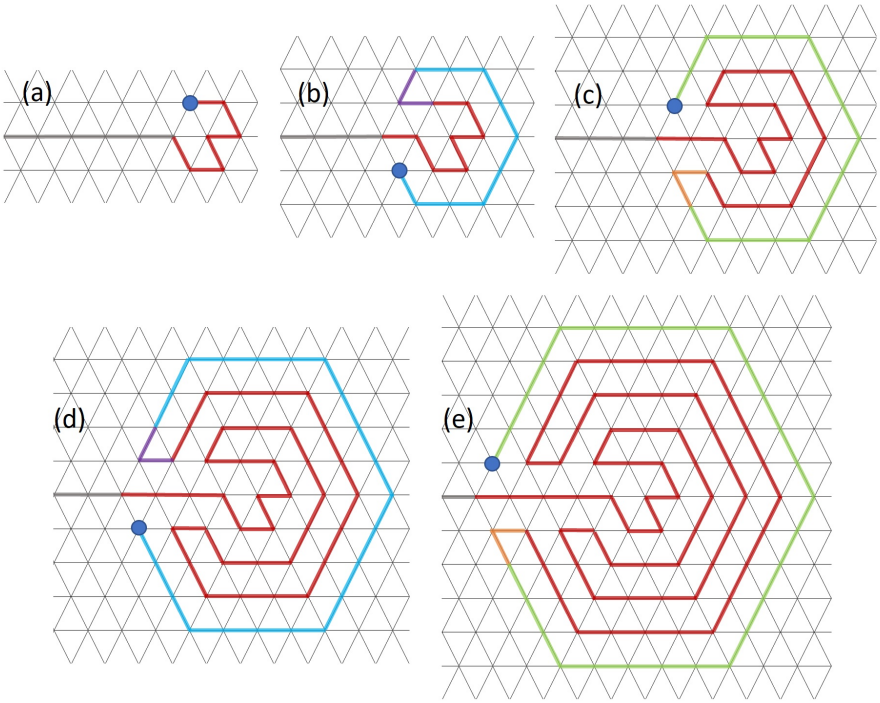


Figure 4.16: Algorithm `Caterpillar-5`: Some example of construction of odd and even layers. The blue particle is the *leader*. The other particles have not been shown in the figure for simplicity. The purple lines and orange lines show rules A and C, respectively. The light blue lines and green lines show rules B and D, respectively. The red area is the area of the hexagon made by the *leader*'s movements. The black line shows the line of particles that have not yet occupied any node in the area of the target shape.

$HEX(2i)$. After performing rule A, all the nodes coating $HEX(2i-1)$ are occupied by particles because of the worm-like strategy, and this means $HEX(2i)$ is successfully formed.

Case 2: Let $HEX(2i)$ be fully formed, and consider the next layer $2i+1$.

The following rules are performed in order (from left to right) by the *leader* particle to form layer $2i+1$: CD.

In order to form $HEX(2i+1)$, Algorithm `Caterpillar-5` extends $HEX(2i)$ by filling all $6(2i+1)$ nodes coating layer $2i$ as depicted in figure 4.4. The *leader* manages the shape extension by moving around the perimeter of $HEX(2i)$ and pulling its preceding line segment of particles.

The *leader* starts the process by performing rule C, according to which, the *leader* moves one step along West, and then it moves one step along South-East. In this way, the *leader's* position is adjusted to have at least one non-bonded neighbor occupying a node of layer $2i$ (i.e., its East and North-East neighbors). Next, the *leader* performs rule D. According to

this rule, the *leader* keeps moving counter-clockwise around the perimeter of $HEX(2i)$ until it fully coats that layer. The *leader* is informed of this when it occupies a node with three non-bonded neighbors. In such a case, two of those non-bonded neighbors are located in layer $2i$, and the third is a follower particle located in the straight part of the caterpillar. This follower neighbor, which is located on the *leader's* South-West, represents the west-most corner of $HEX(2i+1)$. After performing rule D, all the nodes coating $HEX(2i)$ are occupied by particles because of the worm-like strategy, and this means $HEX(2i+1)$ is successfully formed. □

An example can be seen in Figure 4.16-b, where starting from layer 1, layer 2 is formed by the caterpillar following rule A and rule B. Another example is shown in Figure 4.16-d, where a hexagonal ball has been completely extended from layer 3 to layer 4 by performing the same sequence of rules (i.e., AB).

Figure 4.16-c shows how a hexagonal ball has been completely extended from layer 2 to layer 3 by the *leader* particle through performing the following sequence of rules: CD. Another example is shown in Figure 4.16-e where a hexagonal ball has been completely extended from layer 4 to layer 5 by performing the same sequence of rules (i.e., CD).

4.4.5.4 Complexity

In this section, we calculate the complexity of Algorithm `Caterpillar-5`.

Theorem 10. *The total number of movements in Algorithm* `Caterpillar-5` *is* $\Theta(n^2)$.

Proof. The *leader*, starting from its original position 1, moves once and only once through all the positions of $HEX(h)$ except one for each layer $j > 1$. That is, it performs $\sigma(h)-(h-1) = n-h+1$ moves.
Every particle x with $1 < p(x) < n-h+1$ performs exactly the same number of moves as the leader. In fact, it performs $p(x)-1$ movements to reach position 1; it then follows the path travelled by the leader stopping after reaching the $p(x)-1$ particles ahead of it already stopped. That is, each of these particles performs $p(x)-1+n-h+1-(p(x)-1) = n-h+1$ moves, for a total of $(n-h)(n-h+1)$ movements. Of the last $h-1$ particles, each travels $n-h$ for a total of $(h-1)(n-h) = nh-h^2+h-n$ positions. Hence, the total number of movements is $\Theta(n^2)$ as claimed. □

4.5 Experimental Comparison

In section 4.4, we analysed the complexity of the five algorithms, establishing tight asymptotic bounds on the total number of movements performed by the particles during their execution. The analysis has shown that the $\Theta(n^{2.5})$ complexity of algorithm `Caterpillar-1` is asymptotically worse than that of all the other algorithms, which all have the same $\Theta(n^2)$ asymptotic complexity.

To further understand the difference, from an efficiency point of view, between these four algorithms, we have turned to experimental analysis.

In order to carefully assess the efficiency of the algorithms, we implement them using the AmoebotSim simulator[1], which is a visual simulator for the Amoebot model developed by the Self-Organizing Particle Systems (SOPS) Lab at Arizona State University and the University of Paderborn. The simulator and all five curling algorithms presented in this chapter are written in C++. We use the default scheduler of the AmoebotSim simulator which is a sequential one. As already mentioned, this scheduler activates one particle at a time.

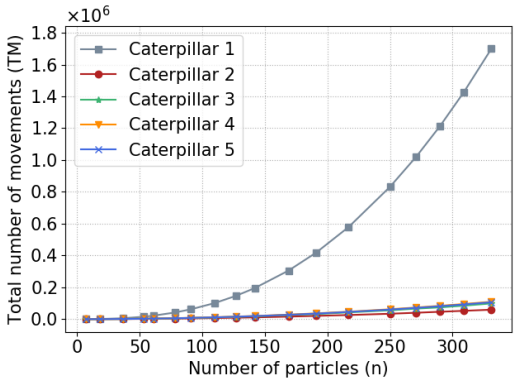
In this section, we compare the performance of Algorithms `Caterpillar-1`, `Caterpillar-2`, `Caterpillar-3`, `Caterpillar-4`, and `Caterpillar-5` based on the results of the experiments conducted in the AmoebotSim simulator. As already noted, in this chapter, we evaluate the performance of an algorithm based on its total number of movements TM.

Table 4.1 and figure 4.17 show the total number of movements performed by the algorithms under different lengths of the caterpillar (i.e., different numbers of particles constituting the caterpillar; n). From the table, figure 4.17-a, and figure 4.17-c, the performance of Algorithm `Caterpillar-1` is substantially lower than those of the other algorithms (i.e., it has the highest complexity). Furthermore, as the number of particles increases, the total number of movements grows for all algorithms and the difference between the performance of the algorithms increases.

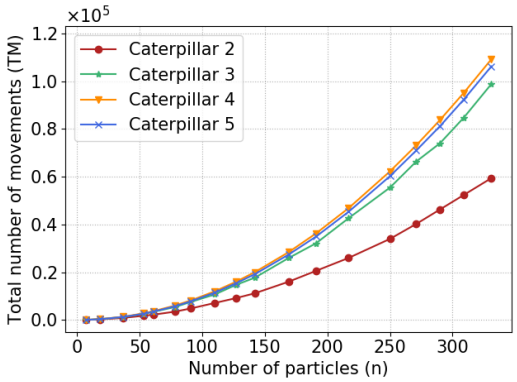
As seen in table 4.1 and figure 4.17, Algorithm `Caterpillar-2` has the lowest number of movements (i.e., the lowest complexity) when the number of particles is equal to or more than 13. With particle numbers less than 13, the performances of `Caterpillar-2`, `Caterpillar-3`, `Caterpillar-4`, and `Caterpillar-5` are very comparable. It is clear from the table, figure 4.17-b, and figure 4.17-d that the number of movements of Algorithm `Caterpillar-5` is slightly less than that of Algorithm `Caterpillar-4`. Excluding the setups with 37 and 38 particles, Algorithm `Caterpillar-5` never reaches a performance equal to or higher than that of Algorithm `Caterpillar-3` when the number of particles is more than 20; of course, their

[1] https://amoebotsim.readthedocs.io/en/latest/

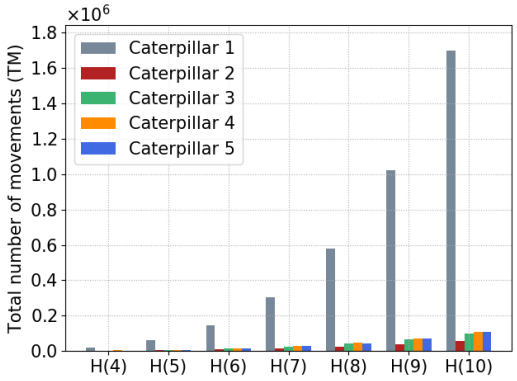
performance difference is negligible in the setups with not more than 38 particles. When the number of particles is less than or equal to 20, the performances of Algorithms `Caterpillar`-5 and `Caterpillar`-3 are very similar.



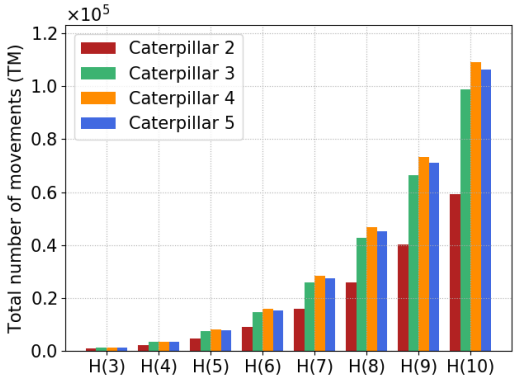
(a) The number of movements required to form hexagonal balls with respect to the number of particles constituting the caterpillar (for all five algorithms).



(b) A closer look to the number of movements required to form hexagonal balls with respect to the number of particles constituting the caterpillar (for four algorithms: C2, C3, C4, C5).



(c) The number of movements required to form complete hexagonal balls with different numbers of layers (for all five algorithms). H represents HEX.



(d) A closer look to the number of movements required to form complete hexagonal balls with different numbers of layers (for four algorithms: C2, C3, C4, C5). H represents HEX.

Figure 4.17: Total number of movements in each algorithm with respect to the number of particles.

Table 4.1: **Total number of movements in each algorithm with respect to the number of particles.** "#P" gives the number of particles. "Type" indicates the type of hexagonal ball that can be formed (Complete or Incomplete) with the given number of particles. "C" denotes that the number of particles is enough to form a complete hexagonal ball. "I" denotes that the number of particles is not sufficient to form a complete hexagonal ball, so an incomplete hexagonal ball can be formed.

#P	Type	`Caterpillar-1`	`Caterpillar-2`	`Caterpillar-3`	`Caterpillar-4`	`Caterpillar-5`
7	C: HEX(1)	50	40	42	42	42
9	I	109	72	62	72	63
11	I	199	104	98	110	99
12	I	257	120	120	132	120
13	I	324	136	144	156	143
15	I	488	168	198	210	195
17	I	695	200	260	272	255
18	I	816	216	294	306	288
19	C: HEX(2)	949	232	330	342	323
20	I	1094	248	352	380	340
21	I	1252	266	376	420	378
22	I	1423	286	411	462	418
23	I	1608	308	447	506	460
28	I	2749	455	682	756	700
37	C: HEX(3)	5818	866	1258	1332	1258
38	I	6247	922	1299	1406	1292
39	I	6695	978	1342	1482	1365
40	I	7162	1034	1398	1560	1440
41	I	7648	1090	1456	1640	1517
45	I	9792	1314	1752	1980	1845
53	I	15086	1762	2508	2756	2597
61	C: HEX(4)	21828	2210	3412	3660	3477
78	I	41487	3452	5384	6006	5694
91	C: HEX(5)	61928	4830	7568	8190	7826
97	I	73073	5550	8278	9312	8827
110	I	101169	7110	10680	11990	11440
127	C: HEX(6)	146573	9150	14692	16002	15367
142	I	195360	11167	17698	20022	19170
169	C: HEX(7)	305452	16012	25940	28392	27378
191	I	417921	20588	32148	36290	34953
217	C: HEX(8)	579216	25996	42658	46872	45353
250	I	831381	33958	55462	62250	60250
271	C: HEX(9)	1021206	40220	66382	73170	71002
290	I	1213626	46300	74056	83810	81200
309	I	1426424	52380	84780	95172	92391
331	C: HEX(10)	1699181	59420	98838	109230	106251

4.6 Conclusions and Future Directions

In this chapter, we proposed *Caterpillar* as a novel high-level leader-based programmable matter, and discussed its implementation in terms of the lower-level programmable particles of the Geometric Amoebot model.

The primitive operations of this high-level matter allow it to be easily programmed to perform complex operations. In this thesis, we studied the implementation of the primitive operation "curl" using deterministic algorithms for programmable particles. We designed five deterministic algorithms for the curl operation (Caterpillar-1, Caterpillar-2, Caterpillar-3, Caterpillar-4, Caterpillar-5), proved their correctness, determined analytically their complexity, and evaluated experimentally their performance.

Both the analytical and experimental analysis were performed considering the total number of movements made by the particles. The results show that Algorithm Caterpillar-2 displays the best performance, and Caterpillar-5 performs substantially worse than the other algorithms. Among the other three algorithms, Caterpillar-3 shows a better overall performance, followed by Caterpillar-5, which slightly performs better than Caterpillar-4. It is worth noting that all the presented algorithms require a small constant memory since, in all of them, the particles use a small number of states.

The correctness was formally established under the sequential activation scheduler, which is traditionally assumed for programmable particles; the proofs however can be extended to hold under the more general semi-synchronous scheduler.

As a future extension of this work, the most immediate and direct is the design and implementation of other potential operations for Caterpillar in addition to the "curl" operations, done in this chapter, and to the "flock" operation, whose implementation is trivial.

Once the basic set of operations for Caterpillars is available, the most important open problem is how to use these operations to program the caterpillar for complex tasks. Indeed, there are many applications in which we can efficiently use caterpillars, such as coating, coverage, and foraging. For example, an important and interesting application is "object coating". Algorithm Caterpillar-2 and Algorithm Caterpillar-5 presented in this thesis can be adjusted to be used for coating objects. All the algorithms designed in this chapter are deterministic. The use of non-determinism (e.g., randomization) for some applications is to be investigated.

It would also be interesting to calculate the exact memory cost and compare the curling algorithms from that point of view as a possible future direction.

Finally, the concept of implementing higher-level organisms in terms of lower-level ones can obviously be applied to other programmable matter models, opening a large new research direction.

Chapter 5

Conclusions and future directions

In this chapter, we summarize our conclusions and briefly present future research directions.

In this thesis, we studied multi-robot systems and programmable matter as two types of multi-agent systems. In multi-robot systems, the focus was on a recently developed robotic concept called 'mergeable nervous system' (MNS) [84, 140] which combines aspects of centralized and decentralized controls and benefits from both by allowing a robot team to establish a communication network through self-organization and yield control of their sensors and actuators temporarily to a single robot. On the other hand, in programmable matter, the focus of the thesis was on a novel high-level leader-based programmable matter (called Caterpillar) which can perform some primitive operations and make programming of programmable matter easier without sacrificing efficiency. An MNS-enabled approach, like the proposed high-level programmable matter, can be considered a high-level system which is guided by a brain robot.

In this thesis, we first investigated if an MNS-enabled approach is a good alternative for centralized and decentralized approaches in multi-robot systems. To this end, we experimentally investigated the performance of an MNS-enabled approach in a collective perception task. We demonstrated that our MNS-enabled approach is high-performing, fault tolerant, and scalable, so it is an appropriate approach for multi-robot systems. As the first goal of the thesis, for the first time, we comprehensively tested and compared centralized and decentralized control architectures with one formalized set of systematic metrics in a multi-robot coverage task. In our comparison, we used an MNS-enabled approach as an appropriate representative approach for hybrid control. As the second goal of the thesis, we proposed Caterpillars as a high-level leader-based programmable matter. As designing individual-level behaviors for a collective mission is not an easy task, high-level programmable organisms with a set of primitive operations can make the process of designing collective behaviors easier. In

the following, we briefly summarize our contributions and give a brief outlook to the future research directions.

In Chapter 2, we assessed the performance of an MNS-enabled approach called HIER in a collective perception task where a robot team should collectively perceive the density of objects distributed in the environment. We experimentally compared the performance of the HIER approach with that of three state-of-the-art decentralized approaches. The results show that the HIER approach is more accurate, more consistent, and faster than the decentralized approaches. We also empirically tested the scalability and fault tolerance of the presented approaches. The results show that the HIER approach does not substantially suffer from scalability or fault tolerance disadvantages compared to the decentralized approaches. In summary, the results of the chapter verify that the MNS-enabled approach can successfully combine aspects of centralized and decentralized approaches.

A possible direction for the future is to investigate the effects of various conditions on the performance of the approaches (e.g., the shape of the arena, different sizes and shapes of obstacles, and features of the stochastic field). Future work on collective perception also could study performance optimization. For example, online adaptation methods which improve the performance of the approaches can be used by any approach in the scenario we described in Chapter 2. Another possible direction for the future is to study other types of failures, such as motion control errors, odometry errors, or complete failure of robots. For more details on the future directions on collective perception please see the Future work section of Chapter 2.

In Chapter 3, we tested the performance of four approaches to multi-robot coverage with varying degrees of (de)centralization, and compared them using experiments and analysis. In our comparison, we used an MNS-enabled approach as the hybrid formation approach, a predetermined control approach, a centralized formation approach and a fully decentralized approach. The results show that the predetermined control approach displays the highest performance and the hybrid formation approach shows a slightly lower performance than the centralized formation approach. The results also indicate that the hybrid formation approach shows scalability and fault tolerance comparable with the fully decentralized approach. In terms of resilience to challenging environments, the results show that unexpectedly more centralized approaches might perform better than the fully decentralized approach in messy environments with randomized obstacles. Furthermore, regarding the possible limitations associated with UAV supervisors, we concluded that the limited operating time of UAVs is more restrictive than their energy efficiency. For more information about the results, please see the Conclusion section of Chapter 3. In summary, the results of the chapter provide the designers of multi-robot systems with a better understanding of the relevant task conditions and constraints when selecting the degree of (de)centralization for multi-robot

coverage control.

Future work should study the performance of the mentioned approaches in a variety of more challenging environments (e.g., environments with different sizes and shapes, environments with more obstacles of different sizes and shapes). It would be interesting to include other approaches in our comparison as a possible future direction. For example, it would be interesting to use a beacon-based approach that uses some static UAVs, which can see ground robots and communicate with them, as beacons in order to reduce the repeated coverage by giving some guidance to the ground robots according to the distribution of their peers in the arena. The effects of ground robots' sensing range on coverage completeness could also be investigated in the future. This can be studied by using different grid resolutions in discrete environments or using different sensor ranges. The results indicate that in order to achieve high coverage completeness in real applications, it is essential to develop autonomous recharging for robots and UAVs. This can be investigated in the future. For more details on the future work on comparing control architectures in multi-robot coverage please see the Discussion section of Chapter 3.

In Chapter 4, we presented *Caterpillar* as a novel high-level leader-based programmable matter made of lower level Amoebot particles and discussed its implementation from a microscopic level. By considering some primitive operations for this high-level matter, we have allowed it to be easily programmed in order to perform complex tasks. We demonstrated how the primitive operation "curl" of a caterpillar can be implemented using deterministic algorithms for programmable particles. More specifically, we first designed five deterministic algorithms for the curl operation namely Caterpillar-1, Caterpillar-2, Caterpillar-3, Caterpillar-4, Caterpillar-5. Then, we proved their correctness, analytically calculated their complexity in terms of total number of movements of the particles, and finally experimentally assessed their performance in terms of the total number of movements using AmoebotSim simulator. The results demonstrate that Algorithm Caterpillar-2 outperforms other presented algorithms and Algorithm Caterpillar-1 shows a performance substantially worse than the others. Among the other three algorithms, generally, Algorithm Caterpillar-3 performs better than Algorithm Caterpillar-5, which in turn slightly outperforms Algorithm Caterpillar-4.

A possible direction for the future is to design and implement the other potential operations for Caterpillar in addition to the "curl" operations presented in Chapter 4. Another possible direction for the future is to investigate the performance of Caterpillar in complex tasks such as coating objects, coverage, and foraging. All the algorithms designed in Chapter 4 are deterministic. However, it would be interesting to investigate the performance of probabilistic algorithms in tasks like coverage, foraging, and leader protection. Therefore, probabilistic algorithms could also be investigated in the future. Finally, the concept of implementing higher level organisms can clearly be applied to other programmable matter models

which will open a large new research direction. For more details on the future directions on Caterpillars please see the Conclusion section of Chapter 4.

References

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

[2] D. Albani, D. Nardi, and V. Trianni. Field coverage and weed mapping by UAV swarms. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4319–4325. IEEE, 2017.

[3] M. Allwright, N. Bhalla, C. Pinciroli, and M. Dorigo. ARGoS plug-ins for experiments in autonomous construction. Technical Report TR/IRIDIA/2018-007, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2018.

[4] M. Allwright, N. Bhalla, C. Pinciroli, and M. Dorigo. Simulating multi-robot construction in ARGoS. In *Swarm Intelligence – Proceedings of ANTS 2018 – Eleventh International Conference*, volume 11172 of *Lecture Notes in Computer Science*, pages 188–200, Berlin, Germany, 2018. Springer.

[5] R. Almadhoun, T. Taha, L. Seneviratne, and Y. Zweiri. A survey on multi-robot coverage path planning for model reconstruction and mapping. *SN Applied Sciences*, 1 (8):847, 2019.

[6] R. Ananthakrishnan and A. Ehrlicher. The forces behind cell movement. *International journal of biological sciences*, 3(5):303, 2007.

[7] D. Arbuckle and A. A. Requicha. Self-assembly and self-repair of arbitrary shapes by a swarm of reactive robots: algorithms and simulations. *Autonomous Robots*, 28(2): 197–211, 2010.

[8] G. S. Avellar, G. A. Pereira, L. C. Pimenta, and P. Iscold. Multi-uav routing for area coverage and remote sensing with minimum time. *Sensors*, 15(11):27783–27803, 2015.

[9] P. Bartashevich and S. Mostaghim. Benchmarking collective perception: new task difficulty metrics for collective decision-making. In *EPIA Conference on Artificial Intelligence*, pages 699–711. Springer, 2019.

[10] P. Bartashevich and S. Mostaghim. Positive impact of isomorphic changes in the environment on collective decision-making. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 105–106, 2019.

[11] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman. Multi-robot search and rescue: A potential field based approach. In *Autonomous robots and agents*, pages 9–16. Springer, 2007.

[12] R. A. Bazzi and J. L. Briones. Stationary and deterministic leader election in self-organizing particle systems. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pages 22–37. Springer, 2019.

[13] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of dna. *Discrete Applied Mathematics*, 71(1-3):79–94, 1996.

[14] S. Campbell, W. Naeem, and G. W. Irwin. A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres. *Annual Reviews in Control*, 36(2):267–283, 2012.

[15] S. Cannon, J. J. Daymude, D. Randall, and A. W. Richa. A markov chain algorithm for compression in self-organizing particle systems. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 279–288, 2016.

[16] K. C. Cheung, E. D. Demaine, J. R. Bachrach, and S. Griffith. Programmable assembly with universally foldable strings (moteins). *IEEE Transactions on Robotics*, 27(4):718–729, 2011.

[17] G. S. Chirikjian. Kinematics of a metamorphic robotic system. In *proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 449–455. IEEE, 1994.

[18] P. Clifford and A. Sudbury. A model for spatial conflict. *Biometrika*, 60(3):581–588, 1973.

[19] F. Comets, S. Popov, G. M. Schütz, and M. Vachkovskaia. Billiards in a general domain with random reflections. *Archive for rational mechanics and analysis*, 191(3):497–537, 2009.

[20] N. Correll and A. Martinoli. Modeling and designing self-organized aggregation in a swarm of miniature robots. *The International Journal of Robotics Research*, 30(5): 615–626, 2011.

[21] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, 2015.

[22] J. J. Daymude, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Improved leader election for self-organizing programmable matter. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 127–140. Springer, 2017.

[23] J. J. Daymude, Z. Derakhshandeh, R. Gmyr, A. Porter, A. W. Richa, C. Scheideler, and T. Strothmann. On the runtime of universal coating for programmable matter. *Natural Computing*, 17(1):81–96, 2018.

[24] J. J. Daymude, K. Hinnenthal, A. W. Richa, and C. Scheideler. Computing by programmable particles. In *Distributed Computing by Mobile Entities*, pages 615–681. Springer, 2019.

[25] J. J. Daymude, R. Gmyr, K. Hinnenthal, I. Kostitsyna, C. Scheideler, and A. W. Richa. Convex hull formation for programmable matter. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, pages 1–10, 2020.

[26] J. J. Daymude, A. W. Richa, and C. Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. In *35th International Symposium on Distributed Computing (DISC 2021)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[27] M. H. DeGroot. Reaching a consensus. *Journal of the American Statistical Association*, 69(345):118–121, 1974.

[28] E. D. Demaine, M. J. Patitz, R. T. Schweller, and S. M. Summers. Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor. *arXiv preprint arXiv:1004.4383*, 2010.

[29] Z. Derakhshandeh, S. Dolev, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Amoebot-a new model for programmable matter. In *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pages 220–222, 2014.

[30] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, T. Strothmann, and S. Tzur-David. Infinite object coating in the amoebot model. *arXiv preprint arXiv:1411.2356*, 2014.

[31] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. An algorithmic framework for shape formation problems in self-organizing particle systems. In *Proceedings of the Second Annual International Conference on Nanoscale Computing and Communication*, pages 1–2, 2015.

[32] Z. Derakhshandeh, R. Gmyr, T. Strothmann, R. Bazzi, A. W. Richa, and C. Scheideler. Leader election and shape formation with self-organizing programmable matter. In *International Workshop on DNA-Based Computers*, pages 117–132. Springer, 2015.

[33] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal shape formation for programmable matter. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 289–299, 2016.

[34] Z. Derakhshandeh, R. Gmyr, A. W. Richa, C. Scheideler, and T. Strothmann. Universal coating for programmable matter. *Theoretical Computer Science*, 671:56–68, 2017.

[35] A. Deshpande, M. Kumar, and S. Ramakrishnan. Robot swarm for efficient area coverage inspired by ant foraging: the case of adaptive switching between brownian motion and lévy flight. In *ASME 2017 dynamic systems and control conference*. American Society of Mechanical Engineers Digital Collection, 2017.

[36] R. K. Dewangan, A. Shukla, and W. W. Godfrey. Survey on prioritized multi robot path planning. In *IEEE international conference on smart technologies and management for computing, communication, controls, energy and materials (ICSTM)*, pages 423–428. IEEE, 2017.

[37] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Mobile ram and shape formation by programmable particles. In *European Conference on Parallel Processing*, pages 343–358. Springer, 2020.

[38] G. A. Di Luna, P. Flocchini, N. Santoro, G. Viglietta, and Y. Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33(1):69–101, 2020.

[39] B. Dieber, C. Micheloni, and B. Rinner. Resource-aware coverage and task assignment in visual sensor networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(10):1424–1437, 2011.

[40] A. Din, M. Jabeen, K. Zia, A. Khalid, and D. K. Saini. Behavior-based swarm robotic search and rescue using fuzzy controller. *Computers & Electrical Engineering*, 70:53–65, 2018.

[41] S. Dolev, S. Frenkel, M. Rosenblit, R. P. Narayanan, and K. M. Venkateswarlu. In-vivo energy harvesting nano robots. In *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*, pages 1–5. IEEE, 2016.

[42] M. Dorigo, G. Theraulaz, and V. Trianni. Reflections on the future of swarm robotics. *Science Robotics*, 5(49):eabe4385, 2020.

[43] D. Doty. Theory of algorithmic self-assembly. *Communications of the ACM*, 55(12): 78–88, 2012.

[44] F. Dressler. A study of self-organization mechanisms in ad hoc and sensor networks. *Computer Communications*, 31(13):3018–3029, 2008.

[45] F. Dufoulon, S. Kutten, and W. K. Moses Jr. Efficient deterministic leader election for programmable matter. *arXiv preprint arXiv:2106.01108*, 2021.

[46] B. Eberhardinger, G. Anders, H. Seebach, F. Siefert, A. Knapp, and W. Reif. An approach for isolated testing of self-organization algorithms. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 188–222. Springer, 2017.

[47] B. Eberhardinger, H. Ponsar, D. Klumpp, and W. Reif. Measuring and evaluating the performance of self-organization mechanisms within collective adaptive systems. In *International Symposium on Leveraging Applications of Formal Methods*, pages 202–220. Springer, 2018.

[48] J. T. Ebert, M. Gauci, and R. Nagpal. Multi-feature collective decision making in robot swarms. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1711–1719, 2018.

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

[50] Y. Emek, S. Kutten, R. Lavi, and W. K. Moses Jr. Deterministic leader election in programmable matter. *arXiv preprint arXiv:1905.00580*, 2019.

[51] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407 (1-3):412–447, 2008.

[52] N. Fujinaga, Y. Yamauchi, H. Ono, S. Kijima, and M. Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

[53] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[54] S. Garnier, C. Jost, R. Jeanson, J. Gautrais, M. Asadpour, G. Caprari, and G. Theraulaz. Aggregation behaviour as a source of collective decision in a group of cockroach-like-robots. In *European conference on artificial life*, pages 169–178. Springer, 2005.

[55] S. Garnier, C. Jost, J. Gautrais, M. Asadpour, G. Caprari, R. Jeanson, A. Grimal, and G. Theraulaz. The embodiment of cockroach aggregation behavior in a group of micro-robots. *Artificial life*, 14(4):387–408, 2008.

[56] N. Gastineau, W. Abdou, N. Mbarek, and O. Togni. Distributed leader election and computation of local identifiers for programmable matter. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 159–179. Springer, 2018.

[57] S. S. Ge and C.-H. Fua. Complete multi-robot coverage of unknown environments with minimum repeated coverage. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 715–720. IEEE, 2005.

[58] A. Giusti, J. Nagi, L. Gambardella, and G. A. Di Caro. Cooperative sensing and recognition by a swarm of mobile robots. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 551–558. IEEE, 2012.

[59] H. Hamann. Towards swarm calculus: Urn models of collective decisions and universal properties of swarm performance. *Swarm Intelligence*, 7(2):145–172, 2013.

[60] R. A. Holley and T. M. Liggett. Ergodic theorems for weakly interacting infinite systems and the voter model. *The annals of probability*, pages 643–663, 1975.

[61] X. Huang, F. Arvin, C. West, S. Watson, and B. Lennox. Exploration in extreme environments with swarm robotic system. In *2019 IEEE International Conference on Mechatronics (ICM)*, volume 1, pages 193–198. IEEE, 2019.

[62] E. R. Hunt, S. Jones, and S. Hauert. Testing the limits of pheromone stigmergy in high-density robot swarms. *Royal Society open science*, 6(11):190225, 2019.

[63] S. Ichikawa and F. Hara. Characteristics of object-searching and object-fetching behaviors of multi-robot system using local communication. In *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 775–781. IEEE, 1999.

[64] A. Jamshidpey and M. Afsharchi. Task allocation in robotic swarms: Explicit communication based approaches. In *Canadian Conference on Artificial Intelligence*, pages 59–67. Springer, 2015.

[65] A. Jamshidpey, W. Zhu, M. Wahby, M. Allwright, M. K. Heinrich, and M. Dorigo. Multi-robot coverage using self-organized networks for central coordination. In *International Conference on Swarm Intelligence*, pages 216–228. Springer, 2020.

[66] A. Jamshidpey, M. Wahby, M. Heinrich, M. Allwright, W. Zhu, and M. Dorigo. Centralization vs. decentralization in multi-robot coverage: Ground robots under uav supervision. Technical Report TR/IRIDIA/2021-008, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, May 2021.

[67] M. R. Jovanović and N. K. Dhingra. Controller architectures: Tradeoffs between performance and structure. *European Journal of Control*, 30:76–91, 2016.

[68] M. Juliá, A. Gil, and O. Reinoso. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots*, 33 (4):427–444, 2012.

[69] E. Kaddoum, C. Raibulet, J.-P. Georgé, G. Picard, and M.-P. Gleizes. Criteria for the evaluation of self-* systems. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 29–38, 2010.

[70] M. Kegeleirs, D. Garzón Ramos, and M. Birattari. Random walk exploration for swarm mapping. In *Annual conference towards autonomous robotic systems*, pages 211–222. Springer, 2019.

[71] Y. Khaluf. Edge detection in static and dynamic environments using robot swarms. In *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 81–90. IEEE, 2017.

[72] Y. Khaluf, S. Van Havermaet, and P. Simoens. Collective lévy walk for efficient exploration in unknown environments. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 260–264. Springer, 2018.

[73] Y. Khaluf, M. Allwright, I. Rausch, P. Simoens, and M. Dorigo. Construction task allocation through the collective perception of a dynamic environment. In *International Conference on Swarm Intelligence*, pages 82–95. Springer, 2020.

[74] A. Khamis, A. Hussein, and A. Elmogy. Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks 2015*, pages 31–51, 2015.

[75] S. Koenig and Y. Liu. Terrain coverage with ant robots: a simulation study. In *Proceedings of the fifth international conference on Autonomous agents*, pages 600–607. Association for Computing Machinery, 2001.

[76] S. Kornienko, O. Kornienko, C. Constantinescu, M. Pradier, and P. Levi. Cognitive micro-agents: individual and collective perception in microrobotic swarm. In *Proc. of the IJCAI-05 Workshop on Agents in real-time and dynamic environments, Edinburgh, UK*, pages 33–42, 2005.

[77] Z. Laouici, M. A. Mami, and M. F. Khelfi. Cooperative approach for an optimal area coverage and connectivity in multi-robot systems. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 176–181. IEEE, 2015.

[78] D. A. Lima and G. M. Oliveira. A cellular automata ant memory model of foraging in a swarm of robots. *Applied Mathematical Modelling*, 47:551–572, 2017.

[79] Y. Liu and R. Bucknall. A survey of formation control and motion planning of multiple unmanned vehicles. *Robotica*, 36(7):1019–1047, 2018.

[80] W. Luo and K. Sycara. Adaptive sampling and online learning in multi-robot sensor coverage with mixture of gaussian processes. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6359–6364. IEEE, 2018.

[81] D. Maftuleac, S. K. Lee, S. P. Fekete, A. K. Akash, A. López-Ortiz, and J. McLurkin. Local policies for efficiently patrolling a triangulated region by a robot swarm. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1809–1815. IEEE, 2015.

[82] N. Majcherczyk, D. J. Nallathambi, T. Antonelli, and C. Pinciroli. Distributed data storage and fusion for collective perception in resource-limited mobile robot swarms. *IEEE Robotics and Automation Letters*, 6(3):5549–5556, 2021.

[83] A. Marjovi, J. G. Nunes, L. Marques, and A. De Almeida. Multi-robot exploration and fire searching. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1929–1934. IEEE, 2009.

[84] N. Mathews, A. L. Christensen, R. O'Grady, F. Mondada, and M. Dorigo. Mergeable nervous systems for robots. *Nature communications*, 8(439), 2017.

[85] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen, and G. C. de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35), 2019.

[86] G. Mermoud, L. Matthey, W. C. Evans, and A. Martinoli. Aggregation-mediated collective perception and action in a group of miniature robots. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2010)*, number CONF, pages 599–606, 2010.

[87] T. Miki, M. Popović, A. Gawel, G. Hitz, and R. Siegwart. Multi-agent time-based decision-making for the search and action problem. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2365–2372. IEEE, 2018.

[88] M. Mirzaei, F. Sharifi, B. W. Gordon, C. A. Rabbath, and Y. Zhang. Cooperative multi-vehicle search and coverage problem in uncertain environments. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 4140–4145. IEEE, 2011.

[89] F. Mondada, M. Bonani, X. Raemy, J. Pugh, C. Cianci, A. Klaptocz, S. Magnenat, J.-C. Zufferey, D. Floreano, and A. Martinoli. The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and*

competitions, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco, 2009.

[90] G. Morlino, V. Trianni, E. Tuci, et al. Collective perception in a swarm of autonomous robots. In *IJCCI (ICEC)*, pages 51–59, 2010.

[91] A. Naz, B. Piranda, J. Bourgeois, and S. C. Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263. IEEE, 2016.

[92] M. Nazarahari, E. Khanmirza, and S. Doostie. Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm. *Expert Systems with Applications*, 115:106–120, 2019.

[93] A. Nedić, A. Olshevsky, and M. G. Rabbat. Network topology and communication-computation tradeoffs in decentralized optimization. *Proceedings of the IEEE*, 106(5): 953–976, 2018.

[94] S. Oguz, M. K. Heinrich, M. Allwright, W. Zhu, M. Wahby, E. Garone, and M. Dorigo. S-drone: An open-source quadrotor for experimentation in swarm robotics. Technical Report TR/IRIDIA/2022-010, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, October 2022.

[95] M. Otte. Collective cognition and sensing in robotic swarms via an emergent group-mind. In *International Symposium on Experimental Robotics*, pages 829–840. Springer, 2016.

[96] B. Pang, Y. Song, C. Zhang, H. Wang, and R. Yang. A swarm robotic exploration strategy based on an improved random walk method. *Journal of Robotics*, 2019, 2019.

[97] B. Pang, Y. Song, C. Zhang, and R. Yang. Effect of random walk methods on searching efficiency in swarm robots for area exploration. *Applied Intelligence*, pages 1–11, 2021.

[98] M. J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014.

[99] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6(4):271–295, 2012.

[100] J. Prasetyo, G. De Masi, P. Ranjan, and E. Ferrante. The best-of-n problem with dynamic site qualities: Achieving adaptability with stubborn individuals. In *International Conference on Swarm Intelligence*, pages 239–251. Springer, 2018.

[101] J. Prasetyo, G. De Masi, and E. Ferrante. Collective decision making in dynamic environments. *Swarm Intelligence*, 13(3):217–243, 2019.

[102] I. Rekleitis, V. Lee-Shue, A. P. New, and H. Choset. Limited communication, multi-robot team based coverage. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3462–3468. IEEE, 2004.

[103] I. Rekleitis, A. P. New, E. S. Rankin, and H. Choset. Efficient boustrophedon multi-robot coverage: an algorithmic approach. *Annals of Mathematics and Artificial Intelligence*, 52(2):109–142, 2008.

[104] P. W. Rothemund. Folding dna to create nanoscale shapes and patterns. *Nature*, 440 (7082):297–302, 2006.

[105] M. Rubenstein, A. Cornejo, and R. Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[106] E. Sahin, T. H. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. Swarm-bot: Pattern formation in a swarm of self-assembling mobile robots. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 6–pp. IEEE, 2002.

[107] M. Santos, S. Mayya, G. Notomista, and M. Egerstedt. Decentralized minimum-energy coverage control for time-varying density functions. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pages 155–161. IEEE, 2019.

[108] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[109] J. Scherer, S. Yahyanejad, S. Hayat, E. Yanmaz, T. Andre, A. Khan, V. Vukadinovic, C. Bettstetter, H. Hellwagner, and B. Rinner. An autonomous multi-uav system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 33–38, 2015.

[110] T. Schmickl and H. Hamann. Beeclust: A swarm algorithm derived from honeybees. *Bio-inspired computing and communication networks*, pages 95–137, 2011.

[111] T. Schmickl, C. Möslinger, and K. Crailsheim. Collective perception in a robot swarm. In *International Workshop on Swarm Robotics*, pages 144–157. Springer, 2006.

[112] A. Schroeder, S. Ramakrishnan, K. Manish, and B. Trease. Efficient spatial coverage by a robot swarm based on an ant foraging model and the lévy distribution. *Swarm Intelligence*, 11(1):39–69, 2017.

[113] G. D. M. Serugendo, M.-P. Gleizes, and A. Karageorgos. Self-organization in multi-agent systems. *The Knowledge engineering review*, 20(2):165–189, 2005.

[114] Q. Shan and S. Mostaghim. Collective decision making in swarm robotics with distributed bayesian hypothesis testing. In *International Conference on Swarm Intelligence*, pages 55–67. Springer, 2020.

[115] L. Siligardi, J. Panerati, M. Kaufmann, M. Minelli, C. Ghedini, G. Beltrame, and L. Sabattini. Robust area coverage with connectivity maintenance. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2202–2208. IEEE, 2019.

[116] T. Stirling, S. Wischmann, and D. Floreano. Energy-efficient indoor search by swarms of simulated flying robots without global information. *Swarm Intelligence*, 4(2):117–143, 2010.

[117] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[118] V. Strobel, E. Castelló Ferrer, and M. Dorigo. Managing byzantine robots via blockchain technology in a swarm robotics collective decision making scenario. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*, pages 541–549. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[119] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[120] S. Thabit and A. Mohades. Multi-robot path planning based on multi-objective particle swarm optimization. *IEEE Access*, 7:2138–2147, 2018.

[121] T. Toffoli and N. Margolus. Programmable matter: concepts and realization. *Physica. D, Nonlinear phenomena*, 47(1-2):263–272, 1991.

[122] S. Tomforde, J. Kantert, and B. Sick. Measuring self-organisation at runtime-a quantification method based on divergence measures. In *International Conference on Agents and Artificial Intelligence*, volume 2, pages 96–106. SciTePress, 2017.

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

[124] G. Valentini, H. Hamann, M. Dorigo, et al. Self-organized collective decision making: the weighted voter model. In *AAMAS*, pages 45–52, 2014.

[125] G. Valentini, H. Hamann, and M. Dorigo. Efficient decision-making in a self-organizing robot swarm: On the speed versus accuracy trade-off. In *Proceedings of the 2015*

International Conference on Autonomous Agents and Multiagent Systems, pages 1305–1314, 2015.

[126] G. Valentini, D. Brambilla, H. Hamann, and M. Dorigo. Collective perception of environmental features in a robot swarm. In *International Conference on Swarm Intelligence*, pages 65–76. Springer, 2016.

[127] G. Valentini, E. Ferrante, H. Hamann, and M. Dorigo. Collective decision with 100 kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous agents and multi-agent systems*, 30(3):553–580, 2016.

[128] G. Valentini, E. Ferrante, and M. Dorigo. The best-of-n problem in robot swarms: Formalization, state of the art, and novel perspectives. *Frontiers in Robotics and AI*, 4:9, 2017.

[129] J. C. Varughese, R. Thenius, P. Leitgeb, F. Wotawa, and T. Schmickl. A model for bio-inspired underwater swarm robotic exploration. *IFAC-PapersOnLine*, 51(2):385–390, 2018.

[130] M. Wahby, J. Petzold, C. Eschke, T. Schmickl, and H. Hamann. Collective change detection: Adaptivity to dynamic swarm densities and light conditions in robot swarms. In *ALIFE 2019: The 2019 Conference on Artificial Life*, pages 642–649. MIT Press, 2019.

[131] J. E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Distributed Computing*, 17(2):171–189, 2004.

[132] P. K. Wang. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[133] X. Wang, S. Han, Y. Wu, and X. Wang. Coverage and energy consumption control in mobile heterogeneous wireless sensor networks. *IEEE Transactions on Automatic Control*, 58(4):975–988, 2012.

[134] E. Winfree, F. Liu, L. A. Wenzler, and N. C. Seeman. Design and self-assembly of two-dimensional dna crystals. *Nature*, 394(6693):539–544, 1998.

[135] D. Woods. Intrinsic universality and the computational power of self-assembly. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 373(2046):20140214, 2015.

[136] D. Woods, H.-L. Chen, S. Goodfriend, N. Dabby, E. Winfree, and P. Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, pages 353–354, 2013.

[137] J. Yu and S. M. LaValle. Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics. *IEEE Transactions on Robotics*, 32(5):1163–1177, 2016.

[138] M. N. Zafar and J. C. Mohanta. Methodology for path planning and optimization of mobile robots: A review. *Procedia Computer Science*, 133:141–152, 2018.

[139] Y. Zhang, S. Ouguz, S. Wang, E. Garone, X. Wang, M. Dorigo, and M. Heinrich. Self-reconfigurable hierarchical frameworks for bearing-based formation control of robot swarms. Technical Report TR/IRIDIA/2021-004, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, August 2021.

[140] W. Zhu, M. Allwright, M. K. Heinrich, S. Oğuz, A. L. Christensen, and M. Dorigo. Formation control of UAVs and mobile robots using self-organized communication topologies. In *Swarm Intelligence – Proceedings of ANTS 2020 – Twelfth International Conference*, Lecture Notes in Computer Science, Berlin, Germany, 2020. Springer.

[141] K. Zia, A. Din, K. Shahzad, and A. Ferscha. A cognitive agent-based model for multi-robot coverage at a city scale. *Complex Adaptive Systems Modeling*, 5(1):1–13, 2017.

Appendix: Robots available in practice

Given that energy consumption is mostly a practical consideration, we catalog the performance and capabilities of several robot options that are currently available off-the-shelf[1]. Although this catalog is an ad-hoc summary of available products, the spread of these products is one way to assess the scope of energy consumption and efficiency of mobile robots and drones at the technical state of the art.

For ground robots, we catalog the e-puck2[2], Thymio II[3], TurtleBot3 Burger[4], iRobot Roomba e[5], Clearpath robotics Husky[6], Clearpath robotics Warthog[7], ECA Group Cameleon C[8], Autonomous Solutions Inc. (ASI) Chaos[9], SMP Robotics Rover S5 PTZ[10], Mattro Rovo 2[11], and Boston Dynamics Sopt Explorer[12]. The relevant product specifications listed by the manufacturer are given in Table 1. The approximate energy metrics, calculated from the product specifications in Table 1, are given in Table 3. We calculate the approximate energy metrics for ground robots according to energy consumption corresponding to distance traveled.

For drones, we consider those with cameras, LiDAR, or other imaging capabilities suitable for a supervisor role. We do not consider fixed-wing drones, as they are not suitable for slow speeds or hovering. We catalog the DJI Mini 2[13] DJI Mavic 2 Pro[14], Force1 U49WF FPV

[1] When the monetary cost is not listed by the manufacturer, the listed cost is from the distributor Generation Robots: generationrobots.com.

[2] https://www.gctronic.com/doc/index.php/e-puck2

[3] I.e., Thymio Wireless, http://wiki.thymio.org/en:thymiospecifications

[4] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications

[5] https://www.irobot.com/roomba/e-series

[6] https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/

[7] https://clearpathrobotics.com/warthog-unmanned-ground-vehicle-robot/

[8] https://www.ecagroup.com/en/solutions/cameleon-c-ugv-unmanned-ground-vehicle

[9] https://asirobots.com/platforms/chaos/

[10] https://smprobotics.com/products_autonomous_ugv/area-and-perimeter-gas-monitoring-robot/

[11] https://www.mattro.com/en/the-rovo/technical-specifications

[12] https://shop.bostondynamics.com/spot

[13] https://www.dji.com/be/mini-2

[14] https://store.dji.com/be/product/mavic-2

Camera Drone[15], Skydio 2[16], and Autel Evo II[17]. The relevant product specifications listed by the manufacturer are given in Table 2. The approximate energy metrics, calculated from the product specifications in Table 2, are given in Table 4. We calculate the approximate energy metrics for drones according to energy consumption corresponding to flight time.

Table 1: Listed product specifications for mobile robots, unmanned ground vehicles (UGVs), and autonomous ground vehicles (AGVs) that are currently available off-the-shelf.

Product name	Locomotion, estimated max. speed	Estimated operating time, charging time	Estimated hourly energy consumption	Listed monetary cost
e-puck2	two-wheeled, 0.154 m/s	3 h, 2.5 h	600mAh 3.7V	850 CHF
Thymio II	two-wheeled, 0.14 m/s	4 h, 1.5 h	375mAh 3.7V	140 EUR
TurtleBot3 Burger	two-wheeled, 0.22 m/s	2.5 h, 2.5 h	720mAh 11.1V	580 EUR
iRobot Roomba e	two-wheeled, 0.5 m/s	1.5 h, 3 h	1200mAh 14.4V	400 EUR
Clearpath Husky	four-wheeled, 1 m/s	3 h, 4 h	6500mAh 24V	22000 EUR
Clearpath Warthog	four-wheeled, 5 m/s	3 h, 4 h	36500mAh 48V	90000 EUR
ECA Group Cameleon C	tracked, 1.7 m/s	4 h, -	- -	-
ASI Chaos	tracked, 2.2 m/s	3 h, -	- -	-
SMP Rover S5 PTZ	four-wheeled, 1.8 m/s	12 h, 4 h	10000mAh 12V	-
Mattro Rovo 2	tracked, 8.3 m/s	8 h, 3.25 h	11000mAh 100V	-
Spot Explorer	quadruped, 1.6 m/s	1.5 h, 2 h	7000mAh 58V	75000 USD

[15] https://force1rc.com/products/u49w-blue-heron-wifi-drone-with-camera-fpv-drone-w-live-video-altitude-hol
[16] https://www.skydio.com/pages/skydio-2
[17] https://auteldrones.com/pages/evo-ii-detail

Table 2: Listed product specifications for camera or mapping drones or unmanned aerial vehicles (UAVs) that are currently available off-the-shelf.

Product name	Locomotion, estimated max. speed	Estimated flight time, charging time	Estimated hourly energy consumption	Listed monetary cost
DJI Mini 2	quadrotor, 16 m/s	31 minutes, -	1160mAh 7.7V	1500 EUR
DJI Mavic 2	quadrotor, 20 m/s	31 minutes, -	1990mAh 15.4V	1500 EUR
U49WF FPV	quadrotor, -	25 minutes, -	830mAh 7.4V	100 USD
Skydio 2	quadrotor, 16.1 m/s	23 minutes, -	1640mAh 11.4V	1000 USD
Autel Evo II	quadrotor, 20.1 m/s	40 minutes, 1.5 h	4730mAh 11.55V	1500 USD

Table 3: Metrics of energy consumption, energy efficiency, and cost efficiency for ground robots listed in Table 1.

Product name	Approximate maximum distance per charge	Approximate energy consumption (at 0.068 m/s)	Approximate energy consumption by distance	Approximate robot cost by distance per charge
e-puck2	1.7 km	2.9 joules/s	43 joules/m	0.50 EUR/m
Thymio II	2 km	2.7 joules/s	40 joules/m	0.07 EUR/m
TurtleBot3 Burger	2 km	6.2 joules/s	91 joules/m	0.29 EUR/m
iRobot Roomba e	3 km	3.5 joules/s	52 joules/m	0.15 EUR/m
Clearpath Husky	11 km	32.0 joules/s	470 joules/m	2.04 EUR/m
Clearpath Warthog	55 km	71.4 joules/s	1050 joules/m	1.67 EUR/m
ECA Group Cameleon C	25 km	-	-	-
ASI Chaos	25 km	-	-	-
SMP Rover S5 PTZ	80 km	54.4 joules/s	800 joules/m	-
Mattro Rovo 2	240 km	72.1 joules/s	1060 joules/m	-
Spot Explorer	9 km	25.8 joules/s	380 joules/m	7.26 EUR/m

Table 4: Metrics of energy consumption, energy efficiency, and cost efficiency for drones listed in Table 2.

Product name	Approximate maximum distance per charge (at 0.074 m/s)	Approximate energy consumption during flight	Approximate energy consumption by distance (at 0.074 m/s)	Approximate robot cost by distance per charge (at 0.074 m/s)
DJI Mini 2	140 m	8.9 joules/s	120 joules/m	10.71 EUR/m
DJI Mavic 2	140 m	30.6 joules/s	410 joules/m	10.71 EUR/m
U49WF FPV	110 m	6.1 joules/s	82 joules/m	0.76 EUR/m
Skydio 2	100 m	18.7 joules/s	250 joules/m	8.36 EUR/m
Autel Evo II	180 m	54.6 joules/s	740 joules/m	6.97 EUR/m