NORTHWESTERN UNIVERSITY

Decentralized Persistent Shape Formation in Large-Scale Homogeneous Robotic Swarms

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Hanlin Wang

EVANSTON, ILLINOIS

September 2021

Abstract

Decentralized Persistent Shape Formation in Large-Scale Homogeneous Robotic Swarms

Hanlin Wang

This research looks at the robotic shape formation problem, which is one of the fundamental problems in robotic swarm systems. Here, the task is to move a group of robots to form a user-specified shape. In this dissertation, the task of shape formation is divided to four problems: (i) using local information to estimate the swarm size; (ii) using arbitrary number of robots to display a user-specified shape; (iii) localizing a swarm of robots with peer-to-peer measurements; and (iv) assigning each robot a location in the shape and routing each robot to quickly reach its assigned goal location. Accordingly, four algorithms, each solves a problem above, are presented. The presented algorithms are validated using a custom physical 100-robot swarm, and a custom efficient swarm system simulator. The results from both the simulation and physical experiments show that: each presented algorithm is able to solve its corresponding problem efficiently and reliably. Furthermore, those four presented algorithms are brought together into a fully decentralized persistent shape formation algorithm. With the developed simulator and physical robotic swarm, it is further demonstrated that: this integrated algorithm allows the swarms with any size to persistently form arbitrary user-specified shapes, requiring only the use of local information.

Acknowledgments

First, I want to thank my advisor, Michael Rubenstein, who introduced me to the field of swarm-robotic system, and whose support leads to the development of this dissertation. Throughout my four-year-long journey, he has been a talented and experienced researcher that can always identify the right direction for solving the problems; he has been a trusted collaborator that always replied to my questions without any delay; and most importantly, he has been an organized advisor that helped me to keep my research at the perfect pace. Thank you so much, Mike!

I'd also like to thank my parents for their continued support of my decision to pursue a Ph.D.'s degree.

Contents

List of Figures

CHAPTER 1

INTRODUCTION

1.1. Background

A *swarm system* generally refers to a large group of similar agents, often numbering in the hundreds or more, that displays emergent behavior arising from local interactions among the agents [1]. Across countless species such as birds [2], bees [3], and ants [4], nature has shown us that by working together with peers as a unified systems, even the simplest creature, like an ant, can accomplish some extremely complex tasks.

In swarm systems, the swarm's shape is an important and fundamental property. It has been found in nature that shape formation can allow simple creatures to amplify their group ability, for example, flocks of birds fly in an aerodynamically optimum V-shape to achieve energy efficiency [2], army ants form bridges across gaps to allow the swarms to travel efficiently [4] and reach locations not accessible by any single individual, and fish school to better protect themselves from predators. See Fig. 1.1 for a graphical illustration of these examples.

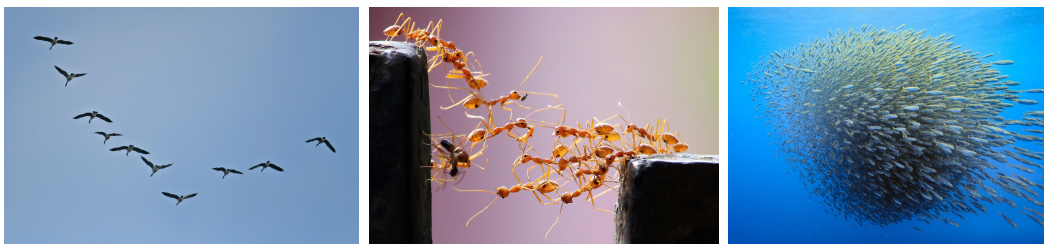

Figure 1.1. Example of shape formations in nature. (Left) flocks of birds fly in V shape [5]. (Middle) Ants build bridge across a gap [6]. (Right) A school of fish [7].

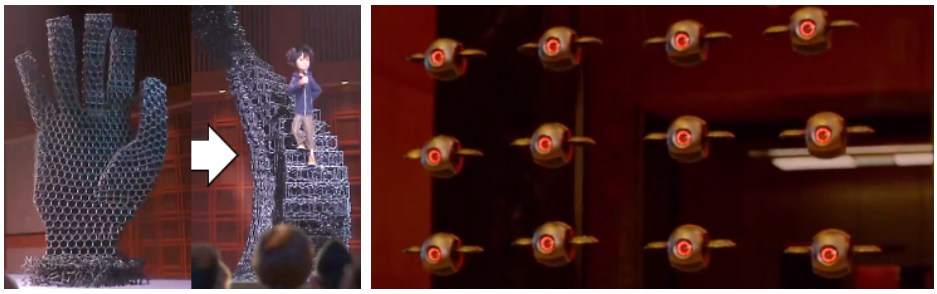

Figure 1.2. Example of robotic shape formation in the work of fiction. (Left) *Microbots* from the film *Big Hero 6*. (Right) The robot swarm from the film *Tau*.

Similar to their natural counterparts, swarm shape plays a critical role in robotic swarms. When mapped to robotic systems, the task of shape formation is often framed as moving a set of robots, which are initially located randomly in space, into a given arbitrary target formation. The concept of shape formation on robotic swarms has frequently been depicted or described in works of fiction, for example, the *microbots* in the film *Big Hero 6* can connect together to form various shapes and perform tasks cooperatively; the robotic swarm in the film *Tau* can work together to guard the house. See Fig. 1.2 for a graphical illustration of these two examples.

Recently, several real-world applications has brought this concept from the fiction to reality. SMORES-EP is a modular robot [8] designed and built at the UPenn, and used by researchers at UPenn and Cornell [9]. Each SMORES-EP module is extremely simple: it has only the capability of moving on the flat surfaces, the capability of connecting with the other modules, and the capability of talking to a central controller. That said, when multiple SMORES-EP module connect with each other and transform into different shapes, they are able to address many complex tasks that are beyond any individual's capability, such as climbing up the stairs, picking up an object, or exploring the environment. See Fig. 1.3 for a picture of SMORES-EP robot.

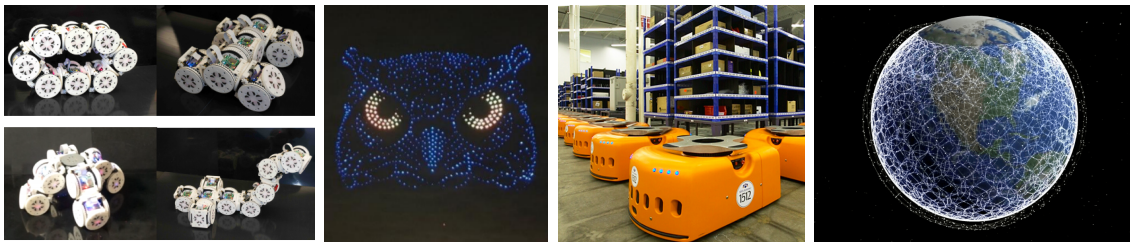

Figure 1.3. Example of robotic shape formation in the reality. From left to right: SMORES-EP robot system from UPenn and Cornell [9]; a drone light show from Intel [10]; Kiva system from Amazon Robotics [11]; the envisioned Starlink system from SpaceX [12].

Another famous example is the drone light show from the company Intel [10], which has been shown to be an eco-friendly alternative to the traditional firework show. The firework show has been playing an important role in a lot of entertainment events, but it has been criticized for contributing to air and water pollution, as well as the extreme high expense. The drone show from the Intel, on the other hand, is noise-free, smoke-free, reusable, and is able to provide almost the same performance compared to the traditional firework show from entertainment perspective. See Fig. 1.3 for a picture of a drone light show from Intel.

Besides the entertainment applications, a more example of a commercial application is Kiva system from the Amazon Robotics. Kiva system is a automated storage and retrieval system that consists of thousands of ground robots. It has been shown that Kiva system can increase the overall efficiency of the picking system by 3.5 times. Kiva robots are capable of delivering a item to a worker every 6 seconds, which results in 600 picks an hour. Other warehouse designs would require about 150 people to complete the same amount of work in a day as 50 could with Kiva system [13]. See Fig. 1.3 for a picture of Kiva system.

Furthermore, beyond the examples on Earth, robotic shape formation also plays an important role in the space applications. Starlink is a satellite constellation constructed by the company SpaceX [12]. This system will consist of thousands of small satellites in low

Earth orbit, working in combination with ground transceivers. By forming a mesh pattern surrounding Earth, these small satellites will be able to bring the internet connectivity to almost everywhere on this planet. See Fig. 1.3 for a picture of Starlink system.

In addition to the examples above, the concept of shape formation can also to found in many other applications, such as manufacturing [14], environment monitoring [15], and more [16–18]. Given countless examples found in both the nature and the real world, it is clear that the robotic shape formation is a problem that has both the scientific and practical importance, motivating the study presented in this dissertation.

In the past, many solutions to the shape formation problem have been proposed. Many past efforts have concentrated on a fixed-scale version of the problem. In the fixed-scale shape formation problem, the swarm size is assumed to stay the same all the times, and the representation of the desired shape is often pre-computed and given to the swarm as an input. There is a related body of work controlling groups of robots that uses centralized planning or controllers, for example, some shape formation in large groups of quadrotors [19–21], or some applications in warehouses [13].

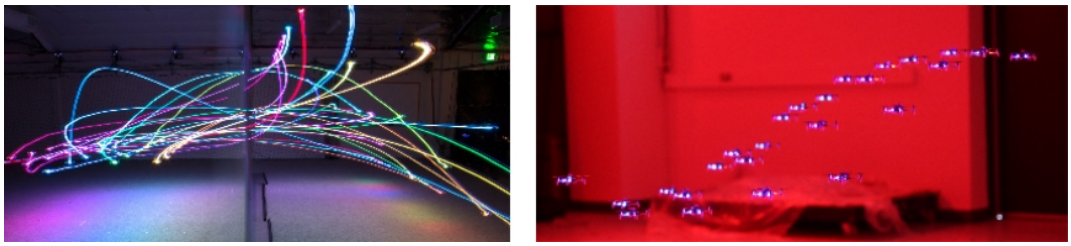

Figure 1.4. Example of centralized shape formation in quadrotor swarms. (Left) Long exposure of 32 Crazyflie quadrotors flying through a hole on a wall [20]; (Right) 25 quadrotors perform a periodic wave motion using the method proposed in [21].

Previous works includes the methods solving the problem in a discrete setting [22–24] and the methods solving the problem in a continuous setting [25–27]. While these systems provide

exciting results, the centralized approach creates a single point of failure which reduces the fault tolerance of the group, and suffers from the extremely high computation complexity as the complexity will exponentially escalate over the swarm size. As a result, centralized method does not scale well to very large groups of robots. To reduce the computational cost, an alternative is to design an artificial potential function to guide the swarm to the goal configuration [28–30]. Despite that this type of method can significantly reduce the computation complexity of the problem, it often takes a long time to converge and lack completeness guarantees [27].

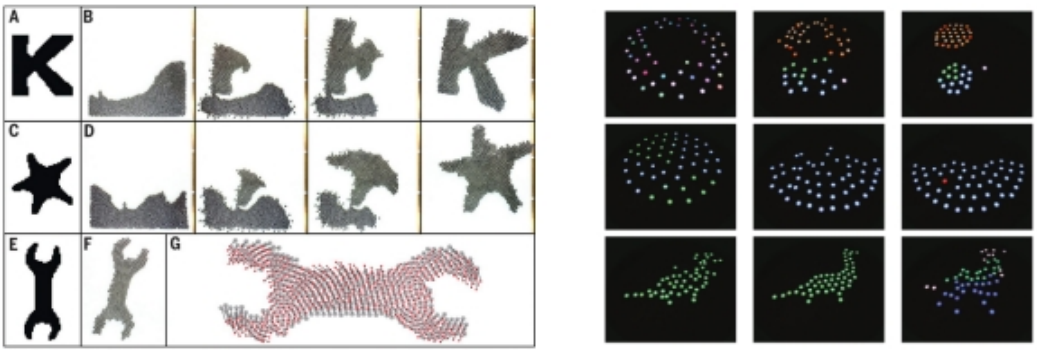

Figure 1.5. Example of decentralized shape formation in ground robotic swarms. (Left) 1024 Kilobots cooperatively form various shapes [31]; (Right) 50 Elisa robots display a sequence of images [32].

One strategy that helps to break the problem's computational bottleneck is distributing the computational cost among the agents. The decentralized methods have been shown to empirically scale well with swarm size [27, 31–36]. The decentralized methods can be categorized into progressive methods [31, 33, 37], and parallel methods [32, 35, 36]. In progressive methods, the shape "grows" from an pre-determined leader agent, which is essentially a sequential process. These methods require an additional leader selection phase to select the leader agents, furthermore, its has not been shown that these methods are collision-free. To the contrary, in parallel methods [27, 32, 35, 36], each agent takes on the same role and

cooperatively forms the shape in parallel. Compared to the progressive methods, the parallel methods are more efficient and do not require the leader selection phase. For the parallel methods, when including the collision avoidance in algorithm design, one challenge is that: in reality, agents have limited communication range and bandwidth, which makes it hard for the agents to get access to global information, therefore, the approaches in the past have not provided completeness guarantees.

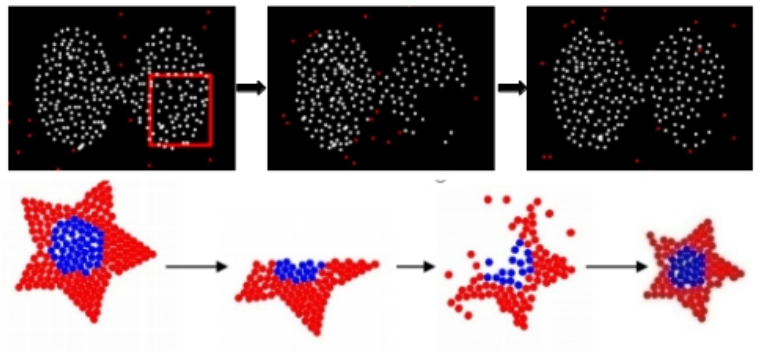

Figure 1.6. Example of persistent robotic shape formation. In both examples, the formed shape is damaged by the robot removal. (Top) The swarm recovers the desired shape by changing the robot density [38]; (Bottom) The swarm repairs the desired shape by forming the shape at a smaller scale [33]

Different from the fixed-scale shape formation, the other more advanced version of the problem is so called *persistent* (or *scale-independent*) shape formation [38–40]. In this version of shape formation problem, robots can be removed from or added to the swarm in real-time. When the swarm size changes, two strategies allowing the swarm to adapt have been proposed [38–40]: one strategy is to keep the scale of the desired shape fixed, and change the density of the robots [38]; the other strategy is to keep the density of the robot fixed, and change the size of the desired shape [39, 40]. It has been shown that, both of those two methods [39, 40] allow the system to adapt to the swarm size change. On the other hand, these previous methods either have an implicit requirement on the swarm size to perfectly display

the desired shape [38, 40], or require the swarm to wait "long enough" to sense the swarm size change, making the time for the swarm to adapt to the swarm size change relatively long [39]. See Fig. 1.6 for a graphic illustration of persistent shape formation.

1.2. Summary

In this dissertation, I present a method that allows a large swarm of identically programmed (homogeneous) robots to persistently form a user-specified 2D shape. The presented method requires only the use of local information (decentralized), in addition, for each robot, besides the desired shape, no other information needs to be known a priori. Furthermore, when the formed shape is damaged by external disturbances such as the addition, the removal, or the shifting of robots, the presented method will enable the swarm to recover the desired shape (it is persistent).

The organization of this dissertation is as follows:

• After chapter 1, chapter 2 states the problem to be solved and introduces the model of the robots used in this research;

• Chapter 3 presents a custom 100-robot physical swarm as well as two swarm system simulators. These systems are important tools for developing and validating the algorithms presented in the later chapters;

• In chapter 4, an overall pipeline for persistent robotic shape formation is presented. This pipeline reduces the task of shape formation to four problems: (i) *swarm size estimation* – using local information to estimate the swarm size; (ii) *goal configuration generation* – using arbitrary number of robots to represent a user-specified shape; (iii) *cooperative localization* – localizing a swarm of robots with peer-to-peer measurements; and (iv) *task assignment and formation control* – assigning each robot its location in shape and routing each robot to reach its assigned goal location;

• Next, from chapter 5 to chapter 8, four algorithms, each solves a problem above, are presented. For each presented algorithm, a comprehensive literature survey is conducted

in the corresponding chapter. In addition, each presented algorithm is validated via the experiments running on both the simulated swarm and the physical swarm;

- Chapter 9 brings together the four algorithms presented in chapter 5 - 8 into a fully decentralized persistent shape formation algorithm. This integrated algorithm is further demonstrated using both the simulated swarm and the physical swarm;
- Lastly, I conclude in the chapter 10.

1.3. Contributions

The major contribution of this dissertation is the design and analysis of a collection of algorithms such that when combined, they allow the swarms with any size to persistently form arbitrary user-specified shapes without the use of any global information. In particular, I present the following work:

- A provably correct decentralized algorithm for estimating the number of robots in a swarm. In addition, this algorithm is able to adapt to the addition or removal of robot, making it possible for the robot to constantly monitor the swarm size at run time [41].

- An algorithm that generates the goal configuration for a given swarm to display the given shape. The generated goal configuration is given in the format of a set of goal points such that: the number of goal points exactly matches the swarm size. This algorithm is able to generate the goal configurations for the swarms with arbitrary sizes, in addition, when the swarm size changes and the swarm needs to transform to a new configuration corresponding to the new swarm size, the algorithm will minimize the swarm's effort required [42].

- A provably correct decentralized algorithm that allows a swarm of identically programmed agents to cooperatively estimate their global poses using the local range and bearing measurements. The novelty of this algorithm is that: it does not require the robot to actively maintain the communication links between itself and its neighbors, or synchronous its local clock with the others, making it possible for the algorithm to work in the situations where the swarm's communication topology is dynamically changing [43].

- A provably correct decentralized algorithm for assigning a set of goal locations to a group of robots, and routing each robot to its assigned goal location. As stated in [44], to the best of my knowledge, this algorithm is the first provably correct fully decentralized

shape formation algorithm that can also provide absolute collision-free and deadlock-free guarantees, requiring only the use of local communication [44].

• These four presented algorithms above into a fully decentralized persistent shape formation algorithm.

In addition, to verify the presented algorithms, the following swarm systems are developed in this dissertation:

• A 100-robot platform that allows researchers to easily test their work;

• A simulator that simulates the developed physical robotic swarm in a very realistic way. It is designed in a way that the user can use exactly the same copy of code to control both the simulated swarm and real swarm. It allows the user to test their codes before implementing them on the real swarm, which helps to avoid the potential hardware damage introduced by inappropriate user code.

• An efficient simulator for swarm systems that is able to simulate large-scale swarms ($\geq$ 1000 robots) in the real time. It allows the users to quickly and safely test their ideas on large-scale swarms.

CHAPTER 2

PRELIMINARIES

In this chapter, I will formally state the problem to be solved in this research, and introduce the robot model used in the design and analysis of the presented algorithms.

2.1. Problem Statement

The main objective of this research is to develop a fully decentralized method that enables a large swarm of identically programmed (homogeneous) robots to persistently form a user-specified 2D shape. To be more specific: It is assumed that the desired shape is given to the swarm in the format of a binary image. For each robot, this input binary image is the only information known a priori. Note that the number of in-shape pixels on this input image does not necessarily match the number of robots. Given the desired shape, the developed method must enable the swarm to reliably form this input shape without the use of any global information (decentralized). Furthermore, besides *forming* the desired shape, the developed method must also enable the swarm to *maintain* the desired shape, that is: when the formed shape is damaged by external disturbances such as robot addition, removal, or the shifting, the swarm should be able to recover the desired shape (persistent). An example of up to 100 robots persistently forming a shape "N" is shown in Fig. 2.1: (a) The input given to each robot, which is a binary image of a shape "N"; (b) initially, 70 robots are randomly dispersed on a 2D plane; (c) the robots form the input shape; (d) 30 robots are added to the swarm, damaging the formed shape; (e - f) the swarm form the shape "N" at

a bigger scale so as to recover the desired shape; (g) the formed shape is damaged by robot shifting; (h - i) the swarm recover the desired shape; (j) 23 robots are removed from the swarm, which damages the formed shape; (k - l) the swarm recover the desired shape by forming the shape at a smaller scale.

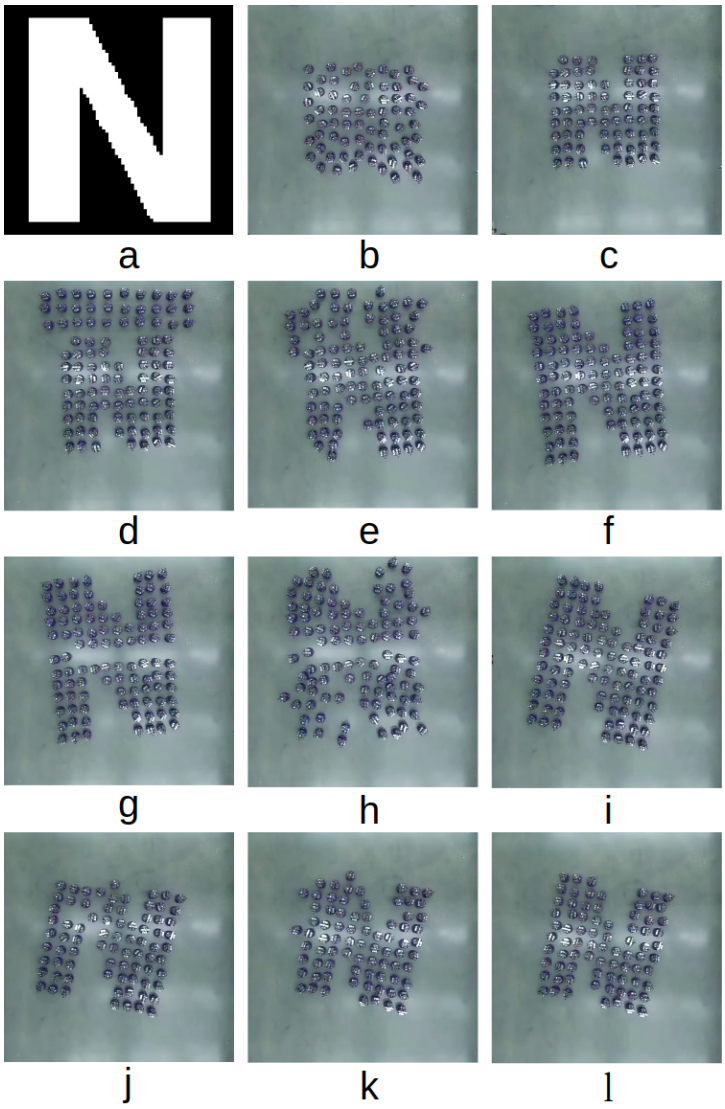

Figure 2.1. An example of the persistent shape formation on a swarm of up to 100 robots. Time for each frame: b - 0s; c - 166s; d - 210s; e - 420s; f - 605s; g - 644s, h - 740s; i - 860s; j - 990s; k - 1167s; l - 1320s.

2.2. Robot Model

When developing the algorithm, one key consideration is the required robot capabilities to execute the algorithm. For the real-world robotics swarms, it is apparent that the complexity and cost of individual robots are the important factors in limiting the size of the swarm. Therefore, in order to make it possible for the developed algorithm to work on a large swarm, the required robot capabilities to execute the algorithm must not be complex. Similar to [31, 33, 34, 38–40, 45–52], in the design and analysis of the presented algorithms, it is assumed that the robot only has the capability communicating with physically nearby neighbors, the capability of sensing the nearby robots' relative positions, and the capability of moving on a 2D plane. To be more specific, I assume the following:

- Each robot is in a disk shape with a radius of r, moreover, it is able to move on a 2D plane;
- Each robot's clock has the same frequency but can be asynchronous in phase;
- Each robot can communicate with any nearby robots lying within its communication range $R \geq 2\sqrt{2}r$;
- Each robot has a locally unique ID. It was shown in [33, 45] that each robot can easily generate this locally unique ID in a probabilistic way on the fly;
- Each robot is identically programmed;
- Each robot holds a local coordinate frame where the local coordinate frame's origin is fixed on the center of the robot and the x-axis' direction is aligned with the robot's heading. In addition, considering the fact that most real-world sensors have a right-handed coordinate system, it is assumed that each robot's local coordinate frame is always right-handed. When an robot a_i receives a message from a neighbor a_j, the robot a_i is able to sense

the transmitter a_j's relative bearing angle $\mathcal{B}_{ij}$ and the distance d_{ij} in its local coordinate frame. See Fig. 2.2 for a graphical illustration of robot's sensing capabilities;

- Each robot has the odometry capability.

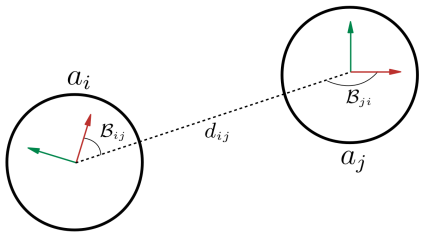

Figure 2.2. Graphical illustration of the robot's sensing capability. Each disk is a robot. The red and green arrow lines are each robot's local coordinate frame, where red arrow line is the x-axis

CHAPTER 3

SWARM SYSTEMS

As the proverb goes: "You can't make bricks without straw", we won't get anywhere without the appropriate resource. Thus, before introducing the algorithms, in this chapter, I will present a 100-robot physical swarm, and two swarm simulators. These three systems are important tools for developing and validating the algorithms presented in the later chapters.

3.1. General Swarm Robotic Simulator

For swarm-robotic research, the simulation is a powerful and safe tool to prototype the theories and algorithms. For this task, I developed an simulator that can realistically and efficiently simulate robot's locomotion, communication, and sensing. In simulation, each robot is modeled as an omni-directional robot able to sense its position and orientation in a common coordinate system. In addition, the simulated robots are able to communicate with any other robot who lies within an user-specified communication range. The simulator consists of two major components: the graphical engine, which graphically displays the simulated results to the user, and the world engine, which handles the computation of simulation of the swarm. The graphical engine is developed using OpenGL (Open Graphics Library). The world engine is developed using C language and accelerated by the multithreading technique. It has been shown that the simulator is efficient enough to simulate large-scale swarm ($\geq$ 2000) in real time. This simulator offers a safe and convenient way to develop and test new

algorithms. The source code and associated documentation for the developed simulator can be found at: https://tinyurl.com/ypm5avbk.

3.2. Coachswarm

Often, simulations are used as a safe and convenient way to develop and validate swarm theories and algorithms. However, it is not possible for a simulator to capture all real-world uncertainties, such as communication uncertainty, sensing error, and imperfect robot motion. These real-world uncertainties can affect algorithm performance in unexpected ways [48]. It is therefore imperative for swarm-robotics researchers to validate their work on physical swarm systems.

Many previous swarm-robotic platforms have been developed [17, 53–64] in which great efforts were made to reduce the robot's cost and footprint so as to increase the size of swarms [56–60]. Unsurprisingly, this often results in swarms with limited computation, communication, and locomotion capabilities. E-puck [55], r-one [53], and swarmbot [54] are three more powerful robots. However, the cost and complexity of constructing these robots makes it hard for them to be used for large-scale swarm-robotic research.

Two robots that attempt to achieve a balance between the capability of robot, its cost, and the complexity of its construction are Colias IV [62] and mROBerTO 2.0 [64]. Colias IV is a small robot with a camera that allows a user to deploy vision-based algorithms on a swarm of small robots. mROBerTO 2.0 is a robot with stepper motors and an associated control algorithm that enables reliable open-loop motion control. Although these robots offer users powerful per-robot performance, they have not yet been deployed in large numbers.

Only a handful of works considered the complexity of swarm operation [17, 60, 61]. The Kilobot swarm [60] was developed with the objective of deploying decentralized algorithms on

a large-scale, 1,000-robot swarm. Unfortunately, the communication from operator to Kilobot swarm is one-directional, making it hard for the operator to collect detailed experimental data directly from the swarm, such as the state of each robot. Zooid [17] is another platform that allows the user to control many small robots. However, Zooid has not yet been shown to perform decentralized algorithms, and the robots are manually and individually charged, increasing the difficulty of maintaining the swarm. Robotarium [61] is a swarm-robotic platform that offers users remote access. When executing user's code, Robotarium uses several additional layers to guarantee the safety of the operations. These intermediate layers will interfere user's algorithms with respect to motion control, robot-to-robot communication, etc, making the experimental results less realistic.

In this section, I present the Coachswarm, a high-performance and low-cost 100-robot platform that allows researchers easily and reliably test their work. This platform offers me a way to test the algorithms' robustness to real-world uncertainties.

The *Coachbot V2.0* platform consists of two key components: 100 Coachbot V2.0 robots, and a central server that manages the swarm.

3.2.1. Coachbot V2.0 Robot

The *Coachbot V2.0* robot is a compact, modular, and inexpensive differential driven mobile robot. *Coachbot V2.0* has a *modular* design, and is composed of three independently-manufactured modules: a locomotion and power module, a computation module, and a sensing module.

The locomotion and power module consists of five elements. First, two wheels driven by two DC gear-motors allow the robot to drive across flat surfaces. Second, an H-bridge

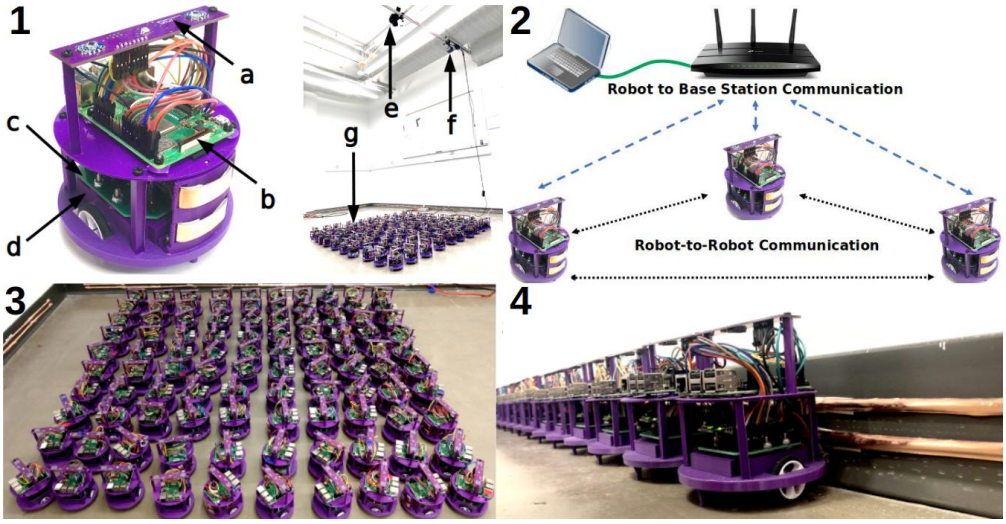

Figure 3.1. Illustration of hardware used in experiments. **(*1*)** Key components of the Coachswarm system: (1 *Left*) Robot used in the experiments. The robot is in a cylinder shape with a height of 0.12m and a radius of 0.05m. Key components are: (a) Localization system based on the HTC Vive, (b) Raspberry PI b+ computer, (c) electronics mother board, (d) rechargeable battery. (1 *Right*) Robot arena used in experiments: (e) overhead camera (only used for recording videos), (f) overhead HTC Vive base station, (g) swarm of 100 robots. **(*2*)** Illustration of the Coachswarm communication network. The green link is an ethernet connection between the base station and the Wi-Fi router. The blue links are TCP/IP connections, and the black links are layer 2 broadcasting connections. **(*3*)** The swarm of 100 robots. **(*4*)** The robots charging by connecting to two metal strips attached to the wall.

independently controls the speed and direction of each DC motor. Third, a Bluetooth low energy (BLE) module that is able to remotely turn the robots on, or put the robots into a low-energy sleep mode. The reason for having this BLE based remote switch is that: manually switching the power on or off on a large swarm becomes impractical, as the time to perform this necessary operation increases linearly with swarm size. Fourth, a 4.2v, 2500mAh lithium ion battery powers the robot continuously for about four hours, or allows the robot to sleep for over one month. Finally, two metal pads are mounted on the front of the robot chassis to allow the robot to charge its battery by contacting an external power bus (see Fig. 3.1

(4)). A custom PCB connects all five of these elements, simplifying assembly and increasing system robustness and reliability.

The robot uses Raspberry Pi 3B+ as its main computer. The Raspberry Pi 3B+ was selected due to its powerful computation and communication capabilities and low power consumption. A Raspberry Pi 3B+ is approximately the size of a credit card, and contains a 1.4GHz 64-bit quad-core CPU, 1GB RAM, and dual-band 802.11ac wireless LAN (2.4GHz and 5GHz).

The *Coachbot V2.0* robot's position and orientation sensor is based on the HTC Vive virtual reality system. This sensor consists of two TS3633-CM1 light-to-digital converters and an Atmel ATtiny87 microcontroller mounted on a custom PCB. The two TS3633-CM1 modules convert the infrared signals from a HTC vive base station mounted on the ceiling to a digital output. The microcontroller uses the timing of these digital outputs to determine the position of each sensor relative to the base station, and transmits this information to the computation module via UART. Using this information, the the computational module can accurately and robustly estimate the robot's position and orientation.

In addition, a custom 3D-printed chassis holds those three modules together. When fully-assembled, the robot is a cylinder with a height of 0.12m and a radius of 0.05m, see Fig. 3.1 (1) for a picture of *Coachbot V2.0* robot.

On the software side, in order to make it easy for user to operate the robot, I developed *COS* (Coachbot operating system), a lightweight communication middleware built on the top of the Raspbian operating system. *COS* focuses on satisfying on the following requirements:

First, to make it easy for user to operate the robot: *COS* must reliably execute user commands, such as starting or stopping the experiment, updating robot code, etc. Moreover,

COS should abstract all hardware resources (sensing module, actuation module, etc) such that a user can manipulate the robot without considering low-level hardware.

Second, to offer reliable performance. *COS* must enable the robot's hardware modules to efficiently and reliably exchange data. To perform decentralized algorithms, *COS* must also allow the robots to efficiently and reliably communicate with each other with low latency. To perform centralized algorithms, *COS* must offer a reliable communication channel to operate the robots as well as to collect telemetry and reports from the robots.

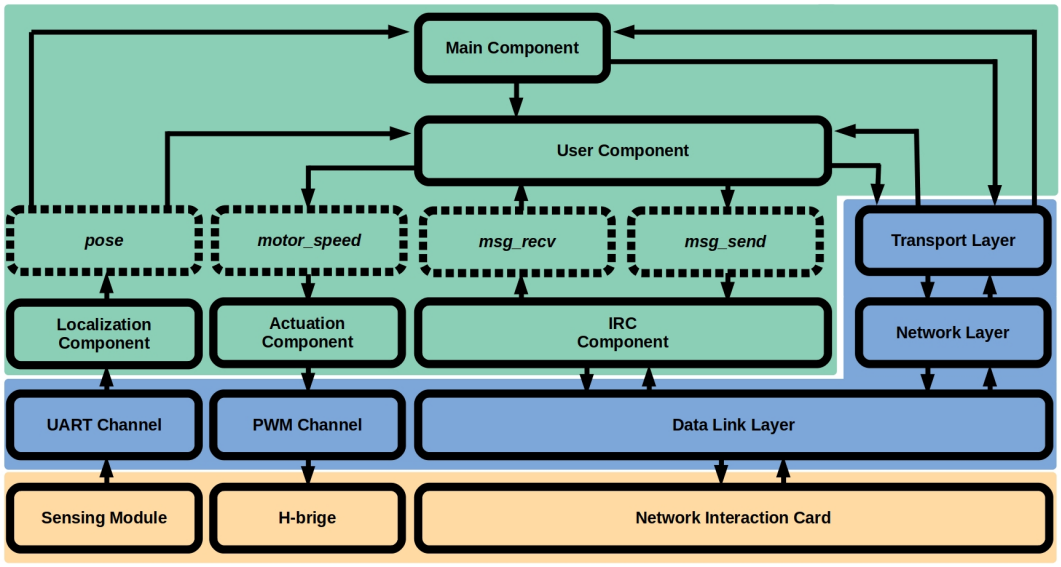

Figure 3.2. The architecture of *COS*. The elements in the orange region are hardware modules, the elements in the blue region are part of the Raspbian OS, and the elements in the green region are the *COS* elements. Among *COS* elements, solid boxes are processes and dashed boxes are shared memory. Arrows indicate the direction of data flow between two elements.

The design of *COS* uses the component model, and its architecture is illustrated in Fig. 3.2. In *COS*, each component is a process in Raspbian OS that exposes several pieces of shared memory as the interface. The current *COS* consists of five components: three components to read the data from the localization sensor, drive the motor, and handle the

inter-robot communication (IRC); a user component to execute the user code; and a main component to coordinate data from the other four components, listen to operator commands, and manage the user component according to user's command. The communication between and function of each module is described in detail below.

Inter-process communication (IPC) among components: *COS* handles IPC using shared memory. Four pieces of shared memory are used in *COS* to allow the components to exchange data:

- *pose*: Contains robot's x-y position and orientation.
- *msg_recv*: Contains the messages received from other robots. *msg_recv* is managed as a ring buffer, meaning that it will only keep the latest incoming messages once full.
- *msg_send*: Contains the message that the robot intends to transmit to other robots.
- *motor_speed*: Contains the user's desired directions and speeds for robot's left and right wheels.

User Component: The process that loads and executes the user code. User code is given to the robot as a Python script. Hardware resources are abstracted to several system calls and availible to the user as APIs. The system calls that *COS* offers to the user are:

- *get_pose():* Returns robot's current orientation and x-y position.
- *get_msg():* Returns the messages received from the other robots, then cleans the shared memory *msg_recv*.
- *send_msg(msg):* transmits the message *msg* to the other robots.
- *drive_robot(l, r):* Allows the user to send the speeds and directions of robot's left and right wheels.

- *set_led(r, g, b):* Allows user to configure the color as well as the brightness of the led on top of the robot.
- *get_clock():* Returns the time elapsed since the user code started.

Localization Component: This process reads the data from the localization sensor via UART, calculates robot's position and orientation, then writes this information into the shared memory *pose*.

Actuation Component: This process takes the user's desired velocities of robot's left and right wheels from shared memory *motor_speed*, converts the velocities into PWM signals that it outputs to the H-bridge.

IRC Component: This process writes all messages received from other robots into the shared memory *msg_recv*, and transmits the message in shared memory *msg_send* to the other robots. Currently, *COS* offers the user a vanilla broadcast IRC channel that is built directly on the data link layer, that is, *COS* passes/takes the IRC packets directly to/from the data link layer without going through transport layer and network layer. By bypassing the OS's default network stack, *COS* avoids the unnecessary communication overhead and latency, as well as the possible congestion incurred by the router. Depending on the user's demand, it is also possible to integrate other data distribution service (DDS) components into *COS* to offer user more advanced communication model.

Main Component: The main component continually monitors the status of the other components, recovering them from crashes, and, if commanded by a user, starting or stopping user code by sending the user component an OS signal. The main component communicates with the user through a TCP/IP communication channel, guaranteeing the reliable delivery of user commands to the robot. Moreover, the same TCP/IP channel is also used by the

main component to report the robot's status (such as the robot's position, battery voltage, etc.) to the user, making it easy to manage the swarm.

3.2.2. Central Server

A computer workstation manages all robots in the swarm at once, allowing a single person to easily operate the entire swarm without any direct interaction. However, the workstation does not control robots during experiments. The workstation can communicate with all robots using Wi-Fi, and can power the robots on and off using Bluetooth.

On the workstation, a custom coordination system makes use of the communication on the star network (TCP/IP links) to operate and manage the swarm in a easy, scalable way. This system has three software components: a FTP server, a monitor module, and a broadcaster module. The FTP server is used to update the code running on the robots. It can push new software to all robots at once. The monitor module is used to monitor the status of all robots, which is transmitted from each robot to the base station. It monitors information such as battery voltage, firmware version, etc. and displays it to the operator. See Fig .3.3 for a screenshot of the operator information display. The broadcaster module is used to send commands to the swarm which help in its operation, such as starting and stopping execution of the user code, turning the robots on/off, and moving robots to a charging station. The broadcaster module makes use of both Wi-Fi and Bluetooth communication.

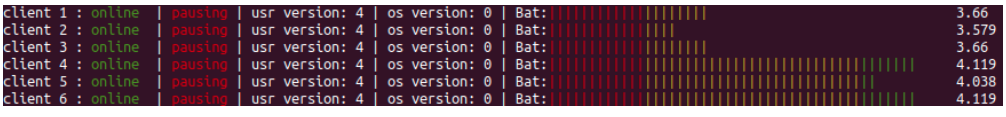

Figure 3.3. The screenshot of the operator information display, showing status of 6 (of 100) robots displayed to the operator.

3.3. Coachswarm Simulator

To assist users to execute their code on Coachswarm, I also developed a Coachswarm simulator. This simulator offers users a convenient way to develop and test their codes on their own machine before running them on the real Coachswarm. It helps to avoid the potential hardware damage caused by inappropriate user codes, in addition, it also simplifies the development of user code. This simulator consists two main components: a world engine written in C that simulates robot's on-board hardware resources, and an user-code loader written in python that executes user's code. The world engine simulates the *Coachbot V2.0* robot's motion, sensing and communication in a very realistic way: the specifications of all the simulated hardware, including the maximal speed of robot's wheel, sampling rate of robot's positioning sensor, throughput of the inter-robot communication channel, etc, are made to be consistent with the real robot. In addition, the user-code loader is designed in a way that: the code used to operate the simulated robot can be used to operate the actual *Coachbot V2.0* robot without any modification. The Docker image and an associated documentation for the developed Coachswarm simulator can be found at: https://hub.docker.com/r/hanlinwang/coachswarm

CHAPTER 4

OVERALL PIPELINE DESCRIPTION

When the given task is complex, it has been shown that the *modular programming* [65] is an useful design technique. The idea is to separate the overall task into sub-problems and solve each sub-problem independently. It helps to simplify the debugging processes, moreover, the developed solution to each sub-problem can also be reused by other applications. In this chapter, I present a pipeline that breaks the overall task of persistent shape formation into four independent problems. This presented pipeline also serves as the roadmap of this dissertation.

The pipeline for persistent shape formation can be briefly described as follows: given the desired shape (a binary image), each robot will first estimate the swarm size n, and then encode the desired shape to a set of n goal points. In addition, each robot will constantly monitor the swarm size on the fly, once it detects that the swarm size has changed, it will update the goal point set accordingly. At the same time, the robots will use the local sensing and communication to actively and cooperatively occupy those goal points corresponding to the current swarm size, forming the input shape persistently.

Following the pipeline above, the overall method is made up four algorithms:

- *swarm size estimation*: a decentralized algorithm that uses the local information to estimate the number of robots currently in the swarm.
- *goal configuration generation*: an algorithm that uses arbitrary number of robots to represent a user-specified shape.

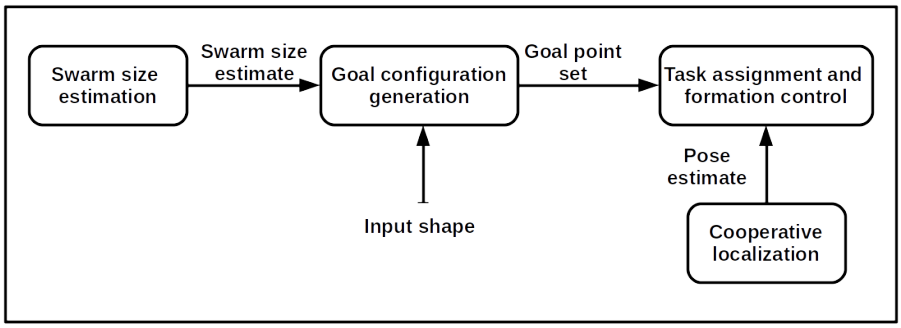

Figure 4.1. Diagram illustrating the robot's overall behavior. Each box indicates an algorithm developed in this dissertation. Arrows indicate the data exchanged between components. Note that each algorithm is executed in parallel, allowing the algorithm to adapt to the unexpected external disturbances at run time.

- *cooperative localization*: a decentralized algorithm for localizing a swarm of robots with peer-to-peer measurements.
- *task assignment and formation control*: a decentralized algorithm that assigns each robot its location in shape and routes each robot to reach its assigned goal location.

From chapter 5 to chapter 8, I will present four algorithms that solve those four problems above. Furthermore, in chapter 9, I will describe in detail how to assemble all those four developed algorithms into a full persistent shape formation algorithm. See Fig.4.1 for an diagram illustrating how each developed algorithm module will interact with each other. Note that, each presented algorithm is not the only solution to its corresponding subproblem. Depending on the user's application, it is possible to replace each presented algorithm module with some other different algorithms to obtain a better performance.

CHAPTER 5

SWARM SIZE ESTIMATION

This chapter describes a decentralized algorithm for computing the number of robots in a swarm, only requiring communication with neighboring robots. The algorithm can adjust the estimated count when the number of robots in the swarm changes, such as the addition or removal of robots. Probabilistic guarantees are given, which show the accuracy of this method, and the trade-off between accuracy, speed, and adaptability to changing numbers. The proposed approach is demonstrated in simulation as well as a real swarm of robots. The algorithm presented in this chapter has been published as [41].

5.1. Background

The number of robots in a robotic swarm is used in a wide variety of applications; from self-assembling a shape at a scale proportional to the number of robots in a swarm, to optimizing behaviors based on swarm size. While some approaches make use of the swarm size given to it a priori [31], having the swarm learn its size on the fly offers more flexibility to behaviors and make it more tolerant to failures that may inadvertently change the swarm size.

One application in swarm robotics that explicitly or implicitly uses the swarm size is shape formation. In [31] the shape is sized to fit the number of robots in the swarm. For [34, 40] the number of robots is implicitly learned by building the shape at larger and larger scales until the shape can no longer be built completely with the swarm. Building these

intermediate sizes increases assembly time, and in some cases [40], it limits the types of shapes that can be formed. Quickly knowing the size of the swarm would greatly increase assembly speed.

Other application to swarm robotics could take advantage of knowing the swarm size to optimize various behaviors of robots. For example in task allocation [66] when assigning tasks to individuals, inefficiencies can occur for larger swarms [67] due to interference and traffic, if the number of robots was known, the allocation of tasks could be adjusted for optimality. Other tasks, such as computing a distributed consensus value [68] require waiting for messages to reach all individuals in the swarm. To ensure this, implementations of these algorithms assume a maximum number of robots in the swarm and set the wait time to allow the message to reach all robots in a swarm that size. If the swarm size was known, a less conservative wait time could be used, speeding up the distributed computation.

Additionally, knowing the swarm size could allow for new approaches that change behavior based on swarm size. For example, a swarm could use more conservative behaviors for small numbers, but could use riskier behaviours for larger sized swarms, where the loss of individuals would have less of an impact on outcome.

Some previous approaches for counting in robot swarms and adhoc networks include electing a leader then using distributed token passing [69] and counting the number of times a token is passed, or building a network tree [70] and propagating counting information from the leaves to the root. These types of approaches are difficult to implement in a moving robotic swarm where the communication topology can be dynamic.

Other approaches are based on a distributed gossip algorithm [71], however there is an implicit maximum number allowed, based on the time allowed for each gossip round. The

approach by [72] and extended by [73] can measure swarm size in dynamic systems, but either requires long range communication or an additional messaging overhead to propagate local messages globally. Similar approaches first elect a single leader and then run distributed consensus between the leader's value of 1 and all others with a value of 0 [74]. The average here will be $\frac{1}{n}$ where n is the swarm size. This approach requires synchronized communication and has an implicit maximum size based on the time the distributed consensus algorithm is run.

5.2. Algorithm Description

This chapter proposes an algorithm that can estimate the number of robots in the swarm, which is scalable, can adapt to changing numbers of robots, works in the dynamic communication environments of swarms and does not have an implicit maximum it can count.

The proposed algorithm is inspired by the following simple idea: Let $s_1, ...s_n$ be n independent samples that are taken from a uniform probability distribution between 0 and 1, we compute the max value of all samples, $\max_{1\leq i\leq n} s_i$. The more samples that are taken, i.e. the larger n, the closer $\max_{1\leq i\leq n} s_i$ is expected to be to 1. Furthermore, the value of $\max_{1\leq i\leq n} s_i$ can be used to estimate the number of samples, n. This can be repeated multiple

Algorithm 1: Pseudo code for swarm size estimation

```
1  maxs ← {} // the variable to record sample maximums in all m rounds
2  for j ← 0 to m do
3      max ← 0  // the variable to find the sample maximum in current round
4      for i ← 0 to n do
5          s_i ← X_i ∼ U(0, 1)
6          if s_i > max then
7              max ← s_i
8      maxs ← {max} ∪ maxs
9  k← average of all the elements in maxs
10 n* = k/(1−k)  // calculate the estimation of the swarm size
```

rounds, where in each round, j, a new set of n samples are generated and the maximum is computed just for the samples in that round $\max^{j}_{1\leq i\leq n} s_i$. The average of these max samples found in all m rounds, $k = \frac{\sum_{j=1}^{m} \max^{j}_{1\leq i\leq n} s_i}{m}$, can be used to provide an estimate of n with less variance when compared to any single rounds $\max_{1\leq i\leq n} s_i$. See Algorithm 1 for the detailed pseudo code.

This idea is mapped to a robotic system by having each robot generate a random number sampled from a uniform probability distribution between 0 and 1. The swarm then communicates amongst themselves to compute the largest number generated by any robot, and uses that number to estimate the number of robots in the swarm. As before, this process can be repeated for multiple rounds to give better estimates of n.

5.3. Theoretical Results

This section studies the correctness and efficacy of the proposed algorithm. I first use lemma 5.1 to show that the algorithm is correct, and then use the theorem 5.1 to show that the error of estimation will linearly converge to 0 in probability.

Definition 5.1. *Let n be the actual swarm size, n^* be the estimation obtained from Algorithm 1, I define the **RE** (relative error) of the estimation n^* as:*

$$RE(n^*) = \frac{|n^* - n|}{n^* + 1}$$

Lemma 5.1. *The sample maximum of n i.i.d. $\mathcal{U}(0,1)$ random variables is an unbiased estimator of $\frac{n}{n+1}$.*

Proof. See Appendix 1. □

Theorem 5.1. *Given an error margin ϵ, the probability that the RE(n^*) exceeds ϵ will linearly decay over number of trials m. Specifically:*

$$Pr\{RE(n^*) \geq \epsilon\} < \frac{1}{m\epsilon^2}$$

Proof. See Appendix 2. □

Remark: The result obtained in Theorem 5.1 suggests that the convergence rate of *RE(n^*)* is independent of number of samples n, which suffices to show that the algorithm is **scalable** from a theoretical perspective.

5.4. Algorithm Implementation

In practice, when implementing the proposed algorithm, each agent needs to wait "long enough" between trials so as to give the swarm sufficient time to find the largest number in each trial, i.e, enable the random number generated by itself to propagate through the entire swarm. However, this waiting time is essentially a function of swarm size, which incurs the "chicken and egg" paradox, as the task here is to estimate the swarm size.

To tackle this problem, I developed an error-driven method with which each agent a_i can use the local-only observation to maintain an estimation of the necessary waiting time. The idea is shown as following: It is assumed that each agent a_i transmits the messages at the same constant frequency f. Through the experiment, each agent will maintain an estimation of swarm's communication diameter d_i, i.e. the longest distance (in term of communication hop) between one agent to another in swarm, and it use this d_i to calculate the waiting time. To be specific, between two adjacent trials, a_i waits $\frac{4d_i}{f}$ amount of time.

To estimate d_i, during each trial j, each agent uses the hop-count algorithm [48] to estimate the distance (in term of communication hop) hop_i^j between itself and the agent that generates the largest number of the trial. If the agent **underestimates** the communication diameter, then two error events may occur, where a_i will update d_i:

- a_i sees a neighbor a_j such that $d_j > d_i$: This implies some other agent has seen a larger pairwise communication distance. In this case, a_i set $d_i = d_j$;
- d_i is less than the hop_i^j: This suggests the agent's diameter estimation is less than the communication distance between itself and some other agent. This is a more serious error because hop_i^j is already a lower bound of the communication diameter, hence in this case we punish d_i by setting $d_i = 2\ hop_i^j$.

It is straight forward to examine that the waiting time $\frac{4d_i}{f}$ is sufficient to enable the agent to detect these two errors before it starts the next trial.

The other challenge I have when developing a decentralized implementation of the algorithm is that: In reality, agent's clocks are not perfectly synchronized. To solve this problem, I embedded a sequence number in each trial, and enforce the agent to only use the information coming from the message that has the same sequence number. See Algorithm 2 for implementation details.

The algorithm is implemented in a "listen-think-talk" manner. Specifically, the algorithm consists of three modules: main module, broadcast module, and message handler module. The broadcast module constantly transmits messages to neighbors at a fixed frequency f; the main module handles computation and memory management, and message handler modifies the local variables according to the incoming messages. These three modules can be implemented using three separate threads that communicate through shared memory. The

sketches of these three modules are shown in Algorithm 2, 3, and 4. Note that all the variables are thread-public.

5.4.1. Main module

The *Main module* takes two inputs: f and m where f is the communication frequency and m is the number of trials that will be used for estimation. The algorithm first claim and initialize the variables that will be used in the later calculation, to be specific (Algorithm 2, Line 1-10):

- *buff*: a ring buffer whose length is m, it is used to store the most recent m trials' results in the history;
- *index*: the variable that helps to operate *buff*;
- *seq_#*: each trial's sequence number;
- *sample_max*: the largest number of current trial;
- n^*: agent's estimation of swarm size;
- *diameter*: agent's estimation of the communication diameter of swarm;
- *hop* and *max_hop_seen*: the variables that help to update *diameter*;
- *last_check*: the beginning time of current trial.

After the initialization phase, the agent enters the main loop. At the beginning of each iteration, the agent first forges the message that will be transmitted by Broadcast Module (Algorithm 2, Line 12), then checks whether it has already waited long enough to start the next trial (Algorithm 2, Line 13). If the agent has waited long enough already, it first checks whether the current estimation of swarm's communication diameter needs to be updated (Algorithm 2, Line 14-18), and if the current diameter estimation is correct, the agent then

adds the result to *buff* (Algorithm 2, Line 19), updates the estimation n^* (Algorithm 2, Line 21-22), and initiates another trial (Algorithm 2, Line 13-25).

Algorithm 2: Main Module

```
   Input: m, f
1  buff ← {0}
2  index ← 0
3  seq_#← 0
4  sample_origin ← rand(0, 1)
5  sample_max ← sample_origin
6  n* ← 0
7  hop ← 0
8  max_hop_seen ← 1
9  diameter ← max_hop_seen
10 last_check ← clock()
11 while agent is active do
12 |  msg ← {seq_#, sample_max, hop, max_hop_seen}
13 |  if clock() - last_check > 4 diameter / f then
14 |  |  last_check ← clock()
15 |  |  if hop > max_hop_seen then
16 |  |  |  max_hop_seen ← hop
17 |  |  if diameter < max_hop_seen then
18 |  |  |  diameter ← max_hop_seen
19 |  |  |  continue
20 |  |  buffer[index] ← sample_max
21 |  |  index = (index + 1) mod m
22 |  |  k ← average(buff)
23 |  |  n* ← k / (1−k)
24 |  |  seq_#← seq_# + 1
25 |  |  sample_origin ← rand(0, 1)
26 |  |  sample_max ← sample_origin
```

Algorithm 3: Broadcast Module

```
1 while agent is active do
2 |  transmit msg
3 |  sleep 1/f
```

5.4.2. Message Handler

When receiving a message, the Message Handler first compares *msg.seq_#* with the local *seq_#*. There are three possible cases:

- If *msg.seq_# - 1 > seq_#*, it implies that the local clock is slower than the neighbor by more than one trial, the agent then sets *seq_# = msg.seq_# - 1* so as to catch up with the neighbor (Algorithm 4, Line 3-4).
- If *msg.seq_# < seq_#* , it suggests that there is a neighbor that is slower than the agent, then the agent will delay the next trial so as to give the neighbor time to catch up.
- If the neighbor and the agent have the same sequence number, then the agent will use the information in the message to update its local variables (Algorithm 4, Line 8-15): Algorithm 4, Line 8-9 is for finding the largest random number in the current trial; Algorithm 4, Line 10-13 is for finding the distance (in term of communication hop) between itself and the agent that generates the largest number of the trial; Algorithm 4, Line 14-15 is for finding the largest pairwise communication distance in swarm.

5.4.3. Complexity

First, one can easily examine that the algorithm's memory complexity is dominated by the size of buffer to store the trials' results in Algorithm 2, In the other words, the algorithm's memory complexity is $\mathcal{O}(m)$.

Next, if it is assumed that the computation complexity of querying a random number generator is $\mathcal{O}(1)$, then the computational cost of each iteration of the algorithm is dominated by the calculation to find the average over *buff*, as a result, the algorithm's computation complexity is $\mathcal{O}(m)$.

Algorithm 4: Message Handler

```
1  while agent is active do
2      if receive a msg then
3          if msg.seq_# - 1 > seq_# then
4              seq_# ← msg.seq_# - 1
5          if msg.seq_# < seq_# then
6              last_check ← clock()
7          if msg.seq_# == seq_# then
8              if msg.sample_max > sample_max then
9                  sample_max ← msg.sample_max
10             if msg.hop < hop then
11                 hop ← msg.hop + 1
12             if sample_max == sample_origin then
13                 hop ← 0
14             if max_hop_seen < msg.max_hop_seen then
15                 max_hop_seen ← msg.max_hop_seen
```

Last, it is straight forward to examine that the algorithm's communication complexity, i.e., the length of each message, is $\mathcal{O}(1)$.

Remark: The results I obtained in this section suggests that the algorithm's cost is independent of swarm's size n with respect to computation complexity, memory complexity, and communication complexity, which suffices to show that the algorithm is **scalable** from a engineering perspective.

5.5. Performance Evaluation

5.5.1. Basic Simulations

To simulate the estimation algorithm on a large number of robots, and on a large number of experiments, I first implemented the behavior on a simplified simulation in python. In this simplified simulation, It is assumed robots are all synchronized and have global communication. Here, the non-ideal effects of a robotic implementation are ignored to allow for fast

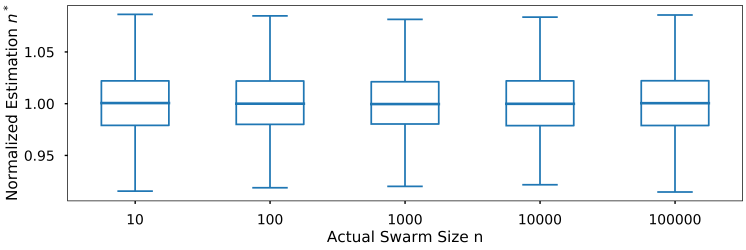

Figure 5.1. The distribution of estimates for the number of robots for varying swarm size. The distribution is normalized by the true number of robots. Each boxplot represents the distribution of 2000 experiments in which the swarm used 1000 rounds to estimate the number of robots.

and efficient experiments on up to 100,000 simulated robots. These first simulations were used to validate predicted properties of the algorithm such as scalability and the effect the number of rounds has on precision of estimates.

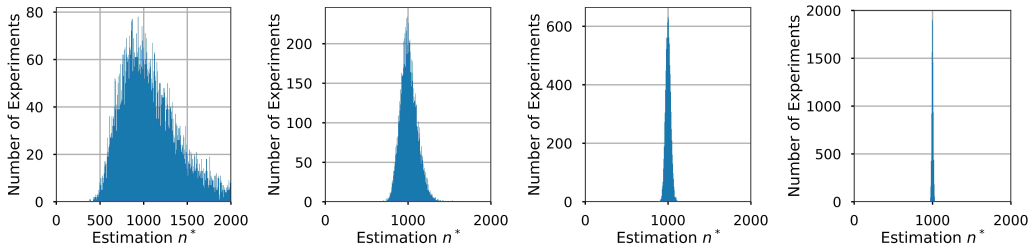

Figure 5.2. From left to right: the histograms of estimates of the swarm size for a 1000 robot swarm after 10, 100, 1000, or 10000 rounds. Each histogram represents the range of estimates of 10,000 experiments.

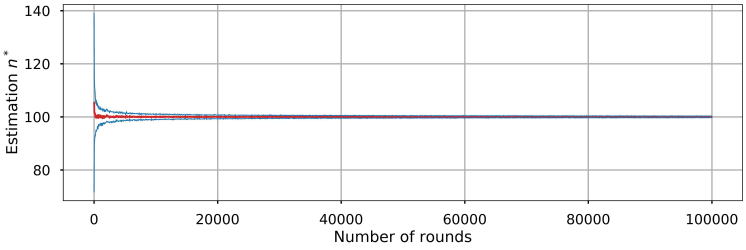

Figure 5.3. The average, and 1 standard deviations above and below average of 1000 experiments with 100 simulated robots as increasing number of rounds are included in the size estimate.

One property of this algorithm predicted in our analysis is that the normalized error (i.e. the percentage of error) is only dependent on the number of rounds, not the swarm size. To show this, I looked at swarms sized from 10 to 100,000 and ran the estimation for 1000 rounds. This was repeated for 2000 experiments. Figure 5.1 shows that the normalized accuracy of swarm size estimates are indeed independent of swarm size.

Next, the simplified simulation was used to show how the number of rounds used to estimate swarm size has an effect on the accuracy of the estimate. In this test, the swarm size is set to 1000, in each experiment, the robots use 10, 100, 1000, and 10000 rounds trials to estimate the swarm size. Fig. 5.2 shows the estimation of 10,000 experiments for varying numbers of rounds. As expected, the result of this test shows that increasing the number of rounds produces less variance in the estimated swarm size.

A third test quantitatively studies how the variance of swarm size estimate changes over the number of rounds included in the estimate. In this test, the swarm size is set to 100, the task is to estimate this swarm size using different numbers of trials ranging from 10 to 100,000. The result of this test is shown in Fig. 5.3. As we can see in the plot, the variance of the swarm size estimate sharply decreases as more rounds are included in the estimate.

5.5.2. Kilobot Simulations

To validate the method's performance on a large-scale swarm system, I tested the algorithm using an agent-based simulator that is originally developed for [75]. The simulator implements both the Kilobot's motion and communication in a very realistic way. Moreover, the user interface of the simulator is exactly the same as the one that is on actual Kilobot. In the simulation, each agent communicates with neighbors at a frequency of 15Hz and the maximum transmission unit is 9 bytes.

First, I use simulation to investigate the effect of the buffer size m on the algorithm's accuracy and adaptability to changes. In simulation, up to 1000 agents execute our algorithm to estimate the swarm size using varying of trials in the past. The initial swarm size is 1000, and after 1000 seconds, I remove half of agents from the swarm. Fig. 5.4 shows the each agent's estimation about swarm size over time.

In the second test, I use simulation to investigate the convergence rate of the method. In these experiments, swarms of size 10 to 1000 agents use all the trials in the history to estimate the swarm size. The result is shown in Fig. 5.5.

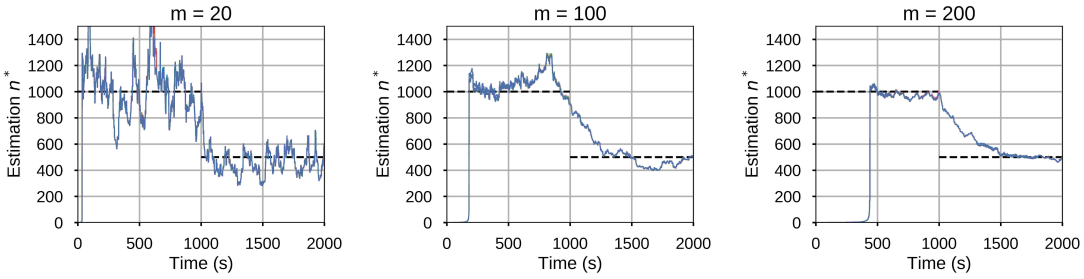

Figure 5.4. The results for the test where up to 1000 agents estimate the swarm size using fixed number of trials. From left to right: $m = 20, 100, 200$. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time. The different colors indicates the results from different agents, plots are overlapping as many have similar estimates.

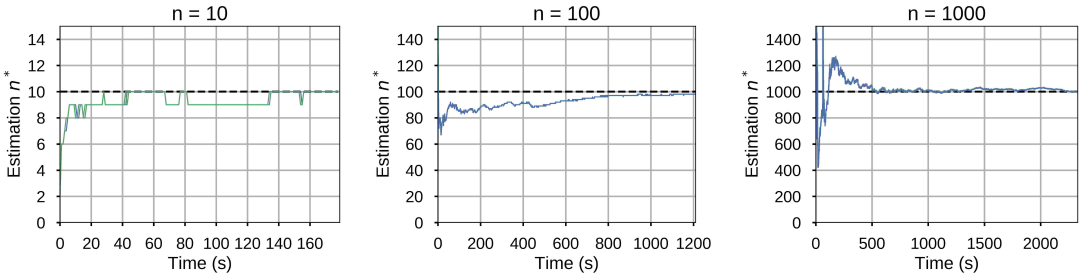

Figure 5.5. The results for the tests where agents estimate the swarm size using all the trials in the history. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time, plots are overlapping as many have similar estimates.

A third simulation is given to demonstrate our algorithm's performance on the swarm with extremely large size. In this test, 5000 simulated agents perform two experiments: In first experiment, agents use 200 trials in the past to estimate the swarm size, and the swarm size is initialized to be 5000 then drops to 2500 after 3000 seconds, the result of this test is shown in Fig. 5.6 (Left). In the second experiment, 5000 simulated agents use all the trials in the history to do the estimation, and the swarm size does not change in this experiment, the result of this second test is shown in Fig. 5.6 (Right).

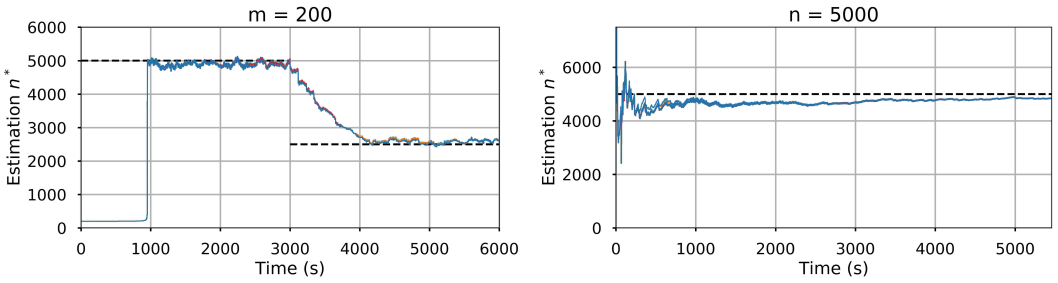

Figure 5.6. The experiments to demonstrate the algorithm's performance on a swarm of 5000 simulated agents. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time. The different colors indicates the results from different agents, plots are overlapping as many have similar estimates.

5.5.3. Real Kilobot implementation

To validate our algorithm's performance beyond simulation, I implemented our algorithm on a swarm of 35 Kilobots [51]. In reality, it is hard to collect the data from each individual in real-time. As a result, instead of paying attention to each individual's estimation, I use a "probe message" to collect the minimal and maximal estimation amongst the swarm.

I performed two tests on the physical Kilobot swarm: The first test is given to demonstrate the algorithm's convergence rate, and the second second test is given to show the

algorithm's accuracy and adaptability to changes. In the first test, 35 Kilobots are tasked to use all the trials in their history to estimate the swarm size, the result for this test is shown in Fig. 5.7 (top); in the second test, Kilobots use 100 most recent trials to estimate the swarm size, moreover, the swarm size is initialized to be 35 then drops to 25 after 2600 seconds. The result for the second test is shown in Fig. 5.7 (bottom).

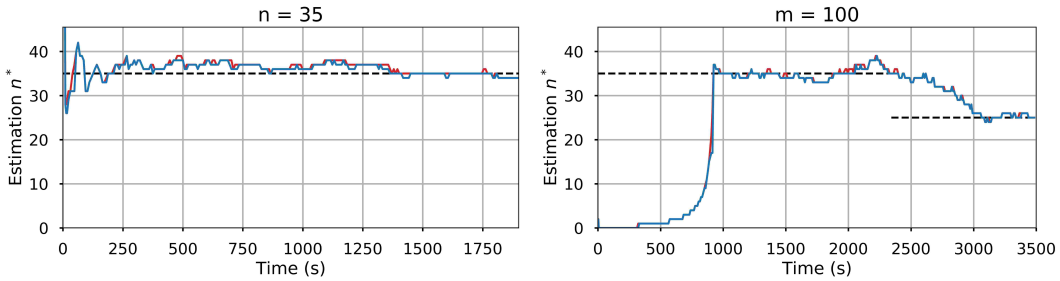

Figure 5.7. The results from physical experiments. The black dotted line is the actual swarm size n, the red and blue solid lines are largest and smallest estimation amongst the swarm, respectively.

5.5.4. Real *Coachbot V2.0* robot implementation

Due to the fact that the Kilobot's hardware capabilities are fairly limited, in the experiments presented in Section 5.5.3, it took a long time for the Kilobot swarm to accurately estimate the swarm size. In this section, I implement the algorithm on the *Coachbot V2.0* swarm. The result of the experiment shows that, for the robots with more advanced hardware capabilities, *Coachbot V2.0* for example, the presented algorithm can actually estimate the swarm size pretty fast. Initially, 100 robots are located in the arena, then, at $t = 50$ s, the robots that are located in the left half of the arena are removed from the swarm. In this experiment, The robot's communication rate is set to 25 hz, each robot uses 300 most recent trials to estimate the swarm size. The result of this experiment is shown in Fig. 5.8. As we can see

in the plot, *Coachbot V2.0* swarm is able to accurately estimate the swarm size within in 15 s, in addition, when the swarm size changes, the swarm is able to adapt within 10 s.

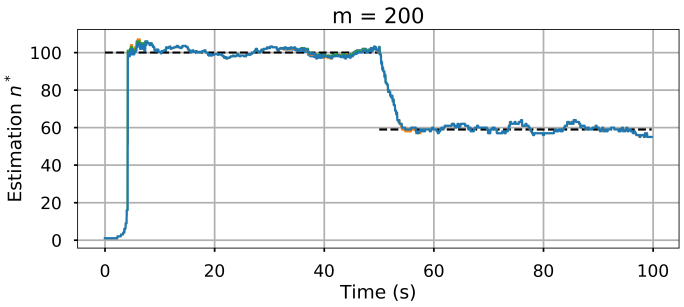

Figure 5.8. The experiment running on a swarm of up to 100 *Coachbot V2.0* robots. The black dotted line is the actual swarm size n and the solid colored lines are agents' estimations over time. The different colors indicates the results from different agents, plots are overlapping as many have similar estimates.

CHAPTER 6

GOAL CONFIGURATION GENERATION

This chapter presents an algorithm that automatically encodes a user-defined complex 2D shape to a set of cells on a grid each characterizing a robot currently in the swarm. The algorithm is validated via up to 200 simulated robots as well as up to 100 physical robots. The results show that the goal configurations generated by the algorithm for the swarms with any size are consistent with the input shapes, moreover, it allows the swarm to adapt to the swarm size change quickly and robustly. The algorithm presented in this chapter has been published as [42]. A summary video of the presented algorithm can be found in [76].

6.1. Background

In this past, many methods to represent the desired shapes has been presented, including curves or regions explicitly described by a mathematical formula [77, 78], potential fields [79], masked grid [39, 40, 44], and more [49, 50]. The mathematical formula-based representations [77–80] can help to derive the formation control laws when the robot's kinematic or dynamic constraints need to be considered. However, when the desired shape is complex, it is time-consuming (sometimes even impossible) to encode the desired shape to a mathematical formula. Masked grid, also known as "binary image" in 2D case [39] or "binary volumes" in 3D case [40], is a grid where each cell is labeled with either a 1 or a 0, indicating whether the cell is in the shape or not, and the desired shape is described as the set of in-shape cells on the grid [39, 40, 44]. Masked grid is a convenient way to encode the complex shape, in

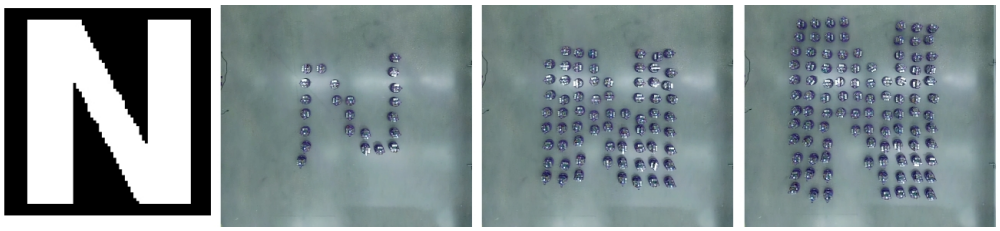

Figure 6.1. From left to right: the target shape – "N"; the swarms with different sizes forming the configurations generated by the proposed algorithm

addition, for the swarm of modular robots that are with discrete attachment locations [8, 81], the masked grid is a natural way to describe the collective's configuration.

Among previous methods, most of them assume that the swarm size does not change [44, 77–80] , only a few consider the situations where the swarm size could dynamically change [38–40]. When the removal or addition of the robots occurs, there are two strategies for the swarm to adapt. One option is to keep the scale of the desired shape fixed, and change the density of the robots [38]. One drawback of this method is that: when the robot's physical size is finite, the size of swarm to display the shape will be limited, as one can fit only finite amount of robots in a unit of space. On the contrary, the other option is to keep the density of the robot fixed and change the size of the desired shape [39, 40]. When using the masked grid to describe the target shape, there are two options to scale the goal configuration: change the number of robots in each cell and fix the number of in-shape cells [39], or, change the number of in-shape cells and fit exactly one robot to each cell [40]. As shown in [39, 40], both of these two methods can offer the swarm the capability of self-healing, making the system resilient to the removal and addition of robots. On the other hand, for the algorithm proposed in [39], when the swarm size changes, it takes the swarm a long time to adapt, as the swarm needs to wait "long enough" to sense the change of the swarm size. Moreover,

the algorithm presented in the [40] only works for certain types of shapes, and the generated configurations can be perfectly formed only by the swarms with certain sizes.

This chapter presents an algorithm that automatically encodes an input 2D shape (given by an binary image) to a masked square grid where each in-shape cell characterizes a robot current in the swarm. Given an input shape and the swarm size n, the algorithm will first use naive binary image scaling methods to generate two reference grids, in which one has slightly more than n in-shape cells and the other has slightly less than n in-shape cells, then use a second subroutine, called *interpolation*, to refine those two reference grids so as to obtain the final output – a masked grid with exactly n in-shape cells. The algorithm is validated via both the simulated and physical experiments, the results show that the goal configurations generated by the algorithm are consistent with the original input shapes, moreover, when the swarm size changes, it allows the swarm to adapt quickly and robustly.

6.2. Preliminaries

In this section, I will formally state the problem, and introduce the notations frequently used in the rest of the chapter.

6.2.1. Goal Configuration Generation: Problem Statement

The proposed algorithm takes two inputs: an binary image describing the desired shape, and the size of the swarm to display the desired shape. Note that a binary image is essentially a 2D masked grid, therefore, for the sake of description, in the rest of this chapter, I use the word "pixel" and the word "cell" interchangeably. The output of the algorithm is a masked grid such that: the number of in-shape cells must equal to the input swarm size.

When designing the algorithm, there are two factors to be considered: first, the generated goal configurations should be consistent with the input shape; second, in order to allow the swarm to quickly adapt to the removal and addition of the robots, the goal configurations generated for different swarm sizes should be similar to each other as well.

6.2.2. Notations

Let $\mathcal{G}_i$ be a 2D $m \times m$ masked square grid i, $c_i^{(x,y)} \in \{0, 1\}$ denotes the label of the cell in the x-th row and y-th column from the left-top corner, and $\mathcal{S}(\mathcal{G}_i) = \{(x, y) \mid c_i^{(x,y)} = 1\}$ denotes the set of the coordinates of all in-shape cells on $\mathcal{G}_i$. Given a set $\mathcal{A}$, $|\mathcal{A}|$ denotes its cardinality. For a pair of sets $\mathcal{A}$ and $\mathcal{B}$: $\mathcal{A} \cup \mathcal{B}$ denotes the union of set $\mathcal{A}$ and set $\mathcal{B}$; $\mathcal{A} \cap \mathcal{B}$ denotes the intersection of set $\mathcal{A}$ and set $\mathcal{B}$; $\mathcal{A} - \mathcal{B}$ denotes the set of all the elements that are in set $\mathcal{A}$ but not in set $\mathcal{B}$. For $n \geq 3$ sets $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{n-1}$, their union is denoted as $\bigcup_{i=0}^{n} \mathcal{A}_i$, in addition, $\bigcup_{i=a}^{a} \mathcal{A}_i = \emptyset$.

6.3. Approach

It is assumed that the desired shape is given to the swarm in the format of a 100×100 binary image, however this approach can be generalized to any size binary image. The

Algorithm 5: Pipeline for proposed algorithm

Input: Input shape $\mathcal{G}_{in}$, swarm size n
Output: Configuration for the swarm $\mathcal{G}_o$

$\mathcal{G}_o^l, \mathcal{G}_o^h \leftarrow scaling(\mathcal{G}_{in}, n)$
if $|\mathcal{S}(\mathcal{G}_o^l)|$ *is* n **then**
  $\mathcal{G}_o \leftarrow \mathcal{G}_o^l$
else
  if $|\mathcal{S}(\mathcal{G}_o^h)|$ *is* n **then**
    $\mathcal{G}_o \leftarrow \mathcal{G}_o^h$
  else
    $\mathcal{G}_o \leftarrow interpolation(\mathcal{G}_o^l, \mathcal{G}_o^h, n)$
return $\mathcal{G}_o$

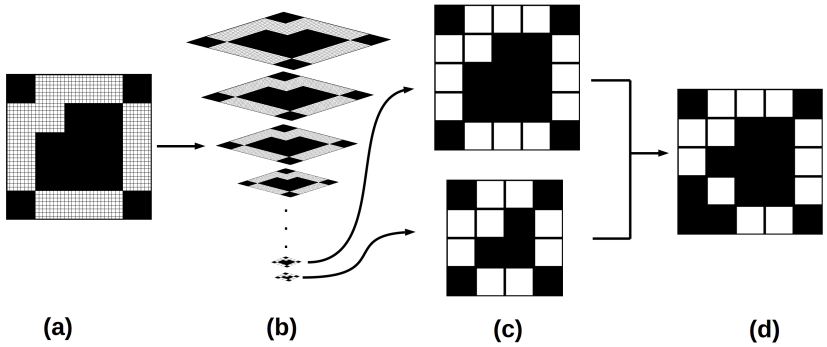

Figure 6.2. The graphical illustration of the overall pipeline of the presented algorithm. From left to right: (a) The input shape, which is given in the format of a binary image. In this example, the task is to find a goal configuration for a swarm of 12 robots; (b) The *scaling* subroutine is applied to the input masked grid so as to find two reference grids with approximately 12 in-shape cells; (c) Two reference masked grids with each pixel enlarged for the visualization purpose; (d) A configuration with 12 in-shape cells is constructed by the *interpolation* subroutine using those 2 reference grids in (c).

proposed algorithm consists of two subroutines – *scaling* and *interpolation*. Given the desired shape $\mathcal{G}_{in}$ and the swarm size n, the algorithm will first use the *scaling* subroutine to find two reference masked grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ such that: $\mathcal{G}_o^h$ has slightly more in-shapes than n and $\mathcal{G}_o^l$ has slightly less in-shapes than n, then apply the *interpolation* subroutine to $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ so as to obtain an output that is with exactly n in-shape cells. A graphical illustration of the overall pipeline is shown in Fig. 6.2 and a detailed description of algorithm's overall pipeline is shown in Alg. 5.

6.3.1. Scaling

The *scaling* subroutine first uses the *image scaling* to change the number of in-shape pixels. Image scaling is a well studied topic [82, 83], here, the task is to create a new version of the image with a different width and/or height in pixels. Many strategies to scale a binary

Algorithm 6: *scaling* subroutine

Input: Input shape $\mathcal{G}_{in}$, swarm size n
Output: Two reference masked grids $\mathcal{G}_o^l$, $\mathcal{G}_o^h$

1 $m \leftarrow 100$ // inialize m to the size of $\mathcal{G}_{in}$
2 $iter \leftarrow 1$
3 $last \leftarrow \mathcal{G}_{in}$ // variable to store the last scaled grid
4 **if** $|\mathcal{S}(\mathcal{G}_{in})|$ *is* n **then**
5 $\mathcal{G}_o^l \leftarrow \mathcal{G}_{in}$, $\mathcal{G}_o^h \leftarrow \mathcal{G}_{in}$
6 **if** $|\mathcal{S}(\mathcal{G}_{in})| < n$ **then**
7 **while** *1* **do**
8 $m \leftarrow m + 1$ // gradually increase scaled grid size
9 $cur \leftarrow$ scale the grid $\mathcal{G}_{in}$ to size $m \times m$
10 **if** $|\mathcal{S}(cur)| \geq n$ **then**
11 $\mathcal{G}_o^l \leftarrow last$, $\mathcal{G}_o^h \leftarrow cur$
12 **break**
13 **else**
14 $last \leftarrow cur$
15 $iter \leftarrow iter + 1$
16 **if** $|\mathcal{S}(\mathcal{G}_{in})| > n$ **then**
17 **while** *1* **do**
18 $m \leftarrow m - 1$ // gradually decrease scaled grid size
19 **if** $m < 15$ **then** // switch to skeletonization
20 $cur \leftarrow$ skeletonize the grid $last$
21 **if** $|\mathcal{S}(cur)| \leq n$ **then**
22 $\mathcal{G}_o^l \leftarrow cur$, $\mathcal{G}_o^h \leftarrow last$
23 **break**
24 **else**
25 Abort: the input n is too small.
26 $cur \leftarrow$ scale the grid $\mathcal{G}_{in}$ to size $m \times m$
27 **if** $|\mathcal{S}(cur)| \leq n$ **then**
28 $\mathcal{G}_o^l \leftarrow cur$, $\mathcal{G}_o^h \leftarrow last$
29 **break**
30 **else**
31 $last \leftarrow cur$
32 $iter \leftarrow iter + 1$
33 **return** $\mathcal{G}_o^l$, $\mathcal{G}_o^h$

image have been proposed in the past, in the presented algorithm, I use the *nearest neighbor interpolation* [82] as the *image scaling* method.

Given an input shape $\mathcal{G}_{in}$ and swarm size n, there are three possible cases: If the number of in-shape cells on the input grid $|\mathcal{S}(\mathcal{G}_{in})|$ equals to n, the algorithm will return $\mathcal{G}_{in}$ directly (Alg. 6, Line 4-5). If the $|\mathcal{S}(\mathcal{G}_{in})| < n$, the algorithm will first keep upscaling the $\mathcal{G}_{in}$ (Alg. 6, Line 6-15) until a grid that has more than n in-shape cells is found (Alg. 6, Line 10 - 12), then return the two scaled images obtained most recently, and exit this subroutine (Alg. 6, Line 11-12). Similarly, if $|\mathcal{S}(\mathcal{G}_{in})| > n$, the algorithm will keep downscaling the $\mathcal{G}_{in}$ until finding a grid that contains less than n in-shape cells (Alg. 6, Line 16-32).

One issue for using the *image scaling* to reduce the number of in-shape cells is that: When the size of the scaled image is too small ($\leq 15 \times 15$ according to the experiments), it often fails to preserve the main structure of the input shape. Therefore, to prevent the main

structure of the shape in scaled image being distorted by over-downsampling, the size of the scaled image cannot be smaller than a threshold (Alg. 6, Line 20). This limits the minimal number of in-shape cells in the outputs that can be generated.

Besides the *image scaling*, an alternative to reduce the pixels required to display a shape is the operation *skeletonization* [84]. The operation *skeletonization* generates a "thinner" version of the input shape that emphasizes shape's geometrical and topological properties. The *scaling* subroutine uses the operation *skeletonization* to extend the range of swarm sizes for which the presented method can work: if the image scaled with the minimal size still has more than n in-shape cells, the algorithm then applies the operation *skeletonization* to this scaled image so as to obtain the shape's skeleton, which is a masked grid with fewer in-shape cells (Alg. 6, Line 20). It is possible that the number of in-shape cell on the skeleton is still more than n, if that happens, algorithm will abort as the input swarm size n is too small for displaying the desired shape $\mathcal{G}_{in}$ (Alg. 6, Line 25). See Alg. 6 for the detailed pseudo code for the *scaling* subroutine.

Remark: By combining the *image scaling* and the operation *skeletonization*, one desirable feature that *scaling* subroutine offers is that: when swarm size is large enough, the generated masked grids will preserve the input shape's details; on the other hand, when the swarm size is small, the algorithm will prioritize the shape's main structure so as to make the generated configuration be "conceptually similar" to the input shape.

6.3.2. Interpolation

The high-level idea behind the *interpolation* subroutine can be described as follows: Say we have a $l \times l$ binary image $\mathcal{G}_o^l$ and a $h \times h$ binary image $\mathcal{G}_o^h$ with a and b amount of in-shape

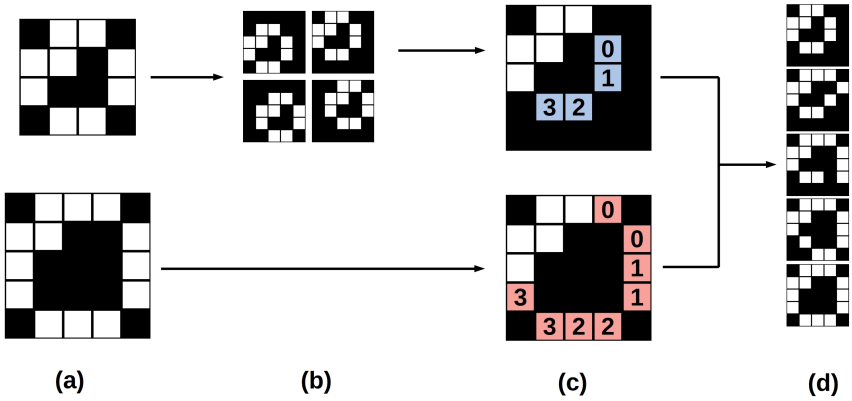

Figure 6.3. The graphical illustration of the *interpolation* subroutine. From left to right: (a) Two reference grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with 9 and 13 in-shape cells, respectively; (b) The $\mathcal{G}_o^l$ is aligned to $\mathcal{G}_o^h$ (Alg. 7, Line 1). There are 4 possible locations to place $\mathcal{G}_o^l$ on a 5×5 gird, and the algorithm chooses the one in the right-up corner because the *difference score* between this grid and $\mathcal{G}_o^h$ is the lowest; (c) The algorithm calculates the set $\mathcal{D}_{l-h}$, which is the set of cells filled with blue color, and the set $\mathcal{D}_{h-l}$, which is the set of cells filled with red color, and then split these two sets into two sets of 4 subsets $\{d_{l-h}^0, \ldots, d_{l-h}^3\}$ and $\{d_{h-l}^0, \ldots, d_{h-l}^3\}$, the number on each cells indicates the subset that it belongs to (Alg. 7, Line 2-24); (d) 5 configurations generated using $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with different input swarm size n s. From top to bottom: the configuration generated for the swarm with a size of $9, 10, 11, 12, 13$, respectively (Alg. 7, Line 25-27).

cells, respectively. Assume $h > l$ and $b > a$, we want to generate a sequence of $h \times h$ binary images $\mathcal{G}_a, \ldots, \mathcal{G}_b$, in which each generated image $\mathcal{G}_i$ has exactly i amount of in-shape cells. To do so, the algorithm will first place the input grid $\mathcal{G}_o^l$ on a empty $h \times h$ grid at a location such that the overlapping between the newly formed binary image and the $\mathcal{G}_o^h$ is maximized. Then, the algorithm will calculate the difference between the newly formed $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, which can be characterized by the cells that are with different labels on those two grids. The sequence of the binary images $\mathcal{G}_a, \ldots, \mathcal{G}_b$ can be constructed by gradually toggling the labels of those difference cells on the grid $\mathcal{G}_o^l$. To be more specific, for the aligned $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, their difference can be characterized by two sets: $\mathcal{D}_{l-h} = \mathcal{S}(\mathcal{G}_o^l) - \mathcal{S}(\mathcal{G}_o^h)$, which is the set of cells that are in the shape on $\mathcal{G}_o^l$ but off the shape on $\mathcal{G}_o^h$, and $\mathcal{D}_{h-l} = \mathcal{S}(\mathcal{G}_o^h) - \mathcal{S}(\mathcal{G}_o^l)$, which

is the set of cells that are in the shape on $\mathcal{G}_o^h$ but off the shape on $\mathcal{G}_o^l$. (Alg. 7, Line 3-4). Next, let $k = |\mathcal{S}(\mathcal{G}_o^h)| - |\mathcal{S}(\mathcal{G}_o^l)|$ be the difference between the numbers of in-shape cells on $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ (Alg. 7, Line 2), the algorithm splits those two sets into two groups of k disjointed subsets $d_{l-h}^0, \ldots, d_{l-h}^{k-1}$ and $d_{h-l}^0, \ldots, d_{h-l}^{k-1}$ such that: $\bigcup_{i=0}^{k-1} d_{l-h}^i = \mathcal{D}_{l-h}$ and $\bigcup_{i=0}^{k-1} d_{h-l}^i = \mathcal{D}_{h-l}$. In addition, for a pair of subsets d_{h-l}^i and d_{l-h}^i, the algorithm enforces their sizes to be such that: $|d_{h-l}^i| = 1 + |d_{l-h}^i|$ (Alg. 7, Line 5-24). With this constraint on each subset's cardinality, given an swarm size n, its corresponding configuration can be constructed as follows: first, set the labels the cells $\bigcup_{i=0}^{n-|\mathcal{S}(\mathcal{G}_o^l)|} d_{l-h}^i$ to 0 on $\mathcal{G}_o^l$, then, set the labels of the cells $\bigcup_{i=0}^{n-|\mathcal{S}(\mathcal{G}_o^l)|} d_{h-l}^i$ to 1 on $\mathcal{G}_o^l$ (Alg. 7, Line 25-27). It is straight forward to examine that the masked grid constructed via the procedure above will have exactly n in-shape cells. The pseudo code for the *interpolation* subroutine is shown in Alg. 7. A graphical illustration of the *interpolation* subroutine is shown in Fig. 6.3.

Assume that the reference grid $\mathcal{G}_o^l$ has a size of $l \times l$ and $\mathcal{G}_o^h$ has a size of $h \times h$, where $h \geq l$ according to Alg. 6. The *interpolation* subroutine will first draw the $\mathcal{G}_o^l$ on an empty $h \times h$ grid (Alg. 7, Line 1). Before drawing $\mathcal{G}_o^l$ on this empty $h \times h$ grid, the algorithm needs to determine the location to place the $\mathcal{G}_o^l$ on this $h \times h$ grid, as there might be multiple choices since $h \geq l$. To do so, I first define a metric called *difference score* as follows:

Definition 6.1. *Given two masked grids $\mathcal{A}$ and $\mathcal{B}$, the difference score between $\mathcal{A}$ and $\mathcal{B}$ is given by $|\mathcal{S}(A) - \mathcal{S}(\mathcal{B})| + |\mathcal{S}(B) - \mathcal{S}(\mathcal{A})|$, i.e, the number of cells that are in shape on $\mathcal{A}$ but not in shape on $\mathcal{B}$ plus the number of cells that are on in shape on $\mathcal{B}$ but not in shape on $\mathcal{A}$.*

With this metric, the location to place the grid $\mathcal{G}_o^l$ on the new $h \times h$ grid can be determined as follows: the algorithm will exhaustively search all possible translations, and choose

Algorithm 7: *interpolation* subroutine

Input: Reference grids $\mathcal{G}_o^l, \mathcal{G}_o^h$, swarm size n
Output: The generated masked grid $\mathcal{G}_o$

1 $\mathcal{G}_o^l \leftarrow$ align $\mathcal{G}_o^l$ to $\mathcal{G}_o^h$
2 $k \leftarrow |\mathcal{S}(\mathcal{G}_o^h)| - |\mathcal{S}(\mathcal{G}_o^l)|$ // calculate the difference of in-shape numbers on two reference grid
3 $\mathcal{D}_{l-h} \leftarrow \mathcal{S}(\mathcal{G}_o^l) - \mathcal{S}(\mathcal{G}_o^h)$ // the set of cells that are in the shape on $\mathcal{G}_o^l$ but off the shape on $\mathcal{G}_o^h$
4 $\mathcal{D}_{h-l} \leftarrow \mathcal{S}(\mathcal{G}_o^h) - \mathcal{S}(\mathcal{G}_o^l)$ // the set of cells that are in the shape on $\mathcal{G}_o^h$ but off the shape on $\mathcal{G}_o^l$
5 Initialize $sz^0, \ldots, sz^{k-1}$ to be all 0s // the size of each subset used in interpolation
6 **for** $i \leftarrow 0, \ldots, k-1$ **do** // determine each subset's size
7 **if** $i \leq |\mathcal{D}_{l-h}| \ mod \ k$ **then**
8 $sz^i \leftarrow \lfloor \frac{|\mathcal{D}_{l-h}|}{k} \rfloor + 1$
9 **else**
10 $sz^i \leftarrow \lfloor \frac{|\mathcal{D}_{l-h}|}{k} \rfloor$
11 $dt(\mathcal{G}_o^l) \leftarrow$ apply *distance transform* to $\mathcal{G}_o^l$ // calculate each in-shape cell's distance to the boundary
12 *buf_sub*$\leftarrow$use each cell's value in $dt(\mathcal{G}_o^l)$ as the key to sort $\mathcal{S}(\mathcal{G}_o^l)$ in ascending order
13 Initialize $d_{l-h}^0, \ldots, d_{l-h}^{k-1}$ to be all $\varnothing$s // the subsets of cells to be turned off on $\mathcal{G}_o^l$
14 **for** $i \leftarrow 0, \ldots, k-1$ **do** // assemble the subsets of cells to be turned off on $\mathcal{G}_o^l$
15 $d_{l-h}^i \leftarrow d_{l-h}^i \cup \{$the first sz^i cells in *buf_sub*$\}$
16 *buf_sub*$\leftarrow$*buf_sub* - d_{l-h}^i
17 Initialize $d_{h-l}^0, \ldots, d_{h-l}^{k-1}$ to be all $\varnothing$s // the subsets of cells to be turned on on $\mathcal{G}_o^l$
18 *buf_add*$\leftarrow \mathcal{D}_{h-l}$
19 **for** $i \leftarrow 0, \ldots, k-1$ **do**
20 $d_{h-l}^i \leftarrow d_{h-l}^i \cup \{sz^i$ amount of cells in *buf_add* that are closest to the cells in $d_{l-h}^i\}$
21 *buf_add*$\leftarrow$*buf_add* - d_{h-l}^i
22 **for** $i \leftarrow 0, \ldots, k-1$ **do**
23 $d_{h-l}^i \leftarrow d_{h-l}^i \cup \{$the first cell in *buf_add*$\}$
24 *buf_add*$\leftarrow$*buf_add* - d_{h-l}^i
25 $\mathcal{G}_o \leftarrow$a copy of $\mathcal{G}_o^l$ // make a copy of $\mathcal{G}_o^l$ and toggle the labels of cells on it so as to construct the output
26 set labels of all the cells in $\bigcup_{i=0}^{n-|\mathcal{G}_o^l|} d_{l-h}^i$ to 0 on $\mathcal{G}_o$
27 set labels of all the cells in $\bigcup_{i=0}^{n-|\mathcal{G}_o^l|} d_{h-l}^i$ to 1 on $\mathcal{G}_o$
28 **return** $\mathcal{G}_o$

the translation that gives the minimal *difference score* between the grid $\mathcal{G}_o^h$ and the newly generated $\mathcal{G}_o^l$ (Alg. 7, Line 1).

After aligning $\mathcal{G}_o^l$ to $\mathcal{G}_o^h$, the algorithm will first calculate the difference between $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ (Alg. 7, Line 2-4), then pack the set $\mathcal{D}_{l-h}$ into k subsets (Alg. 7, Line 5-16). sz^i denotes the size of the subset d_{l-h}^i of $\mathcal{D}_{l-h}^i$, the algorithm will first calculate the size of each subset d_{l-h}^i (Alg. 7, Line 5-10). After determining the size of each subset d_{l-h}^i, the algorithm then start

to determine the contents of each subset d^i_{l-h}. When removing the cells from the shape, it is desired to remove the cells following the order such that: the cells closer to shape's boundary will be removed first. This order helps to avoid generating "holes" in the remaining shape. To address this design consideration, for all the cells in $\mathcal{D}_{l-h}$, the algorithm will first calculate each cell's Manhattan distance to the boundary using the operation *distance transform* [85] (Alg. 7, Line 11), and then use each cell's distance to boundary as the key to sort all the cells in $\mathcal{D}_{l-h}$ in ascending order (Alg. 7, Line 12). The sorted $\mathcal{D}_{l-h}$ is stored in the variable *buf_sub*. Next, the algorithm will start to assemble each subset d^i_{l-h} according to the determined pack size sz^i (Alg. 7, Line 14-16): for each subset d^i_{l-h}, the algorithm pack the first sz^i cells in *buf_sub* into it (Alg. 7, Line 15), and then delete those sz^i cells from the *buf_sub* right after so as to avoid the case where the same cell shows up in two difference subsets (Alg. 7, Line 16).

Next, the algorithm starts to assemble the subsets $d^0_{h-l}, \ldots, d^{k-1}_{h-l}$ (Alg. 7, Line 18 - 24). It will first make a copy of $\mathcal{D}_{h-l}$ and store it to variable *buf_add* (Alg. 7, Line 18). Note that when removing a subset d^i_{l-h} of cells from the shape, the algorithm will damage the structure of the shape. To reduce the effect of the removal of d^i_{l-h}, when packing each subset d^i_{h-l}, which are the sets to be added to the remaining shape, it is desired that the cells in d^i_{h-l} are as "close" to cells in d^i_{l-h} as possible (Alg. 7, Line 20). To be more specific, given a subset d^i_{l-h} and the set *buf_add*, the algorithm treats each cell in d^i_{l-h} as a "job" and each cell currently in *buf_add* as a "worker", and the cost for each "worker" doing each "job" is given by the Manhattan distance between those two cells. The algorithm uses the Hungarian algorithm [86] to assign exactly 1 "worker" to each "job" in each subset d^i_{h-l} such that the total cost is minimized, and these assigned "workers" will be packed into d^i_{h-l} (Alg. 7, Line 20). Recall that as stated in the overall description of the *interpolation* subroutine,

there is a constraint on the each pair of subsets' cardinalities that: $|d^i_{h-l}| = 1 + |d^i_{l-h}|$. On the other hand, it is straight forward to examine that after Alg. 7, Line 19-21, each pair of subsets d^i_{h-l} and d^i_{l-h} have the same cardinality. In order to satisfy the cardinality constraint above, in Alg. 7, Line 22-24, the algorithm will add one extra cell to each of those subsets $d^0_{h-l}, \ldots, d^{k-1}_{h-l}$.

So far, both the subsets $d^0_{l-h}, \ldots, d^{k-1}_{l-h}$ and the subsets $d^0_{h-l}, \ldots, d^{k-1}_{h-l}$ have been assembled, the algorithm then constructs the desired masked grid $\mathcal{G}_o$ following the procedure described at the beginning of the Section 6.3.2 (Alg. 7, Line 25-27). See Fig. 6.3 for a graphical illustration of the *interpolation* subroutine.

In *interpolation* subroutine, one key element is the way to determine sz^i, which is the size of each subset d^i_{h-l} (Alg. 7, Line 6-10). Given two masked grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, there might be multiple feasible combinations of each subset's size. One can consider a case where $|\mathcal{D}_{l-h}| = 2, |\mathcal{D}_{h-l}| = 4, k = 2$, one way to split $\mathcal{D}_{l-h}$ and $\mathcal{D}_{h-l}$ is: $|d^0_{l-h}| = 1, |d^1_{l-h}| = 1, |d^0_{h-l}| = 2, |d^1_{h-l}| = 2$, and the other way is: $|d^0_{l-h}| = 2, |d^1_{l-h}| = 0, |d^0_{h-l}| = 3, |d^1_{h-l}| = 1$. According to the overall pipeline of the algorithm (Alg. 5), for the same pair of reference grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, they could be used to construct $|\mathcal{S}(\mathcal{G}^h_o)| - |\mathcal{S}(\mathcal{G}^l_o)| + 1$ different configurations with $n = |\mathcal{S}(\mathcal{G}^l_o)|, |\mathcal{S}(\mathcal{G}^l_o)| + 1, \ldots, |\mathcal{S}(\mathcal{G}^h_o)|$ amount of in-shape cells, respectively. It is trivial to see that different ways to determine each subset's size will result in different *difference score*s among these generated masked grids. Recall that as stated in the Section 7.2.2, to allow the swarm to quickly adapt to the swarm size change, one of my design considerations is: the configurations generated for different swarm sizes should be similar to each other. Responding to this design consideration, given a pair of reference grids $\mathcal{G}^l_o$ and $\mathcal{G}^h_o$, for the configurations generated from them, a desirable way to determine each subset's size should

make the *difference score* between any pair of masked grids with adjacent number of in-shape cells as small as possible. In the following, I will show that: given two reference grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, the way that the algorithm determines each subset's size (Alg. 7, Line 6-10) is actually the optimal way that can minimize the maximal *difference score* between any pair of generated masked grids whose difference of in-shape cell number is one.

Problem 6.1. *(Fair packing) Given a set $\mathcal{A}$ and an integer k, $\mathcal{P}_k(\mathcal{A})$ denotes a k-partition of the set $\mathcal{A}$, which is a set of k subsets $\{a_0, \ldots, a_{k-1}\}$ such that: (i) $\bigcup_{i=0}^{k-1} a_i = \mathcal{A}$, and (ii) $\forall i \neq j, a_i \cap a_j = \varnothing$. The task is to find a $\mathcal{P}_k(\mathcal{A})$ that minimizes the maximal cardinality among all those k subsets $a_i \in \mathcal{P}_k(\mathcal{A})$. That is, given a set $\mathcal{A}$ and an integer k, find a partition $\mathcal{P}_k^*(\mathcal{A})$ such that:*

$$\mathcal{P}_k^*(\mathcal{A}) = argmin \max_{a_i \in \mathcal{P}_k^*(\mathcal{A})} |a_i|$$

To interpret Problem 1, one can consider a simple instance of it: Say we have 10 balls and we are tasked to put those 10 balls into 3 bins. The task is to find a way to assign those 10 balls to those 3 bins such that the maximal number of balls among all 3 bins is minimized. Next, in the Lemma 6.1, I will show an sufficient condition for a solution to be optimal to Problem 1.

Lemma 6.1. *Given a set $\mathcal{A}$ and an integer k, let $\mathcal{P}_k^{'}(\mathcal{A})$ be a k-partition of the set $\mathcal{A}$, if the $\mathcal{P}_k^{'}(\mathcal{A})$ is made such that:*

$$\max_{a_i \in \mathcal{P}_k^{'}(\mathcal{A})} |a_i| - \min_{a_i \in \mathcal{P}_k^{'}(\mathcal{A})} |a_i| \leq 1 \tag{6.1}$$

Then $\mathcal{P}_k^{'}(\mathcal{A})$ is an optimal solution to Problem 1.

PROOF. See Appendix 3 □

Theorem 6.1. *(Smooth transition) Given two reference masked grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$ with a and b amount of in-shape cells, respectively, let $\mathcal{G}_a, \mathcal{G}_{a+1}, \ldots, \mathcal{G}_b$ be the masked grids generated for the swarms with a size of $a, a+1, \ldots, b$. Among all the ways to determine the size of each subset used in interpolation subroutine, Alg. 7 Line 6-10 is the optimal way for minimizing the following objective:*

$$\max_{a \leq i \leq b-1} |\mathcal{S}(\mathcal{G}_i) - \mathcal{S}(\mathcal{G}_{i+1})| + |\mathcal{S}(\mathcal{G}_{i+1}) - \mathcal{S}(\mathcal{G}_i)| \tag{6.2}$$

PROOF. See Appendix 4 □

6.4. Performance Evaluation

In this section, I empirically study the performance of the presented algorithm. Given a goal shape, in Section 6.4.1, I first study the quality of configurations generated for the swarms with different sizes. Then, in Section 6.4.2, a swarm of simulated robots used the shape formation algorithm proposed in [44] to form those generated configurations. In addition, beyond simulation, in Section 6.4.3, the generated configurations were formed by a swarm of physical robots. The results from both the simulations and physical experiments show that: the proposed algorithm can indeed make the swarm adapt to the swarm size change quickly and robustly.

In the experiments, I use four complex shapes as the goal shapes: the "N", the "star", the "wrench", and the "circle". These four goal shapes are shown in the Fig. 6.4.

6.4.1. Experiments on Generated Configurations

First, given each goal shape, I use the proposed algorithm to generate the goal configurations with in-shape cell number ranging from around 20 to 1024. Recall that as stated in the Section 7.2.2, we have two design considerations: the similarity between each generated configuration and the goal shape, and the similarity between the configurations generated for difference swarm sizes. The videos of the configurations generated for different swarm sizes can be found in [76]. In addition, I introduce two metrics to qualitatively evaluate the similarity between each pair of the binary images whose difference of in-shape cell number is one: the *normalized difference score* (NDS), and the *normalized inter-shape distance* (NISD). The NDS is the ratio between those two configurations' *difference score* and the sum of two configurations' in-shape cell numbers. The NISD is defined as follows: given two masked grid $\mathcal{A}$ and $\mathcal{B}$ where $|\mathcal{S}(\mathcal{A})| \leq |\mathcal{S}(\mathcal{B})|$, we assign cells in $\mathcal{S}(\mathcal{B})$ to the cells in $\mathcal{S}(\mathcal{A})$ in a way such that: (i) for each in-shape cell on $\mathcal{A}$, we assign exactly one in-shape cell on $\mathcal{B}$ to it, in addition, (ii) each in-shape cell on $\mathcal{B}$ can be assigned to no more than one cell on $\mathcal{A}$. The cost of each pair of in-shape cell on $\mathcal{A}$ and its assigned in-shape cell from $\mathcal{B}$ is given by the Manhattan distance between them in cells. The NISD is the ratio between the minimal total cost that any feasible assignment can achieve and sum of two configurations' in-shape cell numbers. Intuitively, the NDS shows the "mismatch" between two configurations, and NIDS

Figure 6.4. Goal shapes used in the experiments. From left to right: the shape "letter N", the shape "star", the shape "wrench", and the shape "circle".

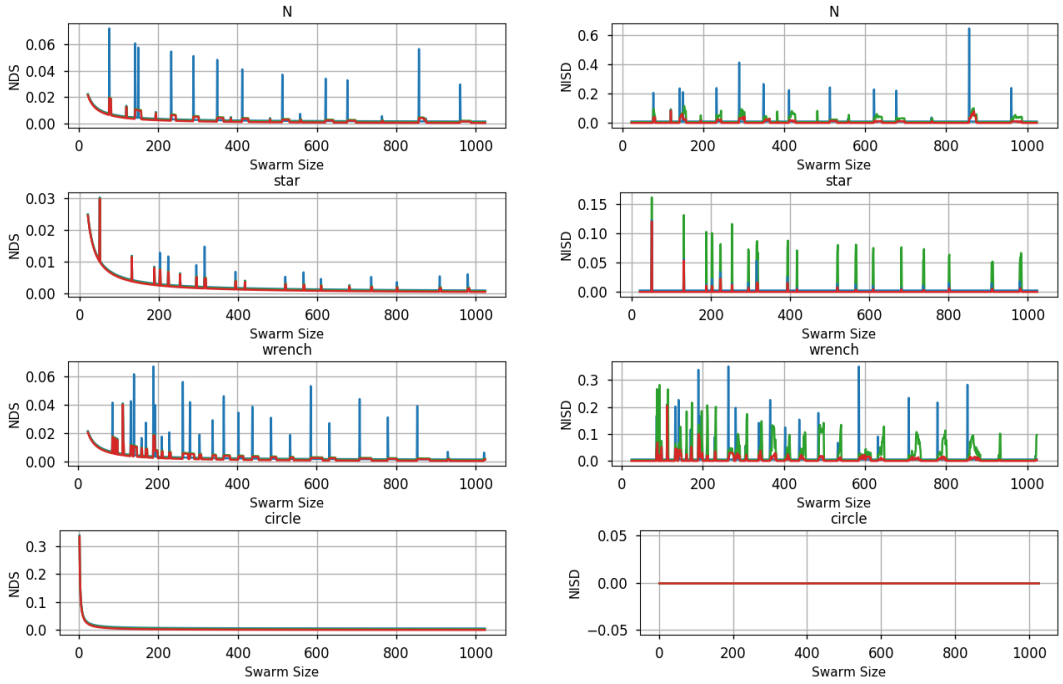

Figure 6.5. Comparison between the proposed algorithm and two baselines. For each swarm size n, its corresponding data points on the plots are the NDS and NISD between the generated masked grids with n and $n+1$ in-shape cells, respectively. The red plots are the results for the proposed algorithm, the blue plots are the results for the baseline 1, and the green plots, which overlap with the red plots in all NDS plots, are the results for the baseline 2.

essentially characterizes the minimal average distance traveled by the swarm to transform from one configuration to the other. The plots showing these two metrics over difference in-shape cell numbers for each goal shape are shown in Fig. 6.5. In addition, I also compare the proposed algorithm with two baselines. Both of those two baselines use the same pipeline as the presented algorithm does. The difference between the presented algorithm and the baseline 1 is that: when executing the *interpolation* subroutine, instead of using Alg. 7 Line 6-10 to determine each subset's size, the baseline 1 will aggressively set sz^0 to be $|\mathcal{D}_{l-h}|$ and set $sz^i \dots sz^{k-1}$ to be 0. The difference between the presented algorithm and the baseline 2 is that: when executing the *interpolation* subroutine, instead of using Alg. 7 Line 11-24 to

assemble each subset, the baseline 2 will naively fit the cells into each subset by the lexical order of each cell's coordinate. In the *interpolation* subroutine, there are two subproblems to be solved: how to determine each subset's size, and how to determine the content of each subset. These two baselines are essentially two naive solutions to those two subproblems.

As expected, in these plots, we can see that the presented algorithm outperforms two baselines with respect to both NDS and NISD for all four goal shapes, confirming that the presented way to determine the size and content of each subset in *interpolation* subroutine (Alg. 7 Line 6-10, Line 11-24) can indeed make the transition between goal configurations generated for different swarm sizes more "smooth". Note that the presented algorithm and baseline 2 uses the same way to determine each subset's size in *interpolation* subroutine, as a result, in all NDS plots, the presented algorithm (black) overlaps with baseline 2 (green). The other counter-intuitive observation here is that: the NISD for the shape "circle" is 0 for all the swarm sizes, this is because: using the pipeline presented in the chapter, for any swarm size n, the "circle" generated for the swarm size n will always be "inside" the "circle" generated for the swarm size $n+1$, therefore, the "circle's" NISD is by definition 0 for all the swarm sizes.

6.4.2. Experiments on Simulated Robotic Swarm

In the simulation, a swarm of up to 100 simulated *Coachbot* robots were tasked to use the shape formation algorithm proposed in [44] to form the configurations generated from the presented algorithm. In the simulation, the communication rate is $20hz$, the maximal speed of robot's wheel is $0.1m/s$, and each edge on the grid has a length of $0.3m$. The demonstration videos of the simulated swarms with different sizes forming the goal shapes can be found in [76].

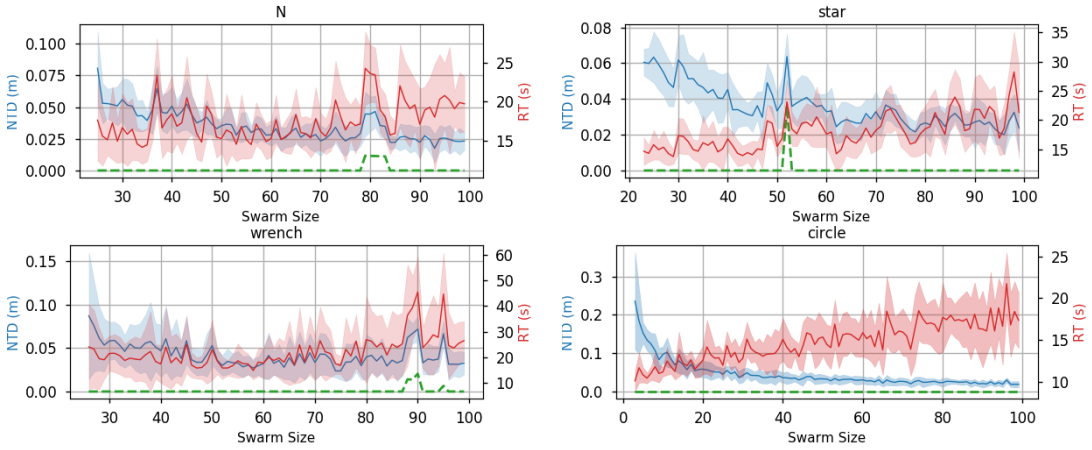

Figure 6.6. The results from the addition experiment. Each solid line is the average result from 50 trials, and the colored shade areas show the confidence intervals for NTD and RT over swarm size at a confidence level of two σ. Each green dotted line is the shape's NISD obtained from the previous section.

In the first experiment, I study the how the addition of the robots will affect the swarm's behavior. In this experiment, the swarm size is initialized to be around 20. Every time when robots currently in the swarm complete forming the shape, I add one more robot in a random location near the swarm and broadcast the new swarm size to the swarm. The robots will update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process will be repeated until the swarm size gets to 100. For each goal shape, I repeat this experiment 50 times, and in each trial, I study two metrics: the response time (RT), which is time between the swarm size change and the swarm forming the shape at the new scale, and the normalized travel distance (NTD), which the average distance traveled by the swarm to form the shape at the new scale. The results from 50 trials are shown in Fig. 6.6.

In addition, besides the addition of the robots, I am also interested in effect of the removal of robots on the swarm's behavior. In the second experiment, the swarm size is initialized to 100, and similar to the first experiment, every time when the current swarm complete

forming the shape, I randomly choose one robot currently in the swarm, remove it from the swarm, and broadcast the new swarm size to the robots. The robots will update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process will be repeated until the swarm size gets to minimal swarm size required to display the shape. For each goal shape, I repeat this experiment 50 times, and in each trial, the metric RT and NTS are investigated, see Fig. 6.7 for the results from all 50 trials. As we can see in the plots, in both addition and subtraction experiments, every time when the swarm size change occurs, the swarm is able to adapt quickly, within 40 s to be more specific. Moreover, one can observe that for each shape, at some certain swarm sizes, the RT and NTD change sharply in both subtraction and addition experiments. For example, for the shape "wrench", the RT and NTD spike at the swarm size 52. To investigate the cause of these spikes, I compare the NISD obtained from previous section (green dotted line) with the NTD and RT obtained from the simulation. Unsurprisingly, the results show that the swarm sizes where the NTD and RT spike are consistent with the swarm sizes where the shape's NISD spikes, in the other words, the swarm sizes where the RT and NTD spike are the swarm sizes where the generated goal configurations change greatly.

Recall that the swarm size estimation algorithm presented in chapter 5 is a probabilistic method, making it possible for each robot's swarm size estimate to be slightly different from the actual swarm size. In addition, due to the asynchrony of each robot's local clock, it is also possible that each robot's swarm size estimate is different from each other. In this test, I study how the each robot's swarm size estimate error, and the inconsistency between each robot's swarm size estimate, will affect the algorithm's performance. In this test, a swarm of 200 simulated robots are tasked to use the algorithm presented in [44] to form a shape "N". In addition, every 1.5 seconds, each robot will update its swarm

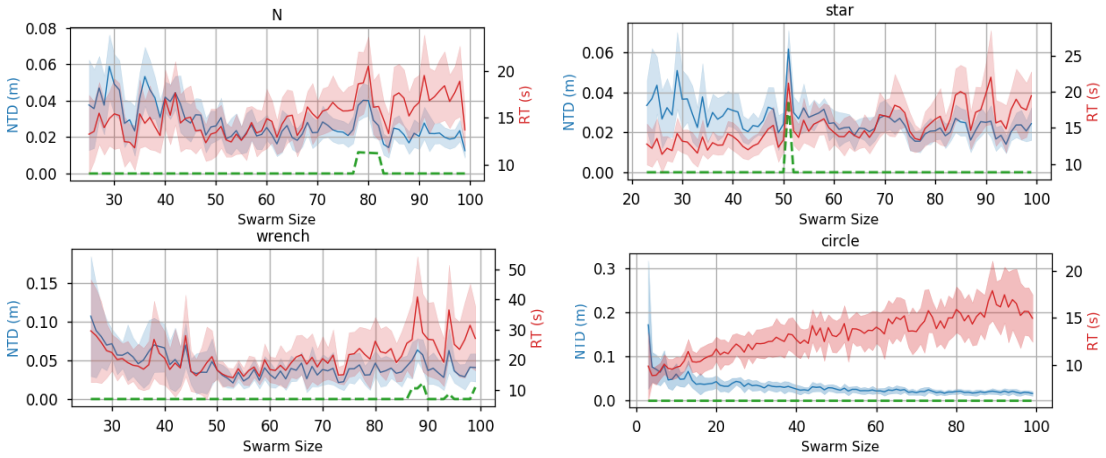

Figure 6.7. The results from the substraction experiment. Each solid line is the average result from 50 trials, and the colored shade areas show the confidence intervals for NTD and RT over swarm size at a confidence level of two σ. Each green dotted line is the shape's NISD obtained from the previous section.

size estimate with a probability of 10%. This probabilistic periodic swarm size estimate update is for simulating the asynchrony of each robot's local clock. When updating its swarm size estimate, each agent will sample from a Gaussian distribution $\lfloor \mathcal{N}(200, \sigma_s^2) \rfloor$, and update its goal configuration accordingly. This noise injection is for simulating each robot's swarm size estimate error. Three noise profiles are used in this test: $\sigma_s = 2$, $\sigma_s = 10$, and $\sigma_s = 20$. Furthermore, each time when a robot updates its goal configuration, it will check if the current goal point is in the new goal configuration, if not, it will set its goal point to the closest one in the new goal point set. For each noise profile, 10 trials were run. I study how the different noise profile will affect the quality of formed configuration, which is characterized by the NDS between the goal configuration and the swarm's configuration at each time step, as well as the robot's effort to form the shape, which is characterized by the swarm's NTD over time. The results are shown in Fig. 6.8.

As we can see in the plots, instead of converging to a stationary point, the swarm's NTD keeps increasing over time, in the other words, the swarm's configuration keeps changing

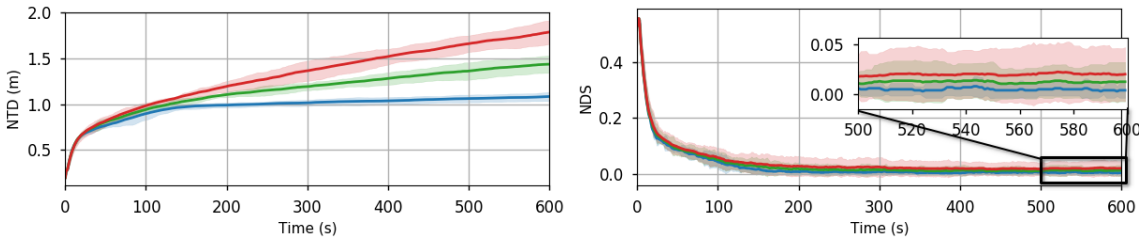

Figure 6.8. Each solid line is the average result from 10 trials, and the colored shade areas show the confidence intervals for NTD and NDS over swarm size at a confidence level of two σ. The color indicates the noise profile used in the experiment: blue – $\sigma_c = 2$, green – $\sigma_c = 10$, red – $\sigma_c = 20$

over time. This is because: in this test, every time when a robot updates its swarm size estimate, it is always possible for this robot to change its goal point, as a result, at any time of the experiment, it is always possible for some robots to move, increasing the swarm's NTD. As expected, in the plots, we can see that the higher swarm size estimate error will result in a larger NTD, in the other words, the higher the swarm size estimate error is, the less stable the swarm's configuration will be. In addition, the swarm size estimate error will also affect the quality of the formed shape: the higher the swarm size estimate error is, the less consistent the swarm's configuration will be to the desired configuration.

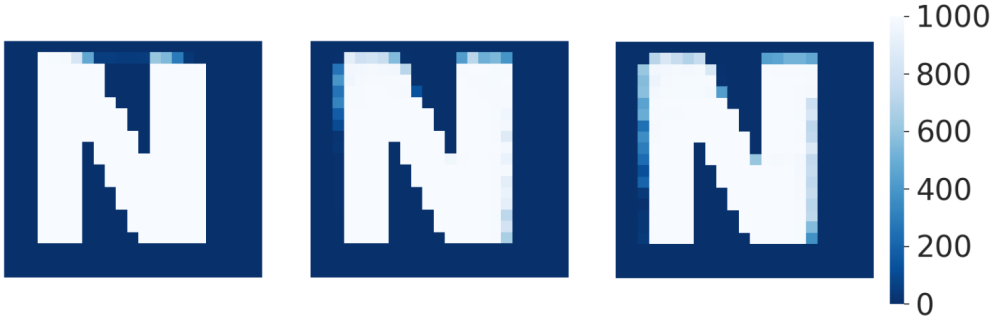

Figure 6.9. The change in swarm's configuration over time. The intensity of each cell indicates the amount of time that cell is occupied by a robot. From left to right: $\sigma_c = 2$, $\sigma_c = 10$, and $\sigma_c = 20$

As we discussed before, in this test, the swarm's configuration will keep changing over time. A graphical illustration of the change in the swarm's configuration over time can be found in Fig. 6.9. In this figure, each heatmap shows the swarm's configuration during the last 1000 seconds of a 3000-second-long trial, the intensity of each cell indicates the amount of time that cell is occupied by a robot. Unsurprisingly, a higher swarm size estimate error will make the swarm's configuration less stable.

6.4.3. Experiments on Physical Robotic Swarm

Beyond the simulations, I also experiment on a swarm of up to 100 real *Coachbot* robots. In the experiment, the robots are tasked to form the shape "N". The swarm size starts as 100, every time when the current swarm complete forming the shape, I remove a batch of robots from the swarm and then broadcast the new swarm size to the robots. The robots use the presented algorithm to update their goal configuration according to the new swarm size, and then start to form the new goal configuration. This process is repeated until the swarm size gets to 23. See Fig. 6.1 for the still images from the experiment, and the video for this experiment can be found in [76]. As we can see in the video, when the swarm size changes, the robots adapt quickly and robustly.

CHAPTER 7

COOPERATIVE LOCALIZATION

This chapter presents a decentralized algorithm that allows a swarm of identically programmed agents to cooperatively estimate their global poses using the local range and bearing measurements. The design of the presented algorithm explicitly considers the asynchrony of each agent's local clock, moreover, the theoretical analysis about the effect of each agent's sensing noise and communication loss is given. The presented algorithm is validated via experiments running on a swarm of up to 256 simulated robots, and a swarm of 100 physical robots. The results from the experiments show that the presented algorithm allows each agent to estimate its global pose quickly and robustly. The algorithm presented in this chapter has been published as [43], a summary video of the presented algorithm can be found in [76].

7.1. Background

Localization plays an important role in swarm systems. In many collective tasks, such as shape formation [44, 49, 50], shepherding [87], and more [88], it is valuable to have each agent knowing its pose in a global coordinate system.

It has been shown that when the swarm works in a well-controlled environment, agents can obtain their poses through an external infrastructure such as HTC Vive Virtual Reality system [44] or GPS system [89]. However, when the swarm is deployed in the environments where the external infrastructures are not available, underground or underwater for example,

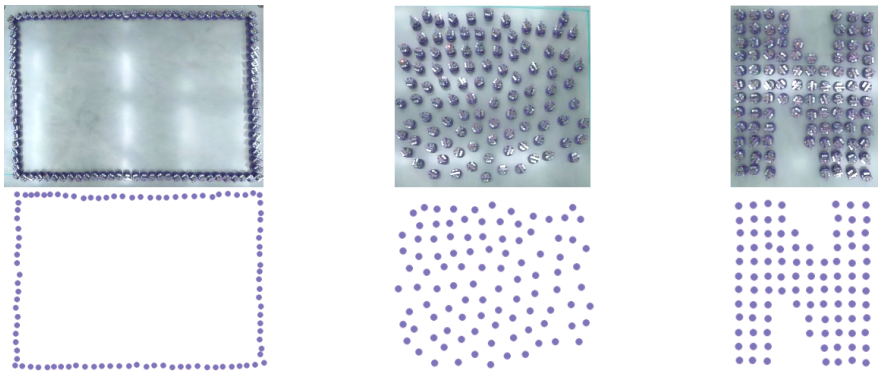

Figure 7.1. (Top) 100 Coachbot V2.0 robots are placed in three patterns; (Bottom) The robots' position estimates obtained via the proposed algorithm.

those infrastructure-based solutions will become impractical. A more flexible alternative to infrastructure-based methods is the local-measurement-based methods. Here, it is assumed that each agent is able to communicate with nearby agents, in addition, each agent is also able to measure its distance [31, 90–94], bearing [95, 96], or both [97–99], relative to its neighbors. The agents will calculate their global poses using the local communication and relative measurements. The type of methods only require the use of peer-to-peer information therefore permit the operations in environments without external infrastructure.

The previous local-measurement-based methods can be categorized into centralized methods [100, 101] and decentralized methods [31, 91, 92, 94, 96, 102]. In centralized methods [100, 101], a central controller will collect all the peer-to-peer measurements from the entire swarm and use these collected measurements to calculate each agent's pose. The centralized methods are generally more computationally efficient, and can be more robust to the sensing noise. However, it is not easy for them to scale to the swarms with large sizes because they suffer from the *single point of failure* problem.

On the contrary, the decentralized methods are inherently more scalable and more robust to the failures. The decentralized methods can be categorized into two types: progressive

methods [31, 91, 92, 96, 102] and concurrent methods [94]. In progressive methods, each agent has two possible states: localized and unlocalized. The unlocalized agents will localize themselves using the inter-agent measurements and the pose estimates from their already localized neighbors, then, these unlocalized agents will become localized agents and their pose estimates will later be used by their unlocalized neighbors. It was shown in [31, 91, 102] that with certain amount of pre-localized anchoring agents, it is possible to accurately localize a densely placed swarm using the local information only. However, each agent's position error will increase over the communication hops away from the anchoring agents. In addition, this type of methods require an additional phase to set certain agents as anchoring agents. The beacon-free solutions are proposed in [92, 96]. In [92], agents use inter-agent distance information to organize *robust quads* with their nearby agents. The adjacent *robust quads* can recover their relative positions using the information from the set of agents in common between two quads. However, it has not been shown that this method can estimate agent's orientation. The work presented in [96] allows agents to estimate their orientation using the inter-agent bearing information, in addition, this methods is also able to localize only a subset of the agents in the swarm. However, the coordinate system established by this method is scale-free, requiring an additional step to recover the coordinate system's scale [103].

Different from the progressive methods, in concurrent methods [94, 104, 105], agents do not explicitly hold a Boolean state of being localized or unlocalized. Instead, all the agents will constantly and cooperatively refine their pose estimates using their local measurements such as distance [94], bearing [104], or both [105]. The concurrent methods are generally more robust to the sensing noise, in addition, these types of methods are more robust to the unexpected external disturbances such as the removal or the addition of the agents. One

drawback for this type of methods is that compared to the progressive methods, the concurrent methods' commutation cost is often more expensive, as each agent needs to frequently exchange its pose estimates with its neighbors. This chapter presents a fully decentralized concurrent localization algorithm for localizing a swarm of identically programmed agents. As stated in chapter 2, it is assumed that each agent can transmit messages to nearby agents, in addition, each time when an agent receives a message from a neighbor, it can sense the transmitter's bearing and distance. The agents will estimate their global poses by cooperatively minimizing the disagreement between the inter-agent measurements and their pose estimates. The novelty of the presented algorithm is that: the design of the algorithm explicitly considers the asynchrony of the agent's local clock, moreover, the algorithm does not require the agent to actively maintain the communication links between itself and its neighbors, which helps to avoid the unnecessary communication overhead and makes the presented algorithm more flexible. The theoretical analysis about how the communication loss and sensing noise will affect the swarm's behavior is given, in addition, the algorithm's performance is thoroughly investigated via the experiments running on a swarm of up to 256 simulated agents as well as a swarm of 100 physical robots. The results from the experiments show that the presented algorithm is able to localize the swarms with different sizes and configurations quickly and robustly.

7.2. Preliminaries

This section introduces the agent model used in the analysis of the algorithm, and formally states the decentralized localization problem.

7.2.1. Agent model

Given the robot capabilities proposed in chapter 2, in this chapter, the robot is modeled as follows: It is assumed that the agents are placed on a 2D plane. Each agent holds a local coordinate frame where the local coordinate frame's origin is fixed on the center of the agent and the x-axis' direction is aligned with the agent's heading. It is assumed each agent's clock has the same frequency but can be asynchronous in phase. In addition, it is assumed that each agent has a locally unique ID. Each agent is able to broadcast messages to all the physically nearby agents lying within its communication range R. Moreover, when an agent a_i receives a message from a neighbor a_j, the agent a_i is able to sense the transmitter a_j's relative bearing angle $\mathcal{B}_{ij}$ and the distance d_{ij}.

Furthermore, in order to make the agent model used in this chapter more realistic, it is assumed that the inter-agent communication channel is lossy, that is, for an agent a_i, when a nearby neighbor a_j transmits a message, a_i will receive this message with a probability of λ, where $0 < \lambda < 1$. In addition, it is assumed that the agent's bearing and range sensor is noisy. That is: let $\mathcal{B}_{ij}$ and d_{ij} be agent a_j's actual bearing angle and distance in agent a_i's local coordinate frame, respectively. When agent a_i attempts to measure those two quantities, the actual measurements returned from the sensor will be two random variables $\hat{\mathcal{B}_{ij}} = \langle \mathcal{B}_{ij} + \epsilon_{\mathcal{B}} \rangle_{2\pi}$ and $\hat{d}_{ij} = d_{ij} + \epsilon_d$, where $\langle \cdot \rangle_{2\pi} := \cdot + 2\pi \lfloor \frac{\pi - \cdot}{2\pi} \rfloor$ denotes the 2π modulus operator [106], which is an operator that wraps an angle from the real space to the interval

$(-\pi, \pi]$, and $\epsilon_B \sim \mathcal{N}(0, \sigma_{\mathcal{B}}^2)$, $\epsilon_d \sim \mathcal{N}(0, \sigma_d^2)$ are two independent zero-mean Gaussian random variables. The distribution used to model the agent's bearing sensor noise is also known as *wrapped normal distribution* [107].

7.2.2. Problem statement

The task is to design an algorithm that enables each agent to use the local information to estimate its orientation and position in a global coordinate frame. In the presented algorithm, this task is achieved by enforcing agents to constantly refine their pose estimates to reduce the inconsistency between their pose estimates and the inter-agent relative measurements. To be more specific, let $\mathcal{A} = [a_1, a_2, ..., a_n]$ be a set of n agents, the undirected graph $\mathcal{G} = \{\mathcal{A}, \mathcal{E}\}$ is used to describe the network's communication topology, for a pair of agents a_i and a_j, $(i, j) \in \mathcal{E}$ iff they are located in each other's communication and sensing range R, and for each agent a_i, the set of all the agents located in its communication range is denoted as $N_i = \{a_j | a_j \in \mathcal{A}, (i, j) \in \mathcal{E}\}$. Each agent a_i uses the vector $[\theta_{ix}, \theta_{iy}]$ and vector $[x_i, y_i]$ to describe its orientation estimate and position estimate, respectively, where $\theta_i = \arctan 2(\theta_{iy}, \theta_{ix})$ indicates the agent's orientation estimate, and x_i, y_i are agent's estimate of its xy position. The agents will estimate their pose by cooperatively solving the following two problems:

Problem 7.1. *(Orientation Estimation): Let $\mathcal{W}_{ij}$ be the true orientation difference between two agent a_i and a_j, we define the swarm's orientation estimate error f_o as:*

$$\frac{1}{4} \sum_{a_i \in \mathcal{A}} \sum_{a_j \in N_i} \frac{1}{|N_i|} \frac{1}{|N_j|} \left\| \begin{bmatrix} \cos(\mathcal{W}_{ij}), -\sin(\mathcal{W}_{ij}) \\ \sin(\mathcal{W}_{ij}), \quad \cos(\mathcal{W}_{ij}) \end{bmatrix} \begin{bmatrix} \theta_{ix} \\ \theta_{iy} \end{bmatrix} - \begin{bmatrix} \theta_{jx} \\ \theta_{jy} \end{bmatrix} \right\|_2^2$$

The task is to find each agent a_i's orientation estimate $[\theta_i^{x*}, \theta_i^{y*}]$ that minimizes the objective f_o above.

Problem 7.2. *(Position Estimation): For each agent a_i, given a_i's orientation estimate $\theta_i = \arctan 2(\theta_{iy}, \theta_{ix})$, we define the swarm's position estimate error f_p as:*

$$\frac{1}{4}\sum_{a_i\in\mathcal{A}}\sum_{a_j\in N_i}\frac{1}{|N_i|}\frac{1}{|N_j|}\left\|\begin{bmatrix}x_i\\ y_i\end{bmatrix}+d_{ij}\begin{bmatrix}\cos(\mathcal{B}_{ij}+\theta_i)\\ \sin(\mathcal{B}_{ij}+\theta_i)\end{bmatrix}-\begin{bmatrix}x_j\\ y_j\end{bmatrix}\right\|_2^2$$

The task is that: given each agent's orientation estimate obtained from problem 7.1, find each agent a_i's position estimate $[x_i^*, y_i^*]$ that minimizes the objective f_p above.

The objective f_o and f_p can be intuitively interpreted as the normalized sum of the inconsistency between each pair of agents' pose estimates and their distance, orientation difference, as well as bearings relative to each other.

7.2.3. From bearing angle to orientation difference

One can see that in order to solve problem 7.1, each agent a_i needs to be able to measure nearby agent a_j's orientation difference $\mathcal{W}_{ij}$. On the other hand, as stated in chapter 2, agent a_i is not able to measure the neighbor a_j's orientation difference directly, the only two measurements it can obtain from the sensor are: a_j's bearing and distance. As pointed out in [96], for two agents a_i and a_j, given their bearing angles in the other's local frame $\mathcal{B}_{ji}$ and $\mathcal{B}_{ij}$, their orientation difference $\mathcal{W}_{ij}$ can be explicitly written as:

$$\mathcal{W}_{ij} = \langle \mathcal{B}_{ij} - \mathcal{B}_{ji} + \pi \rangle_{2\pi}$$

where $\langle\cdot\rangle_{2\pi}$ denotes the 2π modulus operator introduced in Section 7.2.1.

7.3. Approach

As briefly discussed in Section 7.2.2, in the presented algorithm, the agents will actively refine their pose estimates to minimize the objective f_o and f_p. When doing so, each agent will treat the objective f_o and the objective f_p separately: each agent a_i updates its orientation estimate $[\theta_{ix}, \theta_{iy}]$ only according to objective f_o (Algorithm 8, Line 20-21), and then uses the obtained orientation estimate $[\theta_{ix}, \theta_{iy}]$ to calculate objective f_p so as to update its position estimate $[x_i, y_i]$ (Algorithm 8, Line 22-24).

In the presented algorithm, each agent will consistently broadcast its current pose estimate to the neighbors at a fixed frequency of f (Algorithm 8, Line 28), meanwhile, at the same frequency, it will periodically update its pose estimate using the messages received (Algorithm 8, Line 10-26). For each agent, the message received from another agent can be characterised by a 3-tuple $msg = (data, \hat{\mathcal{B}}, \hat{d})$, where $data$ is the payload of the message, $\hat{\mathcal{B}}$ is the measurement of transmitter's bearing angle, and $\hat{d}$ is the measurement of transmitter's distance. See Algorithm 8 for the detailed pseudo code of the proposed algorithm, and below is a detailed description of the main variables used in the algorithm:

- *id*: the agent's id, which is locally unique;
- α, β: the variables to control the step size of the update of the agent's pose estimate;
- *msg_buff*: the buffer to store the messages received in the latest $\frac{1}{f}$ amount of time;
- *last_check*: the intermediate variable that assistants to execute periodical broadcast;
- *clock()*: the syscall that returns the time elapsed since the program started;
- $\theta_{ix}, \theta_{iy}, x_i, y_i$: the agent a_i's pose estimate;
- *echo_id, echo_*$\mathcal{B}$*, echo_msg*: the variables that assist the agent to execute the "random echo" protocol so as to obtain its own bearing angle measured in the other's local frame;

- *msg_out*: the message to be transmitted.
- *msg_in*: a 3-tuple that contains the payload of received message, the measurement of message's transmitter's bearing angle, and the measurement of the message's transmitter's distance.

For each agent in swarm, every time when it receives a message, it will store it in the buffer *msg_buff* (Algorithm 8 Line 31-32). Each agent will periodically process the messages in buffer *msg_buff* (Algorithm 8, Line 9-26) and then empty the buffer (Algorithm 8, Line 29) at a fixed frequency of f (Algorithm 8, Line 8). When an agent a_i attempts to update its orientation estimate $[\theta_{ix}, \theta_{iy}]$, the first task is to measure the orientation differences from its neighbor a_j. As stated in Section 7.2.3, for an agent a_i, the calculation of its orientation difference from its neighbor a_j requires two measurements: $\hat{\mathcal{B}}_{ij}$, which is the measurement of a_j's bearing angle in a_i's local frame, and $\hat{\mathcal{B}}_{ji}$, which is the measurement of a_i's bearing angle in a_j's local frame. On the other hand, the second measurement $\hat{\mathcal{B}}_{ji}$ cannot be obtained by a_i via local sensing directly. In order to allow each agent to obtain its own bearing angle measured by its neighbor, the agents will cooperatively execute a "random echo" protocol: for each agent in the swarm, every time when forging *msg_out*, which is the message to be transmitted, it will first uniformly and randomly select a message *echo_msg* in the buffer *msg_buff*, then embeds the id and the measurement of bearing angle of this *echo_msg*'s transmitter in the *msg_out* (Algorithm 8, Line 11-13, Line 26). By cooperatively doing so, each agent a_i will be able to obtain its own bearing angle measured by any neighbor a_j with a non-zero probability. See Fig. 7.2 for a minimal working example for this "random echo" protocol. In this example, there are three agents involved. Recall that it is assumed that the agents' clocks are asynchronous in phase, therefore, from a global observer's perspective, despite that each agent is programmed to broadcast at the same frequency, they still might

Algorithm 8: Proposed algorithm (runs on each agent a_i)

Input: *id*, f, α, β

msg_buff $\leftarrow \emptyset$
echo_id $\leftarrow$ *id*
*echo_*$\mathcal{B}$ $\leftarrow 0$
last_check $\leftarrow$ *clock()*
Initialize the pose estimation $\theta_{ix}, \theta_{iy}, x_i, y_i$
msg_out $\leftarrow \{$*id*, $\theta_{ix}, \theta_{iy}, x_i, y_i$, *echo_id*, *echo_*$\mathcal{B}\}$
while *True* **do**
  if *clock() - last_check* $> \frac{1}{f}$ **then**
    last_check $\leftarrow$ *clock()*
    if *msg_buff is* ***not*** *empty* **then**
      echo_msg $\leftarrow$ uniformly and randomly choose a element in *msg_buff*
      echo_id $\leftarrow$ *echo_msg.data.id*
      *echo_*$\mathcal{B}$ $\leftarrow$ *echo_msg.*$\hat{\mathcal{B}}$
      msg_selected $\leftarrow$ uniformly and randomly choose a element in *msg_buff*
      if *msg_selected.data.echo_id* == *id* **then**
        $\theta_{jx}, \theta_{jy}, x_j, y_j \leftarrow$ the pose estimates contained in *msg_selected.data*
        $\hat{\mathcal{B}}_{ij} \leftarrow$ *msg_selected.*$\hat{\mathcal{B}}$
        $\hat{d}_{ij} \leftarrow$ *msg_selected.*$\hat{d}$
        $\hat{\mathcal{B}}_{ji} \leftarrow$ *msg_selected.data.echo_*$\mathcal{B}$
        $\hat{\mathcal{W}}_{ij} \leftarrow \langle \hat{B}_{ij} - \hat{\mathcal{B}}_{ji} + \pi \rangle_{2\pi}$
        $\delta_\theta \leftarrow \begin{bmatrix} \cos\left(\hat{\mathcal{W}}_{ij}\right), \sin\left(\hat{\mathcal{W}}_{ij}\right) \\ -\sin\left(\hat{\mathcal{W}}_{ij}\right), \cos\left(\hat{\mathcal{W}}_{ij}\right) \end{bmatrix} \begin{bmatrix} \theta_{jx} \\ \theta_{jy} \end{bmatrix} - \begin{bmatrix} \theta_{ix} \\ \theta_{iy} \end{bmatrix}$
        $\theta_i \leftarrow \arctan 2(\theta_{iy}, \theta_{ix})$
        $\theta_j \leftarrow \arctan 2(\theta_{jy}, \theta_{jx})$
        $\delta_{xy} \leftarrow \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \frac{\hat{d}_{ij}}{2} \begin{bmatrix} \cos\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \cos\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \\ \sin\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \sin\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix}$
        $\begin{bmatrix} \theta_{ix} \\ \theta_{iy} \end{bmatrix} \leftarrow \begin{bmatrix} \theta_{ix} \\ \theta_{iy} \end{bmatrix} + \alpha\delta_\theta$
        $\begin{bmatrix} x_i \\ y_i \end{bmatrix} \leftarrow \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \beta\delta_{xy}$
    msg_out $\leftarrow \{$*id*, $\theta_{ix}, \theta_{iy}, x_i, y_i$, *echo_id*, *echo_*$\mathcal{B}\}$
    transmit *msg_out*
    msg_buff $\leftarrow \emptyset$
  else
    if *receive a message msg_in* **then**
      msg_buff $\leftarrow$ *msg_buff* $\cup \{$*msg_in*$\}$

transmit messages at different times, as shown in Fig. 7.2. Due to the space constraints, in the example, we will only track the agent 0's behavior, which suffices to demonstrate the "random echo" protocol as all the agents are identically programmed. At t_0 and t_1, agent 0 receives messages from agent 1 as well as agent 2 and stores them in the buffer *msg_buff* (Algorithm 8, Line 33). At t_2, agent 0 generates a message and transmits it out. When generating this message, agent 0 uniformly and randomly selects a message in the *msg_buff* (which is the message from agent 1 at t_2), and embeds the id as well as the measurement of the bearing angle of this selected message's transmitter in the message to be transmitted (Algorithm 8, Line 11-13). This transmitted message will allow the agent 1 to obtain its own bearing angle measured by agent 0. Right after transmitting the message, agent 0 will empty its buffer *msg_buff*. The agent 0's behaviors at t_3, t_4 and t_5 are almost the same as t_0, t_1 and t_2, except that at t_5, it forges the message using the message from agent 2 instead of the message from agent 1.

As discussed before, each agent a_i will use the messages in the buffer *msg_buff* to update its pose estimate at a fixed frequency of f (Algorithm 8, Line 9-29). To be more specific, every $\frac{1}{f}$ amount of time, each agent a_i will first uniformly and randomly select a message in the buffer *msg_buff*, which is denoted as *msg_selected* in Algorithm 8 (Algorithm 8, Line 14). Then, a_i will check if the value of the field *echo_id* in *msg_selected* matches its own id, i.e., if this *msg_selected* contains a_i's bearing angle measured by the other. If so, a_i will use the information in *msg_selected* to update its pose estimate (Algorithm 8, Line 16-26) and then empty the buffer *msg_buff* (Algorithm 8, Line 29). Otherwise, a_i will do nothing but empty the buffer *msg_buff* directly (Algorithm 8, Line 29). Each time when agent a_i updates its pose estimate (Algorithm 8, Line 25-26), the step size is controlled by the variables α and β.

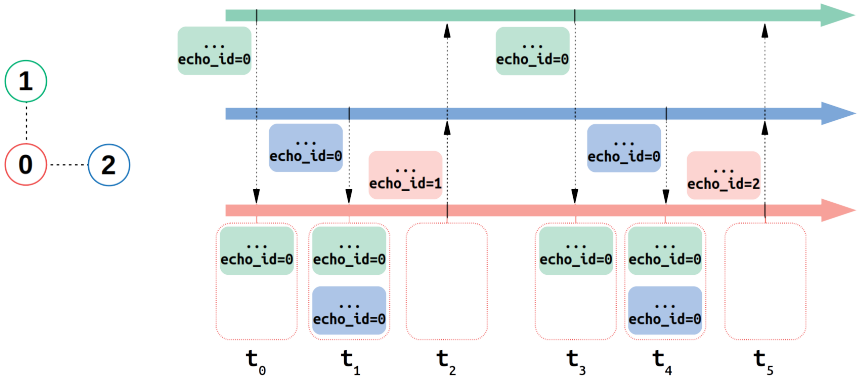

Figure 7.2. (Left) Physical positions of agents: each disk represents an agent, the number on the disk indicates the agent's id, the dotted line connecting two agents indicates that those two agents are located within each other's communication range. (Right) The horizontal colored arrow lines are agents' local clocks running from the left to the right. Each vertical dotted arrow line is a broadcast event, which points from the transmitter to the receiver(s). Each filled box is a transmitted message, the box's color shows its transmitter. The array of unfilled boxes attached to agent 0's clock on the bottom shows the values of agent 0's variable *msg_buff* at different times $t_0, ..., t_5$, each filled box inside the *msg_buff* is a received message currently stored in agent 0's buffer *msg_buff*.

7.4. Theoretical Results

This section studies how each agent's sensing noise and the communication loss will affect the swarm's behavior in expectation, and I show that when the agent's communication loss and sensing noise is low, the swarm's behavior can be approximated as the unbiased stochastic gradient descent (SGD) on the objective f_o and f_p, In addition, the analysis about the algorithm's complexity is given.

7.4.1. Analysis of the swarm's behavior

For the sake of analysis, first, I describe the swarm's behavior from a global observer's perspective. Say there is a global observer holding a clock that has the same frequency as

Algorithm 9: Swarm's behavior (From a global observer's perspective)

1 **for** $k \leftarrow 1, 2, ..., \infty$ **do**
2 **for** *each agent* $a_i \in \mathcal{A}$ **do**
3 Assign each neighbor $a_j \in N_i$ a probability: $\frac{1-(1-\lambda)^{|N_j|}}{|N_j|}\frac{1-(1-\lambda)^{|N_i|}}{|N_i|}$, where $(1-\lambda)$ is the packet loss rate
4 randomly select a neighbor a_j according to its probability assigned above
5 $\hat{\mathcal{B}_{ij}} \leftarrow \langle \mathcal{B}_{ij} + \epsilon_{\mathcal{B}} \rangle_{2\pi}$, where $\epsilon_B \sim \mathcal{N}(0, \sigma_{\mathcal{B}}^2)$
6 $\hat{\mathcal{B}_{ji}} \leftarrow \langle \mathcal{B}_{ji} + \epsilon_{\mathcal{B}} \rangle_{2\pi}$, where $\epsilon_B \sim \mathcal{N}(0, \sigma_{\mathcal{B}}^2)$
7 $\hat{d}_{ij} \leftarrow d_{ij} + \epsilon_d$, where $\epsilon_d \sim \mathcal{N}(0, \sigma_d^2)$
8 $\hat{\mathcal{W}}_{ij} \leftarrow \langle \hat{B}_{ij} - \hat{\mathcal{B}}_{ji} + \pi \rangle_{2\pi}$
9 $\delta_\theta \leftarrow \begin{bmatrix} \cos\left(\hat{\mathcal{W}}_{ij}\right), \sin\left(\hat{\mathcal{W}}_{ij}\right) \\ -\sin\left(\hat{\mathcal{W}}_{ij}\right), \cos\left(\hat{\mathcal{W}}_{ij}\right) \end{bmatrix} \begin{bmatrix} \theta_{jx}^{k-1} \\ \theta_{jy}^{k-1} \end{bmatrix} - \begin{bmatrix} \theta_{ix}^{k-1} \\ \theta_{iy}^{k-1} \end{bmatrix}$
10 $\theta_i \leftarrow \arctan 2(\theta_{iy}^{k-1}, \theta_{ix}^{k-1})$
11 $\theta_j \leftarrow \arctan 2(\theta_{jy}^{k-1}, \theta_{jx}^{k-1})$
12 $\delta_{xy} \leftarrow \begin{bmatrix} x_j^{k-1} \\ y_j^{k-1} \end{bmatrix} - \frac{\hat{d}_{ij}}{2} \begin{bmatrix} \cos\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \cos\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \\ \sin\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \sin\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \end{bmatrix} - \begin{bmatrix} x_i^{k-1} \\ y_i^{k-1} \end{bmatrix}$
13 $\begin{bmatrix} \theta_{ix}^k \\ \theta_{iy}^k \end{bmatrix} \leftarrow \begin{bmatrix} \theta_{ix}^{k-1} \\ \theta_{iy}^{k-1} \end{bmatrix} + \alpha\delta_\theta$
14 $\begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} \leftarrow \begin{bmatrix} x_i^{k-1} \\ y_i^{k-1} \end{bmatrix} + \beta\delta_{xy}$

each agent's local clock. This global observer will record every agent's pose estimate at a fixed frequency f, which is the same as the frequency at which each agent will attempt to update its pose estimate (Algorithm 8, Line 9-29). It is straight forward to examine that: by doing so, this global observer will be able to capture all the changes of each agent's pose estimate, see Proposition 7.1 for the formal proof of this conclusion.

Proposition 7.1. *Let $\theta_{ix}^k, \theta_{iy}^k, x_i^k, y_i^k$ be agent a_i's pose estimate observed by the global observer at time $k\frac{1}{f}$, the swarm's behavior can be described by the Algorithm 9 without loss of any information.*

Proof. See Appendix 5. □

It is worth noting that the objective f_o and f_p are defined using the true inter-agent relative bearing angles and distances, whereas in reality, the actual measurements that each agent uses to update its pose estimate will be corrupted by the sensing noise (Algorithm 9, Line 5-7). Next, I will show how the sensing noise and the communication loss will affect the swarm's behavior.

Lemma 7.1. *Let $\Theta_i^k = [\theta_{ix}^k, \theta_{iy}^k]^\intercal$ be agent a_i's orientation estimate at k^{th} iteration, in addition, let $\frac{\partial f_o}{\partial \Theta_i}^k$ be the value of partial derivative of objective f_o with respect to Θ_i calculated using agents' orientation estimates at k^{th} iteration, we have:*

$$\mathbb{E}(\Theta_i^{k+1} - \Theta_i^k) = -\mu\alpha\{\frac{\partial f_o}{\partial \Theta_i}^k + \gamma_\Theta^k\}$$

In which:

$$\mu = 1 - (1-\lambda)^{|N_i|} \tag{7.1}$$

$$\gamma_\Theta^k = \sum_{a_j \in N_i} \frac{1 - \exp\{-\sigma_\mathcal{B}^2\}\{1-(1-\lambda)^{|N_j|}\}}{|N_j|} \mathcal{R}(-\mathcal{W}_{ij})\Theta_j^k - \frac{(1-\lambda)^{|N_j|}}{|N_j|}\Theta_i^k \tag{7.2}$$

where $\mathcal{R}(\cdot) = \begin{bmatrix} \cos(\cdot), & -\sin(\cdot) \\ \sin(\cdot), & \cos(\cdot) \end{bmatrix}$ *is the rotation matrix constructed from the angle* $\cdot$ *.*

Proof. See Appendix 6. □

Lemma 7.2. *Assume the agents' orientation estimates have converged to a stable state. Let $\mathcal{X}_i^k = [x_i^k, y_i^k]^\intercal$ be agent a_i's position estimate at k^{th} iteration, moreover, let $\frac{\partial f_p}{\partial \mathcal{X}_i}^k$ be the value of partial derivative of objective f_p with respect to $\mathcal{X}_i$ calculated using agents' position*

estimates at k^{th} iteration, we have:

$$\mathbb{E}(\mathcal{X}_i^{k+1} - \mathcal{X}_i^k) = -\mu\beta\{\frac{\partial f_p}{\partial \mathcal{X}_i}^k + \gamma_{\mathcal{X}}^k\}$$

In which:

$$\mu = 1 - (1-\lambda)^{|N_i|} \tag{7.3}$$

$$\gamma_{\mathcal{X}}^k = \sum_{a_j \in N_i} \frac{1 - \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}\{1-(1-\lambda)^{|N_j|}\}}{2|N_j|}\mathcal{P}(i,j)d_{ij} + \frac{(1-\lambda)^{|N_j|}}{|N_j|}\{\mathcal{X}_j^k - \mathcal{X}_i^k\} \tag{7.4}$$

where $\mathcal{P}(i,j) = -\begin{bmatrix}\cos(\mathcal{B}_{ij}+\theta_i) - \cos(\mathcal{B}_{ji}+\theta_j) \\ \sin(\mathcal{B}_{ij}+\theta_i) - \sin(\mathcal{B}_{ji}+\theta_j)\end{bmatrix}$ *is the matrix constructed from a pair of agents a_i and a_j's relative bearing angles as well as their orientation estimates.*

Proof. See Appendix 7. □

Lemma 1 and Lemma 2 suggest that: at each iteration, in expectation, each agent will update its pose estimate along a direction that is slightly deviated from the objective's gradient direction, with a step sizes that is slightly smaller than the user-specified one. As we can see in Lemma 1 and Lemma 2, the bias of the update direction γ_Θ, $\gamma_\mathcal{X}$, and the depreciation factor of the step size μ, are essentially functions of the packet loss rate and the variance of agent's sensing noise, in other words, the deviation of agent's expected update direction from the gradient and the depreciation of the step size are essentially introduced by the agent's communication loss and sensing noise. The equation 7.1 and 7.3 suggest that: in expectation, the communication loss will "shorten" the each agent's step size. Moreover, the effect of the communication loss can be reduced by agent a_i's degree, i.e., the number of

the a_i's neighbors $|N_i|$. As to the biases of expected update direction γ_Θ and $\gamma_\mathcal{X}$, according to the equation 7.2 and 7.4, these biases are a result of both the communication loss and the sensing noise.

On the other hand, one can observe that: the bias of the update direction γ_Θ, $\gamma_\mathcal{X}$, and the depreciation of the step size will super-linearly decay over the packet loss rate as well as the agent's sensing noise. This suggests that: when the communication loss and sensing noise are reasonably small, the bias of update direction γ_Θ, $\gamma_\mathcal{X}$, and the depreciation of the step size will become negligible.

Assumption 7.1. *Each agent a_i's degree $|N_i|$ is not too low and the packet loss rate $1-\lambda$ is not too high such that:* $(1-\lambda)^{|N_i|} \approx 0$.

Assumption 7.2. *The variance of agent's bearing sensing noise $\sigma_\mathcal{B}^2$ is not too big such that:* $\exp\left\{-\frac{\sigma_\mathcal{B}^2}{2}\right\} \approx 1$.

Remark: The assumption 1 and 2 are actually pretty mild. One can consider a case where the packet loss rate $1-\lambda$ is 10% and the lowest degree of any agent in the swarm is 2. In this case, $(1-\lambda)^{|N_i|} = 0.01 \approx 0$. For assumption 2, consider a case where each agent's bearing sensing noise has a standard deviation (std) of 0.3 rad, which is around the same as the std of the angle measurements obtained from the Bluetooth 5.1 devices [98]. In this case, $\exp\left\{-\frac{\sigma_\mathcal{B}^2}{2}\right\} = \exp\{-0.045\} \approx 1$.

Theorem 7.1. *If Assumption 1 and 2 hold, then the swarm's behavior can be approximated as the unbiased stochastic gradient descent on objective f_o and f_p. Namely:*

$$\mathbb{E}(\Theta_i^{k+1} - \Theta_i^k) \approx -\alpha \frac{\partial f_o}{\partial \Theta_i}^k, \qquad \mathbb{E}(\mathcal{X}_i^{k+1} - \mathcal{X}_i^k) \approx -\beta \frac{\partial f_p}{\partial \mathcal{X}_i}^k$$

PROOF. Theorem 1 can be easily obtained by substituting $(1-\lambda)^{|N_i|}$ with 1 and substituting $\exp\left\{-\frac{\sigma_B^2}{2}\right\}$ with 1 in equation 7.1, equation 7.2, equation 7.3, and equation 7.4. □

So far, we have seen that when the communication loss and sensing noise is reasonably small, the swarm's behavior is equivalent to the unbiased SGD on the objective f_o and f_p. In other words, in expectation, the algorithm allows the swarm to constantly reduce the disagreement between their pose estimates and their local measurements, which suffices to show the algorithm's correctness.

7.4.2. Complexity

First, it is straight forward to examine that the memory to execute the algorithm is dominated by the size of buffer *msg_buff*. Therefore, for an agent a_i, the algorithm's memory complexity is $\mathcal{O}(|N_i|)$, where $|N_i|$ is the number of a_i's neighbors.

Next, if it is assumed assume that the complexity of querying a random number generator is $\mathcal{O}(1)$, then, the cost to execute Algorithm 8 Line 8-32 is $\mathcal{O}(1)$. In addition, during a unit of time, each agent a_i can receive at most $f|N_i|$ messages, that is, in a unit of time, a_i will execute Algorithm Line 8-32 at most $f|N_i|$ times. Thus, for an agent a_i, the computation complexity of the algorithm is $\mathcal{O}(f|N_i|)$.

Lastly, given the fact that the length of each message exchanged amongst the agents is $\mathcal{O}(1)$, one can easily conclude that the algorithm's communication complexity, i.e., the amount of data to be transmitted by each agent in a unit of time, is $\mathcal{O}(f)$.

7.5. Empirical Evaluation

This section investigates the performance of proposed algorithm empirically in a 100-robot swarm and in simulation.

To qualitatively evaluate the algorithm's performance, I first introduce two metrics *NOEE* (normalized orientation estimate error) and *NPEE* (normalized position estimate error), which are the metrics to evaluate the error of agents' orientation estimates and position estimates, respectively. The *NOEE* and *NPEE* are defined as follows: $\theta_i^k = \operatorname{atan}2(\theta_{iy}^k, \theta_{ix}^k)$ and $[x_i^k, y_i^k]$ denote agent a_i's orientation estimate and position estimate observed by the global observer at k^{th} iteration, in addition, θ_i^g and $[x_i^g, y_i^g]$ denote agent a_i's true pose in a global coordinate system. At each iteration k, I first calculate the rotation and translation that can match the agents' estimated positions and their true positions the best, namely, find $\theta_*^k \in (-\pi, \pi], t_*^k \in \mathbf{R}^2$ that minimize the following objective:

$$\min \sum_{a_i \in \mathcal{A}} \left\| \begin{bmatrix} \cos(\theta_*^k), -\sin(\theta_*^k) \\ \sin(\theta_*^k), \quad \cos(\theta_*^k) \end{bmatrix} \begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} + t_*^k - \begin{bmatrix} x_i^g \\ y_i^g \end{bmatrix} \right\|_2^2$$

Then, the *NOEE* and *NPEE* at k^{th} iteration are defined as:

$$NOEE^k = \frac{1}{n} \sum_{a_i \in \mathcal{A}} |\langle \theta_i^k + \theta_*^k - \theta_i^g \rangle_{2\pi}|$$

$$NPEE^k = \frac{1}{n} \sum_{a_i \in \mathcal{A}} \left\| \begin{bmatrix} \cos(\theta_*^k), -\sin(\theta_*^k) \\ \sin(\theta_*^k), \quad \cos(\theta_*^k) \end{bmatrix} \begin{bmatrix} x_i^k \\ y_i^k \end{bmatrix} + t_*^k - \begin{bmatrix} x_i^g \\ y_i^g \end{bmatrix} \right\|_2$$

where $\langle \cdot \rangle_{2\pi}$ denotes the 2π modulus operator introduced in Section 7.2.1, and n is the swarm size. The *NOEE* is the normalized geodesic distance between agents' transformed orientation estimates and their true orientations, and *POEE* is the normalized Euclidean distance between agents' transformed position estimates and their true positions.

7.5.1. Simulation

In simulation, the robot's communication range is set to 0.3 m, the robot's communication frequency f is set to 30 Hz. Moreover, the inter-robot communication channel has a packet loss rate of 10%. In each test, the simulated robots are placed in three patterns: the circle, the shape "N", and the random mesh. See Fig. 7.3 for a graphical illustration of these three patterns. In the circle pattern, n robots are evenly distributed on on a circle. The circle's radius is made such that the distance between two adjacent robots is 0.25 m, in addition, each robot's orientation is set to be such that: each agent faces straight towards to the center of the circle. In the shape "N" pattern, the robots' positions form a "N" shape on a grid, moreover, the distance between any pair of adjacent robots is 0.2m. In the random mesh configuration, n agents are randomly placed in a $0.25\sqrt{n}$ m $\times$ $0.25\sqrt{n}$ m space, moreover, when generating each robot's position, the swarm's communication graph is enforced to be connected. In both shape "N" pattern and random mesh pattern, each robot's orientation is set randomly.

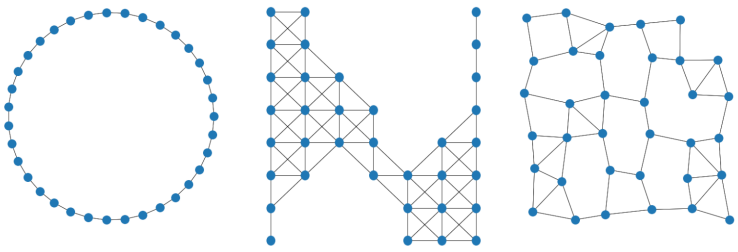

Figure 7.3. From left to right: the circle pattern, the shape "N" pattern, and the random mesh pattern for a swarm of 35 robots. Each dot represents a robot, the line connecting two robots indicates that those two robots are located within each other's communication range.

The first test studies how the sensing noise will affect the algorithm's performance. In this test, a swarm of 100 simulated robots estimate their poses with a step size of $\alpha = \beta = 0.2$.

Each robot a_i's pose estimate is randomly initialized in a way that $x_i^0 \sim \mathcal{U}(-20, 20), y_i^0 \sim \mathcal{U}(-20, 20), \theta_i^0 \sim \mathcal{U}(-\pi, \pi)$, where $\mathcal{U}(a, b)$ stands for uniform distribution between the interval (a, b). I test the algorithm's performance with three noise profiles: $\sigma_\mathcal{B} = 0.05$ rad, $\sigma_d = 0.01$ m; $\sigma_\mathcal{B} = 0.1$ rad, $\sigma_d = 0.02$ m; and $\sigma_\mathcal{B} = 0.2$ rad, $\sigma_d = 0.04$ m, where $\sigma_\mathcal{B}, \sigma_d$ are the standard deviations of robot's bearing sensing noise and range sensing noise, respectively. For each noise profile, 30 trials were run. The results are shown in Fig 7.4. As we can see in the figure, for all three patterns, the convergence rate of swarm's pose estimate is almost the same for different noise profiles. On the other hand, as expected, the sensing noise will affect the accuracy of the swarm's pose estimate: the bigger sensing noise will result in swarm's pose estimate converging to a state with higher error.

A second test studies the algorithm's performance on different swarm size. In this test, swarms of 64, 128, 256 robots estimate their poses with a step size of $\alpha = \beta = 0.2$. Each robot a_i's pose estimate is randomly initialized in a way such that $x_i^0 \sim \mathcal{U}(-20, 20), y_i^0 \sim \mathcal{U}(-20, 20), \theta_i^0 \sim \mathcal{U}(-\pi, \pi)$. In addition, the sensing noise is set to $\sigma_\mathcal{B} = 0.05$ rad, $\sigma_d = 0.01$ m. For each swarm size, 30 trials were run. The results are shown in Fig 7.5. Unsurprisingly,

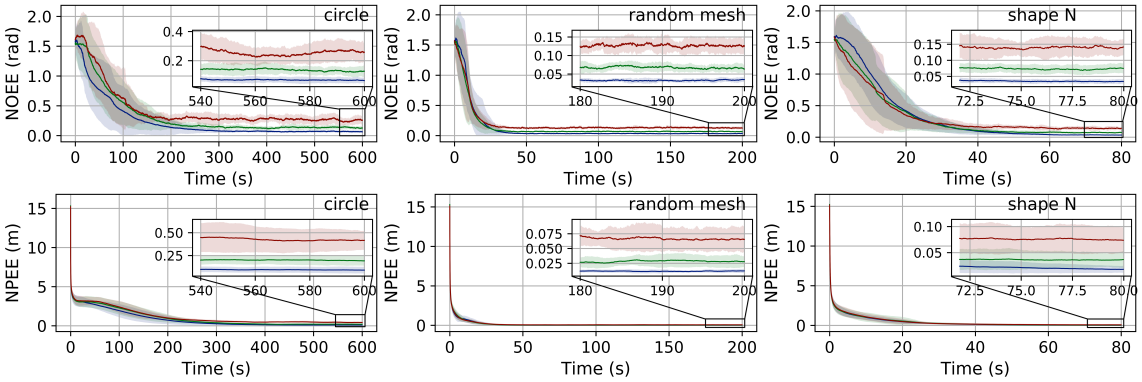

Figure 7.4. Each colored solid line is the mean from 30 trials, and the colored shade areas show the confidence intervals at a confidence level of two σ. The color indicates the noise profile used in experiment: blue – $\sigma_\mathcal{B} = 0.05$ rad, $\sigma_d = 0.01$ m; green – $\sigma_\mathcal{B} = 0.1$ rad, $\sigma_d = 0.02$ m; red – $\sigma_\mathcal{B} = 0.2$ rad, $\sigma_d = 0.04$ m.

the results from experiments suggests that for all three patterns, the swarm size will affect both the convergence rate and the accuracy of the swarm's pose estimate: the larger the swarm size is, the slower the swarm's pose estimate will converge, and the higher the swarm's localization error will be.

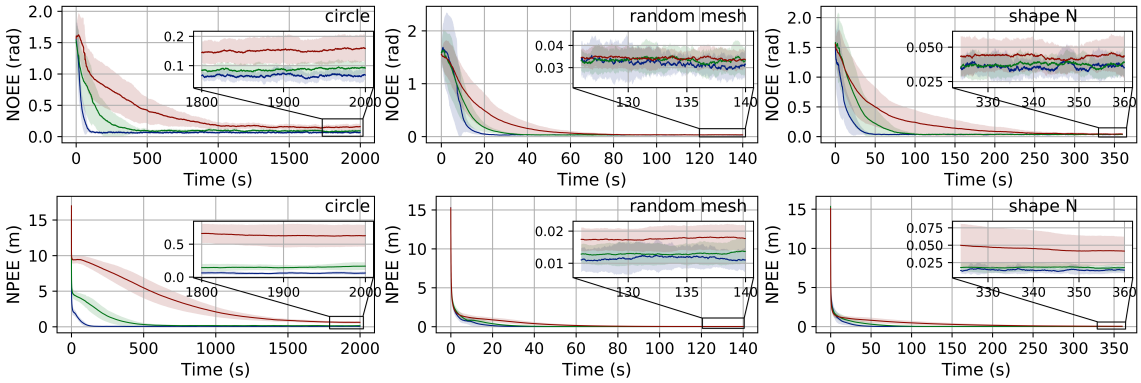

Figure 7.5. Each colored solid line is the mean from 30 trials, and the colored shade areas show the confidence intervals for a confidence level of two σ. The color indicates the step size used in experiment: blue – $\alpha = \beta = 0.1$; green – $\alpha = \beta = 0.2$; red – $\alpha = \beta = 0.3$.

A third test studies the effect of the step size α and β on the algorithm's performance. In this test, a swarm of 100 simulated robots estimate their poses using three different step sizes: $\alpha = \beta = 0.1$, $\alpha = \beta = 0.2$, and $\alpha = \beta = 0.3$. The noise profile used in this test is $\sigma_{\mathcal{B}} = 0.05$ rad, $\sigma_d = 0.01$ m. Each robot a_i's pose estimate is randomly initialized in a way such that $x_i^0 \sim \mathcal{U}(-20, 20), y_i^0 \sim \mathcal{U}(-20, 20), \theta_i^0 \sim \mathcal{U}(-\pi, \pi)$. For each step size, 30 trials were run. The results are shown in Fig 7.6. As we can see in the figure, larger step size will enable the swarm's pose estimate to converge faster, at a cost of swarm's pose estimate's accuracy.

In all three tests, we can see that: compared to the other two patterns, it takes much longer for the circle pattern to converge. One possible explanation is that: given n agents,

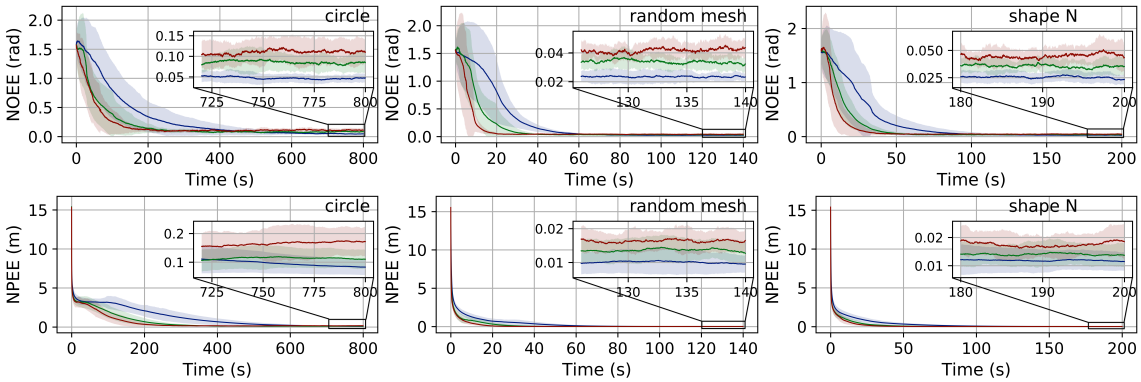

Figure 7.6. Each colored solid line is the mean from 30 trials, and the colored shade areas show the confidence intervals for a confidence level of two σ. The color indicates the step size used in experiment: blue – $\alpha = \beta = 0.1$; green – $\alpha = \beta = 0.2$; red – $\alpha = \beta = 0.3$.

the circle pattern's communication diameter, i.e., the maximal pairwise inter-agent communication hop, is $\mathcal{O}(n)$, whereas the random mesh pattern and shape "N" pattern has a communication diameter of $\mathcal{O}(\sqrt{n})$. The communication diameter essentially characterizes the cost to spread an agent's information across the entire swarm. The larger the communication diameter is, the longer it will take to spread a agent's information across the swarm. As a result, the larger communication diameter makes circle pattern converge much slower than the others. In addition, the difference of each pattern's communication diameter can also be used to explain the observation that the circle pattern has a higher localization error, as the sensing error will accumulate over the communication hop.

7.5.2. Experiments

This section examines the algorithm's performance on a swarm of 100 *Coachbot V2.0* robots. Note that *Coachbot V2.0* robot does not have real bearing and range sensor. In order to implement the presented algorithm on *Coachbot V2.0*, in experiments, I embed the robot's position in the transmitted data packet, and the receiver can calculate the transmitter's

bearing and distance by comparing its own pose with the position contained in the received message. In addition, embedding the transmitter's position in data packet can also be used to simulate the limited communication range. In the experiment, the robot's communication frequency f is 30 Hz, and the step size is set to $\alpha = \beta = 0.2$. The robots are placed in three patterns: the rectangle pattern, the random mesh pattern, and the shape "N" pattern. See Fig. 7.1 for a graphical illustration of these three patterns. In the rectangle pattern, the robots are densely placed on the perimeter of a rectangle, where the distance between two adjacent robots is 0.15 m, and each robot faces straight towards the center of the rectangle. The robot's communication range is set to 0.2 m in rectangle pattern so as to make each robot's degree to the consistent with the circle pattern used in simulation. The remaining two patterns are the same as the ones used in the simulation, and in those two patterns, the robot's communication range is set to 0.3 m. For the random mesh pattern and the shape "N" pattern, 15 trials were run; for the rectangle pattern, 6 trials were run due to its long convergence time. The results are shown in Fig 7.7. In all the trials, the algorithm reliably converge to all the robots accurately localizing themselves. We can see that for each pattern, its convergence rate is approximately the same as the results obtained in the simulation (the rectangle pattern is corresponding to the circle pattern). The random mesh pattern and the shape "N" pattern can converge in less than a minute, while it takes much longer for the rectangle pattern to converge. In addition, one can see that: for random mesh pattern and the shape "N" pattern, the average converged *NPEE*, i.e., normalized position error, is smaller than 2 cm, and the average converged *NPEE* for the rectangle pattern is around 5 cm. Given the fact that the robot is in a disk shape with a diameter of 13 cm, it can be concluded that: for all three patterns, the robots' average converged position error

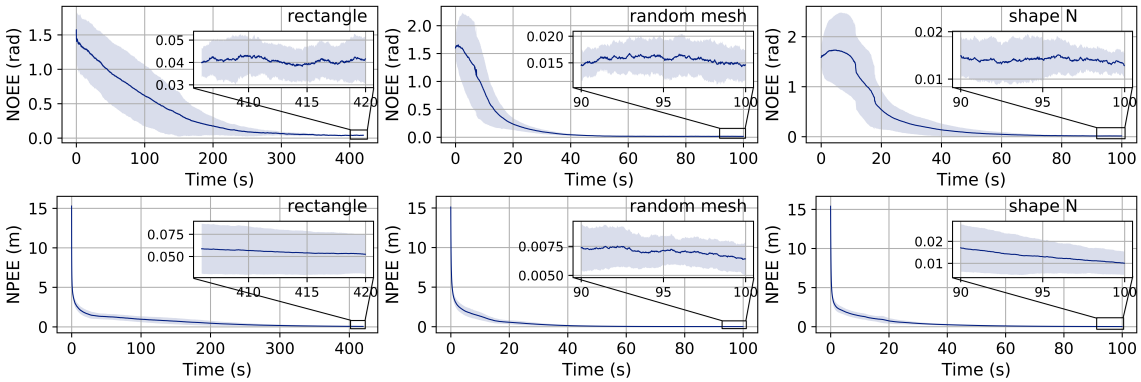

Figure 7.7. Each colored solid line is the mean from multiple trials, and the colored shade areas show the confidence intervals for a confidence level of two σ.

is smaller than the robot's footprint, and in the case of shape "N" pattern and the random mesh pattern, much smaller.

7.5.3. Example use case: decentralized robotic shape formation

One desirable feature of the presented algorithm is that: the algorithm does not require the agent to actively maintain the communication links between itself and its neighbors, or synchronous its lock clock with the others. This feature makes it possible for the algorithm to work in the situations where the swarm's communication topology is dynamically changing. In this demonstration, 100 robots use the presented algorithm to execute the shape formation algorithm presented in Section 8 and [44]. Different from the original version of the algorithm presented in [44], where the agents need to acquire their poses from a global position system, in this experiment, the agents will estimate their poses according to the local measurements and the odometry. Each robot will keep using it on-board odometry to capture the change in its pose (including orientation and position) over time. At a fixed frequency f_{odom}, the robot will add the change in it pose during the past $\frac{1}{f_{odom}}$ amount of time (measured by its

on-board odometry) to its pose estimate. At the same time, it will also constantly refine its pose estimate according to the local measurements using the algorithm presented in this paper. The video of this experiment can be found in [76].

CHAPTER 8

TASK ASSIGNMENT AND FORMATION CONTROL

The task of shape formation in robot swarms can often be reduced to two tasks: assigning goal locations to each robot, and creating a collision-free path to that goal. This chapter presents a distributed algorithm that solves these tasks concurrently, enabling a swarm of robots to move and form a shape quickly and without collision. A user can specify a desired shape as an image, send that to a swarm of identically programmed robots, and the swarm will move all robots to goal locations within the desired shape. This algorithm was executed on a swarm of up to 1024 simulated robots, and a swarm of 100 real robots, showing that it reliably converges to all robots forming the shape. The algorithm presented in this chapter has been published as [44], a summary video of the presented algorithm can be found in [76].

8.1. Background

In general, the complete shape formation problem can be divided into two subproblems: assignment of robots' locations in shape, and routing each robot to reach its assigned location.

The assignment subproblem tries to divide the goal locations among the individuals, often in an optimal way such as minimizing the total distance traveled by the swarm. This problem has been well studied and there are several algorithms which can find the optimal assignment, including the Hungarian algorithm [86], auction algorithms [108, 109], and iterative methods [110, 111]. Recent work shows that some certain assignments which minimize a cost of interest can help to reduce the computational complexity of path planning problems. One

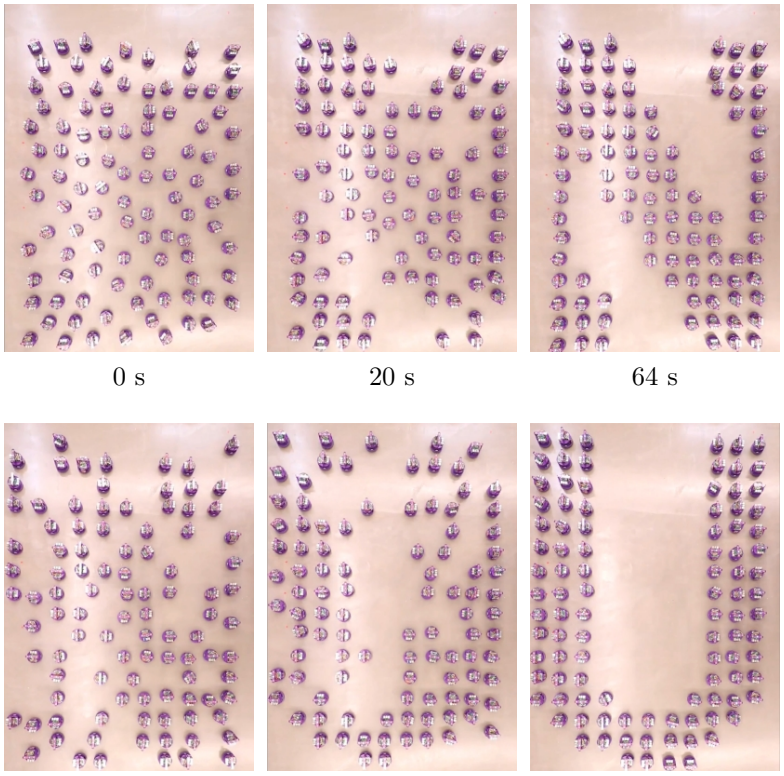

Figure 8.1. Still images from a 100 robot shape formation experiment. The robots start in a random configuration, and move to form the desired "N" shape. Once this shape is formed, they then form the shape "U". The entire sequence is fully autonomous using the distributed algorithm described in this chapter.

typical cost of interest used is the sum of the distance traveled by all agents [27, 112, 113], and the other cost of interests are the sum of the square distance traveled [114] and the maximal distance traveled [115], which help to minimize the total time elapsed. In these methods, the calculation for the assignment is handled by a centralized coordinator. These centralized strategies can deliver a solution to the assignment problem, but do not easily scale to large numbers of robots, present a single point of failure, and do not easily adapt to situations where the number of robots is unknown or can vary.

Unsurprisingly, the distributed assignment methods, on the other hand, can often scale well to the number of robots, and can be more robust to failures [116] and varying numbers of robots. Past efforts try to solve the distributed assignment problem by following an incremental distributed refinement process [117, 118]. Here, the order how agents explore each goal has significant effect on convergence rate. In [117], the agents follow a pre-assigned order which assures that a correct assignment of agents to tasks is always achieved after exploring at most a polynomial number of assignments. In [118], authors obtained an efficient convergence by forcing agents to follow a certain path. This path is collision-free when agents have infinitely small size, but when agents have finite size, the path cannot provide a collision-free guarantee.

After determining the role in shape, each agent then needs to move cooperatively to form the desired shape. In the past, many methods to produce the formation have been proposed. According to the types of actively controlled variables [119] such as agent's position or distances to the neighbors, formation control methods can be categorized into local-measurement-based methods [120–132], and position-based methods [22–30, 35, 46, 47, 52, 112–115, 133–139].

In local-measurement-based methods [120], the agents form the desired shape by actively controlling its distance [121–125], bearing [126], or both [127–131], relative to its neighbors. This type of methods only require the use of relative measurements therefore can be employed in the GPS-denied environments, e.g. indoor environments. For local-measurement-based methods, the challenge is how to obtain the global stabilization to the desired formation using only peer-to-peer information [127].

Some methods achieve the global stabilization by relying on the leader agents [125, 129–131]. In these methods, the leader tracks its desired trajectory, and the non-leader agents

are tasked to maintain certain graph structures rooted from the leader agent where each vertex characterizes an agent and each edge characterizes an inter-agent measurements, such as distance or relative position. These methods allow the swarm to stabilize to a formation that is even dynamically moving, but require an additional leader selection phase to assign a role (leader or non-leader) to each agent.

The methods proposed in [127, 128] are leaderless, these methods enable a group of agents to reliably produce a rigid shape, using the relative positions of agent's neighbors. Nevertheless, in order to achieve global stability, the method proposed in [128] needs the communication graph amongst agents to be complete, and the method proposed in [127] requires the desired formation to satisfy some specific topological conditions.

While local-measurement-based methods permit operations in GPS-denied environments, they often require a centralized coordinator, or the use of a complete communication network, to initially assign each robot a position in final shape. Moreover, without any additional mechanism, it often fails to provide an absolute collision avoidance guarantee when agents have finite size.

To the contrary, in position-based methods, the desired formation is achieved by actively controlling the agents' positions. This type of methods require that each agent is able to measure their own positions with respect to a global coordinate system. Here, the challenge is how to efficiently generate collision-free trajectories where agents can achieve the desired formation by moving to goal locations.

Some previous work tackled the problem in a discrete setting [22–24, 133], while others solved the problem in a continuous setting [25, 26, 134, 135]. As expected, these centralized methods suffer from the curse of dimensionality (because the dimension of swarm's joint

configuration space increase exponentially over swarm size), hence often cannot easily scale to large-scale swarms, such as a swarm of over 1000 agents.

An alternative method is to use an artificial potential function to guide agents to the desired formations using gradient descent. Some authors make use of gradient descent to drive agents to goals [28, 29], and some use the potential function to modify current trajectories locally to prevent collision and maintain connectivity [30]. The drawbacks for this kind of method are that it may take a long time to converge, and there is no guarantee provided that they can form the desired shape [27, 138].

Distributed multi-agent path planning is a well studied topic. Some methods are based on local measurements, either relative velocities [47, 136, 137], or relative positions [35], but none of these methods can provide a deadlock-free guarantee, agents can get stuck in a situation where no action can be made for further progress, yet the shape is incomplete. In fact, in the presented review of distributed path planners [35, 46, 47, 132, 136, 137], none of the methods can provide a deadlock-free guarantee and absolute collision-free guarantee at the same time. This is also suggested in [35], which also claims that no deadlock-free distributed path planner that is with absolute collision-free guarantee exists. Other approaches make use of the communication among agents, but in order to guarantee the correctness of the method, require either a lossless fully connected network [52], or precise velocity control [27], which can be difficult to guarantee when implemented in a physical system. In [46], authors presented a distributed collision avoidance strategy which can resolve some certain types of deadlocks using local communication only, but the method cannot resolve all types of deadlocks. A distributed receding horizon control (RHC) based method is proposed in [132], this method requires only the use of relative sensing in robot's local coordinate frames, and is able to provide mathematical guarantees on the achievement of the rendezvous, however,

it has not been shown that the method can provide collision-free guarantee and absolute deadlock-free guarantee at the same time.

This chapter presents a fully distributed shape formation algorithm where each agent is identically programmed and takes the same input, a set of goal points that describes the desired shape. Each agent will use local communication to actively refine the goal assignment and control its position in a distributed fashion. To the best of my knowledge, and supported by [35], the proposed algorithm is the first provably correct fully distributed shape formation algorithm that can also provide absolute collision-free and deadlock-free guarantees, requiring only the use of local communication. Moreover, the physical experiments and simulations presented show that the presented algorithm is robust to real world non-idealities, such as communication errors, sensing errors, and imperfect robot motion.

8.2. Problem Definition

This chapter proposes an algorithm that when given a set of desired target points (which are described by a set of nodes on a grid), moves a swarm of mobile agents so that each agent is located at a target position, and no target position has more than one agent. For each agent, the set of target positions are known a priori, moreover, all the agents agree on the same global reference frame. This algorithm must distribute the target positions among the agents and then drive the agents to their corresponding target position without collision. The system is distributed, agents are identically programmed, and act based on local information gathered through communication. In this section, I will formally state the problem and introduce the notations used in this chapter.

8.2.1. Agent Model

Given the robot capabilities proposed in chapter 2, in this chapter, each agent is modeled as a 2D omni-directional robots that is in a disk shape with a finite radius r, and can move in any direction at speed v_m, or stop. In addition, with the algorithm presented in chapter 7, it is assumed that each agent can measure its own position and orientation in a global coordinate system at all times. Additionally, each agent is able to communicate with any agent lying within its communication range $R \geq 4\sqrt{2}r$. To simplify the analysis and description, here, it is assumed that:

- Each agent has the same clock frequency f_{clock};
- Each agent is able to constantly transmit messages to the neighbors in communication range at a fixed rate f_{comm};
- The local inter-agent communication is lossless;
- Each agent has the same v_m;
- The communication latency is negligible.

Note that here I do not have any assumption on the phase of the agent's clock relative to each other, they can be asynchronous in phase. When the algorithm is implemented in the real world, these assumptions can be relaxed to accommodate the real-world non-idealities, see Section 8.5.2.1 for detailed discussion.

8.2.2. Notations

For the sake of describing the presented algorithm and formulating the problem, I introduce the notations as follows:

Let $A = \{a_1, a_2, ..., a_n\}$ be a set of agents, where each agent $a_i \in A$ has a position $p_{a_i}(t) \in \mathbb{R}^2$ at time t. For all $p \in \mathbb{R}^2$, p^x, p^y denote p's x and y components, respectively. $||\cdot||$ denotes the Euclidean norm on $\mathbb{R}^2$ space and $\succ$ denotes the lexicographic order on $\mathbb{R}^2$ space, namely, $p_1 \succ p_2$ if and only if: $p_1^x > p_2^x$, or $p_1^x = p_2^x$ and $p_1^y > p_2^y$. Let $Q = \{q_1, ..., q_m\}$ be a set of distinct target locations, where $q_i \in \mathbb{R}^2$, it is assumed that $\forall q_i, q_j \in Q, ||q_i - q_j|| \geq 2\sqrt{2}r$, i.e, in the desired shape no pair of robots collide with each other. Moreover, $T_{a_i}(t) \in Q$ denotes a_i's assigned target position at time t. It is assumed that every agent has the same communication range R and same radius r, and $N_{a_i}(t) \subset A$ denotes the set such that at time t, $\forall a_j \neq a_i$, $||p_{a_i}(t) - p_{a_j}(t)|| \leq R$ if and only if $a_j \in N_{a_i}(t)$, in the other words, $N_{a_i}(t)$ is the set of agents that are able to communicate with a_i at time t.

8.2.3. The Problem Formulation

The task is to design an algorithm to move a swarm of n identical robots, represented by set A, from their initial positions to an arbitrary connected target formation, represented by set Q. To simplify the problem, it is assumed that $|Q| = |A|$. The algorithm should be deadlock-free and collision-free, that is:

- $\exists t_{max} > 0$ such that at any time $t > t_{max}$, it holds that: $\forall a_i \in A$, $p_{ai}(t) = T_{a_i}(t)$, moreover, $\forall a_j \neq a_i, T_{a_j}(t) \neq T_{a_i}(t)$,
- $\forall t \geq 0$, for any two agents $a_i \neq a_j$, $||p_{a_i}(t) - p_{a_j}(t)|| \geq 2r$.

8.3. Approach

To form the goal shape, each agent needs to pick a valid goal, and then move on a collision-free and deadlock-free path towards that goal without any centralized coordination.

Algorithm 10: General Framework for Shape Formation

Input: $Q = \{q_1, q_2, ..., q_m\}$

1 $T_{a_i}(t) \leftarrow$ random element in Q
2 **while** *True* **do**
3 **if** $\exists a_j \in N_{a_i}(t)$, *s.t.* $T_{a_j}(t) = T_{a_i}(t)$ **then**
4 run *new goal selector* (Alg. 13 Line 3-10)
5 run *motion planner* (Alg. 11 Line 25-29)

For this task, two subproblems arise. One of them is solving duplicated assignments, which is caused by the limited sensing ability of the agents. Agents have limited communication range so they have to determine their targets based only on the local information. This makes it possible that there exist multiple robots holding the same target. The other subproblem is planning each robot's motion based on local information so as to generate collision-free and deadlock-free paths toward the goals. Additionally, if every agent's target is unique, the motion planner should guarantee that each agent will reach the target in a finite amount of time.

In the presented algorithm, the task is handled by two modules: the *new goal selector*, which is used to pick a valid goal, and the *motion planner*, which is used to plan the agent's motion. Each agent uses the *motion planner* to move to its current goal, and if it encounters another agent holding the same goal, one of them uses the *new goal selector* to pick another goal, while the other one will stay with its current goal. A detailed description is shown as Alg. 10. Note that this algorithm runs on each agent of the swarm.

8.3.1. Motion planning

To generate a collision-free path, the continuous environment is first converted into a discrete grid, as shown in Fig. 8.2. While this grid representation of the environment will make agent's motion less efficient, it helps to reduce the computational cost of motion planning.

Note that the grid here is the same grid goal points are located on. Let l be the length of the grid edge, where the constraint that $2\sqrt{2}r \leq l \leq \frac{R}{2}$ is enforced for the purpose of collision avoidance. Furthermore, it is assumed that there are no obstacles located in the environment. With this representation, each agent's path is given by a sequence of the waypoints, i.e. the nodes of the grids.

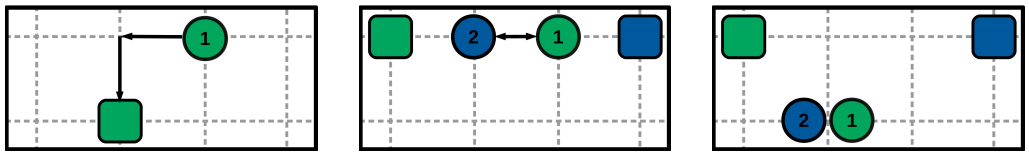

Figure 8.2. Illustration of the grid discretization of space and possible collision cases. The intersections of grey dashed lines represent the feasible waypoints, and agents travel on the edges between waypoints. Each agent's position is shown with a colored circle and its goal point is shown with a square of the same color. Moreover, each agent is labeled with a unique number and the arrow shows agent's incentive for next step. (Left) A valid trajectory for a single agent to move to its goal. The trajectory is shown as a sequence of arrows. (Middle) An edge collision, where blue and green robots both intend to travel on the edge in black, in opposite directions. Here neither can make progress without collision. (Right) Collision happens on a waypoint, where the blue and green robots try to move to the same waypoint at the same time, physically colliding.

For every two adjacent waypoints, the motion controller enforce that the robots travel on the line segment between them. The motion controller plans every robot's motion, such that the following constrains are satisfied:

Constraint 8.1. *At any time t, no agent moves to the waypoint that is currently occupied by the other, and no pair of agents move towards the same waypoint at the same time.*

Constraint 8.2. *At any time t, no pair of agents travel on the same edge in opposite directions.*

For each agent located at any waypoint, there are five possible actions: move *north*, *east*, *south*, *west*, and *wait*. Agents should choose the action that greedily reduces the Manhattan distance to its goal point. Once the agent determines its next action, and if this action is not *wait*, it first uses communication to check whether the waypoint is occupied by any other agent. If it is occupied, the agent executes *wait* and continues using communication to check the availability of the waypoint (Alg. 11 Line 26-27). If there is no other agent occupying this waypoint, the agent then starts to check if any other agent wants to move to the same waypoint as it does. If there are multiple robots intending to go to the same waypoint (x, y) at the same time t, then the robot a_i whose current position $p_{ai}(t)$ is the lexically largest will go first (Alg. 11 Line 28-29).

As the agent moves towards its goal, it continually tries to improve its goal assignment, changing its goal based on local information. When it senses a neighbor with whom a swapped goal would result in a reduced pairwise traveled distance (in terms of Manhattan distance), it swaps goals with that neighbor (Alg. 13 Line 11-15). If swapping goals with a neighbor does not change the pairwise distance traveled, they swap goals with a probability $0 < \beta < 1$ (Alg. 13 Line 16-21). When a goal conflict is sensed, i.e. a neighbor is seen that is holding the same goal point, one of the agents picks a new goal from Q to eliminate the duplicated goal (Alg. 13 Line 3-10). An illustration of the cases in which an agent may change its goal are shown in Fig. 8.3.

It is possible that multiple agents (more than two agents) intend to swap the goals at the same time, for example, at time t, a_i intends to swap the goal with a_j while a_j intends to swap the goal with the third agent a_k. In my implementation, the pairwise goal swap is achieved by 2-way handshake (Alg. 13 Line 12, 18). Only the pair of who successfully handshake with each other can swap the goal. To be specific, when agent a_i intends to swap

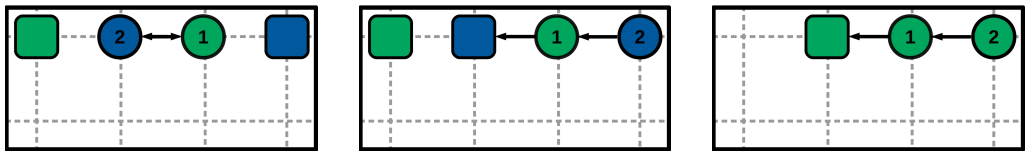

Figure 8.3. Illustration of possibles cases where an agent may change its goal. All the information is encoded in the same way as Fig. 8.2. (Left) For any pair of agents located within each other's communication range, if goal swap can help them to reduce the pairwise total distance traveled (in term of Manhattan distance), then goal swap occurs. (Middle) If the goal swap doesn't affect the total pairwise travel distance, these two agents randomly decide whether to swap. (Right) If both agents hold the same goal, one of them will run the *new goal selector* algorithm to select a new goal from Q.

the goal with a_j, if $p_{a_j}(t) \succ p_{a_i}(t)$, then it will take the role of *client* in this handshake, otherwise if $p_{a_i}(t) \succ p_{a_j}(t)$, a_i will act as *server* in the handshake. A *client* agent a_i will send a handshake request to its intended goal swap peer a_j, which is a handshake *server* since $p_{a_j}(t) \succ p_{a_i}(t)$, and then wait for the acknowledgement (ACK) from this *server* agent a_j for certain amount of time. On the other hand, a *server* agent a_j will wait for the handshake request from its intended goal swap peer a_i for certain amount of time, which is a *client* agent since $p_{a_j}(t) \succ p_{a_i}(t)$, and send back an ACK to a_i after receiving the handshake request from a_i. Note that it is possible that a *server* agent a_j receives the handshake requests from other agents that is not its intended goal swap peer. When this happens, it will answer the requests from these agents with a negative acknowledgement (NACK) (or does not answer these requests at all so as to trigger the handshake timeout). The *client* agent a_i will update its goal only after receiving the ACK from the intended goal peer a_j, and the *server* agent a_j will update its goal after receiving the handshake request from its intended goal swap peer a_i.

Remark: Beyond the situations where the agents' goals are interchangeable, it is also possible to extend the motion planner presented in the section to the situations where some

agents cannot swap their goals with the others, for example, the shape formation tasks where the agents hold different roles [33]. To do so, the main challenge is how to implement the primitive *swap* when the agents cannot swap their goals directly. Here, the idea is that: instead of swapping the goals, the agents will swap their positions. It was shown in [88] that the position swapping between two agents can be achieved by a sequence of intermediates moves without affecting the others' positions, making it possible to extend the presented motion planner to more complex situations.

8.3.2. New goal selecting

As we want each agent to have a unique goal in the end, the algorithm needs to eliminate any duplication in the assigned goals. For the *new goal selector* algorithm, It is desired that the swarm has as many different goals assigned as possible. When the total number of assigned goals is equal to the size of the swarm, no pair of agents will have the same goal. This implies that the *new goal selector* should keep the number of assigned goals growing. Therefore, every time a new goal is selected, the total number of assigned goal points should be non-decreasing.

8.3.2.1. Random selector. A simple *new goal selector* would be a random selector. For every pair of agents a_i, a_j that detect that they hold the same goal, if $p_{a_j}(t) \succ p_{a_i}(t)$, where $\succ$ denotes the lexical order on $\mathbb{R}^2$ space, then a_i randomly picks a new goal from the set Q.

While the random selector can guarantee the process to be almost surely convergent, the probability of picking an unassigned goal will decay over the number of assigned goals, leading to a relatively long convergence time. I therefore introduce a heuristic to speed convergence.

8.3.2.2. Gradient-based selector. The gradient algorithm, which is a well-known collective behavior, also known as hop-count algorithm [48, 49], can be adapted to improve goal selection. It is a simple algorithm that involves two agent roles, the common agent and the anchoring agent, and both roles transmit a single message containing a position q_u and a hop-count h. Each common agent listens to messages from its neighbors in communication range R, finds the message with the lowest hop-count received, (q_u^x, q_u^y, hop), and then transmits the message $(q_u^x, q_u^y, hop + 1)$ (Alg. 11 Line 17-19).

The basic hop-count algorithm from [48, 49] can be modified in the following way to allow for better goal selection. An agent will take on the anchoring role when it is one grid length away from an unassigned goal point (q_u^x, q_u^y) (by "unassigned goal" I mean the goal that is not assigned to any of the agent's neighbors in communication range), and transmit the message $(q_u^x, q_u^y, 0)$ (Alg. 11 Line 20-24). An anchoring agent will become a common agent when it no longer detects a unassigned goal that is one grid length away. Every agent a_i

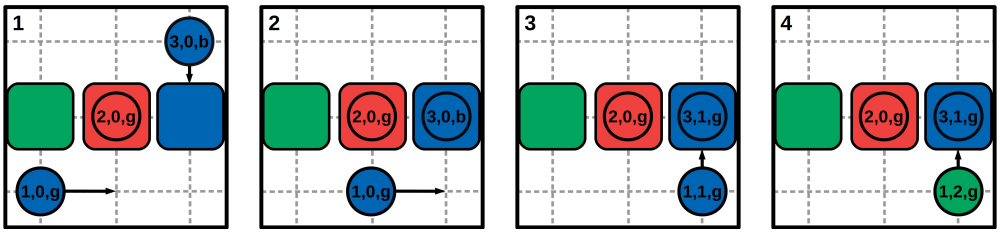

Figure 8.4. An example of agents using gradient-based selector to update their goals. Goal positions are shown as a colored square, agent positions are shown as a circle who's color matches its current goal. Agents are labeled with its index i, hop-count value hop, and candidate goal q_u color (r-red, b-blue, g-green), respectively. For this example, it is assumed that each agent's communication range is one grid length. Initially, in ***frame1***, the goal, T, for agent a_1 and a_3 is blue, and a_2 is red. In ***frame 2*** agent a_1 and a_3 move towards their goal, with a_3 arriving at its goal. In ***frame 3*** the hop-count is updated for agents and a_1 continues towards its current goal. In ***frame 4*** agent a_1 sees a_3 with the same goal, and since $a_3 \succ a_1$, a_1 changes goals, choosing the goal indicated by the hopcount message.

keeps the latest goal point (q_u^x, q_u^y) it transmitted as the candidate goal q_u. When a_i detects that there is another agent a_j holding the same goal and $p_{a_j}(t) \succ p_{a_i}(t)$, it then uses the current candidate goal q_u to update its goal $T_{a_i}(t)$ (Alg. 13 Line 3-10).

This gradient-based selector helps to prevent an agent from selecting a goal that is already occupied, while also propagating information about valid goals throughout the entire swarm. This helps increase the possibility (compared to random goal selector) that the new goal generated from "new goal selector" is valid, i.e. the goal has not been occupied by other agents yet. See Fig. 8.4 for a graphical illustration.

In addition, the swarm can also use the gradient hop-count to detect whether the shape is completed. If the shape is completely formed, there will be no anchor nodes in the swarm, so the gradient value of each agent will increase temporally. If any agent holds a gradient value larger than the number of agents, then the agent knows there are no anchor nodes, and therefore the shape is completed. Once any agent detects that the shape is complete, it will send a message which propagates across the entire swarm telling every other agent that the shape is complete.

8.3.3. Implementation

In this section, I describe the implementation of shape formation algorithm using gradient-based selector in detail. The algorithm consists of three components: main component, broadcast component, and goal manager. The main component coordinates agent's motion based on the information coming from its neighbors in communication range so as to avoid collision; the broadcast component constantly transmits messages to neighbors at a fixed frequency f_{comm}, and the neighbors in communication range will use this information to

coordinate their traffic; the goal manager refines agent's assigned goal so as to eliminate the duplicated assignment and resolve the deadlocks. These three components can be implemented using three separate threads running on each agent that communicate through shared memory. The sketches of these three modules are shown in Alg. 11, Alg. 12, and Alg. 13. Note that all the variables are thread-public.

8.3.3.1. Main Component. In main component, the agent has two tasks: use the messages from its neighbor to plan its motion (Alg. 11 Line 25-29), and perform the gradient algorithm so as to help to propagate the information about unassigned goal through the swarm (Alg. 11 Line 15-24). The variables and system calls that are used in this component are:

- *hop:* Agent's current hop-count value;
- q_u: Agent's candidate goal;
- T: Agent's current goal, i.e., the goal that the agent is moving towards;
- *wp*: The waypoint that agent is claiming, i.e., the waypoint that agent is currently moving to or staying at;
- p: The agent's position;
- *next_step:* Agent's next waypoint;
- Δt: The amount of time such that: If the agent tries to plan its motion at time t, then it will use all the messages received between $t - \Delta t$ and t to do the calculation;
- *clock():* The system call that returns the time elapsed since the program started;
- *last_check:* The variable to record the time when the agent arrived at current waypoint;
- *surroundings:* The set of four waypoints that are one grid length away from current waypoint;
- *msg_buff:* The set of messages that are received in the last Δt amount of time;

Algorithm 11: Main Component

Input: $Q = \{q_1, q_2, ..., q_m\}$, β.

/* Q is the set of goal points, β is a constant where $0 < \beta < 1$. */

$wp \leftarrow$ current waypoint
$hop \leftarrow \infty$
$q_u \leftarrow$ random element in Q
$T \leftarrow$ random element in Q
$next\_step \leftarrow$ current waypoint
$\Delta t \leftarrow \frac{2}{f_{comm}}$
$last\_check \leftarrow clock()$
while *True* **do**
  $surroundings \leftarrow \{(wp^x - l, wp^y), (wp^x, wp^y - l), (wp^x + l, wp^y), (wp^x, wp^y + l)\}$ /* the set of waypoints one grid length away from current waypoint. */
  $wait\_flag \leftarrow 0$
  for *i in surroundings* **do** /* find the next waypoint */
    if *choice of i reduces distance to goal* **then**
      $next\_step \leftarrow$ i
      Break
  $msg\_buff \leftarrow$ all messages received since $clock()$ - Δt
  if *msg_buff is* ***not*** *empty* **then**
    /* find the message that contains the lowest hop-count value. */
    $msg\_min \leftarrow$ the message in msg_buff that contains the lowest $message.hop$
    $hop \leftarrow 1 + msg\_min.hop$
    $q_u \leftarrow msg\_min.q_u$
    for *i in surroundings* **do** /* check if there is any unassigned goal one grid length away */
      if $i \in Q$ ***and*** $\forall\ msg_j$ *in msg_buff,* $msg_j.T \neq i$ **then**
        $q_u \leftarrow i$
        $hop \leftarrow 0$
        Break
    for *i in msg_buff* **do** /* loop through all the messages in msg_buff to check if there is any potential collision */
      if $i.wp == next\_step$ **then**
        $wait\_flag \leftarrow 1$ /* next_step is occupied */
      if $i.next\_step == next\_step$ ***and*** $i.p \succ p$ **then**
        $wait\_flag \leftarrow 1$ /* other agent with higher priority intends to go to the same waypoint */
  if $wait\_flag == 0$ ***and*** $clock()$ - $last\_check > \Delta t$ **then** /* there is no potential collision and the agent has stay at the current waypoint long enough */
    $wp \leftarrow next\_step$
    agent moves to wp
    $last\_check \leftarrow clock()$
  else
    stay at current waypoint

Algorithm 12: Broadcast Component

```
1  while True do
       /* Forge the message to be transmitted, the message contains: agent's current position, current
        waypoint, next waypoint, current goal, candidate goal, and hop-count value.                    */
2      msg←{p, wp, next_step, T, q_u, hop}
3      transmit msg
4      sleep 1/f_comm
```

Algorithm 13: Goal Manager

```
1  while True do
       /* a_i denotes the agent that is executing this thread                                          */
2      if receive a msg_in from any other agent a_j then
3          if a_j holds the same goal then
4              if p_{a_j} ≻ p_{a_i} then
5                  if rand(0, 1.0) > 0.1 then
6                      T ← q_u
7                      last_check ← clock() /* The goal changes, as a result, Alg. 11 Line 11-13 may change next_step,
                        hence we need to reset the timer for safety checking so as to avoid collision  */
8                  else
9                      T ← random element in Q
10                     last_check ← clock() /* Same reason as Alg. 13 Line 7.                          */
11         if the goal swap with a_j can reduce cost then
12             Execute the 2-way handshake with a_j
13             if 2-way handshake succeeds then
14                 updates agent's goal
15                 last_check ← clock() /* Same reason as Alg. 13 Line 7.                              */
16         if the goal swap with a_j doesn't effect cost then
17             if rand(0, 1.0) < β then
18                 Execute the 2-way handshake with a_j
19                 if 2-way handshake succeeds then
20                     updates agent's goal
21                     last_check ← clock() /* Same reason as Alg. 13 Line 7.                          */
```

- *wait_flag:* The flag variable that helps agent to check the potential collisions, specifically, if *wait_flag* is 0, then agent can move to the next waypoint; otherwise if *wait_flag* is 1, then the agent needs to stay at the current waypoint.

Recall that the agents' clocks could be asynchronized in phase. In order to avoid collisions, the agent is enforced to move in a "listen-think-walk" manner, namely, before moving from one waypoint wp_a to the other waypoint wp_b, the agent will wait at wp_a long enough, more

than Δt amount of time to be specific, so as to collect the neighbor's information and broadcast its information to the neighbors. Moreover, when the agent waits at waypoint wp_a, it keeps using the messages received in last Δt amount of time to determine whether it is safe to move to the wp_b. The collision-free guarantees of this traffic scheduling strategy is shown in Section 8.4.1.

8.3.3.2. Goal Manager. In goal manager component, when agent receives a message from its neighbor, the agent will first check that if this neighbor are holding the same goal point as it does, if so, then the one whose current position is lexcially smaller will change its goal (Alg. 13 Line 3-10). After this, the agent will then check whether the conditions (Alg. 13 Line 11, Line 16) for goal swap are triggered, if so, then it tries to execute the *2-way handshake* with the intended agent, and if the *2-way handshake* succeeds, the agents then updates their goals accordingly.

Note that this thread is executed concurrently with the main component (Alg. 11), as a result, when agent's goal changes in this thread (Alg. 13 Line 6, 9, 14, 20), the Alg. 11 (Line 11-14) may change the agent's *next_step*, therefore, in order to avoid the physical collision that is incurred by the concurrency, right after changing the goal in Alg. 13, the agent will reset the timer for the safety checking (Alg. 13 Line 7, 10, 15, 21).

8.4. Theoretical Results

8.4.1. Safety

In this section, I show that if the assumptions proposed in Section 8.2.1 are satisfied, then the presented algorithm is safe, i.e., the algorithm is collision-free.

Recall that as shown in Section 8.3.1, to provide collision-free guarantee, the implementation of motion planner should satisfy the Constraint 8.1 and Constraint 8.2. First, I show that my implementation satisfies Constraint 8.1.

Lemma 8.1. *Let $wp_{a_i}(t)$ be the waypoint that agent a_i is claiming at time t. If agent a_i changes the wp_{a_i} from $wp^0_{a_i}$ to $wp^1_{a_i}$ at time t^*, then there is no agent a_j such that $wp_{a_j}(t^*) = wp^1_{a_i}$.*

Proof. See Appendix 9. □

Theorem 8.1. *At time $t = 0$, if each agent starts with a unique waypoint, then for any time $t > 0$, Constraint 8.1 will be satisfied.*

Proof. If each agent starts with a unique waypoint, then Lemma 8.1 suffices to show that Theorem 8.1 holds. □

Next, I show that my implementation satisfied the Constraint 8.2 as well.

Theorem 8.2. *At time $t = 0$, if each agent starts with a unique waypoint, then for any time $t > 0$, Constraint 8.2 will be satisfied.*

Proof. See Appendix 10. □

8.4.2. Almost sure convergence

In this section, I show that if the *new goal selector* can pick a valid new goal point with non-zero probability, then the algorithm can enable the swarm to successfully form the desired shape with probability 1, regardless of the swarm's initial configuration.

To prove the convergence of the algorithm, for every time step t, I construct the following objective functions:

$$J_1(t) = \sum_{i=1}^{|A|} d_i(t)$$

$$J_2(t) = |A| - \sum_{i=1}^{|Q|} e_i(t)$$

In which, $d_i(t)$ is the Manhattan distance from agent a_i's current position to its current goal at time t, i.e., the number of edges to be traversed in the grid. Moreover, for each goal position q_i, I define $e_i(t)$ as follows:

$$e_i(t) = \begin{cases} 1, & \text{if at time t, } \exists \text{ j s.t. } T_{a_j}(t) = q_i \\ 0, & \text{otherwise} \end{cases}$$

One can see that these two objective functions will both equal 0 only if all agents successfully arrive at a unique goal. Therefore, it is sufficient to show that the presented method can always drive the swarm to the desired final configuration by proving the presented method can make both J_1 and J_2 converge to 0, regardless of the initialization.

Proposition 8.1. *Let $Pr\{\cdot\}$ be the probability that event $\cdot$ will occur, for any function $J(t) \in \mathbb{Z}^{\geq 0}$, it will almost surely converge to 0 if:*

- $\exists C \in \mathbb{Z}^{\geq 0}$, *s.t.* $\forall t$, $J(t) \leq C$
- $\forall\ t_1 \leq t_2$, $J(t_1) \geq J(t_2)$
- $\exists \tau, \epsilon > 0$, *s.t.* $\forall t$, $Pr\{J(t+\tau) \leq J(t) - 1 | J(t) \neq 0\} \geq \epsilon$

Proof. See Appendix 8. □

Next, to prove that both J_1 and J_2 can almost surely converge to 0, I show that both these two functions satisfy all three conditions proposed in Proposition 8.1.

Lemma 8.2. *Both J_1 and J_2 are bounded by a finite constant.*

PROOF. See Appendix 11. □

Lemma 8.3. *J_2 is monotonically decreasing, moreover, J_1 is monotonically decreasing if J_2 equals 0.*

PROOF. See Appendix 12. □

To describe swarm's traffic condition, at every time step t, I construct a directed graph $\mathcal{G}_t = (\mathcal{V}, \mathcal{E}_t)$, in which $\mathcal{V} = \mathrm{A}$ and $\mathcal{E}_t = \{(v_i, v_j)\}$ as its edge set, where $(v_i, v_j) \in \mathcal{E}_t$ if a_j occupies a_i's next waypoint at time t. By definition, each vertex on $\mathcal{G}_t$ essentially characterizes an agent, therefore, in the rest of the section I use the notation a_i and v_i interchangeably. For the sake of the description, at time t, I call agent a_j is agent a_i's successor, or a_i is a_j's predecessor, if $(v_i, v_j) \in \mathcal{E}_t$. Additionally, I call those agent(s) a_i whose out degree $deg^+(a_i) = 0$ the *head* agent(s), in the other words, a *head* agent is an agent that is not blocked by any other agent.

Lemma 8.4. *If $(v_i, v_j) \in \mathcal{E}_t$, then a goal swap between a_i and a_j will happen with a non-zero probability.*

PROOF. See Appendix 13. □

Lemma 8.5. *If $J_2 = 0$, then if $J_1 \neq 0$, J_1 will decrease by at least 1 within a finite amount of time with non-zero probability, independent of the history.*

PROOF. See Appendix 14 □

Lemma 8.6. *The motion planner will make the position of each assigned goal's owner to greedily move toward the position of the goal point with non-zero probability, regardless of the swarm's configuration.*

PROOF. See Appendix 15. □

Lemma 8.7. *If $J_2 \neq 0$, then the event that: two agents that are holding the same goal at the same time are located within distance R, will occur within a finite amount of time with non-zero probability. Namely, at time $\bar{t}$, if $J_2 \neq 0$, then $\exists\ \overline{\tau}, \overline{\epsilon} > 0, a_i \neq a_j$, such that:*

$$Pr\{||p_{a_i}(\bar{t}+\overline{\tau}) - p_{a_j}(\bar{t}+\overline{\tau})|| \leq R, T_{ai}(\bar{t}+\overline{\tau}) = T_{aj}(\bar{t}+\overline{\tau})\} \geq \overline{\epsilon}$$

PROOF. See Appendix 16. □

Lemma 8.8. *If $J_2 \neq 0$, then J_2 will decrease by at least one within a finite amount of time with non-zero probability, independent of the history.*

PROOF. See Appendix 17. □

Theorem 8.3. *Both J_1 and J_2 will almost surely converge to 0, regardless of the swarm's initial configuration.*

PROOF. Using Lemma 8.2, Lemma 8.3, Lemma 8.5, and Lemma 8.8,on can see that both J_1 and J_2 satisfy all three conditions proposed in Proposition 8.1. Hence both J_1 and J_2 will almost surely converge to 0, regardless of the initialization of the swarm. □

8.4.3. Complexity

In the section, I study the cost of implementation of the algorithm proposed in Section 8.3.3 with respect to its time complexity, memory complexity, and communication complexity.

First, I study the time complexity for each agent planning their action, i.e., the time complexity for executing Alg. 11 Line 9-35. One can see that the time cost is dominated by the time complexity for looping through all the messages received in last $\frac{2}{f_{comm}}$ amount of time. In the *msg_buff*, there will be at most $2|N_{a_i}(t)|$ amount of the messages, on the other hand, each agent can have at most $\lfloor\frac{2R}{l}\rfloor^2$ amount of neighbors in communication range, where R is agent's communication range, l is the grid length. This suffices to show that the time complexity for the decision making is $\mathcal{O}(\lfloor\frac{R}{l}\rfloor^2)$.

Next, the algorithm's memory footprint is dominated by the memory to store the input point set, as a result, the algorithm's memory complexity is $\mathcal{O}(|Q|)$.

To investigate the algorithm's communication complexity, i.e., the amount of data each agent will transmit during one unit of time, I first study the amount of data that each agent will transmit within each communication round, i.e., $\frac{1}{f_{comm}}$ amount of time. During each communication round, an agent a_i will broadcast one message with a length of $\mathcal{O}(1)$ (Alg. 12 Line 2), and process at most $|N_{a_i}(t)|$ *2-way handshake*s, as a_i can receives no more than $|N_{a_i}(t)|$ amount of messages from the neighbors in communication range. Additionally, the amount of data exchanged to process a *2-way handshake* is $\mathcal{O}(1)$, as a result, during one communication round, the amount of data that each agent a_i will transmit to its neighbors is $\mathcal{O}(\lfloor\frac{R}{l}\rfloor^2)$, as each agent can receive no more than $\lfloor\frac{2R}{l}\rfloor^2$ neighbors in communication range, where R is agent's communication range, l is the grid length. On the other hand, in a unit

of time, there will be $\mathcal{O}(f_{comm})$ communication rounds, which suggests that for each agent, the amount of data transmitted during a unit of time is $\mathcal{O}(\lfloor \frac{R}{l} \rfloor^2 f_{comm})$.

8.5. Performance Evaluation

To demonstrate the correctness and performance of the algorithm presented in this chapter, the algorithm is implemented and tested in both simulated and physical experiments. For all experimental tests, the shape was successfully formed. I also compared the performance of the proposed algorithm with the centralized algorithm proposed in [113]. In this centralized approach, every agent is initially assigned a unique goal. In addition, this assignment minimizes the total traveling distance. It is shown in [113] that with such optimal initial assignment, the agents' paths will form an acyclic direct graph (DAG), and each agent's motion can be then scheduled via vertex ordering. While the centralized method can produce the distance-optimal solution, which outperforms the presented method, it could suffer from a single-point of failure and therefore is less robust than the presented method.

8.5.1. Simulation

In simulation, each agent is modeled as an omni-directional robot able to sense its position and orientation in a common coordinate system. The simulation treats each agent as a circle with a radius of $0.05m$. Agents are able to communicate with any other agent who lies within its communication range of $0.6m$, and travel at a speed of $0.05m/s$. These values match the physical robot described in chapter 3.

In each test, the goal shape is given to the swarm in the form of a binary figure, i.e. a black and white image. The figure is scaled such that the number of goal pixels on the figure

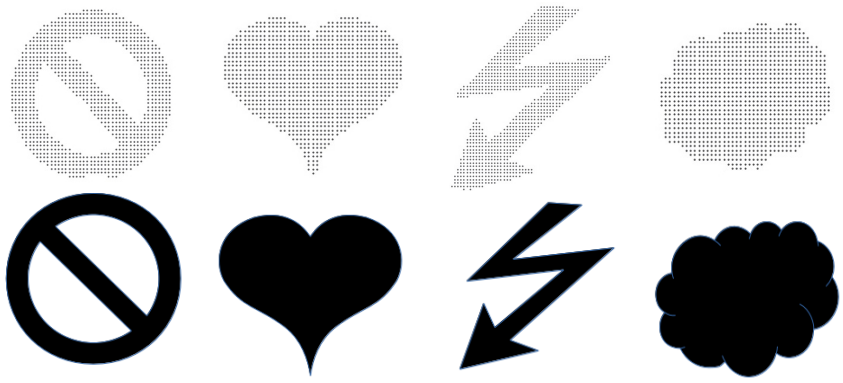

Figure 8.5. 1024 agents form four user-defined shapes. (Bottom) example input binary images and (top) corresponding collective formations.

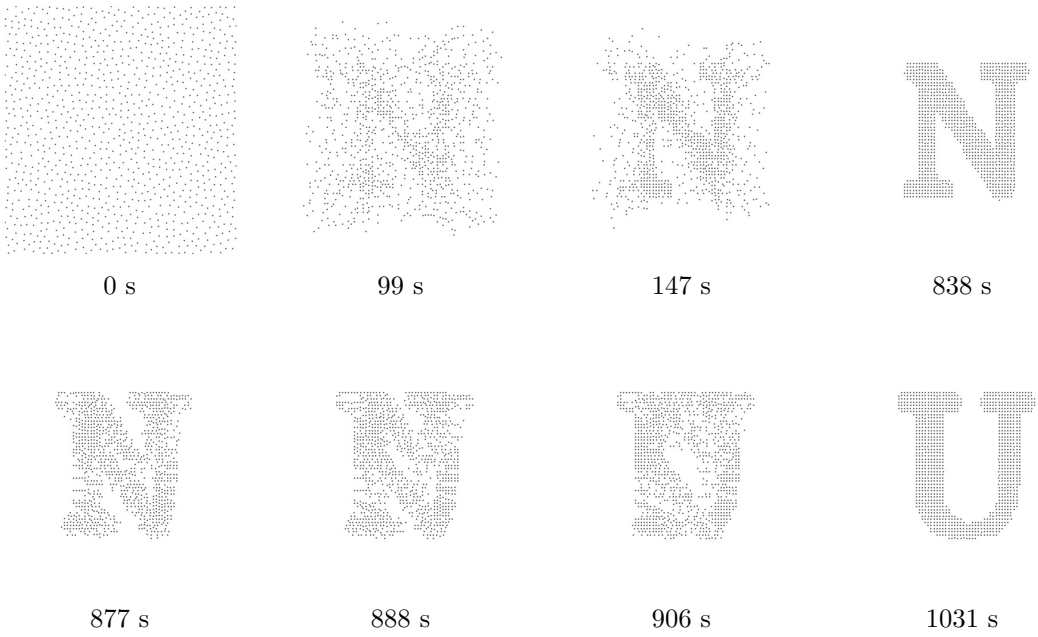

Figure 8.6. Still images from simulation where 1024 agents try to form two different shapes in a row. In this simulation, a swarm of 1024 agents first form a letter "N", then switch to a letter "U" when it is detected that all robots have reached a goal.

equals the number of agents. Example input images and corresponding shapes formed by the collective are shown in Fig. 8.5. See Fig. 8.6 for images from one simulation where 1024 agents formed the letters "N" and "U" in sequence.

First, the simulation is used to investigate the effect of the swarm size on the algorithm's convergence time, i.e. the total time it takes for the swarm to complete shape formation, as well as average robot travel distance, i.e., the total distance traveled by all robots normalized by the number of robots. In this task, swarms of size 16 to 529 agents formed a given target configuration from a random initialization. For every swarm size, 200 trials were run, and in each trial the target shape is randomly generated as a set of connected random positions. The large number of trials are with varying swarm sizes able to eliminate the bias on the final result that is introduced by the target shape, since in each trial the target shape is randomly generated. Fig. 8.7 shows the distributions of convergence time as well as the average distance traveled for different swarm sizes, and a comparison to the centralized method [113].

One counter-intuitive observation here is that the convergence time for centralized methods does not monotonically increase over the swarm size, sometimes even goes down. It is shown in [113] that the worst case convergence time for the centralized method is $|A|+d_{max}-1$ where $|A|$ is the number of agents, and d_{max} is the maximal individual travel distance (the distance from one agent's initial position to its goal) amongst swarm. On the other hand, in simulations, the swarm moved in a fixed-size arena, hence when swarm size $|A|$ increases, i.e., the density of the swarm increases, the d_{max} will decrease. As a result, the convergence time for centralized methods will not necessarily increase over the swarm size. The fact that agents move in a fixed-size arena can also help to explain the trend of the total distance plots. As $|A|$ increases, the swarm's initial position will be "closer" to the target positions (one can consider an extreme case where number of the agents equals to the number of the vertices in arena, in this case, the average distance traveled will be 0), hence the overall trend of the distance plot is that the average distance traveled goes down as number of agents goes up. In

these plots, we can see that the distance traveled incurred by the presented method is only around 20% more than the one that is incurred by the centralized method. Moreover, when the density of the swarm is low (less than 225 agents in arena), the difference on convergence times for both methods are considerably small. When the density of the swarm increase, the convergence time for the presented method sharply increases, this is because in this case, the random goal swaps, i.e., the goal swaps to resolve the deadlock, will more likely happen, as a result, the algorithm will converge slower.

A second test compares the two approaches for *new goal selector*. It measures the convergence rate as well as the total distance traveled by a fixed-size swarm. In this experiment, 400 agents try to form 200 different randomly generated shapes. For each shape, the swarm executes the formation algorithm 200 times, giving a total of 40,000 simulation runs. For each of these runs, agents are initialized with a uniform random distribution centered at the shape's center of mass. For every time step, I measure the average completion rate, the average distance traveled, and the confidence interval at a confidence level of 2σ for both convergence and distance travel, at that time for all 40,000 runs. The results are shown in Fig. 8.8. In these plots, we can see that the gradient-based selector can dramatically increase the algorithm's convergence rate and helps to eliminate the long tail of convergence incurred by the random goal selector. Besides that, unsurprisingly, the gradient-based goal selector can also reduce the variance of the convergence time and total distance traveled.

Simulation was also used to compare the convergence rate as well as swarm's total traveled distance for both the presented method and the centralized one. The experiment contains 40,000 trials; in every trial, 400 agents tried to drive toward a set of randomly generated goal points. First, agents were initialized with random starting locations. Next, either the presented method or the centralized method was used to drive the agents to the goal points.

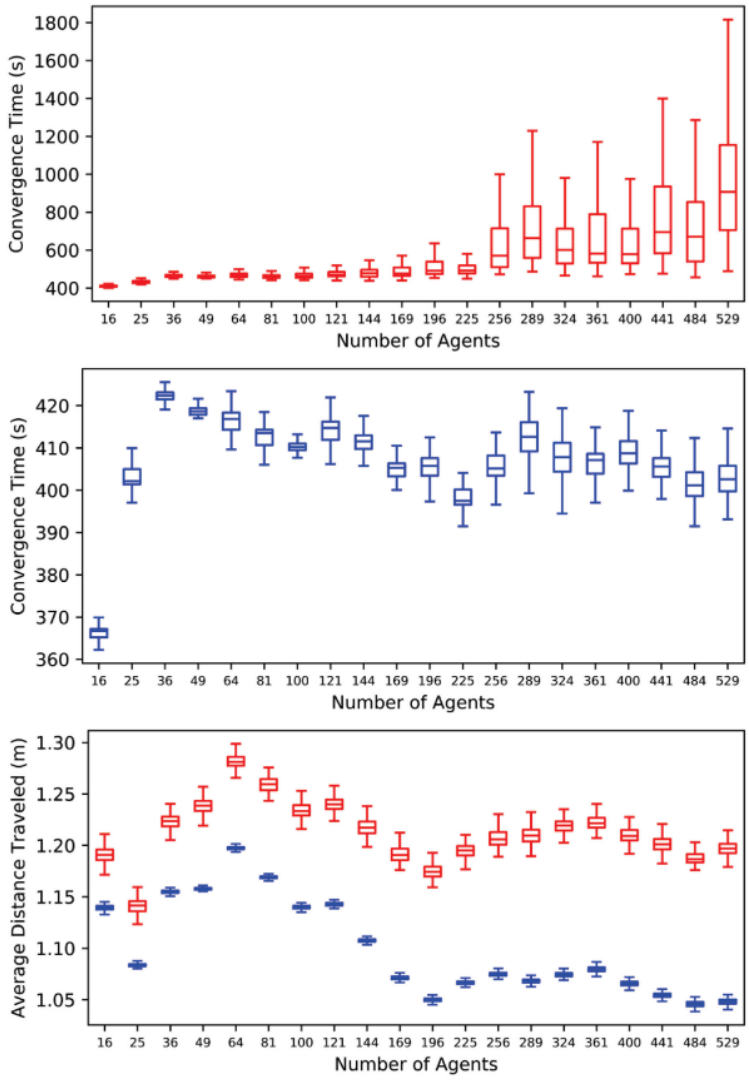

Figure 8.7. Simulation results for both the presented method (red) and the centralized method [113] (blue) in standard box-plot format. For each number of agents, 200 trials were run, and in each trial the target shape was randomly generated.

The simulation results of the presented method and the centralized method are shown in the Fig. 8.9. The difference between Fig. 8.9 and Fig. 8.7 is that: Fig. 8.7 shows the statistics of the final solution's quality, i.e., the total distance and convergence time, whereas the Fig. 8.9 helps to understand how the algorithm's convergence and distance traveled change during

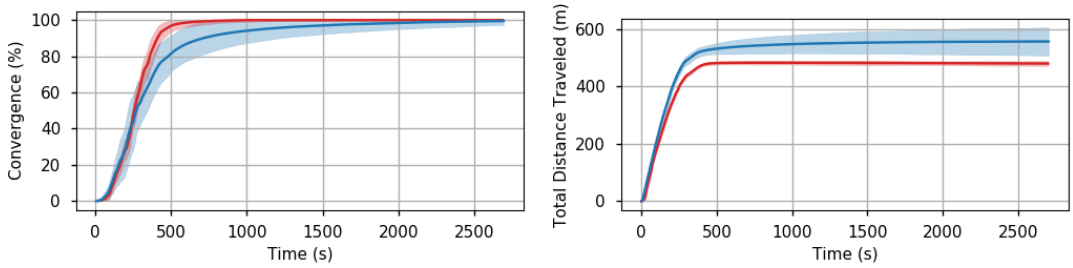

Figure 8.8. Illustration of the improvement made by the gradient-based selector on the algorithm convergence rate and total distance traveled compared to using a random selector. The red plots are the results of gradient-based selector and the blue plots are the results of random selector. Each solid line in the plot is the average result from 40,000 simulations of 400 agents, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

execution. The plot shows average result and the confidence interval with a confidence level of 2σ for both the convergence rates and total distance traveled for both methods. Note that at the beginning, from time 0s to 80s, the presented algorithm makes faster progress than the centralized method. This is because in the centralized method, agents take goals located in the inner area of the shape first so they will not block other moving agents' paths, while in the presented method, agents initially choose goals at random. As a result, the agents are more likely to take the goal points nearby first, giving the presented method a short-term win at the beginning.

8.5.2. Experiments

To validate the correctness and efficiency of the presented algorithm beyond simulation, several physical experiments were performed using 100 *Coachbot V2.0* robots.

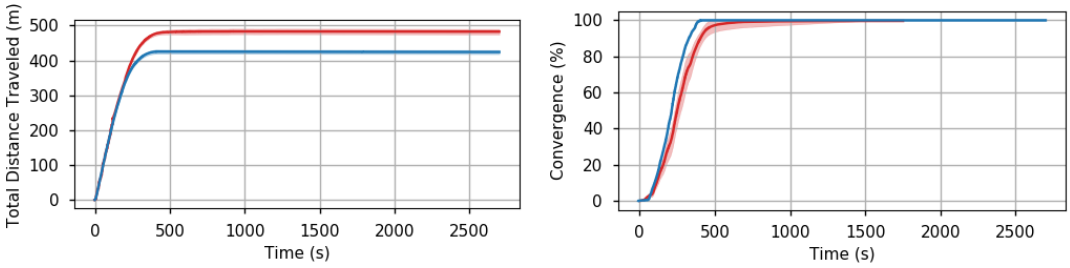

Figure 8.9. Performance comparison between the presented method (red line) and centralized method (blue line). Each solid line in the plot is the average result from 40,000 simulations of 400 agents, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

8.5.2.1. Dealing with real-world non-idealities. In reality, some assumptions proposed in Section 8.2.1 are difficult to be guaranteed in real robot hardware. To compensate the real-world non-idealities such as communication errors, imperfect robot motion, sensing errors, etc., the assumptions proposed in Section 8.2.1 are relaxed as follows:

(a) For any agent, the frequency of its clock is bounded, specifically: $\exists f_{clock}^{max}, f_{clock}^{min}$ s.t. for any agent a_i, we have $f_{clock}^{min} \leq f_{clock}^{i} \leq f_{clock}^{max}$.

(b) Note that for each agent a_i, its communication rate f_{comm}^{i} is defined according to its on-board clock, i.e. the clock that is with frequency f_{clock}^{i}. As a result, even though each agent is programmed to broadcast at the same frequency f_{comm} (Alg. 12 Line 4), from a global observer's perspective, their communication rate could be still different due to the difference on clock's frequency f_{clock}^{i}.

(c) The inter-agent communication packet loss rate is small enough such that: for each robot, if it sends the same message m times in a row, it is guaranteed that this message can be received by all its neighbors in communication range.

(d) For any agent a_i, the speed v_m^i at which it moves on a grid edge is bounded, namely: $\exists v_{max}, v_{min}$ s.t. for any agent a_i, we have $v_{min} \leq v_m^i \leq v_{max}$.

First, note that as shown in Section 8.4.1, the proof of Theorem 8.1 and Theorem 8.2 does not rely on any assumption about robot's physical speed, in the other words, only the relaxation of assumption (a), (b), (c) will effect the correctness of Theorem 8.1 and Theorem 8.2. To preserve the correctness of Theorem 8.1 and Theorem 8.2, in practice, I extend Δt to make the robot to act more conservatively, so as to compensate the difference on robots' clock frequency and packet loss. To be specific, I extend Δt such that:

$$\Delta t \geq \frac{2m}{f_{comm}} \frac{f_{clock}^{max}}{f_{clock}^{min}}.$$

Here, the first term $\frac{2m}{f_{comm}^{min}}$ is for accommodating the communication loss, and the second term $\frac{f_{clock}^{max}}{f_{clock}^{min}}$ is for compensating the difference on robots' clock frequency. The reason for the second term is that when one agent executes Alg. 11 Line 30 and Alg. 12 Line 4, it will use the on board clock to do the calculation, and different clock frequency will yield different results. Therefore, I add the second term to guarantee that for the robot with the fastest clock, it will wait long enough to accommodate the one with the slowest clock.

So far, it is showed that the correctness of Theorem 8.1 and Theorem 8.2 can be preserved when the assumptions proposed in Section 8.2.1 are relaxed. However, when agents move on grids in different speeds, collisions could still happen. To accommodate the difference on robot's physical speeds, I stretch the grid length l so as to give robots some "buffer space". Specifically, we want the l large enough such that:

- When two robots move on two orthogonal adjacent edges, no collision happens, that is:

$$\min \sqrt{(l - v_{max}t)^2 + (v_{min}t)^2} \geq 2r, \ \textit{subject to } t \in [0, \tfrac{l}{v_{max}}].$$

- When two robots move on two collinear adjacent edges, no collision happens, that is:

$$\min(l - v_{max}t + v_{min}t) \geq 2r, \ \textit{subject to } t \in [0, \tfrac{l}{v_{max}}].$$

Solving those two inequalities above, we have:

$$l \geq 2\frac{\sqrt{v_{max}^2+v_{min}^2}}{v_{min}}r.$$

Additionally, in the algorithm, there is the assumption that the robot can move in any direction directly, which does not hold for the *Coachbot V2.0*, as *Coachbot V2.0* is a differential drive robot. When a *Coachbot V2.0* moves from one waypoint wp_a to another waypoint wp_b, it will first spin at waypoint wp_a to adjust its orientation to be parallel with the grid edge connecting wp_a and wp_b, before moving to wp_b. Note that for each step, the robot may change its orientation by 0 *rads* or $\frac{\pi}{2}$ *rads* (because the robot can move both forwards and backwards, hence it does not need to adjust its orientation by more than $\frac{\pi}{2}$ *rads*). Different heading adjustments will take different amount of time, as a result, for those robots that transit to their next waypoints at the same time, the robots who need to spin by $\frac{\pi}{2}$ *rads* will start moving towards their next waypoint later than the ones that do not need to spin, so a collision may occur. To compensate this difference on the adjustments of the robot's heading, I introduce another type of "buffer space" to grid length l, that is to say, assume the robot's minimal spin speed is ω^*, I enforce the grid length l to be:

$$l \geq 2\frac{\sqrt{v_{max}^2+v_{min}^2}}{v_{min}}r + v_{max}\frac{\pi}{2\omega^*}.$$

In experiments, my choice of grid length l is $0.20m$.

8.5.2.2. Results. In these physical experiments, it is demonstrated that the presented algorithm can be easily implemented on a relatively large scale physical swarm, and it can provide reliable performance. Additionally it is robust to real-world noise in both communication, sensing, and motion. In this experiment, 100 robots start randomly dispersed and form the letters "N", "U" in sequence. With the help of hop-count information, robots can

detect when the first letter is completed and then switch to form the second shape, an "U". Images from one of these experiments using the presented algorithm is shown in Fig. 8.1.

I also compared the real-world performance between the presented algorithm and the centralized approach. A shape was formed 15 times with both approaches, and I compared the average convergence rate and average total distance traveled for both approaches. In all these 30 experiments, the shape formation successfully completed. The results from this comparison experiment are shown in Fig. 8.10.

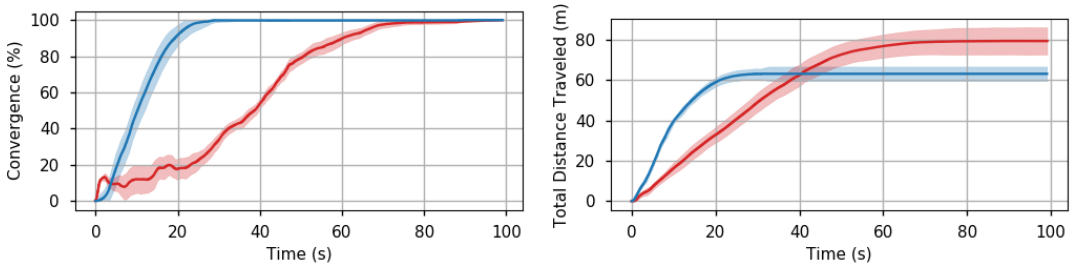

Figure 8.10. Illustration of average performance comparison between the presented method (red line) and the centralized method (blue line). Each solid line is the average result from 15 physical experiments of 100 robots, and the colored shade areas show the confidence interval for convergence and total distance traveled over time at a confidence level of 2σ (two standard deviations above or below the average).

In these plots, we can observe that the presented method gets a short-term win of convergence rate at the beginning compared to the centralized method, which is consistent with the simulation result. On the other hand, one observation here is that in the simulation plots (Fig. 8.9), both the convergence and the distance traveled monotonically increase over time, whereas in Fig. 8.10, during the first 20 seconds, the convergence plot fluctuates. This is because in simulation the agents are tasked to form a set of random shapes from a random initialization of positions whereas in all physical experiments the robots are tasked to form the same shape, letter "N", which will introduce the bias to the final result. Additionally,

noise in robot's motion, communication, and sensing, can not be captured by the simulation very well, and the number of trials are not large enough to eliminate the noise's effect on the convergence rate. As a result, the convergence plot for physical experiment is not monotonically increasing over time.

CHAPTER 9

INTEGRATING THE ALGORITHMS

This chapter assembles those four algorithm modules presented in chapter 5 - 8 into a full persistent shape formation algorithm.

9.1. Approach

In the proposed method, each robot's overall behavior can be described by the state diagram shown in Fig. 9.1.

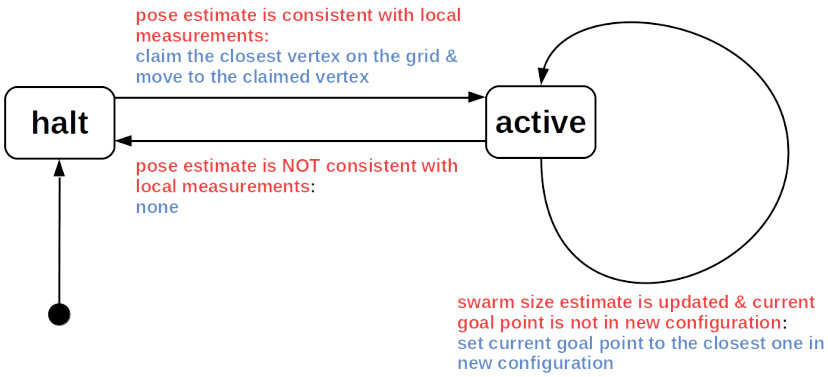

Figure 9.1. The state diagram describing the robot's overall behavior. The black box is the robot's state, the arrow line is the transition between states, the red text is the event triggering the state transition, and the blue text is the action performed by the robot when the transition occurs.

When executing the combined shape formation algorithm, each robot has two possible states: *active* and *halt*. In both states, the robot will use the algorithm presented in chapter 5 to monitor the swarm size, and use the algorithm presented in 6 to generate the goal configuration according to the current swarm size. The robot is initialized to be in the *halt*

state. In the *halt* state, the robot will stay stationary and use the algorithm presented in chapter 7 to estimate its pose. Meanwhile, it will keep checking if the current pose estimate is consistent with the local measurements and the neighbors' pose estimates. If so, the robot will transit to the *active* state. Recall that, in order to execute the formation control algorithm presented in the chapter 8, each robot needs to convert the work space to a discrete grid, and claim a vertex on the grid. When transiting from *halt* state to *active* state, the robot will discretize the work space to a grid in the way that: the grid vertex (i, j) has a position of $(i \times l, j \times l)$, where l is the grid's edge length. In addition, the robot will claim the grid vertex that is closest to its current position, and move to this claimed vertex.

When a robot is in the *active* state, it will estimate its pose according to the local measurements and the odometry using the algorithm presented in chapter 7, and use this pose estimate to execute the algorithm presented in chapter 8 so as to form the generated goal configuration. In addition, every time when the swarm size estimate is updated, the robot will update its goal configuration immediately, and check whether its current goal point is in the updated goal configuration. If the robot's current goal point is not in the new goal configuration, the robot will change its goal point to the closest goal point in the new goal configuration. Meanwhile, it will also keep checking if the current pose estimate is consistent with local measurements and the neighbors' pose estimates. Once the robot senses that its pose estimate is no longer consistent with local measurements and neighbors' pose estimates, it will transit back to the *halt* state.

As stated in chapter 8, for each *active* robot, in order to avoid colliding with the others, every time when it intends to move to the next waypoint, it needs to check if its next waypoint is currently claimed by any other robot. This safety checking requires each robot to actively tell its neighbors about the vertex currently claimed by itself. On the other hand,

it is possible that an *active* robot has a neighbor that is in *halt* state. It is obvious that the *halt* robots cannot participate the safety checking motioned above, as they do not have a vertex claimed yet, which might result in the physical collisions between *active* robots and *halt* robots. To handle the potential physical collisions introduced by the asynchrony of the robots' states, each robot needs to actively broadcast its current state to its neighbors, in addition, for the *active* robots, before moving to the next waypoint, besides checking those two conditions stated in Algorithm 11 Line 26 - 29, they also need to check if there is any *halt* robot nearby. If there is any neighbor currently being in the *halt* state, the robot needs to stay stationary so as to avoid colliding with the nearby *halt* robots.

9.2. Performance Evaluation

To investigate the combined algorithm's performance, the algorithm is implemented and test in both a swarm of up to 200 simulated robots, and a swarm of up to 100 physical robots.

9.2.1. Simulation

In the simulation, a swarm of up to 200 simulated *Coachbot V2.0* robots were tasked to use the proposed method to persistently form a shape "N". In the tests, the robots are placed in a 2D rectangle-shape arena. The communication rate is 25 hz, the maximal speed of robot's wheel is $0.1m/s$, and edge length of the grid is set to $0.3m$. In each test, three metrics are used to quantitatively analyze the swarm's behavior: NPEE, NOEE, and the NDS between the swarm's current configuration and desired configuration. The metric NPEE and NOEE are two metrics to evaluate the swarm's position estimate error and orientation estimate error, which are defined in chapter 7; the metric NDS is the metric to evaluate the difference

between two configurations, which is defined in chapter 6. When calculating the NDS, the swarm's configuration is given by the set of grid vertices claimed by the robots in the swarm, and the desired configuration is the configuration corresponding to the actual swarm size generated by the algorithm presented in chapter 6. In addition, recall that the *halt* robot does not have a vertex claimed yet, in the calculation of the metric NDS, we give each *halt* robot a dummy claimed vertex of (∞, ∞). The videos of simulation can be found in [76].

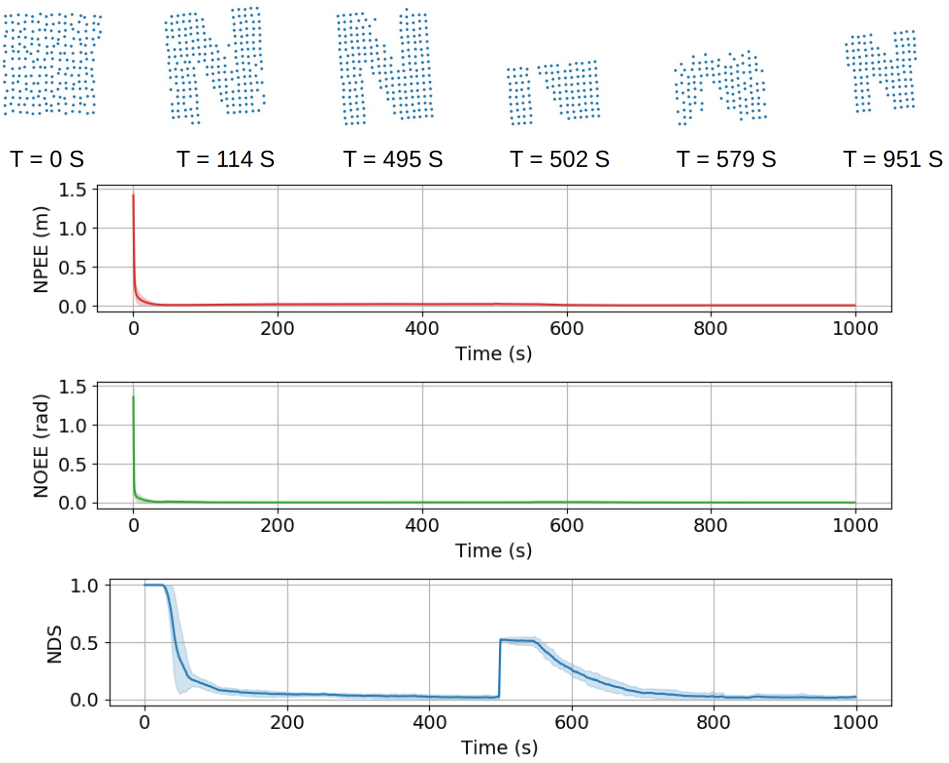

Figure 9.2. From top to bottom: still images from one trial of robot removal test; the NPEE, NOEE, and NDS for the swarm over time. Each colored solid line is the mean from 10 trials, and the colored shade areas show the confidence intervals for a confidence level of 2 σ.

The first test studies the algorithm's adaptability to the event of robot removal. The swarm size is initialized to be 200, then, at $t = 500s$, the robots that are located in the bottom half of the arena are removed from the swarm. As we can see in the simulation, the

proposed algorithm allows the robots to form the desired shape at the right scale, in addition, when the removal of robots happens, the remaining robots are able to sense the change in swarm size, and fix the damaged shape by forming the shape at a smaller scale. The results of this test are shown in Fig. 9.2. One can observe that the NDS sharply increases at $t = 500\ s$, this is because the robot removal that happens at $t = 500\ s$ will damage the formed shape, increasing the NDS. In addition, we can see that: from $t = 500\ s$ to $t = 550\ s$, the NDS stays almost the same, the NDS starts to decrease only after $t = 500\ s$. This is because: it will take some time for swarm to sense the change in the swarm size. Therefore, when the robot removal happens, there will be a delay between the occurrence of robot removal and the swarm's response.

The second test studies the swarm's adaptability to the addition of the robots. The swarm size is initialized to be 100, then, at $t = 500s$, another 100 robots are added to the swarm. As we can see in the simulation, the proposed methods allows the swarm to robustly adapt to the robot addition. The results from this test are shown in Fig. 9.3. Similar to the robot removal test, in this test, at $t = 500\ s$, the NDS sharply increases, as the robot addition will damage the formed shape. The other observation is that: at $t = 500\ s$, the NOEE and NPEE spike as well, this is because: the newly added robots are initialized with random pose estimate, which will corrupt the swarm's original coordinate system. In addition, this corruption of the coordinate system will bring robots into *halt* state. Recall that in the calculation of NDS, we give each *halt* robot a dummy claimed vertex of (∞, ∞). Therefore, since almost all the robots will transit to the *halt* state when the robot addition occurs, at $t = 500\ s$, the NDS goes all the way to 1.

A third test studies the swarm's adaptability to the shifting of the robots. In this test, the swarm size is set to 200. At $t = 500s$, the robots located in top half of the arena are

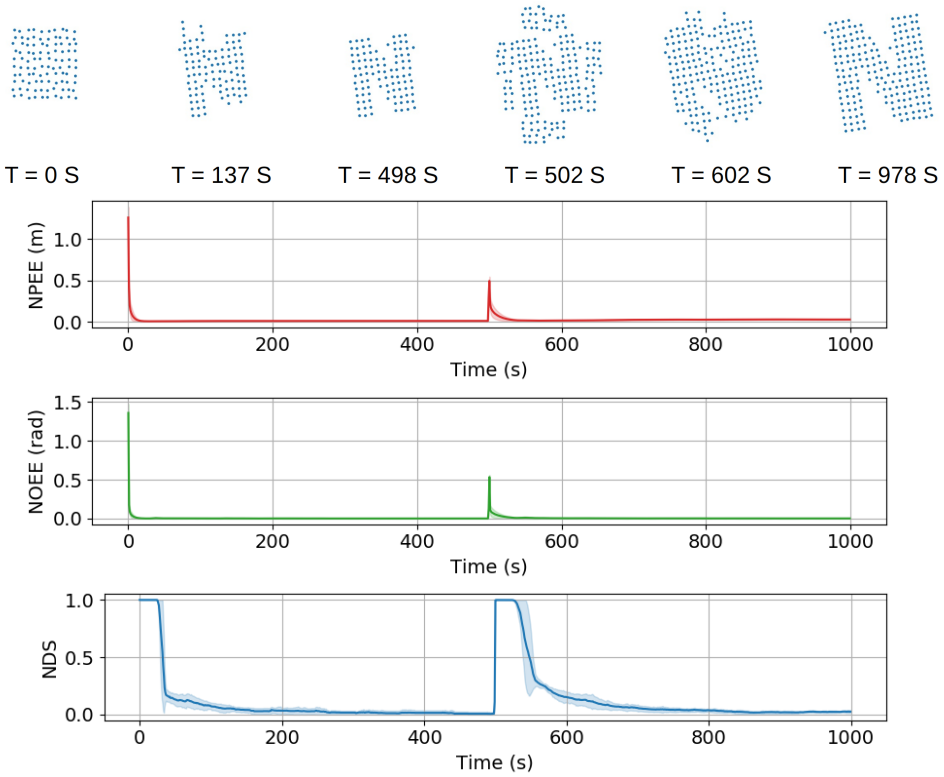

Figure 9.3. From top to bottom: still images from one trial of robot addition test; the NPEE, NOEE, and NDS for the swarm over time. Each colored solid line is the mean from 10 trials, and the colored shade areas show the confidence intervals for a confidence level of 2 σ.

shifted 0.3 m upwards, in addition, each shifted robot's orientation is rotated 0.3 rad counter clockwise. It is assumed that this external disturbance of robot's pose shifting cannot be captured by the robot's on-board odometry. As we can see in the simulation, the proposed method enables the swarm to reliably recover the shape from the event of robot shifting. The results from this test are shown in Fig. 9.4. Similar to the robot addition test, in this test, at $t = 500$ s, all three metrics spikes, as the shifting of robots will not only damaged the formed shape, but also corrupt the swarm's original coordinate system.

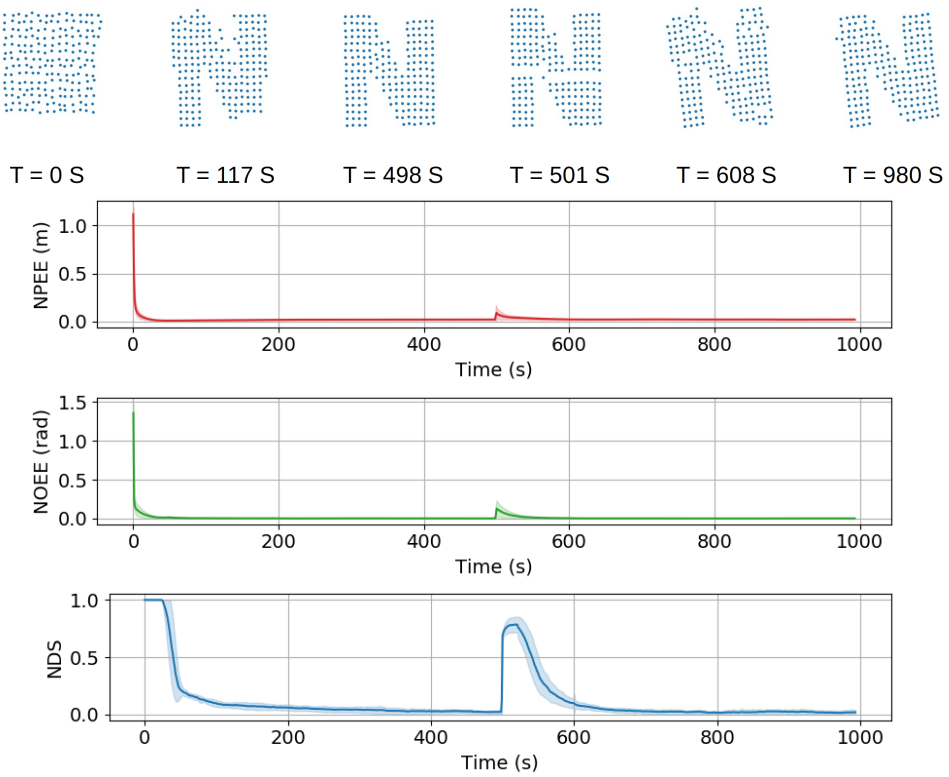

Figure 9.4. From top to bottom: still images from one trial of robot shifting test; the NPEE, NOEE, and NDS for the swarm over time. Each colored solid line is the mean from 10 trials, and the colored shade areas show the confidence intervals for a confidence level of 2 σ.

9.2.2. Experiment

To validate the combined algorithm beyond simulation, I also implement the algorithm on the swarm of up 100 real *Coachbot V2.0* robots. In this experiment, up to 100 robots are tasked to persistently form a shape "N". Initially, 70 robots were randomly dispersed in the arena. After those 70 robots formed the desired shape, I added another 30 robots to the swarm. Those newly added robots damaged the formed shape, the swarm recovered the desired shape by forming it at a bigger scale. Then, I shifted a group of robots' physical positions, and rotated each shifted robot's orientation 0.6 *rad* clockwise. Note that this change in robot's position and orientation will not be captured by the on-board odometry,

damaging both the swarm's coordinate system and the formed shape. With the proposed algorithm, the swarm recovered their coordinate system and recovered the desired shape. Lastly, I removed a group of robots from the swarm, the swarm recovered the desired shape by forming it at a smaller scale. The still images from this experiment can be found in Fig. 2.1, and the video of the experiment can be found in [76]. As we can see in the video, with the proposed algorithm, the swarm is able to reliably form the desired shape, in addition, it can also quickly adapt to the external disturbances.

CHAPTER 10

CONCLUSION AND FUTURE WORK

This dissertation addresses the decentralized robotic-swarm shape formation, a problem that has both scientific and practical importance. In this dissertation, I developed a collection of algorithms such that when combined, they allow the swarms with any size to persistently form arbitrary user-specified shapes without the use of any global information. This collection of algorithm includes: an algorithm that allows the swarm to use local information only to estimate the current swarm size, an algorithm that encodes an user-specified shape into arbitrary number of points, an algorithm localizing a swarm of robot using peer-to-peer measurements, and an algorithm that assigns a set of goal points to a swarm of robots and drives each robot to reach its assigned goal point. For each presented algorithm, both the theoretical analysis and the thoroughly experimental evaluation are provided, In addition, I further assembled these four algorithms into a fully decentralized persistent shape formation algorithm. The combined algorithm was examined using a custom efficient swarm simulator as well as a custom 100-robot swarm. The result shows that: the work presented in this dissertation allows the swarm to robustly form the desired shape, in addition, when the expected external disturbance occurs, such as robot addition or robot removal, the presented method enables the swarm to quickly adapt. I am confident that this dissertation positively contributes to the field of robotic swarm system research.

In this dissertation, I investigated the robotics shape formation in 2D space. For the future research, it might be worth trying to extend the method presented in this dissertation

to 3D space. In addition, the current path planning module requires the robot to represent the work space as a grid. Since the grid representation of the space makes the robots motion less efficient, in the future works, it would be beneficial to remove the requirement of grid representation of the space from the path planner. From the hardware's perspective, currently, the developed platform (*Coachswarm*) still requires users to manually charge the robots and initialize each robot's position, it will be desirable for the users if the operation of *Coachswarm* can be made more automated. Furthermore, in the current implementation, the robot's peer-to-peer sensing capability is emulated by a GPS sensor, it would be desirable if the proposed method could be implemented on the swarms equipped with real peer-to-peer sensors.

Bibliography

[1] Soon-Jo Chung et al. "A survey on aerial swarm robotics". In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 837–855.

[2] Craig W Reynolds. "Flocks, herds and schools: A distributed behavioral model". In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*. 1987, pp. 25–34.

[3] Carl Anderson et al. "Self-assemblages in insect societies". In: *Insectes sociaux* 49.2 (2002), pp. 99–110.

[4] Nathan J Mlot et al. "Fire ants self-assemble into waterproof rafts to survive floods". In: *Proceedings of the National Academy of Sciences* 108.19 (2011), pp. 7669–7673.

[5] [Online] Canadian Geese flying in a V-formation. URL: https://tinyurl.com/ksx2wu72.

[6] [Online] Army ants build bridge. URL: https://tinyurl.com/eujkudye.

[7] [Online] Atlantic Ocean - school of fish. URL: https://tinyurl.com/ejvxnjb4.

[8] Mark Yim. "Modular self-reconfigurable robot systems: Challenges and opportunities for the future". In: *IEEE Robotics Automat. Mag.* 10 (2007), pp. 2–11.

[9] Gangyuan Jing et al. "Accomplishing high-level tasks with modular robots". In: *Autonomous Robots* 42.7 (2018), pp. 1337–1354.

[10] [Online] Illuminate your story with Intel drone light shows. URL: https://tinyurl.com/avs5zsfu.

[11] Peter R Wurman et al. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), pp. 9–9.

[12] [Online] Starlink: Internet from Space. URL: https://tinyurl.com/3rekkhmb.

[13] [Online] Kiva Systems. URL: https://tinyurl.com/3rxmeasr.

[14] Saivipulteja Elagandula et al. "Multi-Robot Path Planning for Cooperative 3D Printing". In: *ASME 2020 15th International Manufacturing Science and Engineering Conference*. American Society of Mechanical Engineers Digital Collection. 2020.

[15] Kai-Chieh Ma et al. "Multi-robot informative and adaptive planning for persistent environmental monitoring". In: *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 285–298.

[16] [Online] OFFensive Swarm-Enabled Tactics (OFFSET). URL: https://tinyurl.com/t757jear.

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

[18] [Online] Transforming Amsterdam's canals with a fleet of autonomous boats. URL: http://roboat.org/.

[19] Gábor Vásárhelyi et al. "Outdoor flocking and formation flight with autonomous aerial robots". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 3866–3873.

[20] James A Preiss et al. "Downwash-aware trajectory planning for large quadrotor teams". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 250–257.

[21] Xintong Du et al. "Fast and in sync: Periodic swarm patterns for quadrotors". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 9143–9149.

[22] Jingjin Yu et al. "Distance optimal formation control on graphs with a tight convergence time guarantee". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*. IEEE. 2012, pp. 4023–4028.

[23] Glenn Wagner et al. "M*: A complete multirobot path planning algorithm with optimality bounds". In: *Redundancy in Robot Manipulators and Multi-Robot Systems*. Springer, 2013, pp. 167–181.

[24] Damjan Miklic et al. "A discrete grid abstraction for formation control in the presence of obstacles". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3750–3755.

[25] Federico Augugliaro et al. "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach". In: *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1917–1922.

[26] Sarah Tang et al. "A complete algorithm for generating safe trajectories for multi-robot teams". In: *Robotics Research*. Springer, 2018, pp. 599–616.

[27] Matthew Turpin et al. "Capt: Concurrent assignment and planning of trajectories for multiple robots". In: *The International Journal of Robotics Research* 33.1 (2014), pp. 98–112.

[28] Luiz Chaimowicz et al. "Controlling swarms of robots using interpolated implicit functions". In: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, pp. 2487–2492.

[29] Péter Molnár et al. "Control of distributed autonomous robotic systems using principles of pattern formation in nature and pedestrian behavior". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 31.3 (2001), pp. 433–435.

[30] Michael M Zavlanos et al. "Potential fields for maintaining connectivity of mobile networks". In: *IEEE Transactions on robotics* 23.4 (2007), pp. 812–816.

[31] Michael Rubenstein et al. "Programmable self-assembly in a thousand-robot swarm". In: *Science* 345.6198 (2014), pp. 795–799.

[32] Javier Alonso-Mora et al. "Image and animation display with multiple mobile robots". In: *The International Journal of Robotics Research* 31.6 (2012), pp. 753–773.

[33] Michael Rubenstein. *Self-assembly and self-healing for robotic collectives*. University of Southern California, 2010.

[34] Michael Rubenstein et al. "Automatic scalable size selection for the shape of a distributed robotic collective". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 508–513.

[35] Dingjiang Zhou et al. "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 1047–1054.

[36] Javier Alonso-Mora et al. "Optimal reciprocal collision avoidance for multiple nonholonomic robots". In: *Distributed autonomous robotic systems*. Springer, 2013, pp. 203–216.

[37] Guannan Li et al. "Decentralized progressive shape formation with robot swarms". In: *Autonomous Robots* 43.6 (2019), pp. 1505–1521.

[38] Jimming Cheng et al. "Robust and self-repairing formation control for swarms of mobile agents". In: *AAAI*. Vol. 5. 2005. 2005.

[39] Michael Rubenstein et al. "Scalable self-assembly and self-repair in a collective of robots". In: *2009 IEEE/RSJ international conference on Intelligent robots and systems*. IEEE. 2009, pp. 1484–1489.

[40] Kasper Stoy et al. "Self-repair through scale independent self-reconfiguration". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2. IEEE. 2004, pp. 2062–2067.

[41] Hanlin Wang et al. "A Fast, Accurate, and Scalable Probabilistic Sample-Based Approach for Counting Swarm Size". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 7180–7185.

[42] Hanlin Wang et al. "Generating Goal Configurations for Scalable Shape Formation in Robotic Swarms". In: *Proceedings 2021 International Symposium on Distributed Autonomous Robotic Systems*. 2021.

[43] Hanlin Wang et al. "Decentralized Localization in Homogeneous Swarms Considering Real-World Non-idealities". In: *IEEE Robotics and Automation Letters* (2021), pp. 1–1. DOI: 10.1109/LRA.2021.3095032.

[44] Hanlin Wang et al. "Shape Formation in Homogeneous Swarms Using Local Task Swapping". In: *IEEE Transactions on Robotics* 36.3 (2020), pp. 597–612.

[45] I Slavkov et al. "Morphogenesis in robot swarms". In: *Science Robotics* 3.25 (2018).

[46] Li Wang et al. "Safety barrier certificates for collisions-free multirobot systems". In: *IEEE Transactions on Robotics* 33.3 (2017), pp. 661–674.

[47] Javier Alonso-Mora et al. "Cooperative collision avoidance for nonholonomic robots". In: *IEEE Transactions on Robotics* 34.2 (2018), pp. 404–420.

[48] Melvin Gauci et al. "Error cascades in collective behavior: a case study of the gradient algorithm on 1000 physical agents". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. 2017, pp. 1404–1412.

[49] Melvin Gauci et al. "Programmable self-disassembly for shape formation in large-scale robot collectives". In: *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 573–586.

[50] Michael Rubenstein et al. "A scalable and distributed approach for self-assembly and self-healing of a differentiated shape". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 1397–1402.

[51] Michael Rubenstein et al. "Kilobot: A low cost robot with scalable operations designed for collective behaviors". In: *Robotics and Autonomous Systems* 62.7 (2014), pp. 966–975.

[52] Vishnu R Desaraju et al. "Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 4956–4961.

[53] Ebrima Jobe et al. "R-One Swarm Robot: Developing the Accelerometer and Gyroscope". In: *Twenty-Fifth International FLAIRS Conference*. Citeseer. 2012.

[54] Francesco Mondada et al. "SWARM-BOT: A new distributed robotic concept". In: *Autonomous robots* 17.2 (2004), pp. 193–221.

[55] Francesco Mondada et al. "The e-puck, a robot designed for education in engineering". In: *Proceedings of the 9th conference on autonomous robot systems and competitions*. Vol. 1. CONF. IPCB: Instituto Politécnico de Castelo Branco. 2009, pp. 59–65.

[56] Gilles Caprari et al. "The Autonomous Micro Robot" Alice": a platform for scientific and commercial applications". In: *MHA'98. Proceedings of the 1998 International Symposium on Micromechatronics and Human Science.-Creation of New Industry*. IEEE. 1998, pp. 231–235.

[57] Yang Li et al. "Distributed camouflage for swarm robotics and smart materials". In: *Autonomous Robots* 42.8 (2018), pp. 1635–1650.

[58] Gilles Caprari et al. "Mobile micro-robots ready to use: Alice". In: *2005 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2005, pp. 3295–3300.

[59] Andrew P Sabelhaus et al. "TinyTeRP: A tiny terrestrial robotic platform with modular sensing". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 2600–2605.

[60] Michael Rubenstein et al. "Kilobot: A low cost scalable robot system for collective behaviors". In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 3293–3298.

[61] Daniel Pickem et al. "The robotarium: A remotely accessible swarm robotics research testbed". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1699–1706.

[62] Cheng Hu et al. "Colias IV: The affordable micro robot platform with bio-inspired vision". In: *Annual Conference Towards Autonomous Robotic Systems*. Springer. 2018, pp. 197–208.

[63] Justin Y Kim et al. "mROBerTO: A modular millirobot for swarm-behavior studies". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 2109–2114.

[64] Kasra Eshaghi et al. "mROBerTO 2.0–An Autonomous Millirobot With Enhanced Locomotion for Swarm Robotics". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 962–969.

[65] Eric Hare et al. "Designing Modular Software: A Case Study in Introductory Statistics". In: *Journal of Computational and Graphical Statistics* 26.3 (2017), pp. 493–500.

[66] Thomas Halva Labella et al. "Self-organised task allocation in a group of robots". In: *Distributed Autonomous Robotic Systems 6*. Springer, 2007, pp. 389–398.

[67] Kristina Lerman et al. "A review of probabilistic macroscopic models for swarm robotic systems". In: *International workshop on swarm robotics*. Springer. 2004, pp. 143–152.

[68] Chih-Han Yu et al. "Self-adapting modular robotics: A generalized distributed consensus framework". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 1881–1888.

[69] Navneet Malpani et al. "Distributed token circulation in mobile ad hoc networks". In: *IEEE Transactions on Mobile Computing* 4.2 (2005), pp. 154–165.

[70] Laurent Mathy et al. "An overlay tree building control protocol". In: *International Workshop on Networked Group Communication*. Springer. 2001, pp. 76–87.

[71] Dionysios Kostoulas et al. "Decentralized schemes for size estimation in large and dynamic groups". In: *Fourth IEEE International Symposium on Network Computing and Applications*. IEEE. 2005, pp. 41–48.

[72] Chris Melhuish et al. "Convoying: using chorusing to form travelling groups of minimal agents". In: *Robotics and Autonomous Systems* 28.2-3 (1999), pp. 207–216.

[73] Manuele Brambilla. (2009). "Group Size Estimation", Thesis.

[74] Matthew L Elwin et al. "A systematic design process for internal model average consensus estimators". In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 5878–5883.

[75] Julia T Ebert et al. "Multi-feature collective decision making in robot swarms". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. 2018, pp. 1711–1719.

[76] Hanlin Wang. Supplementary Materials. URL: https://tinyurl.com/k68kafph.

[77] A Hsieh Mong-ying et al. "Pattern generation with multiple robots". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 2442–2447.

[78] Chien Chern Cheah et al. "Region-based shape control for a swarm of robots". In: *Automatica* 45.10 (2009), pp. 2406–2411.

[79] Mong-Ying Ani Hsieh et al. "Decentralized controllers for shape generation with robotic swarms". In: (2008).

[80] Javier Alonso-Mora et al. "Multi-robot system for artistic pattern formation". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4512–4517.

[81] John W Romanishin et al. "3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 1925–1932.

[82] Olivier Rukundo et al. "Nearest neighbor value interpolation". In: *arXiv preprint arXiv:1211.1768* (2012).
[83] Daniel Vaquero et al. "A survey of image retargeting techniques". In: *Applications of Digital Image Processing XXXIII*. Vol. 7798. International Society for Optics and Photonics. 2010, p. 779814.
[84] YY Zhang et al. "A parallel thinning algorithm with two-subiteration that generates one-pixel-wide skeletons". In: *Proceedings of 13th International Conference on Pattern Recognition*. Vol. 4. IEEE. 1996, pp. 457–461.
[85] Robert Fisher. Distance transform. URL: https://tinyurl.com/22uacjz2.
[86] Harold W Kuhn. "The Hungarian method for the assignment problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
[87] Nathan K Long et al. "A comprehensive review of shepherding as a bio-inspired swarm-robotics guidance approach". In: *IEEE Transactions on Emerging Topics in Computational Intelligence* 4.4 (2020), pp. 523–537.
[88] Hanlin Wang et al. "Walk, stop, count, and swap: decentralized multi-agent path finding with theoretical guarantees". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 1119–1126.
[89] Marco Imperoli et al. "An effective multi-cue positioning system for agricultural robotics". In: *IEEE Robotics and Automation Letters* 3.4 (2018), pp. 3685–3692.
[90] Matthew L Elwin et al. "Distributed Voronoi neighbor identification from inter-robot distances". In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1320–1327.
[91] John Klingner et al. "Fault-tolerant covariance intersection for localizing robot swarms". In: *Robotics and Autonomous Systems* 122 (2019), p. 103306.
[92] David Moore et al. "Robust distributed network localization with noisy range measurements". In: *Proceedings of the 2nd international conference on Embedded networked sensor systems*. 2004, pp. 50–61.
[93] Stefan M Trenkwalder et al. "SwarmCom: an infra-red-based mobile ad-hoc network for severely constrained robots". In: *Autonomous Robots* 44.1 (2020), pp. 93–114.
[94] Frankie KW Chan et al. "Accurate distributed range-based positioning algorithm for wireless sensor networks". In: *IEEE Transactions on Signal Processing* 57.10 (2009), pp. 4100–4105.
[95] Meysam Basiri et al. "On-board relative bearing estimation for teams of drones using sound". In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 820–827.
[96] Alejandro Cornejo et al. "Scale-free coordinates for multi-robot systems with bearing-only sensors". In: *The International Journal of Robotics Research* 32.12 (2013), pp. 1459–1474.
[97] James F Roberts et al. "2.5 D infrared range and bearing system for collective robotics". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3659–3664.
[98] Marco Cominelli et al. "Dead on arrival: An empirical study of the Bluetooth 5.1 positioning system". In: *Proceedings of the 13th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*. 2019, pp. 13–20.

[99] Javier Garcia et al. "Localization using a Particle Filter and Magnetic Induction Transmissions: Theory and Experiments in Air". In: *2020 IEEE Texas Symposium on Wireless and Microwave Circuits and Systems (WMCS)*. IEEE. 2020, pp. 1–6.

[100] Shuangshuang Li et al. "Underwater TDOA Acoustical Location Based on Majorization-Minimization Optimization". In: *Sensors* 20.16 (2020), p. 4457.

[101] J Spletzer et al. "Cooperative localization and control for multi-robot manipulation". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium*. Vol. 2. IEEE. 2001, pp. 631–636.

[102] Radhika Nagpal et al. "Organizing a global coordinate system from local information on an ad hoc sensor network". In: *Information processing in sensor networks*. Springer. 2003, pp. 333–348.

[103] Riccardo Spica et al. "Active decentralized scale estimation for bearing-based localization". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 5084–5091.

[104] Yao Zou et al. "Distributed Localization and Circumnavigation Algorithms for a Multiagent System With Persistent and Intermittent Bearing Measurements". In: *IEEE Transactions on Control Systems Technology* (2020).

[105] Zhiyun Lin et al. "Distributed self localization for relative position sensing networks in 2D space". In: *IEEE Transactions on Signal Processing* 63.14 (2015), pp. 3751–3761.

[106] Luca Carlone et al. "From angular manifolds to the integer lattice: Guaranteed orientation estimation with application to pose graph optimization". In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 475–492.

[107] Kanti V Mardia et al. *Directional statistics*. Vol. 494. John Wiley & Sons, 2009.

[108] Dimitri P Bertsekas et al. "Parallel synchronous and asynchronous implementations of the auction algorithm". In: *Parallel Computing* 17.6-7 (1991), pp. 707–732.

[109] Rafael Mathias de Mendonça et al. "Efficient distributed algorithm of dynamic task assignment for swarm robotics". In: *Neurocomputing* 172 (2016), pp. 345–355.

[110] Dimitri P Bertsekas. "A new algorithm for the assignment problem". In: *Mathematical Programming* 21.1 (1981), pp. 152–171.

[111] Ming S Hung. "A polynomial simplex method for the assignment problem". In: *Operations Research* 31.3 (1983), pp. 595–600.

[112] Meng Ji et al. "Role-assignment in multi-agent coordination". In: (2006).

[113] Jingjin Yu et al. "Shortest path set induced vertex ordering and its application to distributed distance optimal formation path planning and control on graphs". In: *52nd IEEE Conference on Decision and Control*. IEEE. 2013, pp. 2775–2780.

[114] Matthew Turpin et al. "Goal assignment and trajectory planning for large teams of interchangeable robots". In: *Autonomous Robots* 37.4 (2014), pp. 401–415.

[115] Patrick MacAlpine et al. "SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. 1. 2015.

[116] Shuguang Li et al. "Particle robotics based on statistical mechanics of loosely coupled components". In: *Nature* 567.7748 (2019), pp. 361–365.

[117] Nathan Michael et al. "Distributed multi-robot task assignment and formation control". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 128–133.

[118] Stephen L Smith et al. "Target assignment for robotic networks: Asymptotic performance under limited communication". In: *2007 American Control Conference*. IEEE. 2007, pp. 1155–1160.

[119] Kwang-Kyo Oh et al. "A survey of multi-agent formation control". In: *Automatica* 53 (2015), pp. 424–440.

[120] Mehran Mesbahi et al. *Graph theoretic methods in multiagent networks*. Vol. 33. Princeton University Press, 2010.

[121] Ming Cao et al. "Formation control using range-only measurements". In: *Automatica* 47.4 (2011), pp. 776–781.

[122] Dimos V Dimarogonas et al. "Further results on the stability of distance-based multi-robot formations". In: *2009 American Control Conference*. IEEE. 2009, pp. 2972–2977.

[123] Reza Olfati-Saber et al. "Graph rigidity and distributed formation stabilization of multi-vehicle systems". In: *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.* Vol. 3. IEEE. 2002, pp. 2965–2971.

[124] Kwang-Kyo Oh et al. "Formation control of mobile agents based on inter-agent distance dynamics". In: *Automatica* 47.10 (2011), pp. 2306–2312.

[125] J Guo et al. "Adaptive control schemes for mobile robot formations with triangularised structures". In: *IET control theory & applications* 4.9 (2010), pp. 1817–1827.

[126] Shiyu Zhao et al. "Bearing rigidity and almost global bearing-only formation stabilization". In: *IEEE Transactions on Automatic Control* 61.5 (2015), pp. 1255–1268.

[127] Miguel Aranda et al. "Distributed formation stabilization using relative position measurements in local coordinates". In: *IEEE Transactions on Automatic Control* 61.12 (2016), pp. 3925–3935.

[128] Edward A Macdonald. "Multi-robot assignment and formation control". PhD thesis. Georgia Institute of Technology, 2011.

[129] Riccardo Falconi et al. "Graph based distributed control of non-holonomic vehicles endowed with local positioning information engaged in escorting missions". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 3207–3214.

[130] Augie Widyotriatmo et al. "Implementation of leader-follower formation control of a team of nonholonomic mobile robots". In: *International Journal of Computers Communications & Control* 12.6 (2017), pp. 871–885.

[131] Zhiyun Lin et al. "Distributed formation control of multi-agent systems using complex Laplacian". In: *IEEE Transactions on Automatic Control* 59.7 (2014), pp. 1765–1777.

[132] Sven Gowal et al. "Real-time optimization of trajectories that guarantee the rendezvous of mobile robots". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 3518–3525.

[133] Jingjin Yu. "Constant factor time optimal multi-robot routing on high-dimensional grids". In: *2018 Robotics: Science and Systems* (2018).

[134] Sarah Tang et al. "Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 2216–2222.

[135] Yiannis Kantaros et al. "Distributed intermittent connectivity control of mobile robot networks". In: *IEEE Transactions on Automatic Control* 62.7 (2016), pp. 3109–3121.

[136] Jur Van den Berg et al. "Reciprocal velocity obstacles for real-time multi-agent navigation". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 1928–1935.

[137] Jur Van Den Berg et al. "Reciprocal n-body collision avoidance". In: *Robotics research*. Springer, 2011, pp. 3–19.

[138] Steven M LaValle et al. "Rapidly-exploring random trees: Progress and prospects". In: *Algorithmic and computational robotics: new directions* 5 (2001), pp. 293–308.

[139] Kar-Han Tan et al. "Virtual structures for high-precision cooperative mobile robotic control". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*. Vol. 1. IEEE. 1996, pp. 132–139.

APPENDIX

PROOFS

1. Proof of Lemma 5.1

Suppose we have n i.i.d. uniform random variables:

$$\mathcal{X}_i \sim \mathcal{U}(0,1), i = 1, 2, .., n$$

The *CDF* (cumulative density function) of their sample maximum $\mathcal{Y} = \max_{1 \leq i \leq n} \mathcal{X}_i$ is:

$$Pr(\mathcal{Y} \leq x) = \prod_{i=1}^{n} Pr(\mathcal{X}_i \leq x) = \begin{cases} 1, & x > 1 \\ x^n, & 0 \leq x \leq 1 \\ 0, & x < 0 \end{cases}$$

and the *PDF* (probability density function) of $\mathcal{Y}$ is:

$$p(\mathcal{Y} = x) = \frac{\mathrm{d}Pr(\mathcal{Y} \leq x)}{\mathrm{d}x} = \begin{cases} 0, & x > 1 \\ nx^{n-1}, & 0 \leq x \leq 1 \\ 0, & x < 0 \end{cases}$$

From this, it is straight forward to develop the $\mathcal{Y}$'s expected value $\mathbb{E}(\mathcal{Y})$ and variance $\mathbb{V}ar(\mathcal{Y})$:

$$\mathbb{E}(\mathcal{Y}) = \int_0^1 nx^n \mathrm{d}x = \frac{n}{n+1}$$

$$\mathbb{V}ar(\mathcal{Y}) = \int_0^1 nx^{n-1}(x - \frac{n}{n+1})^2 \mathrm{d}x = \frac{1}{(n+1)^2}\frac{n}{n+2}$$

2. Proof of Theorem 5.1

It is well known that for m i.i.d random variables $\mathcal{Y}_1, ..., \mathcal{Y}_m$ that have a expected value μ and a variance σ^2, it holds that:

$$\mathbb{E}(\frac{\sum_{i=1}^{m} \mathcal{Y}_i}{m}) = \mu, \quad \mathbb{V}ar(\frac{\sum_{i=1}^{m} \mathcal{Y}_i}{m}) = \frac{\sigma^2}{m}$$

In our case, specifically, let $k = \frac{\sum_{i=1}^{m} \mathcal{Y}_i}{m}$ and $\mathcal{Y}_i$ be the sample maximum of n *i.i.d.* $\mathcal{U}(0,1)$ random variables, using the result obtained in lemma 5.1, we have:

$$\mathbb{E}(k) = \frac{n}{n+1}, \quad \mathbb{V}ar(k) = \frac{1}{m}\frac{1}{(n+1)^2}\frac{n}{n+2}$$

By *Chebyshev* inequality, we have:

$$Pr(|k - \mathbb{E}(k)| \geq \delta) \leq \frac{\mathbb{V}ar(k)}{\delta^2}$$

On the other hand, let $n^* = \frac{k}{1-k}$, it is straight forward to examine that:

$$RE(n^*) = \frac{|n^* - n|}{n^* + 1} = (n+1)|k - \mathbb{E}(k)|$$

that is:

$$Pr(\frac{RE(n^*)}{n+1} \geq \delta) \leq \frac{1}{m\delta^2}\frac{1}{(n+1)^2}\frac{n}{n+2}$$

Note that $n+1$ is a constant. Let $\epsilon = \delta(n+1)$, by this variable change we have:

$$Pr(RE(n^*) \geq \epsilon) \leq \frac{1}{m\epsilon^2}\frac{n}{n+2} < \frac{1}{m\epsilon^2}$$

3. Proof of Lemma 6.1

For the sake of description, given a partition $\mathcal{P}$, $\overline{\mathcal{P}}$ and $\underline{\mathcal{P}}$ denote the maximal and minimal cardinality of any subsets in $\mathcal{P}$, moreover, $\overline{a_{\mathcal{P}}}$ denotes the subset in $\mathcal{P}$ that holds the maximal cardinality. I prove Lemma 1 by showing that: given the condition (6.1), it is impossible to get $\overline{\mathcal{P}'_k(\mathcal{A})}$ any lower. In order to lower $\overline{\mathcal{P}'_k(\mathcal{A})}$, we need to remove at least 1 element from $\overline{a_{\mathcal{P}'_k(\mathcal{A})}}$, in addition, to satisfy the constraint *(i)* stated in Problem 1, the removed element should be packed into some other subset $a_j^{'}$. On the other hand, given the condition: $\overline{\mathcal{P}'_k(\mathcal{A})} - \underline{\mathcal{P}'_k(\mathcal{A})} \leq 1$, no matter which subset $a_j^{'}$ this removed element will be packed into, let $a_j^{''}$ be the newly formed $a_j^{'}$, we have: $|a_j^{''}| = |a_j^{'}| + 1 \geq \overline{\mathcal{P}'_k(\mathcal{A})}$. That is, let $\mathcal{P}''_k(\mathcal{A})$ be this newly formed k-partition of set $\mathcal{A}$, we have $\overline{\mathcal{P}''_k(\mathcal{A})} \geq |a_j^{''}| \geq \overline{\mathcal{P}'_k(\mathcal{A})}$. This suggests that any attempt to lower $\overline{\mathcal{P}'_k(\mathcal{A})}$ will fail eventually, which completes the proof.

4. Proof of Theorem 6.1

Theorem 6.1 suggests that: given two reference grids $\mathcal{G}_o^l$ and $\mathcal{G}_o^h$, the way that we determine each subset's size (Alg. 7, Line 6-10) is the optimal way that minimizes the maximal *difference score* between any pair of generated masked grids whose difference of in-shape cell number is 1. First, according to Alg. 7 Line 25-27, we have $\mathcal{S}(\mathcal{G}_i) - \mathcal{S}(\mathcal{G}_{i+1}) = d_{l-h}^{i-a}$ and $\mathcal{S}(\mathcal{G}_{i+1}) - \mathcal{S}(\mathcal{G}_i) = d_{h-l}^{i-a}$, we can then rewrite $|\mathcal{S}(\mathcal{G}_i) - \mathcal{S}(\mathcal{G}_{i+1})| + |\mathcal{S}(\mathcal{G}_{i+1}) - \mathcal{S}(\mathcal{G}_i)|$ as $|d_{h-l}^{i-a}| + |d_{l-h}^{i-a}|$. In addition, recall that we have the constraint on each pair of subsets' cardinalities that: $|d_{h-l}^{i-a}| = |d_{l-h}^{i-a}| + 1$, combining this constraint with the result we just obtained, the objective (6.2) can be rewritten as follows:

$$\max_{a \le i \le b-1} 2|d_{l-h}^{i-a}| + 1 \tag{A.1}$$

One helpful observation here is that: finding the optimal way to determine each subset's size that minimizes the objective (A.1) is essentially an instance of Problem 1, with $\mathcal{A} = \mathcal{D}_{l-h}$ and $k = b - a$. Next, it is straight forward to examine that Alg. 7, Line 6-10 satisfy the condition (6.2) stated in Lemma 6.1, that is, Alg. 7, Line 6-10 is the optimal way that minimizes the objective (A.1) by Lemma 6.1, completing the proof.

5. Proof of Proposition 7.1

. The proof of proposition 1 is sketched as follows: First, in Lemma A.1, we show that the global observer is able to capture all the changes of each agent's pose estimate. Next, in the rest of the section, we establish the equivalence relationship between Algorithm 9 and the swarm's behavior.

Lemma A.1. *From the global observer's perspective, between the clock tick* $\frac{k}{f}$ *and the clock tick* $\frac{k+1}{f}$*, each agent will execute Algorithm 8 Line 8-29 exactly one time.*

Proof. We prove Lemma A.1 by contradiction. First, we suppose that there is an agent executing Algorithm 8 Line 8-29 ≥ 2 times during the time interval $(\frac{k}{f}, \frac{k+1}{f}]$, then it implies that the interval between two times agent executing Algorithm 8 Line 8-29 is smaller than $\frac{1}{f}$, which contradicts to the fact that each agent is programmed to execute Algorithm 8 Line 8-29 at a fixed frequency of f.

Next, we suppose that there is an agent executing Algorithm 8 Line 8-29 0 time during the time interval $(\frac{k}{f}, \frac{k+1}{f}]$. This suggests that the time span between the last time it executed Algorithm 8 Line 8-29 and the next time it executes Algorithm 8 Line 8-29 will be greater than $\frac{1}{f}$, where contradiction occurs.

By far, we have showed that it is incorrect to assume an agent can execute Algorithm 8 Line 8-29 ≥ 2 times or 0 time during the time interval $(\frac{k}{f}, \frac{k+1}{f}]$, which suffices to prove Lemma A.1. □

Note that the Algorithm 8 Line 8-29 is the only place where an agent can update its pose estimate. The lemma A.1 suggests that : between two times the global observer record each agent's pose estimate, each agent is able to change its pose estimate no more than one time,

which is sufficient to show that the global observer will not miss any change of any agent's pose estimate. A more intuitive but less formal explanation can be found in Fig. A.1.

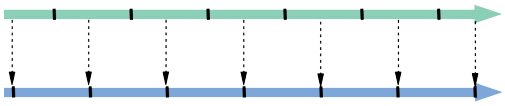

Figure A.1. The graphical illustration of the observer's sampling scheme. The green horizontal arrow line is an agent a_i's local clock and the blue horizontal arrow line is the observer's clock. Each cell on each clock arrow line indicates a time span of $\frac{1}{f}$. Each vertical dotted arrow line indicates an event of the observer recording agent a_i's pose estimate. The ticks on the green arrow line are the clock ticks when a_i execute Algorithm 8 Line 9-29, which is the only place where an agent might change its pose estimate in our algorithm. As we can see in the figure, in this example, a_i has at most seven different pose estimates, despite that the observer's clock is asynchronous with the agent, all these pose estimates will be captured by the observer.

Next, we start to show the equivalence relationship between Algorithm 9 and the swarm's behavior.

Algorithm 9 Line 2-3 suggests that: each time when an agent a_i attempts to update its pose estimate, it will first pick a neighbor a_j with a probability $\frac{1-(1-\lambda)^{|N_j|}}{|N_j|}\frac{1-(1-\lambda)^{|N_i|}}{|N_i|}$, where N_i is the set of all a_i's neighbors, then use this neighbor's information to do the update. In Lemma A.2 and Lemma A.3, we show how this probability of selecting each neighbor is calculated. According to Algorithm 8, Line 14 - 15, for each agent a_i, in order to use neighbor a_j's information to update its pose estimate, two conditions must be satisfied: (i) when a_i randomly and uniformly sample buffer *msg_buff*, it picks the message from a_j, and (ii) this message from a_j contains a_i's bearing measured by a_j. On the other hand, when agent a_j choose the neighbor to echo the measured bearing (Algorithm 8, 11 - 13), this echo neighbor is also randomly and uniformly picked from a_j buffer *msg_buff*. In the other words, the probability of the event "a_i uses a_j to update its pose estimate" = the probability of the

event "a_i picks a_j when sampling its *msg_buff*" $\times$ the probability of the event "a_i picks a_j when sampling its *msg_buff*". Obviously, here, the key is to calculate the probability of one agent picks one of its neighbors when randomly and uniformly sampling its *msg_buff*.

Given the fact that each agent's communication frequency is the same as the frequency at which each agent executes Algorithm 8 Line 8-29, it is straight forward to conclude the following fact:

Lemma A.2. *From an agent a_i's perspective, between the two times it executes Algorithm 8 Line 8-29, each of its neighbors $a_j \in N_i$ will transmit exactly one message.*

Proof. The Lemma A.2 can be proved in the same way as Lemma A.1. □

It is worth noting that due to the communication loss, for each agent a_i, when one of its neighbor transmits a message, it will receive this message with a probability of $\lambda < 1$. Next, in Lemma A.3, we calculate the probability of "a_i picks a_j when sampling its *msg_buff*".

Lemma A.3. *For each agent a_i, when it samples its buffer msg_buff (Algorithm 8, Line 11, 14), the probability of the message from neighbor $a_j \in N_i$ being picked is $\frac{1-(1-\lambda)^{|N_i|}}{|N_i|}$.*

Proof. Given the result from Lemma A.2, when a_i samples its buffer *msg_buf*, for each of its neighbors a_j, the probability that a_j's message is in *msg_buf* is λ, in addition, there will be not more than one message from the same neighbor in the *msg_buff*. Let $\mathbf{e}$ be the event of "a_i picks the message from a_j when sampling *msg_buff*", we here decompose $\mathbf{e}$ into a set of disjointed events:

$$\mathbf{e} = \bigcup_{m=0}^{|N_i|-1} \mathbf{e}_m$$

In which, $\mathbf{e}_m$ is the event that: "a_j's message is in *msg_buff*" $\cap$ "the messages from m other neighbors are also in *msg_buff*" $\cap$ "a_j is picked given the fact that a_j's message is in *msg_buff* and m other neighbors' messages are also in *msg_buff*".

It is straight forward to develop the probability of event $\mathbf{e}_m$:

$$Pr\{\mathbf{e_m}\} = \lambda \binom{|N_i|-1}{m} \lambda^m (1-\lambda)^{|N_i|-1-m} \frac{1}{m+1} \tag{A.2}$$

Note that for any pair of sub-events $\mathbf{e}_m$ and $\mathbf{e}_l$, they are disjointed, namely: $\forall m \neq l, Pr\{\mathbf{e}_m \cap \mathbf{e}_l\} = 0$. Combing this conclusion and the result obtained in equation A.2, we have:

$$Pr\{\mathbf{e}\} = \sum_{m=0}^{|N_i|-1} \lambda \binom{|N_i|-1}{m} \lambda^m (1-\lambda)^{|N_i|-1-m} \frac{1}{m+1} = \frac{1}{|N_i|} \sum_{m=1}^{|N_i|} \binom{|N_i|}{m} \lambda^m (1-\lambda)^{|N_i|-m}$$

On the other hand, it is well known that:

$$\{\lambda + (1-\lambda)\}^n = \sum_{m=0}^{n} \binom{n}{m} \lambda^m (1-\lambda)^{n-m} = 1 \tag{A.3}$$

With equation A.3, the probability of event $\mathbf{e}$ $Pr\{\mathbf{e}\}$ can be rewritten as:

$$Pr\{\mathbf{e}\} = \frac{1-(1-\lambda)^{|N_i|}}{|N_i|}$$

□

As we discussed before, the probability of the event "a_i uses a_j to update its pose estimate" = the probability of the event "a_i picks a_j when sampling its *msg_buff*" $\times$ the

probability of the event "a_i picks a_j when sampling its *msg_buff*". This conclusion, in combination with the result we obtained in Lemma A.3, suffices to show that the probability of a_i using the message from a_j to update its pose estimate is $\frac{1-(1-\lambda)^{|N_j|}}{|N_j|}\frac{1-(1-\lambda)^{|N_i|}}{|N_i|}$.

So far, the remaining part for proving Proposition 7.1 is to compare the way how each agent update its pose estimate given an neighbor's message in Algorithm 8 and Algorithm 9. It is straight forward to examine that the way how each agent update its pose estimate in the Algorithm 9 (Algorithm 9 Line 5-14) is the same as Algorithm 8 (Algorithm 8 Line 16-26), completing the proof of Proposition 7.1.

6. Proof of Lemma 7.1

. At $k+1^{th}$ iteration, given each agent a_i's orientation estimate Θ_i^k at k^{th} iteration, the agent a_i's expected orientation estimate update is:

$$\mathbb{E}(\Theta_i^{k+1} - \Theta_i^k) = \alpha \sum_{a_j \in N_i} \frac{1-(1-\lambda)^{|N_j|}}{|N_j|} \frac{1-(1-\lambda)^{|N_i|}}{|N_i|} \mathbb{E}(\Delta\Theta_{ij}^k) \tag{A.4}$$

In which:

$$\Delta\Theta_{ij}^k = \begin{bmatrix} \cos\left(\hat{\mathcal{W}}_{ij}\right), \sin\left(\hat{\mathcal{W}}_{ij}\right) \\ -\sin\left(\hat{\mathcal{W}}_{ij}\right), \cos\left(\hat{\mathcal{W}}_{ij}\right) \end{bmatrix} \Theta_j^k - \Theta_i^k \tag{A.5}$$

To calculate the term $\mathbb{E}(\Delta\Theta_{ij}^k)$ in equation A.4, we first need to calculate two quantities $\mathbb{E}\{cos(\hat{\mathcal{W}}_{ij})\}$ and $\mathbb{E}\{cos(\hat{\mathcal{W}}_{ij})\}$. According to our agent model, we have $\hat{\mathcal{B}_{ij}} = \mathcal{B}_{ij} + \epsilon_{\mathcal{B}}$, where $\epsilon_{\mathcal{B}} \sim \mathcal{N}(0, \sigma_{\mathcal{B}}^2)$ is agent a_i's bearing sensing noise. On the other hand, by the Algorithm 2, Line 8, we have: $\hat{\mathcal{W}}_{ij} = \langle \langle \mathcal{B}_{ij} + \epsilon_{\mathcal{B}}^i \rangle_{2\pi} - \langle \mathcal{B}_{ji} + \epsilon_{\mathcal{B}}^j \rangle_{2\pi} + \pi \rangle_{2\pi}$, where $\epsilon_{\mathcal{B}}^i$ and $\epsilon_{\mathcal{B}}^j$ are agents a_i and a_j's bearing sensing noise, respectively. Here, one helpful property of 2π modulus operator $\langle \cdot \rangle_{2\pi}$ is that: given an angle θ, by definition we have:

$$cos(\langle \theta \rangle_{2\pi}) = cos(\theta), \quad sin(\langle \theta \rangle_{2\pi}) = sin(\theta) \tag{A.6}$$

That is:

$$\begin{aligned} cos(\hat{\mathcal{W}}_{ij}) &\overset{(A.6)}{=} cos(\langle \mathcal{B}_{ij} + \epsilon_{\mathcal{B}}^i \rangle_{2\pi} - \langle \mathcal{B}_{ji} + \epsilon_{\mathcal{B}}^j \rangle_{2\pi} + \pi) \\ &\overset{(A.6)}{=} -cos(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}}^i) cos(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}}^j) - sin(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}}^i) sin(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}}^j) \end{aligned} \tag{A.7}$$

$$\text{(A.8)}\quad sin(\hat{\mathcal{W}}_{ij}) \overset{(A.6)}{=} sin(\langle \mathcal{B}_{ij} + \epsilon^i_{\mathcal{B}} \rangle_{2\pi} - \langle \mathcal{B}_{ji} + \epsilon^j_{\mathcal{B}} \rangle_{2\pi} + \pi)$$
$$\overset{(A.6)}{=} -sin(\mathcal{B}_{ij} + \epsilon^i_{\mathcal{B}})cos(\mathcal{B}_{ji} + \epsilon^j_{\mathcal{B}}) + cos(\mathcal{B}_{ij} + \epsilon^i_{\mathcal{B}})sin(\mathcal{B}_{ji} + \epsilon^j_{\mathcal{B}})$$

Rearrange the equation A.7 and equation A.8, we have:

$$\text{(A.9)}\quad cos(\hat{\mathcal{W}}_{ij}) = cos(\mathcal{B}_{ij} - \mathcal{B}_{ji} + \pi + \epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}}) = cos(\mathcal{W}_{ij})cos(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}}) - sin(\mathcal{W}_{ij})sin(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})$$

$$\text{(A.10)}\quad sin(\hat{\mathcal{W}}_{ij}) = sin(\mathcal{B}_{ij} - \mathcal{B}_{ji} + \pi + \epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}}) = sin(\mathcal{W}_{ij})cos(\epsilon^i_{\mathcal{B}} - \epsilon^i_{\mathcal{B}}) + cos(\mathcal{W}_{ij})sin(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})$$

Obviously, agent a_i and a_j's bearing sensing noise $\epsilon^i_{\mathcal{B}}, \epsilon^j_{\mathcal{B}}$ are independent with each other, therefore:

$$\text{(A.11)}\quad \mathbb{E}\{sin(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})\} = \mathbb{E}\{sin(\epsilon^i_{\mathcal{B}})cos(\epsilon^j_{\mathcal{B}}) - cos(\epsilon^i_{\mathcal{B}})sin(\epsilon^j_{\mathcal{B}})\}$$
$$\overset{\text{independence}}{=} \mathbb{E}\{sin(\epsilon^i_{\mathcal{B}})\}\mathbb{E}\{cos(\epsilon^j_{\mathcal{B}})\} - \mathbb{E}\{cos(\epsilon^i_{\mathcal{B}})\}\mathbb{E}\{sin(\epsilon^j_{\mathcal{B}})\}$$

$$\text{(A.12)}\quad \mathbb{E}\{cos(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})\} = \mathbb{E}\{cos(\epsilon^i_{\mathcal{B}})cos(\epsilon^j_{\mathcal{B}}) + sin(\epsilon^i_{\mathcal{B}})sin(\epsilon^j_{\mathcal{B}})\}$$
$$\overset{\text{independence}}{=} \mathbb{E}\{cos(\epsilon^i_{\mathcal{B}})\}\mathbb{E}\{cos(\epsilon^j_{\mathcal{B}})\} + \mathbb{E}\{sin(\epsilon^i_{\mathcal{B}})\}\mathbb{E}\{sin(\epsilon^j_{\mathcal{B}})\}$$

On the other hand, it is well known that for an random variable $\epsilon \sim \mathcal{N}(0, \sigma^2)$, it holds that:

$$\text{(A.13)}\qquad \mathbb{E}\{cos(\epsilon)\} = \exp\left\{-\frac{\sigma^2}{2}\right\}, \quad \mathbb{E}\{sin(\epsilon)\} = 0$$

That is:

$$\mathbb{E}\{sin(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})\} \overset{(A.13,A.11)}{=} 0 \tag{A.14}$$

$$\mathbb{E}\{cos(\epsilon^i_{\mathcal{B}} - \epsilon^j_{\mathcal{B}})\} \overset{(A.13,A.12)}{=} \exp\{-\sigma^2_{\mathcal{B}}\} \tag{A.15}$$

By combining equation A.14 with equation A.10 and combining equation A.15 with equation A.9, we have:

$$\mathbb{E}\{cos(\hat{\mathcal{W}}_{ij})\} \overset{(A.9,A.15)}{=} cos(\mathcal{W}_{ij}) \exp\{-\sigma^2_{\mathcal{B}}\} - 0 \; sin(\mathcal{W}_{ij}) = \exp\{-\sigma^2_{\mathcal{B}}\} cos(\mathcal{W}_{ij}) \tag{A.16}$$

$$\mathbb{E}\{sin(\hat{\mathcal{W}}_{ij})\} \overset{(A.10,A.14)}{=} sin(\mathcal{W}_{ij}) \exp\{-\sigma^2_{\mathcal{B}}\} + 0 \; cos(\mathcal{W}_{ij}) = \exp\{-\sigma^2_{\mathcal{B}}\} sin(\mathcal{W}_{ij}) \tag{A.17}$$

That is:

$$\mathbb{E}\{\Delta\Theta^k_{ij}\} = \exp\{-\sigma^2_{\mathcal{B}}\} \begin{bmatrix} \cos(\mathcal{W}_{ij}), \sin(\mathcal{W}_{ij}) \\ -\sin(\mathcal{W}_{ij}), \cos(\mathcal{W}_{ij}) \end{bmatrix} \Theta^k_j - \Theta^k_i \tag{A.18}$$

From this it follows that:

$$\begin{aligned} &\mathbb{E}(\Theta^{k+1}_i - \Theta^k_i) \\ &= \alpha \sum_{a_j \in N_i} \frac{1-(1-\lambda)^{|N_j|}}{|N_j|} \frac{1-(1-\lambda)^{|N_i|}}{|N_i|} \{\exp\{-\sigma^2_{\mathcal{B}}\} \begin{bmatrix} \cos(\mathcal{W}_{ij}), \sin(\mathcal{W}_{ij}) \\ -\sin(\mathcal{W}_{ij}), \cos(\mathcal{W}_{ij}) \end{bmatrix} \Theta^k_j - \Theta^k_i\} \end{aligned} \tag{A.19}$$

On the other hand, given each agent a_i's orientation estimate Θ_i at k^{th} iteration, it is straight forward to develop the value of the partial derivative of objective f_o with respect to

Θ_i at k^{th} iteration:

$$\frac{\partial f_o}{\partial \Theta_i}^k = \alpha \sum_{a_j \in N_i} \frac{1}{|N_i|} \frac{1}{|N_j|} \{\Theta_i^k - \begin{bmatrix} \cos(\mathcal{W}_{ij}), \sin(\mathcal{W}_{ij}) \\ -\sin(\mathcal{W}_{ij}), \cos(\mathcal{W}_{ij}) \end{bmatrix} \Theta_j^k\} \tag{A.20}$$

By rearranging the equation A.19, we have:

$$\mathbb{E}(\Theta_i^{k+1} - \Theta_i^k) = -\mu\alpha\{\frac{\partial f_o}{\partial \Theta_i}^k + \gamma_\Theta^k\}$$

In which:

$$\mu = 1 - (1 - \lambda)^{|N_i|} \tag{A.21}$$

(A.22)

$$\gamma_\Theta^k = \sum_{a_j \in N_i} \frac{1 - \exp\{-\sigma_{\mathcal{B}}^2\}\{1 - (1 - \lambda)^{|N_j|}\}}{|N_j|} \begin{bmatrix} \cos(\mathcal{W}_{ij}), \sin(\mathcal{W}_{ij}) \\ -\sin(\mathcal{W}_{ij}), \cos(\mathcal{W}_{ij}) \end{bmatrix} \Theta_j^k - \frac{(1 - \lambda)^{|N_j|}}{|N_j|} \Theta_i^k$$

7. Proof of Lemma 7.2

. At $k+1^{th}$ iteration, assume each agent a_i's orientation estimate has converged to a stable value θ_i, given each agent a_i's position estimate $\mathcal{X}_i^k$ at k^{th} iteration, the agent a_i's expected position estimate update is:

$$\mathbb{E}(\mathcal{X}_i^{k+1} - \mathcal{X}_i^k) = \beta \sum_{a_j \in N_i} \frac{1-(1-\lambda)^{|N_j|}}{|N_j|} \frac{1-(1-\lambda)^{|N_i|}}{|N_i|} \mathbb{E}(\Delta \mathcal{X}_{ij}^k) \tag{A.23}$$

In which:

$$\Delta \mathcal{X}_{ij}^k = \mathcal{X}_j^k - \frac{\hat{d}_{ij}}{2} \begin{bmatrix} \cos\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \cos\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \\ \sin\left(\hat{\mathcal{B}}_{ij} + \theta_i\right) - \sin\left(\hat{\mathcal{B}}_{ji} + \theta_j\right) \end{bmatrix} - \mathcal{X}_i^k \tag{A.24}$$

According to the agent model stated in Section II.A., $\hat{d_{ij}} = d_{ij} + \epsilon_d$, $\hat{\mathcal{B}_{ij}} = \mathcal{B}_{ij} + \epsilon_{\mathcal{B}}$, and $\hat{\mathcal{B}_{ji}} = \mathcal{B}_{ji} + \epsilon_{\mathcal{B}}$, where $\epsilon_d \sim \mathcal{N}(0, \sigma_d)$, $\epsilon_{\mathcal{B}} \sim \mathcal{N}(0, \sigma_{\mathcal{B}})$. The equation A.24 can be rewritten as:

$$\begin{aligned} \Delta \mathcal{X}_{ij}^k = \mathcal{X}_j^k - \frac{d_{ij}}{2} \begin{bmatrix} \cos(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}} + \theta_i) - \cos(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}} + \theta_j) \\ \sin(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}} + \theta_i) - \sin(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}} + \theta_j) \end{bmatrix} \\ - \frac{\epsilon_d}{2} \begin{bmatrix} \cos(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}} + \theta_i) - \cos(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}} + \theta_j) \\ \sin(\mathcal{B}_{ij} + \epsilon_{\mathcal{B}} + \theta_i) - \sin(\mathcal{B}_{ji} + \epsilon_{\mathcal{B}} + \theta_j) \end{bmatrix} - \mathcal{X}_i^k \end{aligned} \tag{A.25}$$

Next, similar to equation A.16 and equation A.17, we have:

(A.26)

$$\mathbb{E}\{cos(\mathcal{B}_{ij}+\theta_i+\epsilon_{\mathcal{B}})\} \overset{(A.13)}{=} cos(\mathcal{B}_{ij}+\theta_i) \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\} - 0\ sin(\mathcal{B}_{ij}+\theta_i) = \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\} cos(\mathcal{B}_{ij}+\theta_i)$$

(A.27)

$$\mathbb{E}\{sin(\mathcal{B}_{ij}+\theta_i+\epsilon_{\mathcal{B}})\} \overset{(A.13)}{=} sin(\mathcal{B}_{ij}+\theta_i)\exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}+0\ cos(\mathcal{B}_{ij}+\theta_i) = \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}sin(\mathcal{B}_{ij}+\theta_i)$$

From this it follows that:

$$\mathbb{E}\{\Delta\mathcal{X}_{ij}^k\} \overset{(A.26,A.27,A.25)}{=} \mathcal{X}_j^k - \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}\frac{d_{ij}}{2}\begin{bmatrix}\cos(\mathcal{B}_{ij}+\theta_i)-\cos(\mathcal{B}_{ji}+\theta_j)\\ \sin(\mathcal{B}_{ij}+\theta_i)-\sin(\mathcal{B}_{ji}+\theta_j)\end{bmatrix} - \mathbb{E}\{\frac{\epsilon_d}{2}\begin{bmatrix}\cos(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\cos(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\\ \sin(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\sin(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\end{bmatrix}\} - \mathcal{X}_i^k \tag{A.28}$$

It is worth noting that in Section II.A. we assume that $\epsilon_{\mathcal{B}}$ and ϵ_d are independent, therefore:

$$\mathbb{E}\{\frac{\epsilon_d}{2}\begin{bmatrix}\cos(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\cos(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\\ \sin(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\sin(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\end{bmatrix}\} = \mathbb{E}\{\frac{\epsilon_d}{2}\}\mathbb{E}\{\begin{bmatrix}\cos(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\cos(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\\ \sin(\mathcal{B}_{ij}+\epsilon_{\mathcal{B}}+\theta_i)-\sin(\mathcal{B}_{ji}+\epsilon_{\mathcal{B}}+\theta_j)\end{bmatrix}\} = 0 \tag{A.29}$$

That is:

$$\mathbb{E}\{\Delta\mathcal{X}_{ij}^k\} \overset{(A.28,A.29)}{=} \mathcal{X}_j^k - \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}\frac{d_{ij}}{2}\begin{bmatrix}\cos(\mathcal{B}_{ij}+\theta_i)-\cos(\mathcal{B}_{ji}+\theta_j)\\ \sin(\mathcal{B}_{ij}+\theta_i)-\sin(\mathcal{B}_{ji}+\theta_j)\end{bmatrix} - \mathcal{X}_i^k \tag{A.30}$$

Substitute equation A.30 in equation A.23, we have:

$$\mathbb{E}(\mathcal{X}_i^{k+1} - \mathcal{X}_i^k) \overset{(A.30,A.23)}{=} \beta \sum_{a_j \in N_i} \frac{1-(1-\lambda)^{|N_j|}}{|N_j|} \frac{1-(1-\lambda)^{|N_i|}}{|N_i|} \{\mathcal{X}_j^k - \exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\} \frac{d_{ij}}{2} \begin{bmatrix} \cos(\mathcal{B}_{ij}+\theta_i) - \cos(\mathcal{B}_{ji}+\theta_j) \\ \sin(\mathcal{B}_{ij}+\theta_i) - \sin(\mathcal{B}_{ji}+\theta_j) \end{bmatrix} - \mathcal{X}_i^k\} \tag{A.31}$$

On the other hand, given each agent a_i's orientation estimate $\mathcal{X}_i$ at k^{th} iteration, it is straight forward to develop the value of the partial derivative of objective f_p with respect to $\mathcal{X}_i$ at k^{th} iteration:

$$\frac{\partial f_p}{\partial \mathcal{X}_i}^k = \beta \sum_{a_j \in N_i} \frac{1}{|N_i|} \frac{1}{|N_j|} \{\mathcal{X}_i^k + \frac{d_{ij}}{2} \begin{bmatrix} \cos(\mathcal{B}_{ij}+\theta_i) - \cos(\mathcal{B}_{ji}+\theta_j) \\ \sin(\mathcal{B}_{ij}+\theta_i) - \sin(\mathcal{B}_{ji}+\theta_j) \end{bmatrix} - \mathcal{X}_j^k\} \tag{A.32}$$

Rearrange the equation A.31, we have:

$$\mathbb{E}(\mathcal{X}_i^{k+1} - \mathcal{X}_i^k) \overset{(A.31,A.32)}{=} -\mu\beta\{\frac{\partial f_p}{\partial \mathcal{X}_i}^k + \gamma_{\mathcal{X}}^k\} \tag{A.33}$$

In which:

$$\mu = 1 - (1-\lambda)^{|N_i|} \tag{A.34}$$

$$\gamma_{\mathcal{X}}^k = \sum_{a_j \in N_i} \frac{1-\exp\left\{-\frac{\sigma_{\mathcal{B}}^2}{2}\right\}\{1-(1-\lambda)^{|N_j|}\}}{2|N_j|} \begin{bmatrix} \cos(\mathcal{B}_{ij}+\theta_i) - \cos(\mathcal{B}_{ji}+\theta_j) \\ \sin(\mathcal{B}_{ij}+\theta_i) - \sin(\mathcal{B}_{ji}+\theta_j) \end{bmatrix} d_{ij} + \frac{(1-\lambda)^{|N_j|}}{|N_j|}\{\mathcal{X}_j^k - \mathcal{X}_i^k\} \tag{A.35}$$

8. Proof of Proposition 8.1

Proposition 8.1 suggests that: if a bounded non-negative objective is non-increasing and will strictly decrease with a non-zero probability when it is not zero, then it will almost surely converge to zero. Without loss of generality, let $J(t_0) = C$, we have:

$$\lim_{n\to\infty} Pr\{J(t_0 + n\tau) = 0\} \geq \lim_{n\to\infty} \sum_{k=C}^{n} \binom{n}{k} (\epsilon)^k (1-\epsilon)^{n-k}$$

Hence, $\lim_{n\to\infty} Pr\{J(t_0 + n\tau) = 0\} = 1$

9. Proof of Lemma 8.1

Lemma 8.1 suggests that the algorithm can guarantee that when an agent changes the waypoint it is moving to, there is no other agent moving to this waypoint, or staying at this waypoint, at the same time. We prove Lemma 8.1 by contradiction. Let Δt be $\frac{2}{f_{comm}}$, according to the Alg. 11, in order to allow agent a_i to change wp_{a_i} from $wp^0_{a_i}$ to $wp^1_{a_i}$ at time t^*, the following conditions must hold:

- *Condition 1:* For each message msg that a_j received between $t^* - \Delta t$ and t^*, msg should **not** trigger the two conditions stated in Alg. 11 Line 26, 28;
- *Condition 2:* For all the times $t \in (t^* - \Delta t, t^*]$, $wp_{a_i}(t) = wp^0_{a_i}$, as a_i needs to wait at $wp^0_{a_i}$ for more than Δt amount of time before changing wp_{a_i} (Alg. 11 Line 30).

For any other agent $a_j \neq a_i$, $wp^1_{a_j}$ denotes the $wp_{a_j}(t^*)$ and $wp^0_{a_j}$ denotes the waypoint that a_j is claiming prior moving to $wp^1_{a_j}$, moreover, we use $t_j^{0\to1}$ to denote the time that a_j changes wp_{a_j} from $wp^0_{a_j}$ to $wp^1_{a_j}$. Suppose that there is an agent a_j such that $wp^1_{a_j} = wp^1_{a_i}$. We here exhaustively outline two possible cases:

Case 1. $t_j^{0\to1} \leq t^* - 0.5\Delta t$: This case suggests that for all the times $t \in [t^* - 0.5\Delta t, t^*]$, $wp_{a_j}(t) = wp^1_{a_i}$. On the other hand, since a_j transmits the messages at fixed frequency f_{comm}, during time span $[t^* - 0.5\Delta t, t^*]$, it will transmit at least one message to the neighbors, as a result, a_i will receive a message that triggers Alg. 11 Line 26 during $[t^* - 0.5\Delta t, t^*]$, which contradicts to *Condition 1*.

Case 2. $t^* - 0.5\Delta t < t_j^{0\to1} \leq t^*$: By *Condition 2*, in this case we have: For all the times $t \in (t^* - \Delta t, t^* - 0.5\Delta t]$, it holds that $wp_{a_j}(t) = wp^0_{a_j}$ and $wp_{a_i}(t) = wp^0_{a_i}$. Since we assume $wp^1_{a_i} = wp^1_{a_j}$, we can conclude that these two agents intend to go the same waypoint, i.e., $wp^1_{a_i}$, during $(t^* - \Delta t, t^* - 0.5\Delta t]$. On the other hand, the period of time between $t^* - \Delta t$

and $t^* - 0.5\Delta t$ is long enough that a_i, a_j will have communicated *next_step* to each other. Given the fact that a_j changes wp_{a_j} after receiving a_i's *next_step*, one can conclude that a_j has higher priority to move to the $wp^1_{a_j}$, as a result, the message transmitted by a_j during $(t^* - \Delta t, t^* - 0.5\Delta t]$ will trigger Alg. 11 Line 28 on a_i, which contradicts the *Condition 1*, completing the proof.

10. Proof of Theorem 8.2

We prove Theorem 8.2 by contradiction. Suppose that at time t^*, there are two agents a_i, a_j traveling on the edge connecting waypoint wp^0, wp^1 in the opposite direction. Without loss of the generality, we assume a_i is moving from wp^0 to wp^1 and a_j is moving from wp^1 to wp^0. Let $t_i^{0\to1}$ be the time that a_i changes wp_{a_i} from wp^0 to wp^1 and $t_j^{1\to0}$ be the time that a_j changes wp_{a_j} from wp^1 to wp^0, we have two cases:

Case 1. $t_i^{0\to1} \neq t_j^{1\to0}$: This case contradicts to Lemma 8.1, as the agent who changes its wp later will change its wp to a waypoint that is already claimed by the other.

Case 2. $t_i^{0\to1} = t_j^{1\to0}$: By Alg. 11 Line 30, before $t_i^{0\to1}$, there is sufficient time for each of these two agent to sense that its *next_step* is claimed by the other, as a result, Alg. 11 Line 26 will be triggered, and the agents' wps will not change, where contradiction occurs.

11. Proof of Lemma 8.2

At any time t, we can always find a minimal rectangle that covers all agent positions and goal positions. Let cx_t, cy_t be the length of the rectangle's edge in the x and y direction, respectively, and let c_t be $cx_t + cy_t$. Using the fact that each agent moves to its goal point greedily, we have $\forall t_1, t_2$, if $t_1 \leq t_2$, then $c_{t_1} \geq c_{t_2}$, in the other words, $\forall t, c_t \leq c_{t0}$. Note that c_t is always larger than or equal to the Manhattan distance between any pair of points in the rectangle at time t, therefore, we have $J_1 \leq |A| c_{t0}$. On the other hand, since at least one goal will be assigned to the swarm, we have $J_2 \leq |A| - 1$, which completes the proof.

12. Proof of Lemma 8.3

Recall the way that the *new goal selector* works: if two agents realize that they are holding the same goal, then only one of them will select a new goal while the other agent will keep holding the current goal. As a result, no matter whether the new goal is valid or not, J_2 will never increase.

On the other hand, if $J_2 = 0$, then the *new goal selector* will no longer be triggered, so the robots will move greedily toward their goals. Thus when $J_2 = 0$, J_1 will never increase.

13. Proof of Lemma 8.4

If $(v_i, v_j) \in \mathcal{E}_t$, then the goal swap between a_i and a_j will not increase $d_i + d_j$. By Alg. 13 (Line 16-21), the goal swap between a_i and a_j will happen with a non-zero probability, completing the proof.

14. Proof of Lemma 8.5

We prove Lemma 8.5 via case analysis. If $J_1(t) \neq 0$, then at least one agent intends to move to the next waypoint, there are three possible cases:

- ***Case 1. The graph $\mathcal{G}_t$ is cyclic:***

Assume that at time t^0, there exists a cycle $\mathcal{C}$ on $\mathcal{G}_{t^0}$. Let $len(\mathcal{C})$ be the length of $\mathcal{C}$, i.e. the number of the agents that are on cycle $\mathcal{C}$ at time t^0. We use $a^1, a^2, ..., a^{len(\mathcal{C})}$ to denote the agents that are on $\mathcal{C}$ at time t^0, for the sake of description, we order these agents in the way such that:

  - If $i < len(\mathcal{C})$, then $(a^i, a^{i+1}) \in \mathcal{E}_{t^0}$,
  - If $i = len(\mathcal{C})$, then $(a^i, a^1) \in \mathcal{E}_{t^0}$.

Note that it does not matter how we pick the first agent a^1, a^1 could be any agent that is on cycle at time t^0.

We show that Lemma 8.5 holds in case 1 by contradiction.

Suppose that $\forall \tau > 0, Pr\{J_1(t_0+\tau) \leq J_1(t_0)-1\} = 0$, i.e., the probability that J_1 strictly decreases after t_0 is 0. One can conclude that if the assumption is true, then:

- No agent can change its position after t_0, as J_1 will decrease every time when any agent moves, moreover,
- No agent can execute the goal swaps that can decrease J_1.

In the other words, the only two possible actions left for agents are: doing nothing, or executing the goal swaps that cannot change the J_1 (Alg. 13 Line 16-21). It is worth noting that if these two actions are the only ones available for agents, then each time when an

agent tries to decide an action to execute, the action "doing nothing" will be picked with a non-zero probability, as stated in Alg. 13 (Line 16-21).

Next, let q^* be a^1's goal at t^0, combing Lemma 8.4 and the conclusion we just obtained, we have that the following will happen with a non-zero probability:

- q^* propagates among the agents $a^1, ..., a^{len(\mathcal{C})}$ via **swap** in the order that: $a^1 \rightarrow a^{len(\mathcal{C})} \rightarrow a^{len(\mathcal{C})-1} \rightarrow ... \rightarrow a^2$.
- For any agent $a^i \neq a^1$ that is on $\mathcal{C}$ at time t^0, it will **do nothing** but wait to take q^* from its successor (via goal swap) and then pass q^* to its predecessor (via goal swap).

That is, q^* will traverse agents $a^1, ..., a^{len(\mathcal{C})}$ in the opposite direction of $\mathcal{C}$ with a non-zero probability, an graphical illustration of this event is shown in Fig. A.2.

Let t^1 be the time that q^* is passed to a^2, we have $J_2(t^1) \leq J_2(t^0) - len(\mathcal{C})$, because the owners of goals of all the agents that are on $\mathcal{C}$ have moved one step closer to goal via goal swap, see Fig. A.2 for a more intuitive illustration. This observation suffices to show that it is incorrect to assume that the probability that J_1 strictly decreases after t_0 is 0, completing the proof in case 1.

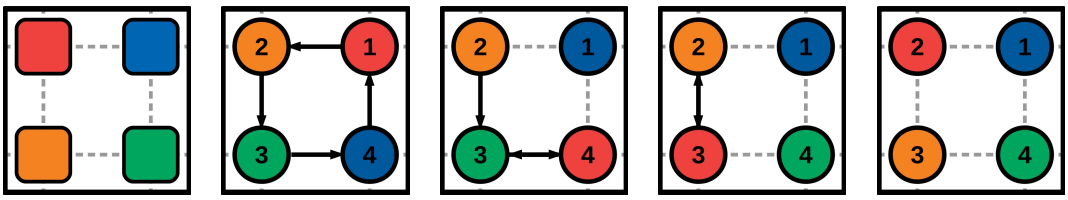

Figure A.2. From left to right: A shape that contains 4 goal points; a sequence of events where J_1 strictly decreases. The frames of this sequence of events are ordered from left to right. All the information is encoded in the same way as Fig. 8.2.

On the other hand, if the $\mathcal{G}_t$ is acyclic, we can find at least one *head* agent, i.e., the agent that is not blocked by any other agent. The *head* agent(s) can either be an agent that already arrived at the goal, or an agent that is still moving to the goal.

- ***Case 2. The graph $\mathcal{G}_t$ is acyclic, moreover, there exists one head agent a_i that has not arrived at its goal $T_{a_i}(t)$ yet:***

If at least one *head* agent has not arrived at its goal yet, J_1 will decrease by one because its next waypoint is open, thus Lemma 8.5 holds in case 2.

- ***Case 3. The graph $\mathcal{G}_t$ is acyclic, moreover, all the head agents have arrived at their goals:***

If all the *head* agents have already arrived at their goals, then there is at least one non-*head* agent who is blocked by a *head* agent. Using Lemma 8.4, we have that the blocked agent can make progress with a non-zero probability by swapping with the blocking *head* agent that has already arrived at its goal. With at most $|A| - 1$ swaps, the agents will either form a cycle, as shown in Fig. A.3 (*right*), which is **case 1**, or generate a *head* agent that hasn't reached its goal, shown in Fig. A.3 (*left*), which is **case 2**. Hence in **case 3**, J_1 will decrease within a finite amount of time with non-zero probability as well, completing the proof.

15. Proof of Lemma 8.6

We prove this lemma via case analysis. At time t, for any agent a, let q be $T_a(t)$, i.e., agent a's target at time t. We here exhaustively outline all two possible cases:

- ***Case 1.*** *a* ***has not arrived at*** *q:*

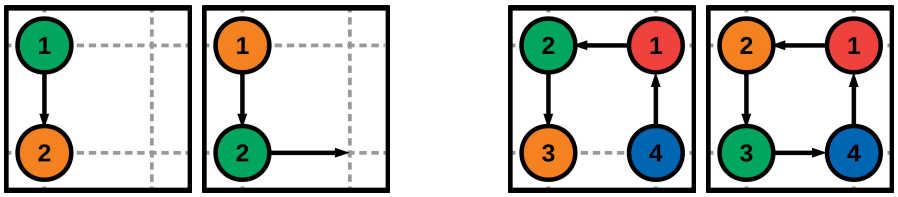

Figure A.3. Illustration of two possible cases (shown by a sequence of two figures on the left and a sequence of two figures on the right) where the *head* agent has already arrived at its goal. The goal shape is shown in Fig. A.2 (*left*). All the information is encoded in the same way as Fig. 8.2.

If a has not arrived at q, a will intend to move to the next waypoint to get closer to q. If the waypoint is unoccupied, then a moves one step closer to q. Otherwise, if the waypoint is occupied, using Lemma 8.4, q will be passed to the agent that is occupying the waypoint with non-zero probability. In either of these two cases, the distance between position of q's owner and q decreases by one with non-zero probability.

- ***Case 2.*** *a* ***is already at*** *q:*

If a is already at q, then a can stay at q with non-zero probability, completing the proof.

16. Proof of Lemma 8.7

If $J_2 \neq 0$, then at least two agents are holding the same goal point. Let q_d be a goal which is assigned to more than one agent. Lemma 8.6 is sufficient to show that two of q_d's owners can concurrently keep moving greedily to q_d with non-zero probability, shown in Fig. A.4 (*left*), unless one of q_d's owners blocks the other owner's way in the trip, shown in Fig. A.4 (*right*). However, one agent being blocked by the other implies that the distance between two agents is one grid length, which is smaller than R, completing the proof.

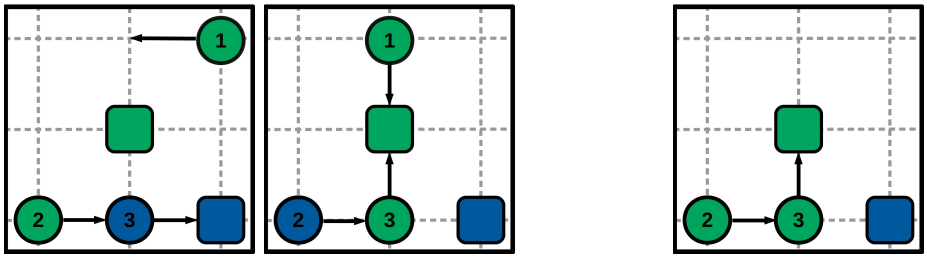

Figure A.4. Illustration of two possible cases (shown by a sequence of two figures on the left and a single figure on the right) where two of q_d's owners (circles in green) meet each other. All the information is encoded in the same way as Fig. 8.2.

17. Proof of Lemma 8.8

Lemma 8.7 suggests that if $J_2 \neq 0$, then two agents holding the same goal will meet each other within a finite amount of time with non-zero probability, which means *new goal selector* will be triggered within a finite amount of time with non-zero probability. Additionally, the when *new goal selector* (Alg. 13 Line 3-10) is triggered, all the goal points in Q will be picked with a non-zero probability, which implies that a goal that has not been assigned to swarm yet will be picked with non-zero probability. As a result, J_2 will decrease with non-zero probability, completing the proof.