THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Zohir Bouzid

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Modèles et Algorithmes pour les Systèmes Émergents

soutenue le 21 Juin 2013

devant le jury composé de :

M. Sébastien Tixeuil	Directeur de thèse
Mme Maria Potop-Butucaru	Encadrant
M. Xavier Urbain	Encadrant
M. Michel Raynal	Rapporteur
M. Jérémie Chalopin	Examinateur
M. Xavier Défago	Examinateur
M. Ralf Klasing	Examinateur
M. Pierre Sens	Examinateur

Acknowledgements

I would like to express my gratitude to my PhD supervisor Prof. Sébastien Tixeuil and to my advisors Prof. Maria Potop-Butucaru and Dr. Xavier Urbain for supervising my work and guiding my first steps in the field of scientific research. I also thank them for giving me the freedom to work independently on subjects not directly related to my PhD.

I would also like to thank my examiners, Prof. Paola Flocchini and Prof. Michel Raynal for the time they spent examining this thesis and for their encouraging feedback. I am very grateful to the members of my jury, Dr. Jérémie Chalopin, Dr. Xavier Défago, Prof. Ralf Klasing, Prof. Michel Raynal and Prof. Pierre Sens for accepting to evaluate my thesis.

My thanks and gratitude go to my coauthors and collaborators, Dr. Shantanu Das, Prof. Shlomi Dolev, Dr. Taisuke Izumi, Dr. Anissa Lamani, Dr. Pierre Sutra, Dr. Corentin Travers and Prof. Koichi Wada.

I especially thank my parents, my brother and my friends Ines, Lamia and Madjid for their constant help and support.

This work was supported by the Digiteo Île-de-France project PACTOLE 2009-38HD.

CONTENTS

Contents **iii**

1 Introduction **1**
1.1 Research Contributions 2

2 Background **5**
2.1 Model 5
2.2 Studied Problems 9
2.2.1 Weber Points 9
2.2.2 Gathering and Convergence 9
2.2.3 The RoboCast Problem 12
2.3 Notations 13

3 Weber Points **15**
3.1 Symmetries in Robot Configurations 15
3.1.1 Symmetricity 18
3.1.2 Regularity 19
3.1.3 Properties of Weber Points 21
3.2 Computation of Weber Points in Regular Configurations 23
3.2.1 Preliminaries 23
3.2.2 Detection of Even Regularity 25
3.2.3 Detection of Odd Regularity 26
3.3 Computation of Weber Points for Quasi-Regular Configurations . . . 30

4 Wait-Free Crash-Resilient Gathering **33**
4.1 Configurations 33
4.2 The Algorithm 36
4.3 Proof of Correctness 39

5 Byzantine Convergence **47**

5.1 Preliminaries . . . 48
5.1.1 Cautious Algorithms . . . 48
5.1.2 Equivalence of Configurations . . . 48
5.1.3 Invariants . . . 50
5.2 Necessity of Strong Multiplicity Detection . . . 52
5.3 Lower Bound on the Number of Faulty Robots in the ATOM[FS] Model 54
5.4 Lower Bound on the Number of Faulty Robots in the ATOM[B(k)] Model . . . 54
5.5 Lower Bound on the Number of Faulty Robots in the ATOM[AS] model 55
5.6 Necessary and Sufficient Conditions for Deterministic Convergence 66
5.6.1 Shrinking algorithms . . . 66
5.6.2 Cautious algorithms . . . 68
5.7 Deterministic Convergence in ATOM[FS] Networks . . . 70
5.7.1 Algorithm 5 is Cautious . . . 70
5.7.2 Algorithm 5 is Shrinking . . . 71
5.8 Deterministic Convergence in NTOM[B(k)] Networks . . . 71
5.8.1 Algorithm 6 is Cautious . . . 73
5.8.2 Algorithm 6 is Shrinking . . . 74
5.9 Deterministic Convergence in NTOM[AS] Networks . . . 80
5.9.1 Algorithm 7 is Cautious . . . 82
5.9.2 Algorithm 7 is Shrinking . . . 83

6 RoboCast **91**
6.1 Introduction . . . 91
6.2 RoboCasting the Local Coordinate System: Two Robots Networks . . 91
6.2.1 Line RoboCast . . . 92
6.2.2 Composing RoboCast . . . 101
6.2.3 RoboCast of the Local Coordinate System . . . 102
6.3 RoboCasting the Local Coordinate System: n-Robots Networks . . . 104
6.4 Avoinding Collisions . . . 111
6.5 Applications . . . 115
6.5.1 Asynchronous Deterministic 2-Gathering . . . 115
6.5.2 Asynchronous Stigmergy . . . 115

7 Conclusion **117**

Bibliography **121**

CHAPTER 1

INTRODUCTION

Distributed systems are playing an increasingly important role in everyday life. Hence, it becomes necessary to base them on solid theoretical ground. The theory of distributed computing addresses this challenge by defining formal models of computation, determining which classes of problems are computable in these models and at what cost in terms of consumed time and resources. Computation models may vary following several parameters like:

- The communication medium (message passing, shared memory, vision ...).
- The temporal model (synchronous, partially synchronous, asynchronous ...).
- The nature of tolerated failures (crash, malicious ...).

Often, changing a single parameter of the model greatly impacts its computational power. Hence, understanding the limiting power of these parameters and their combined effect is a challenging task. Despite important achievements in this field, much remains to be done. Our research attempts to contribute to this goal.

We focus on distributed algorithms for networks of mobile robots which are mobile entities that communicate only through the vision of their positions. In order to use low-cost robots, it is important to study what are the minimal capabilities that robots need to have in order to solve important distributed problems.

Our approach is purely theoretical, it consists of the formal statement of the studied distributed problems and of the mathematical proof of the minimal conditions under which they can be solved. The type of questions we try to answer is, for example, can we solve a given distributed computing problem when the network is asynchronous or when some processes may fail by crashing? and if so, is it possible to do it when robots have no memory ?

A robot network consists of a finite set of very weak mobile entities that are allowed to move in the plane. They cannot communicate *directly* by sending messages to each others. Instead, their communication is *indirect (or spatial)*: a robot writes a value to the network by moving toward a certain position, and a robot reads the state of the network by observing the positions of other robots in terms of its own local coordinate system. In the standard model, robots are anonymous and oblivious: they cannot remember past computations, observations or movements. Hence, as with communication, memory is *indirect*. Since any algorithm must use a kind of memory, resolving problems in the context of robot networks is the art of making them "remember without memory" [25].

Robots are also non-oriented (i.e. they share neither a common coordinate system nor a common length unit). Dealing with the lack of a common notion of spacial information (which results from the disorientation of robots) is at the heart of the design of distributed robot algorithms. Impossibility results, lower bounds, and proofs of correctness are most of the time of geometric nature. Indeed, the effective solvability of a given problem usually translates into the design of adequate spacial invariants that are preserved by robot movements induced by the algorithm. These spacial invariants are in some sense a kind of *indirect* memory which compensate the lack of persistent memory for robots.

1.1 Research Contributions

In this thesis we present the following contributions:

Weber Points. One of the most famous invariants for robot networks algorithms is the Weber point. Given a multiset of points P that are not collinear, the Weber point minimizes the sum of distances over all points of P. It has the key property of being unique for non-collinear points and of remaining unchanged under straight movements of any of the points towards or away from it. Hence, it provides a simple protocol for robots to meet in a single point: simply move towards the Weber point which remains invariant as a result of movements of robots towards it. Unfortunately, computing the Weber point is known to be difficult and was solved only in special cases such as regular polygons [2] and lines [14]. A key result of this thesis is a technique to

compute the Weber point for many configurations [13, 7] that exhibit some kind of symmetry (Chapter 3).

Crash-Resilient Gathering. Gathering and Convergence are two fundamental agreement primitives in robot networks and can serve as a basis in the implementation of a broad class of services. Gathering requires robots to reach a single point within finite time regardless of their initial positions while convergence only requires robots to get close to single point. The second contribution of this thesis is a protocol that solves the gathering problem when robots may incur *any number* of permanent crash failures (Chapter 4). Our protocol [7] relies mainly on our Weber point computation algorithms.

Byzantine-Resilient Convergence. Harder than crash faults, byzantine faults occur when robots start behaving in completely arbitrary way. Tackling them is much more challenging. In this thesis [9, 10, 11, 12] we study under what conditions relating the synchrony of the networks and the number of tolerated byzantine faults, convergence becomes possible to solve in unidimensional robot networks (Chapter 5).

Non-Oblivious Robot Networks. Many fundamental problems in oblivious robot networks are impossible to solve without additional assumptions [32]. It is thus interesting to investigate the power of robots that are endowed with (possibly bounded) memory. This is the object of the last contribution of this thesis [8] in which we implement RoboCast, an all-to-all communication primitive for non-oblivious robots that allow them to send simultaneously to each others any binary message (Chapter 6).

The following chapter gives a formal description of the model of computation and the studied problems. It also compares our results with related works to emphasize our contributions.

CHAPTER 2

BACKGROUND

In this chapter, we firstly present the formal model of robot networks that we use in this thesis (Section 2.1). Then we give a formal description of the studied problems together with the current state of the art regarding these problems (Section 2.2). Finally, we conclude with some notations (Section 2.3) that will be further used in the following chapters.

2.1 Model

Our model is based on the models of Suzuki-Yamashita [34] and Prencipe [33]. The system consists of a set $\mathscr{R}$ of n mobile robots $\{r_1, \ldots, r_n\}$ that communicate only through vision. That is, robots are devoid of any mean of direct communication, the only way of them to communicate is by observing the positions of their peers (a "read") and by moving in the plane (a "write"). Robots do **not obstruct** the vision or movement of other robots. Their visibility is **unlimited**, *i.e.* they are able to sense the entire set of robots. Robots are **anonymous** and **homogeneous**: they do not have identifiers, they are indistinguishable by their appearance and they run the same algorithm. Indices of robots are not known to robots which cannot employ them in their algorithms, we use them to simplify the presentation.

Execution Model. Each robot executes an infinite sequence of **cycles** of three **phases** Look, Compute and Move. During the Look phase, the robot takes a snap-

shot of the environment using its visual sensors. Then it calculates a destination in the Compute phase. Unless the robot is non-oblivious, the chosen destination is based only on the snapshot obtained in the previous Look phase. The chosen destination is based solely on the snapshot it obtained in the preceding phase. Finally, during the Move phase, the robot moves toward its computed destination. A robot can be stopped in its Move phase before it reaches its destination. However, to each robot r_i is associated a constant δ_i such that at every cycle, r_i is guaranteed to move a distance of at least δ_i towards its destination before it can be stopped. Define Δ to be equal tot $\max(\delta_i : r_i \in \mathscr{R})$. We assume that the activations of robots and their movements are controlled by a fictitious entity called the **scheduler** or **adversary**.

Memory. It is generally assumed that robots are **oblivious** in the sense that they have no persistent memory of their past observations, computations and movements. Therefore, the actions they perform and the destinations they choose at every cycle are based entirely on the last observed configuration of the network. Anything that may have happened in the past is forgotten and every cycle is considered by robots as the initial cycle of the system. The network memory is implicit, it is generally inferred from the current configuration. This assumption has two consequences. The first - happy - is that the algorithms designed within this framework are by definition self-stabilizing. The second - unfortunate - is that the lack of memory makes it impossible to solve most of the interesting problems. It is therefore interesting to study robots which are equipped with a memory (potentially **bounded**) to understand the influence of this parameter in the power of the model.

Orientation. Robots are **non-oriented**, that is, each robot has its own local coordinate system with its own origin, axis, and unit of length which may be different from those of other robots. However, robots can share some aspects of their coordinate systems. When they have the same notion of clockwise orientation or handedness, we say that they share the same **chirality**. The local coordinate system of *oblivious* robots may completely change at the beginning of each cycle, however, it remains invariable during the cycle. In contrast, the local coordinate system of a *non-oblivious* robot is assumed to be fixed during the whole run unless it is explicitly modified by the corresponding robot as a result of a computation. We say in this case that robots *remember* their own coordinate systems.

Multiplicity Detection. Each robot is modeled as a point *without volume* on a geometric plane. Thus multiple robots may lie in the same location forming a **multiplicity point**.

When robots are able to detect multiplicity points during their Look phase, we say that they are endowed with the **multiplicity detection** capability. We consider the following three variants of this feature:

- Robots can be endowed with **strong** multiplicity detectors, denoted by $\diamondsuit M$, that allow them to detect the exact number of robots that may simultaneously occupy the same location.
- **Weak** multiplicity detectors, referred to as $?M$, allow robots to detect whether there is one or more robots that are located in some position.
- For ease of presentation, when robots do not have any multiplicity detection capability, we say that they are endowed with a **null** multiplicity detector, denoted hereafter by $\varnothing M$.

Atomicity. The scheduler can be either **atomic** or **non-atomic**. An atomic scheduler, denoted ATOM, activates at each cycle a subset of robots. Upon their activation, the selected robots execute a complete Look-Compute-Move cycle synchronously and in a lock-step manner. Hence, the robots that are activated at the same cycle compute their destinations based on the same picture of the environment.

Under a non-atomic scheduler, denoted NTOM, the phases of the different robots are independent and are not started necessarily at the same time. Moreover, the duration of the different phases are not constant. Each phase of each robot can take an *arbitrary but finite* time. It is possible that for instance a robot executes its Look phase while another robot performs its Move phase, or that a robot executes its Compute phase while its view (obtained during the Look phase) is already outdated. Therefore, a robot can compute its destination based on an outdated information about its environment. Indeed, between the time it finishes its Look phase and the end of its Compute phase, the other robots may have moved and been activated several times.

Observe that ATOM $\subset$ NTOM. That is, ATOM schedules are a strict subset of the schedules allowed by NTOM. Hence, any algorithm designed for the ATOM model works also correctly under NTOM. On the other hand, impossibility results that are proven in the ATOM model still hold in the NTOM model.

Synchrony. Complementary to the atomicity of robots action is the amount of *synchrony* of the scheduler. All the schedulers in this thesis are **fair** *i.e.* each robot

is activated infinitely often in any infinite execution. We consider the following synchrony properties:

- A **fully synchronous** scheduler, denoted FS, operates *all* robots in a lock-step manner forever (hence, it is atomic by definition).
- A k-**bounded** scheduler, denoted B(k), guarantees that between any two successive activations of the same robot, no other robot can be activated more than k times. Hence, it preserves a ratio of k between the most often activated robot and the least often activated robot.
- A **fully asynchronous** scheduler, denoted AS, does not give any guarantee about the activation of robots (apart from fairness).

Observe that FS $\subset$ B(k) $\subset$ AS.

To combine atomicity and synchrony assumptions of schedulers, we refer to them using the following notation $X(Y)$ where $X \in \{\texttt{ATOM}, \texttt{NTOM}\}$ and $Y \in \{\texttt{FS}, \texttt{B}(k), \texttt{AS}\}$. When the reference to Y is omitted, the scheduler is AS by default.

Our notation is not standard, but we found it easier to present the results of this thesis using a notation that decorrelates between the degrees of atomicity and synchrony of the considered scheduler. In the literature, ATOM(FS) is called FSYNCH [26], ATOM(AS) is referred to as either SYM [34] or SSYNCH [26] and NTOM(AS) is traditionally called CORDA [33] or ASYNCH [26].

Faults. The faults that we investigate fall in two categories:

- **Crash faults.** Here, a faulty robot stops executing its cycle forever but it remains visible to other robots.
- **Byzantine faults.** A faulty robot may exhibit arbitrary and unpredictable behavior and movement. When analyzing the correctness of byzantine-resilient algorithms, we assume that the byzantine robots are under the control of the adversary that does everything it can in order to deceive correct robots and make the algorithm fail. Of course, the byzantine fault model encompasses the crash fault model, and is thus harder to address.

The maximum number of the robots that can fail during an execution is indicated by the parameter $f \in [0, n]$. A robot that never incur a fault is said **correct**. The number of correct robots is denoted by m with $n = m + f$. To simplify the proofs, we assume that the indices of correct robots run from 1 to m. A protocol that can handle up to $f = n - 1$ faults is called **wait-free**.

2.2 Studied Problems

This section presents the formal definition of the studied problems followed by a description of related works for each of them.

2.2.1 Weber Points

Definition 1. *The Weber points of a multiset of points P, denoted $WP(P)$, are those points that minimize the sum of distances with points of P. Formally,*

$$WP(P) = \underset{x \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{i=1}^{n} |x, p_i|$$

The Weber point has the key property of remaining unchanged under straight movements of any of the points towards or away from it. Observe that this is not true for the center of gravity of robots which changes as a result of the movement of any robot towards it. Hence, Weber points can be used as a geometric invariant in robots networks algorithms. For example, it is simple to devise a robot protocol that solves gathering: all robots simply move towards the Weber point. Unfortunately, it has been shown [6] that the Weber point is in general not solvable by a formula involving only the usual algebraic operations (addition, subtraction, multiplication, division) and radicals (square roots, cube roots, ...). The problem of finding Weber points has been solved up to now for only a few specific configuration of points (e.g. linear [14] or biangular [2] configurations). A key contribution of our thesis (Chapter 3) is to present techniques for computing the Weber point of a large class of configurations that exhibit some kind of symmetry.

2.2.2 Gathering and Convergence

Definition 2. *The* gathering *problem requires* correct *robots to reach the same, but unknown beforehand, location within finite time regardless of their initial positions. In the weaker* convergence *problem,* correct *robots are only required to asymptotically approach the same position and not necessarily join it.*

Formally, given any initial configuration, convergence requires the existence a point c such that for every $\epsilon > 0$, there exists a time τ_ϵ such that $\forall \tau > \tau_\epsilon$, all correct robots are within a distance of at most ϵ of c at τ.

Note that the definition requires the gathering or convergence property only from *correct* robots. It is impossible to obtain the convergence of all faulty robots since crashed robots stop moving and byzantine robots may exhibit arbitrary behavior and never join the position of correct robots.

Reference	Computation Model	Time
[34]	ATOM	Unbounded
Chapter 6, Section 6.5.1	NTOM	Bounded

Table 2.1: Gathering of Two Non-Oblivious Robots

The gathering problem was introduced in a seminal paper of Suzuki and Yamashita [34] where it was solved in the ATOM model provided that robots are endowed with weak multiplicity detectors $?M$. Later, Prencipe [32] proved that this multiplicity detection capability is necessary to solve the gathering problem in ATOM (and thus in NTOM also). Cieliebak *et al.* [15] proposed a gathering solution for the NTOM model. *Deterministic* convergence was addressed by Cohen and Peleg [16, 17], where algorithms based on convergence to the center of gravity of the system are presented.

Robots with limited visibility were first studied in [4] where the authors addressed the convergence problem in ATOM model. The subsequent work of Flocchini *et al.* [27] proposed a gathering protocol for oblivious robots with limited visibility in the NTOM model where robots share the knowledge of a common direction given by a compass. Recently, Katreniak [30] solved the problem in NTOM without the common direction assumption but only for 1-bounded scheduler. In [19], the authors introduced a new model for robots with volume that are represented by two-dimensional figures rather than points and which block both the motion and visibility of other robots. In this model, they provided a gathering algorithm for three and for four robots. Robots with inaccurate sensors and movements were addressed in [18] and [28].

Gathering of Two Non-oblivious robots Deterministic gathering of two stateless robots has been proved impossible when robots have no common orientation [34]. In [34], the authors also propose a non-oblivious algorithm for deterministic gathering in the ATOM model. In Chapter 6, we extend this result to the NTOM model, using bounded memory and a limited number of movements and cycles (refer to Figure 2.1).

Crash-Resilient Gathering All known *deterministic* solutions to gathering except the two works [1, 23] require robots to be fault-free and initially located on distinct positions. Agmon and Peleg [1] solve gathering with at most one halting fault assuming that no two robots are located on the same position initially. Dieudonné and Petit [23] present a fault-free gathering algorithm (no halting fault) *starting from any initial configuration* of robots provided that their number is *odd*.

Reference	Model	Chirality	Multiplicity Detection	Halting Faults	Initial Configurations
[1]	ATOM	No	Weak (binary)	$f \leq 1$	No multiplicity points
[23]	ATOM	No	Strong	$f = 0$	n is odd
Chapter 4	ATOM	Yes	Strong	$f < n$	not bivalent

Table 2.2: Crash resilience bounds for gathering.

Reference	Problem	Lower Bound	Upper Bound
[1]	Gathering (2 D)	-	$n > 3f$ (ATOM[FS])
Chapter 5	Gathering (1 D)	$n > 2f$	$n > 2f$ (ATOM[FS]) [1]
	Convergence (1 D)	$n > 3f$ (ATOM[B(k)])	$n > 3f$ (NTOM[B(k)])
	Convergence (1 D)	$n > 5f$ (ATOM[AS]) [2]	$n > 5f$ (NTOM[AS])

Table 2.3: Byzantine resilience bounds for convergence/gathering.

Moreover, [23] show that deterministic gathering is impossible if the robots are equally distributed in two points on the plane (the so-called *bivalent* configuration). In Chapter 4, we investigate the possibility of handling more than one halting fault in robot networks in a deterministic setting. In more details, we we show how to solve gathering, when *any number* of robots may crash at any time during the algorithm, assuming that robots share the same chirality (that is, they agree about their handedness). We also assume, as in [23], a strong multiplicity detection mechanism. Our algorithm achieves gathering starting from any configuration (except the bivalent configuration where deterministic gathering is impossible). Our results are summarized in table 2.2.

Byzantine-Resilient Convergence *Deterministic* byzantine-resilient gathering is addressed in [1] where the authors propose an algorithm for the ATOM model with fully synchronous scheduling that tolerates up to f byzantine faults, when the total number of robots is (strictly) greater than $3f$. *Probabilistic* byzantine-resilient gathering was addressed by [21] . In this thesis, we study *deterministic gathering and convergence* in byzantine-prone environments when robots move in a *uni-dimensional* space. We prove several tight bounds that relate the resilience of the system to its degree of synchrony. Our results are presented in Table 2.3 where they are compared with [1]. More details can be found in Chapter 5.

Reference	Model	Time Complexity
[34, 35]	Non-oblivious ATOM	Unbounded
Chapter 6	Non-oblivious NTOM	Bounded

Table 2.4: State of the Art for RoboCast.

2.2.3 The RoboCast Problem

In [34, 35], Suzuki and Yamashita present an algorithm in ATOM that allows a swarm of robots endowed with *memory* to use their motion as a medium to exchange their local coordinate systems. Once robots know the coordinate systems of each others, they can use their positions as a way to encode (and thus transmit) any binary information [34, 35, 22]. This has an important consequence: using this protocol as a building block, robots can emulate any (fault-free anonymous) algorithm that runs in the message passing model in which processes communicate through sending binary messages to each others. This means that in some sense non-oblivious ATOM is stronger than the message passing model. But is this true also for non-oblivious NTOM ?

In this thesis we give a positive answer to this question. We formally specify and implement an all-to-all communication primitive, called RoboCast (see Definition 3 below), that allows non-oblivious robots to exchange various information (*e.g.* their local coordinate axes, unity of measure, *rendez-vous* points, or binary information) using only motion in a two dimensional space. Contrary to the previous solution of [34, 35], our algorithm works in the *fully asynchronous* NTOM model and uses a *bounded* number of movements (see Table 2.4). Then, we use the RoboCast primitive to solve deterministic gathering for two robots in non-oblivious NTOM.

Definition 3. *The* RoboCast *communication abstraction provides a set of robots located at arbitrary positions in a two-dimensional space the possibility to broadcast their local information to each other. The* RoboCast *abstraction offers robots two communication primitives:*

- RoboCast(m)*: sends message m to all other robots.*
- Deliver(m)*: delivers message m to the local robot.*

The message m may consists in the local coordinate system, the robot chirality, the unit of measure, or any binary coded information.

Consider a run at which each robot r_i in the system invokes *RoboCast(m_i)* at some time τ_i for some message m_i. Let τ be equal to $max\{t_1,\ldots,\tau_n\}$. Any protocol solving the RoboCast Problem has to satisfy the following two properties:

- Validity: For each message m_i, there exists a time $\tau'_i > t$ after which every robot in the system has performed *Deliver(m_i)*.
- Termination: There exists a time $\tau_T \geq max\{\tau'_1,\ldots,\tau'_n\}$ after which no robot performs a movement that causally depends on the invocations of *RoboCast(m_i)*.

2.3 Notations

A multiset or a bag S is a generalization of a set where an element can have more than one occurrence. The number of occurrences of an element a in S is referred as its *multiplicity* and is denoted by $\text{mul}_S(a)$ or simply $\text{mul}(a)$. The total number of elements of a multiset, including their repeated occurrences, is referred as the *cardinality* and is denoted by $|S|$. $\min(S)$(resp. $\max(S)$) is the smallest (resp. largest) element of S. If S is nonempty, $\text{range}(S)$ denotes the set $[\min(S),\max(S)]$ and $\text{diam}(S)$ (diameter of S) denotes $\max(S)-\min(S)$.

Denote by $\mathbb{T}$ the set of time instants. We assume that it is equal to the set of positive natural numbers.

Given robots $r_i, r_j \in \mathscr{R}$ and a time $\tau \in \mathbb{T}$, the last position of r_j observed by r_i at τ (or before) is denoted by $P^i_j(\tau)$. the last configuration observed by r_i at τ is given by the multiset $P^i(\tau) = \{P^i_1(\tau),\ldots,P^i_n(\tau)\}$. The subset of positions of correct robots is referred as $U^i(\tau) = \{U^i_1(\tau),...,U^i_m(\tau)\}$ where $U^i_j(\tau) = P^i_j(\tau)$ and $U^i(\tau) \subseteq P^i(\tau)$ (remember that m denotes the number of correct robots). $P^i(\tau)$ and $U^i(\tau)$ are *relative* to the observing robot, that is, they are expressed using the (last) local coordinate system of r_i. We shall drop the superscript i when it is obvious from context.

Let $\mathbb{P}$ be the set of all possible configurations of n robots. Formally, $\mathbb{P} = \mathbb{R}^{2n}$ where $\mathbb{R}$ is the set of real numbers. A configuration is said to be *linear* whenever all robots lie on the same line. Given any robot r, $\text{mul}(r)$ denotes the *multiplicity* of the location occupied by r, that is, the number of robots collocated with r (including r). Given $P \in \mathbb{P}$ a configuration, we denote by $\mathbb{L}(P)$ the *set* of positions in P removing multiplicities (i.e. each point in $\mathbb{L}(P)$ contains at least one robot). Let $\text{SEC}(P)$ and $\text{CH}(P)$ denote respectively the smallest enclosing circle and the convex hull of the point set P. Given a point $c \in \mathbb{R}^2$, let $\overline{P_c}$ denote the configuration that results from removing all the points that are equal to c from P.

Given two distinct points u and v of the plane ($\mathbb{R}^2$), let line$(u, v,)$ denote the straight line passing through these points and let (u, v) (resp. $[u, v]$) denote the open (resp. closed) interval containing all points in this line that lie between u and v. The half-line starting at point u (but excluding point u) and passing through v is denoted by $HF(u, v)$. Formally,

$$HF(u, v) = \{p \in line(u, v), p \neq u : v \in [u, p] \vee p \in [u, v]\}$$

With reference to some point $c \in \mathbb{R}^2 \setminus \{u, v\}$, the clockwise angle between segments $[c, u]$ and $[c, v]$ is denoted by $\sphericalangle(u, c, v, \circlearrowright)$ (or simply $\sphericalangle(u, c, v)$). The anticlockwise angle $\sphericalangle(u, c, v, \circlearrowleft)$ is defined similarly. The Euclidean distance between u and v is denoted by $|u, v|$. The center of any circle G is denoted by center(G).

CHAPTER 3

WEBER POINTS

A key result of this chapter is a technique to compute the Weber points of newly defined classes of configurations that exhibit some kind of symmetry, referred to in the sequel as *symmetric*, *regular* and *quasi-regular* configurations. The formal definitions of these notions and their properties are given in Section 3.1. Then, we present the algorithms for computing the Weber points for such configurations in Sections 3.2 and 3.3.

Non-linear configurations are known to have a unique Weber point while linear configurations may have infinitely many Weber points. The Weber points of a linear configuration P are points in the interval $[min(Med(P)), max(Med(P))]$, where $Med(P)$ denotes the set of median points. If a linear configuration P has a single median, then this point is the unique Weber point WP(P) (see Figure 3.1).

In this chapter, we assume that all configurations are **non-linear**.

3.1 Symmetries in Robot Configurations

Configurations may exhibit several kinds of symmetry. In this section, we firstly consider a specific form of symmetry called *rotational symmetry* which we define precisely and which we show how to quantify. This notion is based on the concept of *views* [34], as described below. We introduce then weaker forms of symmetry called *regularity* and *quasi-regularity* and we explain how all these notions relate to Weber points.

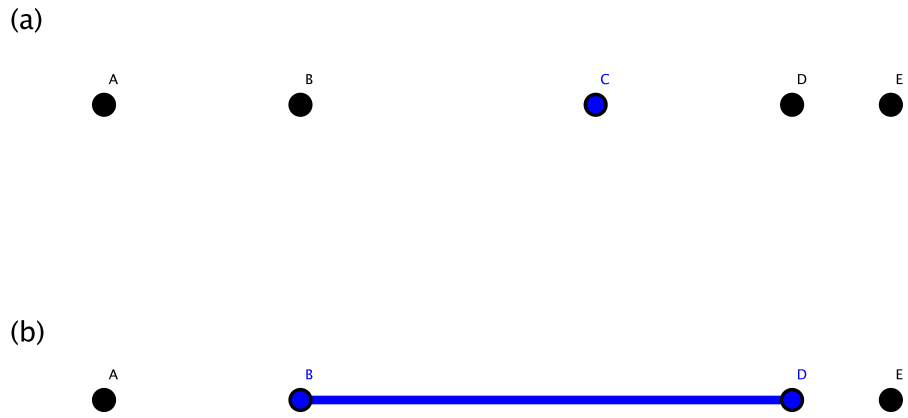


Figure 3.1: Weber points for linear configurations with an (a) *odd* and (b) *even* number of points respectively. In configuration (a), the Weber point is unique and consists of the point C. In configuration (b), the set of Weber points is the line segment $[B,D]$.

Definition 4 (Views)**.** *Let P be a configuration of robots. Given a position $u \in \mathbb{L}(P)$, define the view of u, denoted $\mathcal{V}(u)$, as the expression of P in the polar coordinate system the center of which is u and whose $(1,0)$ point is equal to:*

- *center*$(\text{SEC}(P))$ *if this point is distinct from u.*
- *Otherwise, we take any point in $\mathbb{L}(P)$ that is distinct from u and which maximizes $\mathcal{V}(x)$ according to the lexical order on positions.*

Note that in the definition above, the point $(1,0)$ is not uniquely defined, however the view of any point $u \in \mathbb{L}(P)$ is uniquely defined. Based on the definition of views, we can define an equivalence relation $\backsim$ on the set of robot locations, as follows: $\forall u, u' \in \mathbb{L}(P), (u \backsim u') \Leftrightarrow (\mathcal{V}(u) = \mathcal{V}(u'))$. The corresponding equivalence class for u is denoted by $[u]$.

The following property was observed by Suzuki and Yamashita ([34], Lemma 4.2.) for a different definition of views but it still holds when considering our definition; the argument remains the same. It says that the points of any configura-

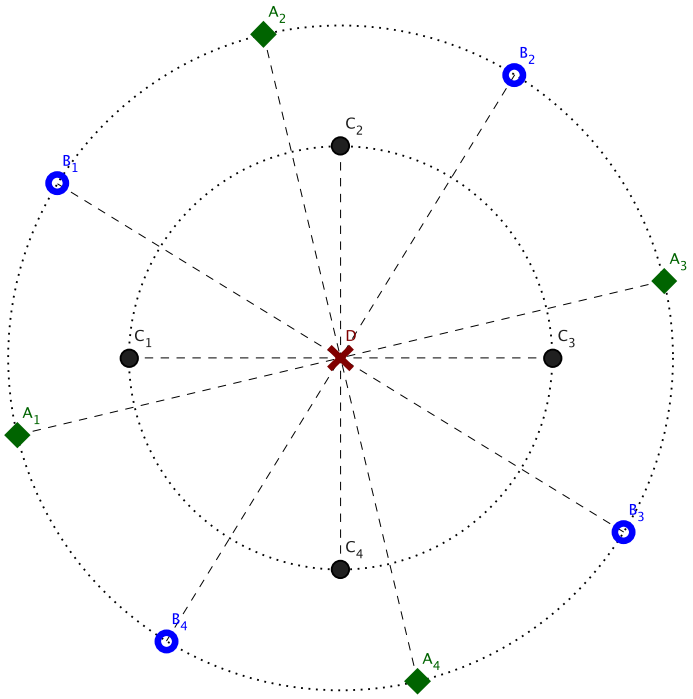


Figure 3.2: A 13-points configuration. The points not lying on the center of the SEC can be partitioned into three equivalence classes of cardinality 4: $\{A_1, A_2, A_3, A_4\}, \{B_1, B_2, B_3, B_4\}$ and $\{C_1, C_2, C_3, C_4\}$.

tion (excluding those lying in the center of the SEC if any) can be partitioned into equivalence classes of the same cardinality (see Figure).

Lemma 3.1.1. *Let P be a configuration and let $c = center(\text{SEC}(P))$. For every $u \in \mathbb{L}(\overline{P_c})$ with $|[u]| = k > 1$, $[u]$ is a regular k-gon with center c and the corners of which have the same multiplicity.*

The following lemma follows from the definition of views:

Lemma 3.1.2. *Let P be a configuration and let $c = center(\text{SEC}(P))$. The following property holds:*
$\exists k > 0, \forall u \in \mathbb{L}(\overline{P_c}) : |[u]| = k$.

Proof. Fix a configuration P. We prove the following equivalent claim.

$$\forall u, w \in \mathbb{L}(\overline{P_c}) : |[w]| \geq |[u]|.$$

Fix $u, w \in \mathbb{L}(\overline{P_c})$ and let $|[u]| = k$. We need to prove that $|[w]| \geq k$. If $k = 1$ then the claim holds trivially. So let us assume in the following that $k > 1$. Let $[u] = \{u_0, \ldots, u_{k-1}\}$. Assume that indices define a clockwise polar ordering of the positions around c. There exists $i \in [0, k-1]$ such that $w \in \sphericalangle(u_i, c, u_{(i+1) \bmod k})$. Observe that both u_i and $u_{(i+1) \bmod k}$ belong to the same equivalence class $[u]$, hence $\mathcal{V}(u_i) = \mathcal{V}(u_{(i+1) \bmod k})$. This means that there is a rotational symmetry of angle $\sphericalangle(u_i, c, u_{(i+1) \bmod k})$ around c. Hence, there exists a position $w' \in \mathbb{L}(\overline{P_c})$, $w' \neq w$ such that $w' \backsim w$ and $w' \in \sphericalangle(u_{(i+1) \bmod k}, c, u_{(i+2) \bmod k})$. By repeating this argument, we find k positions that are equivalent to w (including itself). More precisely, for every $i \in [0, k-1]$, there exists a position $w' \in \mathbb{L}(\overline{P_c})$, $w' \neq w$ with $w' \backsim w$ and $w' \in \sphericalangle(u_i, c, u_{(i+1) \bmod k})$. Consequently, $|[w]| \geq k = |[u]|$. This proves the lemma.

□

3.1.1 Symmetricity

The following definition formalizes the notion of rotational symmetricity.

Definition 5 (Rotational Symmetricity)**.** *The* symmetricity *of a configuration P, denoted $sym(P)$, is the cardinality of the biggest equivalence class defined by $\backsim$ on $\mathbb{L}(P)$. That is, $sym(P) = \max(|[u]| \ : \ u \in \mathbb{L}(P))$.*

If $sym(P) = k > 1$, we say that P is k-symmetric (or symmetric).

The center of symmetricity *of P, denoted $CS(P)$, is the point center*$(\mathrm{SEC}(P))$.

The following property follows from the definition of symmetricity.

Lemma 3.1.3. *For every configuration P, $sym(P) = \max(1, sym(\overline{P_c}))$.*

Proof. Fix a configuration P and let $c = \mathrm{center}(\mathrm{SEC}(P))$. If there exists no robot located at c, then $P = \overline{P_c}$ and the claim holds trivially. Hence, we assume in the following that some robot of P is located at u. That is, $\mathbb{L}(P) = \{c\} \cup \mathbb{L}(\overline{P_c})$. Thus, $\mathrm{sym}(P) = \max(|[u]| \ : \ u \in \{c\} \cup \mathbb{L}(\overline{P_c})) = \max(\{|[c]|\} \cup \{|[u]| \ : \ u \in \mathbb{L}(\overline{P_c})\})$. But the vision of c is unique, hence $|[c]| = 1$. Moreover, $\mathrm{sym}(\overline{P_c}) = \max\{|[u]| \mid u \in \mathbb{L}(\overline{P_c})\}$. Consequently $\mathrm{sym}(\overline{P_c}) = \max(1, \mathrm{sym}(\overline{P_c}))$. □

Now we can adapt a result stated in [20] to our definition of views and symmetricity:

Lemma 3.1.4. *Let P be a configuration such that $sym(P) = k > 1$ and let $c = center(\mathrm{SEC}(P))$. For every $u \in \mathbb{L}(\overline{P_c})$, it holds that $[u]$ is a regular convex k-gon with center c and the corners of which have the same multiplicity.*

Proof. Since $\text{sym}(P) = k > 1$, it follows from Lemma 3.1.3 that $\text{sym}(\overline{P_c}) = k$. This means that there exists an equivalence class in $\overline{P_c}/\backsim$ with cardinality k. But we have seen in Lemma 3.1.2 that all the elements in $\overline{P_c}/\backsim$ have the same cardinality. That is, for all $u \in \mathbb{L}(\overline{P_c})$, $|[u]| = k$. Moreover, according to Lemma 3.1.1, the points of $[u]$ form a regular k-gon with center c and the corners of which have the same multiplicity. □

3.1.2 Regularity

We now define some weaker forms of symmetry called *regularity* and *quasi-regularity*. If we consider any circle G that encloses the points in a configuration P, we may order the points by sweeping the circle G in a clockwise direction and ordering points on the same radius w.r.t. their distance from the center. This idea leads to the following definitions (extending the concepts in [2, 29]).

Definition 6. *Let $P = \{p_1, \ldots, p_n\}$ be a configuration and let $c \in \mathbb{R}^2$.*

- *[Successor] The* clockwise *successor of $p_i \in P$ around c, denoted by $S(p_i, c, \circlearrowright)$ (or simply $S(p_i, c)$) is equal to the point $p_j \in P$ defined as follows:*
  - *Let $X = \{p_k \in P \mid (p_k = p_i) \wedge (k < i)\}$. If $X \neq \emptyset$, then*
  $$p_j = \underset{p_k \in X}{\text{argmax}}\ k$$
  - *Otherwise, let $Y = \{p_k \in P \cap (c, p_i)\}$. If $Y \neq \emptyset$, then*
  $$p_j = \underset{p_k \in Y}{\text{argmax}}\ (|c, p_k|, k)$$
  - *Otherwise, let $Z = \{p_k \in P \mid (\nexists p \in P : 0 < \sphericalangle(p_i, c, p, \circlearrowright) < \sphericalangle(p_i, c, p_k, \circlearrowright)\}$. In this case,*
  $$p_j = \underset{p_k \in Z}{\text{argmax}}\ (|c, p_k|, k)$$
- *[k-th Successor] The k-th successor of p_i around c, denoted $S^k(p_i, c)$ is defined recursively as follows: $S^1(p_i, c) = S(p_i, c)$; and if $k > 1$, $S^k(p_i, c) = S(S^{k-1}(p_i, c))$.*

The k-th anti-clockwise successor *of p_i around c, denoted $S(p_i, c, \circlearrowleft)$ is defined similarly.*

The *string of angles* of P around a point c started in p_i, denoted by $\mathrm{SA}_P(p_i, c)$ is the string $\alpha_1 \dots \alpha_m$ such that $m = n - \mathrm{mul}(c)$ and $\alpha_i = \sphericalangle(S^{i-1}(p_i), c, S^i(p_i))$.

The *size* of $\mathrm{SA}_P(p_i, c)$, denoted by $|\mathrm{SA}_P(p_i, c)|$ is equal to m.

A string SA is k-periodic if it can be written as $\mathrm{SA} = x^k$ where $1 \leq k \leq |\mathrm{SA}|$. The greatest k for which SA is k-periodic is called *the periodicity* of SA and is denoted by $per(\mathrm{SA})$.

Observe that the periodicity of a string of angles does not depend on the process in which it is started. That is, when it comes to periodicity, the important information about a string of angles is only its center c. Hence, in the following we refer to it by writing $\mathrm{SA}_P(c)$ or simply $\mathrm{SA}(c)$ when the related configuration P can be understood from context.

The following theorem shows that strings of angles are computable in $O(n \log n)$ time.

Theorem 3.1.1. *Given is a configuration P with $|\mathbb{L}(P)| > 1$ and a point $c \in \mathbb{R}^2$, there exists an algorithm with running time $O(n \log n)$ that computes $SA(c)$.*

Proof. Fix some $p \in P$ with $p \neq c$. We show how to compute $\mathrm{SA}(p, c)$. We use an array $T[n]$ of $n - \mathrm{mul}(c)$ cells. As a first step, we compute for each $p_i \in P$ with $p_i \neq c$ the angle $\sphericalangle(p, c, p_i, \circlearrowleft)$ and put the result in $T[i]$. Note that this step takes $O(n)$ time. Then we sort T in increasing order ($O(n \log n)$ steps). $\mathrm{SA}(p, c)$ is the string $\alpha_1 \dots \alpha_{n-\mathrm{mul}(c)}$ such that $\forall i \in [1, n - \mathrm{mul}(c) - 1] : \alpha_i = T[i+1] - T[i]$ and $\alpha_{n-\mathrm{mul}(c)} = 2\pi - T[n - \mathrm{mul}(c)]$. The whole algorithm runs in $O(n \log n)$. □

Definition 7 (Regularity)**.** *A configuration P of n points is* regular *if there exists a point $c \in \mathbb{R}^2$ such that the string of angles of P around c is periodic. That is, $\exists k > 1$ such that $per(SA(c)) = k > 1$.*

The regularity *of P, denoted $reg(P)$, is equal to k. Otherwise $reg(P) = 1$.*

The point c is called the center *of regularity and is denoted by $CR(P)$.*

Observe that a configuration that is symmetric is also regular. Precisely, $(\mathrm{sym}(P) > 1) \Rightarrow (\mathrm{reg}(P) = x\mathrm{sym}(P)$ with $x \geq 1)$.

Definition 8 (Quasi-Regularity)**.** *A configuration P of n points is* quasi-regular *if and only if there exist (1) a point $c \in \mathbb{R}^2$ and (2) a regular configuration P' with center of regularity c which can be obtained from P by moving only points located at c if any. Formally, P is quasi-regular with center $c \in \mathbb{R}^2$ if and only if $\exists P' \in \mathbb{P}$ such that $reg(P') > 1$, $CR(P') = c$ and $\forall p \in P' \setminus P : p = c$. In this case, the quasi-regularity of P, denoted $qreg(P)$ is equal to $reg(P')$.*

Its center of quasi-regularity, *denoted $CQR(P)$, is equal to c.*

If P is not quasi-regular then $qreg(P) = 1$.

Note that a configuration that is regular is also quasi-regular (with $P' = P$). Precisely, $(\text{reg}(P) > 1) \Rightarrow (\text{qreg}(P) = \text{reg}(P))$.

The notions of symmetricity, regularity and quasi-regularity are illustrated in Figure 3.3.

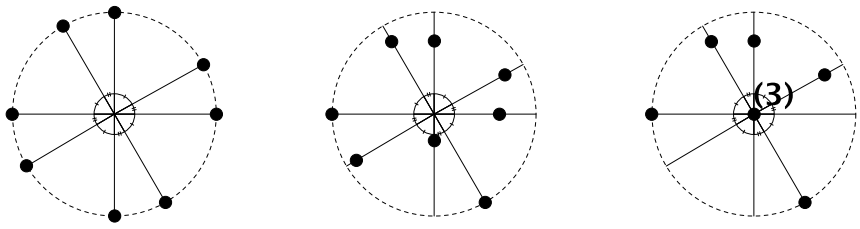


Figure 3.3: Configurations that are (from the left to the right) (i) Symmetric with $\text{sym}(P) = 4$, (ii) Regular with $\text{reg}(P) = 4$, (iii) Quasi-Regular with $\text{qreg}(P) = 4$. The number in parentheses represent the multiplicity of a point.

3.1.3 Properties of Weber Points

The Weber point of a non-linear configuration has the remarkable property or remaining invariant under movement of some of the points towards it (see Lemma 3.1.5). Moreover, we can show that for every configuration P that has a unique Weber point and that is quasi-regular, the center of quasi-regularity $CQR(P)$ coincides with the unique Weber point $\text{WP}(P)$ (see Lemma 3.1.8) and this point can be computed.

Lemma 3.1.5. *Let $P = \{p_1, \ldots, p_n\}$ and $P' = \{p'_1, \ldots, p'_n\}$ two configurations. Let $X = \{x \in WP(P) \mid \forall i \in [1,n] : p'_i \in [p_i, x]\}$. If $X \neq \emptyset$ then $WP(P') = X$.*

Proof. Let $Y = \{x \in \mathbb{R}^2 \mid \forall i \in [1,n] : p'_i \in [p_i, x]\}$. Note that $X = Y \cap \text{WP}(P)$.

Observe that:

$$\sum_{i=1}^{n} |x, p'_i| = \sum_{i=1}^{n} |x, p_i| + \sum_{i=1}^{n} (|x, p'_i| - |x, p_i|)$$

By definition, the points of $\text{WP}(P)$ are those points x that minimize $\sum_{i=1}^{n} |x, p_i|$. Moreover, the points of Y are those that minimize $\sum_{i=1}^{n} (|x, p'_i| - |x, p_i|)$. Hence, the points of $X = Y \cap \text{WP}(P)$ minimize the two sums and minimize their sum also. It follows that the points that minimize $\sum_{i=1}^{n} |x, p'_i|$ are those in X. Thus, $\text{WP}(P') = X$. □

Corollary 3.1.1. *If P is a configuration with a unique Weber point c and if P' is a configuration that is obtained from P by moving robots towards c, then the Weber point of P' is also unique and is equal to c.*

The following lemma states that the center of symmetricity of any configuration P if any, is also its Weber point. The same claim was proved by Anderegg et al. [3] in the cases when $\text{sym}(P) = n$ (equiangular) and $\text{sym}(P) = n/2$ (biangular). Our proof uses the same argument as theirs.

Lemma 3.1.6. *For every non-linear configuration P that is symmetric,*

$$WP(P) = CS(P)$$

Proof. Fix P a non-linear configuration, and assume $\text{sym}(P) = k > 1$. Let $c = CS(P) = \text{center}(\text{SEC}(P))$. We have to prove that $\text{WP}(P) = c$.

Assume for the sake of contradiction that $\text{WP}(P) \neq c$. This implies the existence of a point $c' \neq c$ with $c' \in \text{WP}(P)$.

Let G be the regular k-gon with center c and and c' a vertex of G. Since $c' \in \text{WP}(P)$, it follows by symmetry that all the k points of G belong also to $\text{WP}(P)$. But as P is non-linear, $\text{WP}(P)$ is unique. Contradiction. □

Lemma 3.1.7. *For every non-linear configuration P that is regular,*

$$WP(P) = CR(P)$$

Proof. Let P a non-linear configuration such that $\text{reg}(P) = k > 1$ and let $c = CR(P) = \text{center}(\text{SEC}(P))$. We have to prove that $\text{WP}(P) = c$. Let P' be a configuration obtained from P as follows: For each point $p \in P$ not located at c, move p towards the point that is at the intersection of $HF(c, p)$ and $\text{SEC}(P)$. Clearly, the obtained configuration P' is symmetric with center of symmetricity c and $\text{sym}(P') = \text{reg}(P)$. According to Lemma 3.1.6, $\text{WP}(P') = c$.

Note that P can be obtained from P' by moving points in P' towards c. Hence by applying Corollary 3.1.1, we conclude that $\text{WP}(P) = c$.

□

In the following Lemma we show that the center of quasi-regularity of a configuration is also its Weber point.

Lemma 3.1.8. *For every non-linear configuration P that is quasi-regular,*

$$WP(P) = CQR(P)$$

Proof. Let P a non-linear configuration with $\text{qreg}(P) = k > 1$ and $c = CQR(P)$. We have to prove that $c = \text{WP}(P)$.

By definition of quasi-regularity, there exists a regular configuration P' whose center of regularity is c and which can be obtained from P by moving only points located at c if any. Seen in the reverse sense, P can obtained from P' under straight movement of points towards c. According to Lemma 3.1.7, $\text{WP}(P') = c$. Hence, by Corollary 3.1.1, $\text{WP}(P) = c$. □

3.2 Computation of Weber Points in Regular Configurations

In this section we show how to identify geometric configurations that are regular and to compute their center of regularity. We present two algorithms that detects whether a configuration P of n points given in input is k-regular for some $k > 1$, and if so, they output its center of regularity. The first one detects the regularity only if m is even, it is very simple and runs in $O(n \log n)$ time. The second one can detect any regular configuration, provided that $m \geq 3$. It is a little more involved and runs in $O(n^4 \log n)$ time.

3.2.1 Preliminaries

In this section, we state some technical properties about regular configurations which we use later in our proofs.

Lemma 3.2.1. *Let P be a regular configuration with $reg(P) = k > 1$ and $c = CR(P)$. Let $n' = |\overline{P_c}| = n - mul(c)$. The following property holds:*

$$\forall u \in \overline{P_c} : \sphericalangle(u, c, S^{\frac{n'}{k}}(u, c)) = \frac{2\pi}{k}$$

Proof. Fix $u \in \overline{P_c}$ and let $S = \text{SA}(u, c)$. Note that $|S| = n - mul(c) = n'$. Assume that $S = \alpha_1 \dots \alpha_{n'}$. Since $\text{reg}(P) = k$, it holds that S is k-periodic. Hence, k is a divisor of n'. Let x denote n'/k.

Observe that:

$$\alpha_1 + \alpha_2 + \ldots + \alpha_{n'} = 2\pi$$

Since $n' = kx$, the above sum can be rewritten as follows:

$$(\alpha_1 + \ldots + \alpha_x) + (\alpha_{x+1} + \ldots + \alpha_{2x}) + \ldots + (\alpha_{(k-1)x+1} + \ldots + \alpha_{kx}) = 2\pi$$

The fact that S is k-periodic means that $\forall i \in [1, n'] : \alpha_i = \alpha_{(i+x) mod (n'+1)}$. This implies:

$$k(\alpha_1 + \ldots + \alpha_x) = 2\pi$$

Thus:

$$(\alpha_1 + \ldots + \alpha_x) = \frac{2\pi}{k}$$

But $\sphericalangle(u, c, S^x(u)) = (\alpha_1 + \ldots + \alpha_x)$. Hence $\sphericalangle(u, c, S^x(u)) = \frac{2\pi}{k}$. □

The following lemma is a generalization of Lemma 3.2.1.

Lemma 3.2.2. *Let P be a regular configuration with $reg(P) = k > 1$ and $c = CR(P)$. Let $n' = |\overline{P_c}| = n - mul(c)$. The following property holds:*

$$\forall u \in \overline{P_c} : \forall k' \in [1, k] : \sphericalangle(u, c, S^{\frac{nk'}{k}}(u, c)) = \frac{2k'\pi}{k}$$

Proof.

$$\begin{aligned} S^{\frac{n'k'}{k}}(u, c) &= \underbrace{\sphericalangle(u, c, S^{\frac{n'}{k}}(u)) + \ldots + \sphericalangle(u, c, S^{\frac{n'}{k}}(u))}_{k \text{ times}} \\ &= \frac{2\pi}{k} + \ldots + \frac{2\pi}{k} \\ &\quad \text{(According to Lemma 3.2.1)} \\ &= \frac{2k'\pi}{k} \end{aligned}$$

□

Lemma 3.2.3. *Let P be a regular configuration with $reg(P) = 2k, k > 1$ and $c = CR(P)$. Let $n' = |\overline{P_c}| = n - mul(c)$. The following property holds:*

$$\forall u \in \overline{P_c} : \sphericalangle(u, c, S^{\frac{n'}{2}}(u, c)) = \pi$$

Proof. Fix $u \in \overline{P_c}$. The fact that $\mathrm{reg}(P) = 2k$ means that $\mathrm{SA}(c)$ is $2k$-periodic. Since $|\mathrm{SA}(c)| = n'$, this implies that $2k$ is a factor of n'. Consequently:

$$\sphericalangle(u, c, S^{\frac{n'}{2}}(u)) = \sphericalangle(u, c, S^{\frac{n'}{2k}}(u)) + \sphericalangle(S^{\frac{n'}{2k}}(u), c, S^{\frac{n'}{k}}(u)) + \ldots + \sphericalangle(S^{\frac{(k-1)n'}{2k}}(u), c, S^{\frac{n'}{2}}(u))$$

Since $\mathrm{reg}(P) = 2k$, it holds according to Lemma 3.2.1 that:

$$\forall i \in [1, k-1] : \sphericalangle(S^{\frac{in'}{2k}}(u), c, S^{\frac{(i+1)n'}{2k}}(u)) = \frac{\pi}{k}$$

Therefore:

$$\sphericalangle(u, c, S^{\frac{n'}{2}}(u) = \underbrace{\frac{\pi}{k} + \ldots + \frac{\pi}{k}}_{k \text{ times}} = \pi$$

□

3.2.2 Detection of Even Regularity

In this section we present our algorithm to detect regular non-linear configurations when their regularity is even. It is inspired from Algorithm 2 of Anderegg et al. [3] and runs in $O(n \log n)$ steps. It is based on the notion of median lines adapted from [3].

Definition 9 (median(u))**.** *Given a configuration P with $|\mathbb{L}(P)| > 1$ and $u \in P$, the median line of u, denoted median(u), is the line that passes through u and some other point $u' \in P$ and divides the set of points into two subsets the cardinality of each is smaller or equal to $\frac{n}{2} - 1$.*

Lemma 3.2.4. *Let P be a regular configuration with $reg(P) = 2k, k > 1$ and $c = CR(P)$. Let $n' = |\overline{P_c}| = n - mul(c)$.*

The following property holds:

$$\forall u \in \overline{P_c} : median(u) \text{ passes through } S^{n'/2}(u) \text{ and } c$$

Proof. According to Lemma 3.2.3, $\sphericalangle(u, c, S^{n/2}(u)) = \pi$. Hence the lemma follows. □

As a corollary we have that the center of regularity lies in the intersection of all medians.

Corollary 3.2.1. *Let P be a regular configuration with $|\mathbb{L}(P)| > 1$, $reg(P) = 2k, k > 1$ and $c = CR(P)$. It holds that*

$$\forall u \in P : c \in median(u)$$

We are now ready to show how to compute the center of regularity when it is even:

Theorem 3.2.1. *Let P be a non-linear configuration with $|\mathbb{L}(P)| > 1$ and $reg(P) = 2k, k > 1$. There exists an algorithm running in $O(n \log n)$ steps that computes $reg(P)$ and outputs the center of regularity $CR(P)$.*

Proof. The algorithm is described formally in Figure 3.4. It firstly computes the convex hull CH of the configuration which takes $O(n \log n)$ operations [31]. Then it chooses an arbitrary point u in CH and computes its median(u) which is well defined since $|\mathbb{L}(P)| > 1$ and reg(P) $= 2k, k > 1$. After that, it picks another point u' that belongs to CH but not to median(u). The fact that P is non-linear guarantees that u' is well-defined. Since $u' \notin$ median(u), it follows that median(u) $\neq$ median(u'). Hence, c, the intersection of median(u) and median(u') (line (5)) is

unique. According to Corollary 3.2.1, if the configuration is regular, its center of regularity lies necessarily in c. To check if c is indeed a center of regularity, it suffices to compute SA(c) ($O(n \log n)$ operations as shown in Lemma 3.1.1) and to test if it is periodic ($O(n)$ operations [5]). If SA(c) is periodic, then P is regular with $\text{reg}(P) = per(\text{SA}(c))$. If not, P is either not-regular or its regularity is not even.

```
(1)  CH ← CONVEX HULL(P)
(2)  u ← an arbitrary point on CH
(3)  u' ← a point on CH but not on median(u)
(4)  If (median(u) ∩ median(u') ≠ ⊥)
(5)     c ← median(u) ∩ median(u')
(6)     SA ← SA(c)
(7)     k ← per(SA)
(8)     If (k > 1) RETURN (P IS REGULAR, CR(P) = c, REG(P) = k) endif
(9)  endif
(10) return (P IS NOT REGULAR OR ITS REGULARITY IS ODD)
```

Figure 3.4: Detection of Even Regularity.

□

3.2.3 Detection of Odd Regularity

In this section we present our algorithm for detecting regular non-linear configurations when their regularity is greater or equal to 3. Its time complexity is $O(n^4 \log n)$.

We start by defining the notion of capacious arcs (see Figure 3.5) and we prove some properties about them that we use to detect regularity.

Definition 10 (Capacious Arc)**.** *Let $u, v \in \mathbb{R}^2$ be any two* distinct *points of the plane and let $\alpha \in (0, \pi)$ be a given angle.*

- *The set of points $x \in \mathbb{R}^2$ such that $\sphericalangle(u, x, v, \circlearrowright) = \alpha$ is an open[1] arc of the circle that passes through u and v and the center of which c is such that $\sphericalangle(u, x, v, \circlearrowright) = 2\alpha$.*

  Let $X_\alpha(u, v, \circlearrowright)$ denote this arc and let $C_\alpha(u, v, \circlearrowright)$ denote the circle that include it.

- *$X_\alpha(u, v, \circlearrowleft)$ and $C_\alpha(u, v, \circlearrowleft)$ are defined in a similar way.*

[1] excluding u and v

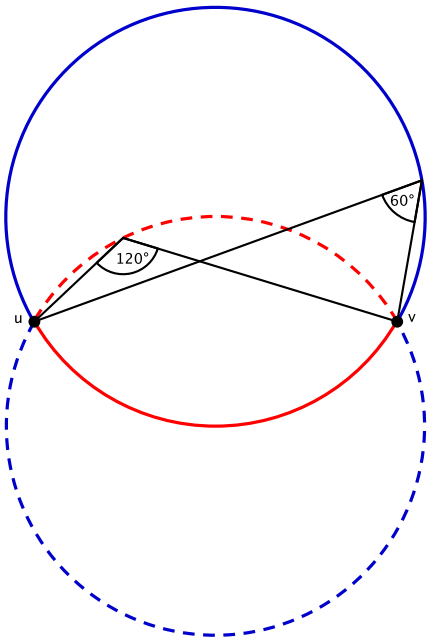


Figure 3.5: The figure illustrates the capacious arcs $X_{\pi/3}(u,v,\circlearrowleft)$ (continuous blue line) and $X_{\pi/3}(u,v,\circlearrowright)$ (dashed line). $X_{\pi/6}(u,v,\circlearrowleft)$ and $X_{\pi/6}(u,v,\circlearrowright)$ are depicted in red.

- *Let $X_\alpha(u,v)$ be the set of points $x \in \mathbb{R}^2$ such that $\sphericalangle(u,x,v) = \alpha$. Clearly, $X_\alpha(u,v) = X_\alpha(u,v,\circlearrowright) \cup X_\alpha(u,v,\circlearrowleft)$.*

Lemma 3.2.5. *Let P be a configuration with $|\mathbb{L}(P)| > 2, reg(P) = k > 2$ and $c = CR(P)$.*

Let $n' = |\overline{P_c}| = n - mul(c), k' = \lceil \frac{k}{2} \rceil - 1$ and $\alpha = \frac{2k'\pi}{k}$.

Let $u \in P$ with $u \neq c$ and denote by v, w the points $S^{\frac{k'n'}{k}}(u,c,\circlearrowright)$ and $S^{\frac{k'n'}{k}}(u,c,\circlearrowleft)$ respectively. It holds that:

(i) $c \in X_\alpha(u,v)$.

(ii) For every $A \in \{C_\alpha(u,v,\circlearrowright), C_\alpha(u,v,\circlearrowleft)\}$, for every $B \in \{C_\alpha(u,w,\circlearrowright), C_\alpha(u,w,\circlearrowleft)\}$, if $A = B$ then $c \notin A$.

Proof. (i) First, we show that $k' \in [1,k]$. Recall that $k' = \lceil k/2 \rceil - 1$. Hence, $k' \leq k$ by definition. Since $k > 2$, it follows that $k' \in [1,k]$.

Consequently, according to Lemma 3.2.2 we have $\sphericalangle(u,c,S^{\frac{n\cdot k'}{k}}(u,c)) = \frac{2k'\pi}{k} = \alpha$. That is, $\sphericalangle(u,c,v) = \alpha$. Therefore $c \in X_\alpha(u,v)$.

(ii) Fix $A \in \{C_\alpha(u,v,\circlearrowright), C_\alpha(u,v,\circlearrowleft)\}$ and $B \in \{C_\alpha(u,w,\circlearrowright), C_\alpha(u,w,\circlearrowleft)\}$. Assume towards contradiction that $A = B$ and $c \in A$. This means that all the points u, v, w and c belong to the same circle A. As a consequence, the point c lies strictly **outside** the convex hull of the points u, v, w. We show that this leads to a contradiction.

Remember that v and w correspond to the points $S^{\frac{k'n'}{k}}(u,c,\circlearrowright)$ and $S^{\frac{k'n'}{k}}(u,c,\circlearrowleft)$ respectively. It follows that $\sphericalangle(u,c,v,\circlearrowright) = \frac{2k'\pi}{k} = \alpha$ and $\sphericalangle(w,c,u,\circlearrowright) = \alpha$. Thus, $\sphericalangle(v,c,w,\circlearrowright) = 2\pi - 2\alpha$.

Since $k' = \lceil k/2 \rceil - 1$ and $k > 2$ then $\alpha = \frac{2k'\pi}{k} \in [\pi/2, \pi)$ and $2\pi - 2\alpha \in (0,\pi]$. Consequently, the angles $\sphericalangle(u,c,v,\circlearrowright), \sphericalangle(w,c,u,\circlearrowright)$ and $\sphericalangle(v,c,w,\circlearrowright)$ are not reflex (less than π), which means that c is **inside** the triangle formed by the points u, v, w. Contradiction.

□

Lemma 3.2.6. *Let P be a configuration with $|\mathbb{L}(P)| > 2, reg(P) = k > 2$ and $c = CR(P)$.*
Let $n' = |\overline{P_c}| = n - mul(c), k' = \lceil k/2 \rceil - 1$ and $\alpha = \frac{2k'\pi}{k}$.
Let $u \in P$ with $u \neq c$. It holds that :

(i) There exist $v, w \in P$ such that $c \in F(v,w)$ where
$F(v,w) = \{A \cap B \mid (A \in \{X_\alpha(u,v,\circlearrowright), X_\alpha(u,v,\circlearrowleft)\}) \wedge (B \in \{X_\alpha(u,w,\circlearrowright), X_\alpha(u,w,\circlearrowleft)\}) \wedge (A \notin C_\alpha(u,w,\circlearrowright) \cup C_\alpha(u,w,\circlearrowleft)) \wedge (B \notin C_\alpha(u,v,\circlearrowright) \cup C_\alpha(u,v,\circlearrowleft))\}$

(ii) For every $v, w \in P$, $|F(v,w)| \leq 4$.

Proof. (i) . Let v, w be the points $S^{\frac{k'n'}{k}}(u,c,\circlearrowright)$ and $S^{\frac{k'n'}{k}}(u,c,\circlearrowleft)$ respectively. According to Lemma 3.2.5, it holds that:

$$c \in X_\alpha(u,v) \cap X_\alpha(u,w)$$

That is,

$$c \in (X_\alpha(u,v,\circlearrowright) \cup X_\alpha(u,v,\circlearrowleft)) \cap (X_\alpha(u,w,\circlearrowright) \cup X_\alpha(u,w,\circlearrowleft))$$

But according to Lemma 3.2.5 (ii), if $A \in \{X_\alpha(u,v,\circlearrowright), X_\alpha(u,v,\circlearrowleft)\}$ and $A \subseteq (C_\alpha(u,w,\circlearrowright) \cup C_\alpha(u,w,\circlearrowleft))$, then $c \notin A$. Similarly, if $B \in \{X_\alpha(u,w,\circlearrowright), X_\alpha(u,w,\circlearrowleft)\}$ and $B \subseteq (C_\alpha(u,v,\circlearrowright) \cup C_\alpha(u,v,\circlearrowleft))$, then $c \notin B$.

Hence $c \in F(v,w)$.

(ii) Observe that $F(v,w) \subseteq G\ \{u\}$ where $G = \{A \cap B \mid (A \in \{C_\alpha(u,v,\circlearrowright), C_\alpha(u,v,\circlearrowleft)\} \wedge (B \in \{C_\alpha(u,w,\circlearrowright), C_\alpha(u,w,\circlearrowleft)\} \wedge \setminus \{\mathtt{uC}^\alpha_{uw}, \mathtt{nC}^\alpha_{uw}\}) \wedge (B \in \{\mathtt{uC}^\alpha_{uw}, \mathtt{nC}^\alpha_{uw}\}) \setminus \{\mathtt{uC}^\alpha_{uv}, \mathtt{nC}^\alpha_{uv}\})\}$.

If two distinct circles intersect, they do so either in two distinct points or in one degenerate point. Hence, taking any two distinct circles A and B with $A \in \{C_\alpha(u,v,\circlearrowright), C_\alpha(u,v,\circlearrowleft)$ and $B \in \{C_\alpha(u,w,\circlearrowright), C_\alpha(u,w,\circlearrowleft)$, they intersect in point u and possibly another point. Consequently G consists of at most five points. It follows that $|F(v,w)| \leq 4$.

□

Theorem 3.2.2. *Given P a configuration with $|\mathbb{L}(P)| > 2$ and $k > 2$. There exists an algorithm running in $O(n^4 \log n)$ steps that detects if $reg(P) = k$ and if so, it outputs its center of regularity.*

Proof. The algorithm is the following. We fix any points $u \in P$ and we test if it is a center of regularity by computing $\mathrm{SA}_P(u)$ ($O(n \log n)$ time) and testing if it is periodic ($O(n)$ operations). If so, P is regular with $\mathrm{reg}(P) = per(\mathrm{SA}_P(u))$.

Otherwise, we proceed as follows. For every $v \in \mathbb{L}(P)$, for every $w \in \mathbb{L}(P) \setminus \{v\}$, we compute the set $F(v,w)$ as defined in Lemma 3.2.6, and we test for every point x in $F(v,w)$, if x is a center of regularity or not by computing the periodicity of $\mathrm{SA}_P(x)$. If $\mathrm{SA}_P(x)$ is periodic than P is regular with $\mathrm{reg}(P) = per(\mathrm{SA}_P(x))$.

Lemma 3.2.6 guarantees that if P is regular then the test will be conclusive for at least one pair (v,w) of points of $\mathbb{L}(P)$. The whole algorithm executes in $O(n^3 \log n)$: we browse all the possible pairs (v,w), and for each pair we generate up to four candidates for the center of regularity by computing the corresponding set $F(v,w)$ (Lemma 3.2.6) , hence we have $O(n^2)$ candidates. Then, $O(n \log n)$ time is needed to test each candidate x by computing the periodicity of $\mathrm{SA}_P(x)$.

Note that our algorithm follows the same patterns as those presented in [3]: generating a restricted set of candidates (points) and testing whether each of them is a center of regularity. □

Theorem 3.2.3. *Given P a configuration of n distinct points. There exists an algorithm running in $O(n^4 \log n)$ steps that detects if P is k-regular with $k \geq 3$, and if so, it outputs k and the center of regularity $CR(P)$.*

Proof. It suffices to generates all the divisors k of n that are greater than 2. Then, for each k, we test if P is k-regular as shown in Theorem 3.2.2. When the test is conclusive, this algorithms return the center of regularity c, so we can output (P IS REGULAR, $CR(P) = c$, $\text{REG}(P) = k$). If test was inconclusive for every generated divisor k, we simply output (P NOT REGULAR). □

Theorem 3.2.4. *Given P a non-linear configuration of n distinct points. There exists an algorithm running in $O(n^4 \log n)$ steps that detects if P is k-regular with $k \geq 2$, and if so, it outputs k and the center of regularity $CR(P)$.*

Proof. We combine the algorithms of Theorems 3.2.1 and 3.2.3. First, we test if P is k-regular for some even k using the algorithm of Theorem 3.2.1 ($O(n \log n)$ steps). If so, we output k and the center of regularity c which are provided by the called algorithm. Otherwise, we test odd regularity using the algorithm of Theorem 3.2.3 ($O(n^4 \log n)$ steps) but by restricting the analysis to only the *odd* divisors of n (the even divisors were already tested). □

3.3 Computation of Weber Points for Quasi-Regular Configurations

In this section we show how to compute Weber points for quasi-regular configurations. Let us start with the following lemma which proves a property about regular configurations:

Lemma 3.3.1. *Let P be any configuration and let $p \in \mathbb{R}^2$. Let $m \in \mathbb{N}^+$.*

P is regular with center c and reg$(P) = m$ iff $\forall x \neq c \in \mathbb{R}^2 : \forall k \in [1, m] : \forall y = N(x, c, \frac{2k\pi}{m}) :$ it holds that $HF(c, x)$ and $HF(c, y)$ contain the same number of robots of P.

Proof. Remind that P is regular with $\text{reg}(P) = m$ and center c iff there exists a configuration P' that is rotational symmetric with respect to c such that P can be obtained from P' by moving robots of P' towards c without reaching it. Moreover $\text{sym}(P') = m$.

Fix x and $y \in \mathbb{R}^2$ such that $y = N(x, c, \frac{2k\pi}{m})$ for some $k \in [1, m]$. We have to prove that $HF(p, x)$ and $HF(p, y)$ contain the same number of robots of P. Observe that the number of robots in $HF(c, x)$ and $HF(c, y)$ remains invariant when we transform P' into P as robots are allowed to move only towards c without reaching it. So no robot joins or leaves either $HF(c, x)$ or $HF(c, y)$. Hence, to prove our claim it suffices to show that $HF(c, x)$ and $HF(c, y)$ contain the same number of robots of P'.

Since $\text{sym}(P') = m$, it follows that P' remains invariant if we rotate it around c with an angle of $\frac{2k\pi}{m}$. Note that $HF(c,x)$ can be seen as the result of rotating $HF(c,y)$ by an angle of $\frac{2k\pi}{m}$. Hence $HF(c,x)$ contains the same number of robots of P' as $HF(c,y)$. This proves the lemma. □

We give some definitions:

Definitions:

- Given a multiset of points P and any point $c \in \mathbb{R}^2$, we define $\text{CIRC}(P,c)$ as the smallest circle centered at c that encloses all points in P.
- Given a point $x \in P$ and some $\alpha \in [0, 2\pi]$, the clockwise neighbor of x with respect to point c and angle α, denoted by $N(x,c,\alpha)$ is the point y such that $|c,x| = |c,y|$ and $\sphericalangle(x,c,y,\circlearrowright) = \alpha$.
- Let Pbe a configuration with $|\mathbb{L}(P)| > 1$ and let $c \in P$. Let $m \in \mathbb{N}^+$. We define $X_m(P,c)$ as the following set of points $\{x \in \text{CIRC}(P,c) \mid \exists k \in [1,m] : \exists y = N(x,c,\frac{2k\pi}{m}) : (c,y]$ contains at least one robot$\}$.
- For each point $x \in X_m(P,c)$, let $\text{LOC}(P,x,c)$ (or $\text{LOC}(P,x)$) denote the number of robots of P that are located in $(c,x]$ and let $\text{OBJ}(P,x)$ denote $\max\{\text{LOC}(P,y) | (y = N(x,p,\frac{2k\pi}{m})) \wedge (k \in [1,m])\}$.

Lemma 3.3.2. *Given a configuration P and a point $c \in P$, P is q-regular with center c and $qreg(P) = m > 1$ if and only if:*

$$mul(c) \geq \sum_{x \in X_m(P,c)} (\text{OBJ}(P,x) - \text{LOC}(P,x)) \qquad (\alpha)$$

Proof. According to Definition 7, P is quasi-regular with center c and $\text{qreg}(P) = m$ iff (i) there exists a configuration P' that is regular with center c, (ii) $\text{reg}(P') = m$ and (iii) P can be transformed into P' by moving only robots located at c. To prove the lemma it suffices to show that $(i) \wedge (ii) \wedge (iii) \Leftrightarrow (\alpha)$.

$\Leftarrow$) Assume (α) holds. For each $x \in X_m(P,c)$, we move $(\text{OBJ}(P,x) - \text{LOC}(P,x))$ robots from c to x. Let P' be the resulting configuration. Since (α) is satisfied, there are enough robots located in c to perform this action. This proves (iii).

Note that $X_m(P',c) = X_m(P,c)$.

By construction of P', it holds that $\forall x \in X_m(P',c) : \text{LOC}(P',x) = \text{OBJ}(P,x)$. But by definition of $\text{OBJ}(P,x)$, we have $\forall x \in X_m(P,c) : \forall k \in [1,m] : \forall y = N(x,c,\frac{2k\pi}{m}) : \text{OBJ}(P,x) = \text{OBJ}(P,y)$. It follows that

$$\forall x \in X_m(P', c) : \forall k \in [1, m] : \forall y = N(x, c, \frac{2k\pi}{m}) : \text{LOC}(P', x) = \text{LOC}(P', y)$$

Consequently, according to Lemma 3.3.1, it holds that (i) P' is regular with center c and (ii) $\text{reg}(P') = m$.

$\Rightarrow$) Assume $(i) \wedge (ii) \wedge (iii)$. Since P' can be obtained from P by moving only robots located at c according to (iii), it follows that $X_m(P, c) \subseteq X_m(P', c)$. Moreover,

$$\forall x \in X_m(P, c) : \text{LOC}(P', x) \geq \text{LOC}(P, x) \qquad (\beta)$$

Since P' is regular with center c and $\text{reg}(P') = m$, it holds according to Lemma 3.3.1 that $\forall x \in X_m(P, c) : \forall k \in [1, m] : \forall y = N(x, c, \frac{2k\pi}{m}) : \text{LOC}(P', x) = \text{LOC}(P', y)$. But $\text{LOC}(P', y) \geq \text{LOC}(P, y)$ according to Equation (β). Hence,

$$\forall x \in X_m(P, c) : \forall k \in [1, m] : \forall y \in N(x, c, \frac{2k\pi}{m}) : \text{LOC}(P', x) \geq \text{LOC}(P, y)$$

It follows that:

$$\forall x \in X_m(P, c) : \text{LOC}(P', x) \geq \text{OBJ}(P, x)$$

Hence,

$$\forall x \in X_m(P', c) : (\text{LOC}(P', x) - \text{LOC}(P, x)) \geq (\text{OBJ}(P, x) - \text{LOC}(P, x))$$

But all the robots that are in $(\text{LOC}(P', x) - \text{LOC}(P, x))$ moved there from c. Consequently:

$$\text{mul}(c) \geq \sum_{x \in X_m(P,c)} (\text{OBJ}(P, x) - \text{LOC}(P, x))$$

□

Now we are ready to prove the main theorem of our chapter:

Theorem 3.3.1. *Given a non-linear configuration P of n robots, there exists an algorithm that detects if P is quasi-regular and if so it outputs its center of quasi-regularity $CQR(P)$.*

Proof. As shown in Lemma 3.1.8, $CQR(P) = \text{WP}(P)$ hence it is unique. If $\text{WP}(P) \in P$, then it can be found by applying Lemma 3.3.2 as follows: for each $p \in P$ we test p is the center of q-regularity of P. Otherwise, $\text{WP}(P) \notin P$ which means that P is regular. Consequently, $\text{WP}(P)$ can be computed as shown in Theorem 3.2.4. □

CHAPTER 4

WAIT-FREE CRASH-RESILIENT GATHERING

In this Chapter we present a wait-free protocol for gathering in the ATOM model in which robots share a common chirality and are endowed with strong multiplicity detectors. The next section present the different classes of configurations that are used by the algorithm and their properties. Section 4.2 gives the protocol and its formal proof of correctness.

4.1 Configurations

In the gathering algorithm, robots compute their next destinations based on the current configuration. Before presenting the algorithm, we present a classification of the robot configurations which will simplify the algorithm description. In the following, we formally define six classes of configurations and prove that they constitute a partition of the set $\mathbb{P}$ of all possible configurations of n robots.

Bivalent($\mathscr{B}$) $\mathscr{B} = \{P \in \mathbb{P} \mid (\forall u \in \mathbb{L}(P) : \text{mul}(u) = n/2)\}$. $\mathscr{B}$ is the set of configurations where the robots are equally distributed over two points in the space.

Multiple ($\mathscr{M}$) $\mathscr{M} = \{P \in \mathbb{P} \mid \exists u \in \mathbb{L}(P) : \forall v \neq u \in \mathbb{L}(P) : \text{mul}(v) < \text{mul}(u)\}$. A configuration P belongs to $\mathscr{M}$ if it has a point u whose multiplicity is greater than that of any other distinct point in P.

Colinear($\mathscr{L}$) $\mathscr{L} = \{P \in \mathbb{P} \mid (P \text{ is linear}) \wedge (P \notin \mathscr{B} \cup \mathscr{M})\}$. We define the subsets $\mathscr{L}1\mathscr{W}$ and $\mathscr{L}2\mathscr{W}$ of colinear configurations depending on whether their We-

ber point is unique or not. That is, $\mathscr{L}1\mathscr{W} = \{P \in \mathscr{L} \mid (WP(P) \text{ is unique})\}$ and $\mathscr{L}2\mathscr{W} = \mathscr{L} \setminus \mathscr{L}1\mathscr{W}$.

Q*Regular ($\mathscr{Q}\mathscr{R}$) $\mathscr{Q}\mathscr{R} = \{P \in \mathbb{P} \mid (\text{qreg}(P) > 1) \wedge (P \notin \mathscr{B} \cup \mathscr{M} \cup \mathscr{L})\}$.

Asymmetric ($\mathscr{A}$) $\mathscr{A} = \{P \in \mathbb{P} \mid (\text{sym}(P) = 1) \wedge (P \notin \mathscr{B} \cup \mathscr{M} \cup \mathscr{L} \cup \mathscr{Q}\mathscr{R}\}$.

Let $\mathbb{X} = \{\mathscr{B}, \mathscr{M}, \mathscr{L}, \mathscr{Q}\mathscr{R}, \mathscr{A}\}$. It is easy to see that $\mathbb{X}$ is a partition of $\mathbb{P}$. By definition, the classes are mutually disjoint. All linear configurations belong to the set $\mathscr{B} \cup \mathscr{M} \cup \mathscr{L}$. For a non-linear configuration P either $\text{sym}(P) > 1$ which implies $P \in \mathscr{Q}\mathscr{R} \cup \mathscr{B} \cup \mathscr{M}$, or $\text{sym}(P) = 1$ which implies that $P \in \mathscr{A} \cup \mathscr{B} \cup \mathscr{M}$. Thus $\bigcup \mathbb{X} = \mathbb{P}$.

Properties of Configurations

Lemma 4.1.1. *Let P be a linear configuration. The following properties hold:*

1. $(|\mathbb{L}(P)| = 2) \Rightarrow (P \in \mathscr{B} \cup \mathscr{M})$
2. $(|\mathbb{L}(P)| = 3) \Rightarrow (P \in \mathscr{M} \cup \mathscr{L}1\mathscr{W})$
3. $(P \in \mathscr{L}2\mathscr{W}) \Rightarrow (|\mathbb{L}(P)| \geq 4)$

Proof. 1. Assume $|\mathbb{L}(P)| = 2$. That is, $\mathbb{L}(P)$ consists in two distinct points u_1 and u_2. If $\text{mul}(u_1) = \text{mul}(u_2)$, then $P \in \mathscr{B}$. Otherwise, $P \in \mathscr{M}$ as either $(\text{mul}(u_1) > \text{mul}(u_2))$ or $(\text{mul}(u_2) > \text{mul}(u_1))$.

2. Assume $|\mathbb{L}(P)| = 3$, *i.e.* $\mathbb{L}(P)$ consists in three distinct points, let them be u_1, u_2, u_3. Suppose w.l.o.g. that $u_2 \in [u_1, u_3]$. We assume that $P \notin \mathscr{L}1\mathscr{W}$ and we prove that $P \in \mathscr{M}$. The fact that $|\mathbb{L}(P)| = 3$ implies that $P \notin \mathscr{B}$. Since P is linear and $P \notin \mathscr{L}1\mathscr{W} \cup \mathscr{B}$, it follows from the definition of $\mathscr{L}$ that $P \in \mathscr{L}2\mathscr{W} \cup \mathscr{M}$. To prove that $P \in \mathscr{M}$ it suffices then to show that $P \notin \mathscr{L}2\mathscr{W}$. Assume towards contradiction that $P \in \mathscr{L}2\mathscr{W}$. This means that the set $Median(P)$ is not a singleton. Hence, there are at least two points in $\mathbb{L}(P)$ that are in $Median(P)$. Consequently, either u_1 or u_3 belongs to $Median(P)$ (together with u_2). Assume w.l.o.g that $u_1 \in Median(P)$. This implies that $\text{mul}(u_1) \geq \lceil n/2 \rceil$. Hence $\text{mul}(u_2) + \text{mul}(u_3) \leq n - \lceil n/2 \rceil$. That is, $\text{mul}(u_2) + \text{mul}(u_3) \leq \lceil n/2 \rceil$. Since $\text{mul}(u_2) \geq 1$ and $\text{mul}(u_3) \geq 1$, it follows that $(\text{mul}(u_2) \leq \lceil n/2 \rceil - 1)$ and $(\text{mul}(u_3) \leq \lceil n/2 \rceil - 1)$. But we showed that $\text{mul}(u_1) \geq \lceil n/2 \rceil$. Consequently we have $\text{mul}(u_2) < \text{mul}(u_1)$ and $\text{mul}(u_3) < \text{mul}(u_1)$. This means that $P \in \mathscr{M}$ which contradicts our assumption that $P \in \mathscr{L}2\mathscr{W}$. This finishes the proof of $P \in \mathscr{M}$ (assuming $|\mathbb{L}(P)| = 3$ and $P \notin \mathscr{L}1\mathscr{W}$). Hence:

$$(|\mathbb{L}(P)| = 3) \Rightarrow (P \in \mathscr{M} \cup \mathscr{L}1\mathscr{W})$$

3. This follows from the above two results.

□

Definition 11 (Safe points)**.** *Given a configuration P, a robot position $p \in P$ is* safe *iff $\forall q \in \mathbb{R}^2 \setminus \{p\}$: $HF(p,q)$ contains at most $(\lceil n/2 \rceil - 1)$ robots of P.*

The notion of safe points is important because any safe point can be used as a gathering point without the possibility that the robots form the bivalent $\mathscr{B}$ configuration while moving towards it. We can show the following properties for safe points.

Lemma 4.1.2. *Any non linear configuration contains a safe point.*

Proof. Let P be a non linear configuration. We say that $Q \subseteq P$ is a quorum iff: (i) $|Q| \geq \lfloor n/2 \rfloor + 1$ and (ii) all points of Q are collinear and Q is maximal for this property, that is, for any $Q' \supset Q$, the points of Q' are not collinear. Let $\text{LINE}(Q)$ denote the line in which are located the points of Q.

Let Q_1 and Q_2 any two *distinct* quorums of P. Condition (i) implies that Q_1 and Q_2 intersect, *i.e.* $Q_1 \cap Q_2 \neq \emptyset$ and the maximality condition in (ii) implies that $\text{LINE}(Q_1) \neq \text{LINE}(Q_2)$.

We show in the following that any point that is not safe belongs necessarily to a quorum. Let $p \in P$ that is not free. We prove the existence of quorum Q to which p belongs. Since p is not safe, there exists $q \in \mathbb{R}^2 \setminus \{p\}$ such that $HF(p,q)$ contains at least $(\lceil n/2 \rceil)$ robots located in it. Hence, $p \cup HF(p,q)$ contains at least $\lceil n/2 \rceil + 1$ robots. Let S denote the multiset of positions of these robots. Since $(|S| \geq \lceil n/2 \rceil + 1)$ and the points of S are collinear, there exists a set Q with $Q \supseteq S$ that is a quorum with $p \in Q$.

We prove the lemma by contradiction. Assume that no point of P is safe, i.e. each point belongs to a quorum. This implies, as P is not linear, that there are at least two distinct quorums because a single quorum cannot contain all elements of P, otherwise the configuration would be linear. Let Q_1 and Q_2 be any two quorums of P with $Q_1 \neq Q_2$ and let p be a point in $Q_1 \cap Q_2$. Since p is not safe according to the contradiction assumption, there exists some $q \in \mathbb{R}^2 \setminus \{p\}$ such that $HF(p,q)$ contains at least $\lceil n/2 \rceil$ robots positions. Denote by X the multiset containing these positions. Note that $|X| \geq \lceil n/2 \rceil$

As $Q_1 \neq Q_2$, it follows according to the maximality of property (ii) of quorums that $\text{LINE}(Q_1) \neq \text{LINE}(Q_2)$. Hence, either $line(p,q) \neq \text{LINE}(Q_1)$ or $line(p,q) \neq \text{LINE}(Q_2)$. Assume *w.l.o.g* that $line(p,q) \neq \text{LINE}(Q_1)$. Since $p \in line(p,q)$, $p \in Q_1$

and $line(p,q) \neq \text{LINE}(Q_1)$ it follows that $line(p,q) \cap \text{LINE}(Q_1) = \{p\}$. Hence, the robots positions that are in $HF(p,q)$ do not belong to Q_1 which means that $X \cap Q_1 = \emptyset$. Hence, $|X \cup Q_1| = |X| + |Q_1| \geq (\lceil n/2 \rceil) + (\lfloor n/2 \rfloor + 1)$. That is $|X \cup Q_1| \geq n+1$. But $|X \cup Q_1| \subseteq P$, a contradiction! Thus the lemma holds. □

Lemma 4.1.3. *If $P \in \mathscr{B} \cup \mathscr{L}2\mathscr{W}$, then P does not have a safe point.*

Proof. Assume towards contradiction that there exists a position $p \in P$ that is safe. Since P is linear, this means that $|\{q \in P \mid q < p\}| \leq (\lceil n/2 \rceil - 1)$ and $|\{q \in P \mid q > p\}| \leq (\lceil n/2 \rceil - 1)$.

But $P \in \mathscr{B} \cup \mathscr{L}2\mathscr{W}$, it follows that P has two distinct median positions, let them be u_1 and u_2 and assume that $u_1 < u_2$. It holds that either $(u_2 > p)$ or $(u_1 < p)$. Assume *w.l.o.g.* that $(u_1 < p)$. Since u_1 is a median position in P, it holds that $|\{q \in P \mid q \leq u_1\}| \geq \lceil n/2 \rceil$. Hence, as $u_1 < p$, it follows that $|\{q \in P \mid q < p\}| \geq \lceil n/2 \rceil$; A Contradiction! □

4.2 The Algorithm

The following lemma is a simple generalization of Lemma 3.1 in [1] that takes into account configurations containing multiplicity points and general adversaries characterized using their cores. Given a configuration P, an algorithm $\mathscr{A}$, denote by $M(P,\mathscr{A})$ the set of positions of robots that $\mathscr{A}$ instructs to move in P [1].

Lemma 4.2.1. *A convergence or gathering algorithm $\mathscr{A}$ is tolerant against an adversary $\mathscr{X}$* only if *at each configuration P, either (1) $M(P,\mathscr{A})$ is a superset of a core of $\mathscr{X}$ or (2) $|\mathbb{L}(P \setminus M(P,\mathscr{A}))| \leq 1$).*

As a consequence, since we want our algorithm to be wait-free ($(n-1)$-tolerant), it must be the case that at each configuration P, there is at *most* one location $c \in \mathbb{L}(P)$ such that the robots at c are allowed to stay in the same position when activated, while all other robots must choose a destination different from the one they are currently occupying. The algorithm must also ensure that robots never reach the configuration $\mathscr{B}$, due to the following impossiblity result adapted from [23].

Lemma 4.2.2. *Starting from a configuration of type $\mathscr{B}$, there is no algorithm that achieves gathering even in fault-free* ATOM *model with strong multiplicity detectors $\diamondsuit M$ and common chirality.*

Proof. A similar result was proved by Dieudonné and Petit [23] without the assumption of common chirality, but their argument still holds if we add it. □

We now define more precisely the objective of a fault-tolerant gathering algorithm. At any time τ during the execution of the algorithm, we define $F(r,\tau) = true$ if robot r has crashed at time $t_i \leq \tau$. The set of non-faulty robots at time τ is denoted by $Live(\mathscr{R},\tau) = \{r_i \in \mathscr{R} | F(r_i,\tau) = false\}$. The set of non-faulty robots at time τ is denoted by $Live(\mathscr{R},\tau) = \{r_i \in \mathscr{R} | F(r_i,\tau) = false\}$.

Definition 12. *Given a set of robots $\mathscr{R}$ that form configuration P at time τ,* GATHERED$(\mathscr{R},\tau)$ = *true iff* $(|\mathbb{L}(Live(\mathscr{R},\tau))| = 1)$ *and* $(M(P,\mathscr{A}) \cap \mathbb{L}(Live(\mathscr{R},\tau)) = \emptyset$.

Input: P (The observed configuration during the precedent LOOK phase).
Output: The destination of the robot.
COMPUTE():

(1) $r \leftarrow$ MY POSITION IN P

(2) **if** $P \in \mathscr{M}$ **then**
(3) $elected \leftarrow \underset{p \in P}{\text{argmax}}\ \text{mul}(p)$
(4) **if** $(r = elected) \vee (\nexists p \in P : p \in (r, elected))$ **then**
(5) **return** $elected$
(6) **else**
(7) $X \leftarrow \{p \in P : p \notin HF(elected, r)\}$
(8) $v \leftarrow \underset{p \in X}{\text{argmin}}\ (k \mid p = S^k(r, elected))$
(9) **let** $d \in \mathbb{R}^2$ *s.t.* $((|d, elected| = |r, elected|) \wedge (\sphericalangle(r, elected, d) = \sphericalangle(r, elected, v)/3))$
(10) **return** d

(11) **if** $P \in 2\mathscr{R} \cup \mathscr{L}1\mathscr{W}$ **then**
(12) **return** $WP(P)$

(13) **if** $P \in \mathscr{A}$ **then**
(14) $X \leftarrow$the set of **safe** points in $\mathbb{L}(P)$.
(15) $elected \leftarrow \underset{p \in X}{\text{argmax}}\ (\text{mul}(p), \frac{1}{\sum_{q \in P} dist(p,q)}, \mathscr{V}(p))$
(16) **return** $elected$

(17) **if** $P \in \mathscr{L}2\mathscr{W}$ **then**
(18) $c \leftarrow center(P)$
(19) **if** $r \notin CH(P)$ **then**
(20) **return** c
(21) **else**
(22) **let** $d \in \mathbb{R}^2$ *s.t.* $(|d, c| = |r, c|) \wedge (\sphericalangle(r, c, d) = \pi/4)$
(23) **return** d

Figure 4.1: Gathering Algorithm: COMPUTE Phase.

Gathering Algorithm

We now describe the algorithm in terms of actions taken by a robot r based on the current configuration and the position of the robot r within the configuration. A more technical description is given in Figure 4.1.

Configuration $P \in \mathscr{M}$
Let c be the unique point of maximum multiplicity in P. If robot r is located at c, it does not move. Otherwise, if there are no robots between r and c, robot r moves directly towards c and if not, it does a side-step i.e. it moves to the closest point d on a half-line $HF(c,d)$ such that the angle between half-line $HF(c,d)$ and half-line $HF(c,r)$ is less than or equal to 1/3 of the angle between half-line $HF(c,p)$ and half-line $HF(c,r)$, for any other robot location $p \in P$. This ensures the robot does not collide with another robot, i.e. it does not create a new point of maximum multiplicity. Note that there may be multiple robots colocated with r, these robot may make the same move as r. However the value of mul(r) would never increase unless r reaches the point c. Thus, the algorithm ensures that the robots remain in a configuration of type $\mathscr{M}$ until gathering is achieved.

Configuration $P \in \mathscr{L}1\mathscr{W}$
By definition, we know that configuration P contains a unique Weber-point c which is also the median and can be computed easily. Each robot r moves directly towards the Weber point c which remains invariant during the movement. Eventually the configuration changes to $\mathscr{M}$ or a gathered configuration.

Configuration $P \in \mathscr{Q}\mathscr{R}$
In this case, robot r moves to the center c of quasi-regularity of P (which is also the Weber-point). Thus, the Weber-point c remains invariant during the movement and eventually the configuration changes to $\mathscr{M}$ or a gathered configuration.

Configuration $P \in \mathscr{A}$
Since P is not linear, we know that there exists a safe point in $\mathbb{L}(P)$. When there are multiple safe points, the algorithm selects a unique point c from among the safe points in $\mathbb{L}(P)$. This is always possible since the configuration is asymmetric (i.e. the view of each point is unique). The algorithm chooses the point c based on the multiplicity(c), the sum of distances of all other robots to c, and finally the view of c (in this order, and maximizing the first parameter, minimizing the second parameter and maximizing the third parameter). Each robot r moves towards this unique point c. We will show that the configuration P' obtained after one step of the algorithm is of type $\mathscr{M}$, $\mathscr{Q}\mathscr{R}$, $\mathscr{L}1\mathscr{W}$, or $\mathscr{A}$ (but not $\mathscr{B}$ or $\mathscr{L}2\mathscr{W}$). Further if

the next configuration is again of type $\mathscr{A}$ then either the maximum multiplicity increases or the minimum sum of distance decreases. This ensures that the algorithm converges towards a configuration of type $\mathscr{M}$ or a gathered configuration.

Configuration $P \in \mathscr{L}2\mathscr{W}$
In this case, there are at least 4 distinct points in the configuration. The algorithm instructs the robots at the two end-points of the line to move away from the line. Any robot that is not located in one of the end-points is instructed to move towards the center of the line. If any of the robots at the end-points move then the next configuration would be non-linear and thus, the algorithm switches to one of the other cases above. Otherwise, if the robots are the end-points never move (i.e. they are crashed) then the configuration remains linear but the sum of distances between correct robots decreases, and the robots would eventually converge to a gathered configuration or a configuration of type $\mathscr{M}$.

4.3 Proof of Correctness

We now show that starting from any configuration except the bivalent configuration $\mathscr{B}$, the algorithm described in Figure 4.1 eventually forms a gathered configuration. The proof is divided into several parts, each dealing with configuration of a different type.

Configurations of type $\mathscr{M}$

Lemma 4.3.1. *Let $P_{\mathscr{R}}(\tau) \in \mathscr{M}$. There exists a time $\tau' > \tau$ such that* GATHERED$(\mathscr{R},\tau)$=*true.*

Proof. Let $elected(\tau)$ be the destination chosen by the robots in configuration $P(\tau)$, i.e. $elected(\tau) = \underset{p \in P(\tau)}{\text{argmax}}\ \text{mul}(p)$. A robot position $p \in P(\tau)$ is said to be **free** with respect to $elected(\tau)$ if no robot is located in the interval (p,c). The lemma follows from the following two claims that we prove below:

C1: $(P(\tau) \in \mathscr{M}) \Rightarrow ((P(\tau+1) \in \mathscr{M}) \wedge (elected(\tau+1) = elected(\tau)))$

C2: $(\forall \tau_i \geq \tau : (P(\tau_i) \in \mathscr{M}) \wedge (elected(\tau_i) = elected(\tau))) \Rightarrow (\exists \tau' \geq \tau :$ GATHERED$(\mathscr{R},\tau')$=true$)$.

Proof of C1: Let $c = elected(\tau)$ be the point of maximum multiplicity in the configuration $P(\tau)$. We need to show that c remains the point of maximum multiplicity in $P(\tau+1)$. In fact we show a stronger result that no two robots that were in

distinct locations at τ can be at the same location at time $\tau+1$, unless the robots are at c. Let us assume the contrary, i.e. let r_1 and r_2 be robots that occupied distinct locations in $P(\tau)$ but occupy the same location $p \neq c$ in $P(\tau+1)$. Note that neither of the robots r_1 and r_2 are located at c at time τ (since otherwise the algorithm would instruct them to remain at c and they would not be at p at $\tau+1$).

According to the algorithm any robot r in configuration $P(\tau) \in \mathscr{M}$ can make two possible moves: (i) either robot r moves directly towards c (Line (5) of algorithm) or, (ii) robot r moves to a point d such that rcd is an isosceles triangle with central angle $0 < \theta < \pi/3$ at c (Line (9) of algorithm). If both the robots r_1 and r_2 both make move of type (i), then they are distinct free points and in this case their paths may not intersect except at c. Otherwise, suppose one of the robots (say r_1) makes a move of type (ii) directly towards a point d. Consider the triangle r_2cd and let $\sphericalangle(r_2,c,d) = \theta$. The other robot r_2 is located either on the half-line $HF(c,r_1)$ or, on a different half-line $HF(c,r_2)$ which forms an angle greater than $3*\theta$ with the half-line $HF(c,r_1)$ w.r.t. point c. In the second case, the path of robot r_2 will never intersect the line segment between c and d. In the first case, either robot r_2 is on a free point (and thus, it will move on the line segment $[c,r_2]$ which doesnot intersect the line segment $[c,d]$) or robot r_2 is not free and thus it makes a move on a line segment parallel to $[c,d]$. In both cases, there is no common point p in the path of the two robots.

Proof of C2: In this case, if $c = elected(\tau)$ is the point of maximum multiplicity in $P(\tau)$ then c is the unique point of maximum multiplicity in all subsequent configurations. Whenever a robot on a free point is activated it moves closer to the point c. Whenever a blocked (i.e. not free) robot is activated, at least one robot moves from a blocked position to a free point. Once a robot r moves to a free point at time τ_i, it may be blocked in subsequent steps by only robots that moved with robot r in that same time step (i.e. these robots were *live* at that time step). Thus an adversary can prevent a non-faulty robot r from reaching point c only by changing a live robot to a crashed robot after each step in which robot r is activated. After a finite time, the adversary will run out of live robots. Thus all live robots will eventually reach c. Once a robot reaches c, the algorithm never instructs the robot to move (since c is the unique point of maximum multiplicity). Thus, GATHERED$(\mathscr{R},\tau)$ will be true at that time. □

Configurations of type $\mathscr{L}1\mathscr{W}$

Lemma 4.3.2. *Let $P(\tau) \in \mathscr{L}1\mathscr{W}$. There exists a time $\tau_c > \tau$ such that either $(P(\tau_c) \in \mathscr{M})$ or, $(\text{GATHERED}(\mathscr{R},\tau_c) = true)$.*

Proof. The lemma follows from the following two claims that we prove below:

C1: $(P(\tau) \in \mathscr{L}1\mathscr{W}) \Rightarrow ((P(\tau+1) \in \mathscr{M} \cup \mathscr{L}1\mathscr{W}) \wedge (WP(P(\tau+1)) = WP(P(\tau))))$

C2: $(\forall \tau' \geq \tau : (P(\tau') \in \mathscr{L}1\mathscr{W}) \wedge (WP(P(\tau')) = WP(P(\tau)))) \Rightarrow (\exists \tau_c \geq \tau :$ GATHERED$(\mathscr{R}, \tau_c) = true)$.

Proof of C1: Since $P(\tau) \in \mathscr{L}1\mathscr{W}$, it follows that $WP(P(\tau)) = c$ is unique and $P(\tau+1)$ is obtained by moving robots in $P(\tau)$ towards c (line 12 of the algorithm). Hence, according to Corollary 3.1.1, $WP(P(\tau+1)) = WP(P(\tau)) = c$. Moreover, $P(\tau+1)$ is linear. This, combined with the fact that its Weber point is unique implies that $P(\tau+1)$ cannot be of type $\mathscr{B}$ or $\mathscr{L}2\mathscr{W}$. Therefore, $P(\tau+1) \in \mathscr{M} \cup \mathscr{L}1\mathscr{W}$.

Proof of C2: Whenever a robot in configuration $\mathscr{L}1\mathscr{W}$ is activated it moves towards the Weber-point c and Weber-point remains invariant due to this movement. Thus, for all configurations $P(\tau')$ the Weber-point is the same point c. For each non-faulty robot r the distance between r and c decreases every time the robot r is activated (unless r is already at c). Thus all non-faulty robots are at the point c at some time τ_c and GATHERED$(\mathscr{R}, \tau_c)$=true. □

Configurations of type $\mathscr{Q}\mathscr{R}$

Lemma 4.3.3. *Let $P(\tau) \in \mathscr{Q}\mathscr{R}$. There exists a time $\tau_c > \tau$ such that* $(P(\tau_c) \in \mathscr{M} \cup \mathscr{L}1\mathscr{W}) \vee$ (GATHERED$(\mathscr{R}, \tau_c) = true$).

Proof. Let $c = WP(P(\tau))$. The lemma follows from the following two claims that we prove below:

C1: $(P(\tau) \in \mathscr{Q}\mathscr{R}) \Rightarrow (P(\tau+1) \in \mathscr{M} \cup \mathscr{L}1\mathscr{W} \cup \mathscr{Q}\mathscr{R}) \wedge (WP(P(\tau+1)) = WP(P(\tau))))$

C2: $(\forall \tau' \geq \tau : (P(\tau') \in \mathscr{Q}\mathscr{R}) \wedge (WP(P(\tau')) = WP(P(\tau))) \Rightarrow (\exists \tau_c \geq \tau :$ GATHERED$(\mathscr{R}, \tau_c) = true)$.

Proof of C1: Since $P(\tau) \in \mathscr{Q}\mathscr{R}$, robots that are activated at τ move towards $WP(P(\tau)) = CQR(P(\tau))$ according to line (12) of the code. Hence, since $P(\tau)$ is Q-regular, the obtained configuration $P(\tau+1)$ is Q-regular also with the same center of Q-regularity as $P(\tau)$ Hence, $WP(P(\tau+1)) = CQR(P(\tau+1)) = CQR(P(\tau)) = WP(P(\tau))$.

As $P(\tau+1)$ is Q-regular, it holds according to the definition of configurations $\mathscr{Q}\mathscr{R}$ that if $P(\tau+1) \notin \mathscr{B} \cup \mathscr{M} \cup \mathscr{L}$ then $P(\tau+1) \in \mathscr{Q}\mathscr{R}$. Therefore, $P(\tau+1) \in \mathbb{B} \cup \mathscr{M} \cup \mathscr{L} \cup \mathscr{Q}\mathscr{R}$. It remains to show that $P(\tau+1) \notin \mathscr{B} \cup \mathscr{L}2\mathscr{W}$. For this, it suffices to show

that $P(\tau+1)$ has a unique Weber point. But this follows Corollary 3.1.1 and the fact that $WP(P(\tau))$ is unique.

Proof of C2: Since all the configurations after τ are of type $\mathcal{2R}$ and since the Weber point remains invariant after τ, it follows that all activated robots after τ choose the same destination point: $WP(P(\tau))$. Hence, there is a time τ_c at which all live robots have reached this point. That is, $\text{GATHERED}(\mathcal{R},\tau_c) = true$. □

Configurations of type $\mathcal{A}$

Lemma 4.3.4. *Let $P(\tau) \in \mathcal{A}$. There exists a time $\tau_c > \tau$ such that $(P(\tau_c) \in \mathcal{M} \cup \mathcal{L}1\mathcal{W} \cup \mathcal{2R}) \vee (\text{GATHERED}(\mathcal{R},\tau_c) = true)$.*

Proof. Given a configuration P, let $\phi(P)$ be the couple of values defined by $(mult, sum) = max\{(\text{mul}(p), \frac{1}{\sum_{q\in P}|p,q|}) \mid p \in P\}$.

The lemma follows from the claims C1 and C3 below. Claim C2 is used to prove C3.

C1: $(P(\tau) \in \mathcal{A}) \Rightarrow (P(\tau+1) \in \mathcal{M} \cup \mathcal{L}1\mathcal{W} \cup \mathcal{2R} \cup \mathcal{A})$

C2:

$$\begin{array}{lcl} (P(\tau) \in \mathcal{A}) & \Rightarrow & (P(\tau+1) = P(\tau)) \\ & \vee & (\phi(P(\tau+1)).mult > \phi(P(\tau)).mult) \\ & \vee & (\phi(P(\tau+1)).mult = \phi(P(\tau)).mult) \wedge (\phi(P(\tau+1)).sum^{-1} < \phi(P(\tau)).sum^{-1})) \end{array}$$

C3: $(\forall \tau' \geq \tau : P(\tau') \in \mathcal{A}) \Rightarrow (\exists \tau_c \geq \tau : \text{GATHERED}(\mathcal{R},\tau_c) = true)$.

Proof of C1: $P(\tau) \in \mathcal{A}$ means that $sym(P(\tau)) = 1$. Hence, each position in $\mathbb{L}(P(\tau))$ has a unique view. This guarantees that the "elected" position computed in line 15 is unique and common to all activated robots at τ, let us denote it by u. Moreover, u is safe in $P(\tau)$. Hence, all activated robots at τ move towards the same safe point u which results in configuration $P(\tau+1)$. We observe that u is also safe in $P(\tau+1)$ since for all $x \in \mathbb{R}^2 \setminus \{u\}$, the number of robots that are located at $HF(u,x)$ does not increase between τ and $\tau+1$ (it may even decrease if some of them reach u). Thus, $P(\tau+1)$ contains at least one safe point (u). According to Lemma 4.1.3 this implies that $P(\tau+1) \notin \mathcal{B} \cup \mathcal{L}2\mathcal{W}$ which suffices to prove the claim.

Proof of C2: Assume that $P(\tau+1) \neq P(\tau)$. Let $u \in P(\tau)$ be the common "elected" position chosen by the algorithm (line 15). Hence, by definition, $(\text{mul}(u), \sum_{p_i(\tau)\in P(\tau)} |u, p_i(\tau)|) = \phi(P(\tau))$. Since all activated robots at τ move to the same destination u, it follows that the resulting configuration $P(\tau+1)$ satisfies:

$$\sum_{p_i(\tau+1)\in P(\tau+1)} |u, p_i(\tau+1)| \leq \sum_{p_i(\tau)\in P(\tau)} |u, p_i(\tau)| = \phi(\tau).sum^{-1}$$

Since $(P(\tau+1) \neq P(\tau))$, there exists at least one robot r_i whose position at $\tau+1$ is distinct from its position at τ. That is $p_i(\tau+1) \neq p_i(\tau)$. Note that since all robots move towards u, it follows that $p_i(\tau+1) \in [p_i(\tau), u]$. We distinguish between two cases:

1. $p_i(\tau+1) = u$. In this case $\text{mul}(u)$ is incremented. Note that u is still safe in $\tau+1$ (as shown in the proof of C1). Hence, $\phi(\tau+1).mult = \text{mul}(u)^{\tau+1} > \phi(\tau).mult$

2. $p_i(\tau+1) \neq u$. That is, r_i is stopped by the scheduler before it reaches u. But since the scheduler guarantees to each robot to move by a distance of at least Δ before it can stop it, it follows that $(|u, p_i(\tau+1)| \leq |u, p_i(\tau)| - \Delta)$ which, combined with the above inequality gives:

   $$\sum_{p_i(\tau+1)\in P(\tau+1)} |u, p_i(\tau+1)| \leq (\sum_{p_i(\tau)\in P(\tau)} |u, p_i(\tau)|) - \Delta = \phi(P(\tau)).sum^{-1} - \Delta$$

   Note that the multiplicity of u does not decrease even if no robot reaches it, *i.e.* $\text{mul}(u)^{\tau+1} \geq \phi(P(\tau)).mult$. Note that u is still safe in $\tau+1$ (as shown in the proof of C1). Hence,

   $$\begin{aligned} \phi(P(\tau+1)) &\geq (\text{mul}(u), \tfrac{1}{\sum_{p_i(\tau+1)\in P(\tau+1)} |u, p_i(\tau+1)|}) \\ &\geq (\phi(\tau).mul, \tfrac{1}{\phi(\tau).sum^{-1} - \Delta}) \end{aligned}$$

   Therefore, either $(\phi(\tau+1).mul > \phi(\tau))$ or $((\phi(\tau+1).mul = \phi(\tau)) \wedge (\phi(\tau+1).sum^{-1} \leq \phi(\tau).sum^{-1} - \Delta)$.

This proves the claim.

Proof of C3: Assume that $(\forall \tau' \geq \tau : P(\tau') \in \mathscr{A})$. We have to prove that:

$$\exists \tau_g \geq \tau : \forall \tau' \geq \tau_g : (P(\tau')) = P(\tau))$$

That is, the configuration does not change after τ_g which implies the existence of a time after τ_g at which all the live robots lie on the same position. That is $\exists \tau_c \geq \tau_g : \text{GATHERED}(\mathscr{R}, \tau_c) = true$.

The claim follows from claim C2 above. There exists a time $\tau_1 \geq \tau$ after which $\phi().mult$ cannot increase since the multiplicity of points is upper bounded by n. Moreover, there exists a time $\tau_2 \geq \tau_1$ after which $\phi().sum^{-1}$ cannot decrease since the sum of distance is lower bounded by 0. Hence, according to claim C2, after time τ_2, the configuration remains the same and the claim follows by setting $\tau_c = \tau_2$.

□

Configurations of type $\mathscr{L}2\mathscr{W}$

Definition 13. *Assume that $P(\tau)$ is linear. Let $u^-(\tau)$ and $u^+(\tau)$ denote $min(\mathbb{L}(P(\tau)))$ and $max(\mathbb{L}(P(\tau)))$ respectively. Denote by $S^-(\tau)$ and $S^+(\tau)$ the set of robots located at $u^-(\tau)$ and $u^+(\tau)$ respectively and let $S^0(\tau) = \mathscr{R} \setminus S^-(\tau) \cup S^+(\tau)$.*

If $P(\tau) \in \mathscr{L}2\mathscr{W}$, then Lemma 4.1.1 implies that $|\mathbb{L}(P(\tau))| \geq 4$. Hence, the sets $S^-(\tau)$, $S^0(\tau)$ and $S^+(\tau)$ in this case are non empty and pairwise disjoint.

Lemma 4.3.5. *If $P(\tau) \in \mathscr{L}2\mathscr{W}$, then $P(\tau+1) \notin \mathscr{B}$.*

Proof. It suffices to show that $|\mathbb{L}(P(\tau+1))| \geq 3$. This simply follows from the fact that robots of $S^-(\tau)$, $S^0(\tau)$ and $S^+(\tau)$ occupy distinct positions and these groups remain disjoint when the robots activated at τ move towards their destinations (computed in line 18 for $S^0(\tau)$ and line 22 for $S^-(\tau)$ and $S^+(\tau)$). □

Lemma 4.3.6. *Assume $P(\tau) \in \mathscr{L}2\mathscr{W}$. If at least one robot in $S_-(\tau) \cup S^+(\tau)$ is activated at τ, then $P(\tau+1) \notin \mathscr{L}2\mathscr{W}$.*

Proof. Let a and b denote $u^-(\tau)$ and $u^+(\tau)$ respectively and let c be the midpoint of $[a,b]$. Due to Lemma 4.1.1 we know that $|\mathbb{L}(P(\tau))| \geq 4$ and thus other than a and b, there exists at least two other points in $\mathbb{L}(P)$. The scenario considered in this lemma can be partitioned into the following three cases: (i) No robot located at a are activated at step τ (ii) No robot located at b are activated at step τ (iii) At least one robot from each of a and b are activated. We will show that in each case, $P(\tau+1) \notin \mathscr{L}2\mathscr{W}$. Let $L = line(a,b)$. Note that all robots lie on L at time τ.

Case (i): In this case, at least one robot r located at point b is activated and according to the algorithm, the robot r moves towards a point p such that $\sphericalangle(b,c,p) = \pi/4$. The new position p' reached by the robot r lies in (b,p) and thus, $p' \notin L$. Note that any robot $r' \in S^0(\tau)$, still remains on line L at some point distinct from a (since robots in $S^0(\tau)$ are allowed to move only towards $c \in L$). Thus, $\mathbb{L}(P(\tau+1))$ contains the points $p' \notin L$, $a \in L$, and at least one other point in L that is distinct from a. Hence $P(\tau+1)$ is not linear, which implies that $P(\tau+1) \notin \mathscr{L}2\mathscr{W}$.

Case (ii): This case is exactly symmetrical to case (i) above and the same result holds.

Case (iii): If not all the robots located at a and b are activated at time τ then we can use similar arguments as above to show that the configuration $P(\tau+1)$ is not linear. Thus the only interesting case to consider is when all robots at a move to the same location a' and all robots at b move to the same location b'. Note that $a' \notin L$ and $b' \notin L$ and $a' \neq b'$. Thus, $line(a', b')$ is distinct from line L. However all the robots $\in S^0(\tau)$ must remain on line L in step $(\tau+1)$. If the configuration $P(\tau+1)$ is linear then all robots $\in S^0(\tau)$ must be located on the same point at step $(\tau+1)$ and this point must be the point of intersection of L and $line(a', b')$. In other words, $|\mathbb{L}(P(\tau+1))| = 3$, which implies that $P(\tau+1) \notin \mathscr{L}2\mathscr{W}$ (due to Lemma 4.1.1). □

Lemma 4.3.7. *Let $P(\tau) \in \mathscr{L}2\mathscr{W}$. There exists a time $\tau_c > \tau$ such that $(P(\tau_c) \in \mathscr{M} \cup \mathscr{L}1\mathscr{W} \cup \mathscr{Q}2\mathscr{R} \cup \mathscr{A}) \vee (\text{GATHERED}(\mathscr{R}, \tau_c) = true)$.*

Proof. Assume for the sake of contradiction that $\forall \tau' \geq \tau : (P(\tau') \in \mathscr{L}2\mathscr{W} \cup \mathscr{B}) \wedge (\text{GATHERED}(\mathscr{R}, \tau') = false)$.

But since $P(\tau) \in \mathscr{L}2\mathscr{W}$ and Lemma 4.3.5 says that a configuration of type $\mathscr{B}$ cannot come after a configuration of type $\mathscr{L}2\mathscr{W}$, it follows that:

$$\forall \tau' \geq \tau : (P(\tau') \in \mathscr{L}2\mathscr{W}) \wedge (\text{GATHERED}(\mathscr{R}, \tau') = false)$$

According to Lemma 4.3.6, this is only possible if the robots located at the endpoints are never activated, which means that they are all faulty. Hence, the center of the configuration $c = center(P(\tau))$ remains constant during all the execution and all correct robots eventually reach this point (line 18 of the algorithm). Hence, there is a time $\tau_c > \tau$ at which all correct robots are located at c. Thus, $\text{GATHERED}(\mathscr{R}, \tau_c) = true$. A contradiction. □

CHAPTER 5

BYZANTINE CONVERGENCE

In this chapter we prove necessary conditions for the convergence of mobile robots despite a subset of them being Byzantine, when these robots can move in a uni-dimensional space. First, we prove that the strong multiplicity detection capability is necessary (Section 5.2). Then, we prove several lower bounds relating synchrony assumptions and byzantine resilience. Recall that n denotes the total number number of robots while f is the maximum number of faulty robots.

In more details, we show that byzantine-resilient convergence in unidimensional networks is impossible to solve:

1. in FS if $n \leq 2f$ (Section 5.3).
2. in B(k) with $k > 1$ if $n \leq 3f$ (Section 5.4).
3. in AS if $n \leq 5f$ and the convergence algorithm is *cautious*, *i.e.*, if it always instructs correct robots to move inside the range of all positions held by the correct robots regardless of the locations of byzantine ones (Section 5.5).

These impossibility results hold whether the model is atomic or not.

We then show that these bounds are tight by providing matching upper bounds. For this, we present three convergence algorithms:

1. The first algorithm (Section 5.7) solves gathering in the ATOM[FS] model provided that $n > 2f$.

2. The second proposed algorithm (Section 5.8) works in NTOM[B(k)] when $n > 3f$.

3. The last algorithm (Section 5.9) achieves convergence in the most general setting assuming that $n > 5f$: the NTOM[AS] model.

5.1 Preliminaries

In this section we propose a framework used further in deriving our lower bounds and proving our impossibility results. First, we give a formal definition of cautious algorithms. Then, we present the notion of equivalence between configurations and define in a precise way assuming various multiplicity detection capability assumptions Finally, we prove some important technical results that will be used in proving the necessity of strong multiplicity detection and the lower bounds of resilience in the following sections.

5.1.1 Cautious Algorithms

A natural way to solve convergence is to never let the algorithm increase the diameter of correct robot positions. We say in this case that the algorithm is *cautious*. A cautious algorithm is particularly appealing in the context of byzantine failures since it always instructs a correct robot to move inside the range of the positions held by the correct robots regardless of the locations of byzantine ones. The notion of cautiousness was introduced in [24] in the context of classical byzantine-tolerant distributed systems. In the following, we customize this notion to robot networks.

Definition 14. *Let $D_i(\tau)$ be the last destination calculated by the robot r_i before time τ and let $U^i(\tau)$ the positions of the correct robots as seen by robot r_i before time τ*[1]*. An algorithm is* cautious *iff $\forall \tau \in \mathbb{T}$, $D_i(\tau) \in range(U^i(\tau))$ for each robot r_i.*

5.1.2 Equivalence of Configurations

In this subsection, we formalize the notion of equivalence between configurations. Informally speaking, two configurations are equivalent if they are completely indistinguishable to individual robots. It follows that the exact definition of this notion will depend on the assumptions on the system and on the capabilities of robots. For example, a configuration of a single point located at position (+1) is different from that consisting of a single point located at position (0) if robots have

[1] If the last calculation was executed at time $\tau' \leq \tau$ then $D_i(\tau) = D_i(\tau')$.

a global common coordinate system. However, both configurations are equivalent when, as we consider in this paper, each robot has its own local coordinate system. In this case, any two configurations such that each one can be obtained from the other by way of translation, symmetry or rotation are considered to be equivalent. This results from the absence of a common coordinate system between robots.

Similarly, a configuration consisting of a single robot is equivalent to that consisting of two robots co-located at the same position only if these robots are devoid of any multiplicity detection capability. We observe that the weaker the capabilities of robots and the assumptions of the system are, the larger the equivalence classes of configurations are. In the sequel, we define three equivalence relations between configurations under the model presented in the previous section while distinguishing between three assumptions on the multiplicity detection capability of robots.

Definition 15. *(equivalence) Two configurations P and P' are said to be* null equivalent*, denoted $P \underset{\varnothing M}{\sim} P'$, if there exists a one-to-one mapping $\pi : \mathbb{L}(P) \mapsto \mathbb{L}(P')$ and a real factor $\alpha > 0$ such that $\forall p, q \in \mathbb{L}(P) : |p, q| = \alpha \cdot |\pi(p), \pi(q)|$.*

If moreover it holds that $\forall p \in \mathbb{L}(P) : (mul(p) > 1) \Leftrightarrow (mul(\pi(p)) > 1)$, then P and P' are weak equivalent *and we denote it by writing $P \underset{?M}{\sim} P'$.*

Finally, if $P \underset{\varnothing M}{\sim} P'$ and $\forall p \in \mathbb{L}(P) : mul(p) = mul(\pi(p))$, then the two configuration are strong equivalent *and we write $P \underset{\Diamond M}{\sim} P'$.*

The definition means that if $P \underset{\varnothing M}{\sim} P'$ and robots do not have any multiplicity detectors, they behave the same in P and P' since they cannot distinguish between these two configurations and they run the same deterministic algorithm. A similar observation can be made about the equivalence relations $P \underset{?M}{\sim} P'$ and $P \underset{\Diamond M}{\sim} P'$ when robot have weak and strong multiplicity detectors respectively.

Note that configurations that are weak (resp. strong) equivalent are necessarily null (resp. weak) equivalent, but the converse is not necessarily true. In order to be able to fully characterize weak (resp. strong) equivalence, robots must be endowed with weak (resp. strong) multiplicity detectors (or stronger ones). Thus, the multiplicity detection capability allows robots to more accurately characterize the equivalence between configurations.

The following property follows from the definition.

Property 5.1.1. *Let P and P' be two configurations of n mobile robots. It holds that:* $(P \underset{\Diamond M}{\sim} P') \Rightarrow (P \underset{?M}{\sim} P') \Rightarrow (P \underset{\varnothing M}{\sim} P')$.

This property means that two configurations that are equivalent when robots are equipped with a multiplicity detector are also equivalent when no multiplicity detection or when a weaker kind of it is assumed, but the converse is not true. Thus, endowing robots with multiplicity detectors allows them to distinguish between configurations that otherwise would have been indistinguishable to them.

5.1.3 Invariants

The following theorem is fundamental to our proof. It states that if at least $n-f$ robots are co-located in the same position, then any convergence algorithm will instruct them to stay in this position. Formally, we have:

Theorem 5.1.1. *Let $P(\tau), \tau \in \mathbb{T}$ be any configuration and let $SetX$ be a subset of at least $(n-f)$ robots in $P(\tau)$ that are colocated at the same position, let it be x; then all destinations computed by robots of $SetX$ at τ using any convergence algorithm are equal to x regardless of the robots' multiplicity detection capabilities.*

Proof. Fix $P_1 = P(\tau)$ and let $SetX \subseteq P_1$ with $|setX| \geq n-f$ and $\mathbb{L}(setX) = \{x\}$.

The proof of our lemma proceeds by contradiction. We assume there exists a convergence algorithm $\mathscr{A}$ that instructs the robots of $SetX$ to move to some position $y \neq x$ when they are activated by the scheduler. Assume without loss of generality that $y < x$ (y is at the left of x). This order is only given for ease of presentation and is unknown to robots that can not use it in their algorithms.

We now inductively create an execution in which the correct robots that run algorithm $\mathscr{A}$ form a moving multiplicity point. That is, they stay always together but they move indefinitely to the left by a distance equal to $|p,q|$ at each movement. Hence, convergence is prevented.

Let P_2 be a configuration such that $P_1 \underset{\diamondsuit M}{\sim} P_2$ (take $\alpha = 1$), but where all the correct robots are located at x. This is possible since $|SetX| \geq n-f$. We denote by $SetX_2$ the set of robots located at x in P_2. Note that $SetX_2$ may contain also some Byzantine robots. According to Property 5.1.1, since $P_1 \underset{\diamondsuit M}{\sim} P_2$, it holds also that $P_1 \underset{?M}{\sim} P_2$ and $P_1 \underset{\varnothing M}{\sim} P_2$. So, P_1 and P_2 are completely indistinguishable to individual robots regardless of their multiplicity detection capabilities. Thus, robots must behave the same in both configurations as they run the same deterministic algorithm.

Hence, when the correct robots of $SetX_2$ are activated in P_2, their computed destination by $\mathscr{A}$ must be equal to y similarly to the robots of $SetX$ in P_1. Assume that the scheduler activates all the robots of $SetX_2$ simultaneously and does not stop them before they reach their destination y. At the same time, each Byzantine robot is moved by the scheduler to the left by a distance equal to $|x,y|$. Denote by

P_3 the resulting configuration. Clearly, $P_3 \underset{\Diamond M}{\sim} P_2$. Therefore, by repeating the same actions indefinitely, the adversary is able to make the correct robots move at each cycle by a distance equal to $|x,y|$. Thus, convergence is prevented which contradicts the assumption of $\mathscr{A}$ being a correct convergence algorithm. This proves our lemma. □

Lemma 5.1.1. *Let $P(\tau), \tau \in \mathbb{T}$ be a configuration of robots endowed with ?M. If $(|\mathbb{L}(P(\tau))| + |\{p \in \mathbb{L}(P(\tau)) \mid mul(p) > 1\}|) \leq f+2$; then all robots that are located at multiplicity points do not move if activated at τ.*

Proof. Fix $P = P(\tau)$ to be a configuration of robots (equipped with ?M) such that:

$$(|\mathbb{L}(P)| + |\{p \in \mathbb{L}(P) \mid \text{mul}(p) > 1\}|) \leq f+2$$

Let S denote the set $\{p \in \mathbb{L}(P) \mid \text{mul}(p) > 1\}$. Let l and k be respectively the cardinalities $|\mathbb{L}(P)|$ and $|S|$. Hence $l+k \leq f+2$. Note that $k \leq l$. If $k = 0$, then all robots are byzantine and the lemma holds trivially. So we consider in the following that $k \geq 1$.

Denote by $q_1, \ldots, q_k$ the elements of S.

Now we construct a configuration $P_{i1} = \pi_1(P)$ such that $\text{mul}(\pi_1(q_1)) = n-l-k+2$, $\text{mul}(\pi_1(p)) = 2$ for every $p \in S$ with $p \neq q_1$ and $\text{mul}(\pi_1(p)) = 1$ for every $p \in P \setminus S$.

To show that P_{i1} is well defined it suffices to prove that the sum of the multiplicities of its points is equal to n.

$$\begin{array}{rcl} \sum_{p \in P_{i1}} \text{mul}(p) & = & \sum_{p \in P} \text{mul}(\pi_1(p)) \\ & = & \text{mul}(\pi_1(q_1)) + \sum_{p \in S, p \neq q_1} \text{mul}(\pi_1(p)) + \sum_{p \in P \setminus S} \text{mul}(\pi_1(p)) \\ & = & (n-l-k+2) + 2(k-1) + (l-k) \\ & = & n \end{array}$$

Hence P_{i1} is well defined. Recall that $\text{mul}(\pi_1(q_1)) = n-l-k+2$, But we assumed that $l+k \leq f+2$, that is, $l+k-2 \leq f$. This means that $\text{mul}(\pi_1(q_1)) \geq n-f$. Therefore, according to Theorem 5.1.1, the robots located at q_1 do not move if activated at τ. It is easy to show that $P \underset{?M}{\sim} P_{i1}$, which implies that the robots of P that are located at q_1 do not move neither if activated at τ.

We construct the configurations $P_{i2}, \ldots, P_{ik}$ similarly to P_{i1} and we can prove by using the same argument as above that the robots of P located at $q_2, \ldots q_k$ do not move if activated at τ. This proves the lemma. □

Corollary 5.1.1. *Let $P(\tau), \tau \in \mathbb{T}$ be a configuration of robots endowed with ?M. If $(|\mathbb{L}(P(\tau))| + |\{p \in \mathbb{L}(P(\tau)) \mid mul(p) > 1\}|) \leq f+2$ and byzantine robots never move*

after τ; then all robots that are located at multiplicity points do not move if activated after τ.

Proof. It suffices to prove that for all $\tau' \geq \tau$, $(|\mathbb{L}(P(\tau'))| + |p \in \mathbb{L}(P(\tau')) \mid \text{mul}(p) > 1\}|) \leq f+2$ so that we can apply lemma 5.1.1 for each $\tau' \geq \tau$.

The proof proceeds by induction on the time instants $\tau' \geq \tau$. The basis is τ and the claim is true according to Lemma 5.1.1.

For the inductive step, we assume the claim holds for time $\tau' - 1 \geq \tau$ and we prove it for τ'. Hence, we suppose that $(|\mathbb{L}(P(\tau'-1))| + |p \in \mathbb{L}(P(\tau'-1)) \mid \text{mul}(p) > 1\}|) \leq f+2$.

Recall that we assumed that byzantine robots never move after τ, hence they do not move at $\tau' - 1$. The correct robots located at multiplicity points at $\tau' - 1$ do not move either according lemma 5.1.1 applied to the induction assumption. Hence, only correct single robots can move in $\tau' - 1$.

When a single robot moves, it can either 1) remain single, 2) join an already existing multiplicity point or 3) form a new multiplicity point with one or more other single robots. In the first case, the movement of the robot does not affect either of the two terms of the right side of the inequality which remains true at τ'. For the second case, the number of locatition (represented by the term $|\mathbb{L}(P(\tau'))|$) decreases by one (compared to $|\mathbb{L}(P(\tau'-1))|$) without affecting the number of multiplicity points (as indicated by the term $|p \in \mathbb{L}(P(\tau')) \mid \text{mul}(p) > 1\}|$). Thus, the inequality holds also at τ'. For the last case, the movement of the robot increases the number of multiplicity points by 1 while decreasing the number of positions by at least 1 and the inequality is true for τ'. □

The following lemma and corollary can be proved by using the same arguments as the proofs of Lemma 5.1.1 and Corollary 5.1.1.

Lemma 5.1.2. *Let $P(\tau)$ be a configuration of robots endowed with $\varnothing M$. If $|\mathbb{L}(P(\tau))| \leq f+1$; then all correct robots do not move if activated at τ.*

Corollary 5.1.2. *Let $P(\tau)$ be a configuration of robots endowed with $\varnothing M$. If $|\mathbb{L}(P(\tau))| \leq f+1$ and no Byzantine robot moves at any time $\tau' \geq \tau$; then all correct robots do not move if activated at τ'.*

5.2 Necessity of Strong Multiplicity Detection

In [32], Prencipe studied the problem of gathering in both ATOM and NTOM models and showed that the problem is impossible without endowing robots with weak multiplicity detectors ($?M$) that are able to detect if there is more than one robot

in a given location. Interestingly, when only convergence is requested, no multiplicity detection capability is needed to solve the problem (*e.g.* the algorithm proposed in [17] where no such condition is assumed). In the following we prove that even the weak multiplicity detection capability turns out to be insufficient to achieve convergence when robots are prone to byzantine failures. That is, a stronger form of multiplicity detection is necessary which allows robots to know the exact number of robots that share the same location simultaneously. The result is proved for the weaker ATOM model, and thus directly extends to the NTOM model.

The following lemma shows that no convergence is possible in presence of byzantine robots without multiplicity detection.

Lemma 5.2.1. *It is impossible to reach convergence in byzantine-prone environments in* ATOM *without multiplicity detection, even in the presence of a single byzantine robot.*

Proof. Take any initial configuration $P(\tau_0)$ such that (i) $|\mathbb{L}(P(\tau_0))| \geq f+1$ and (ii) not all correct robots are located in the same multiplicity point. Following Corollary 5.1.2, if the adversary does not instruct any byzantine robot to move after τ_0, then correct robots will never move and convergence is prevented. This proves our lemma. □

The next lemma proves that even endowing robots with weak multiplicity detectors ($?M$) does not allow them to achieve convergence.

Lemma 5.2.2. *It is impossible to reach convergence in byzantine-prone environments in* ATOM *with weak multiplicity detection, even in the presence of a single byzantine robot.*

Proof. Fix an initial configuration $P(\tau_0)$ such that: (i) $|\mathbb{L}(P(\tau_0))| + |\{p \in \mathbb{L}(P(\tau_0)) \mid \text{mul}(p) > 1\}| \leq f+2$ and (ii) there are at least two positions in $\mathbb{L}(P(\tau_0))$ contain at least two correct robots each. Hence, in accordance with Corollary 5.1.1, if no byzantine robot is instructed to move after τ_0, no correct robot located at a multiplicity point will ever move which prevents convergence and proves our lemma. □

In the following chapter, we present three algorithms that rely on strong multiplicity detection and allow robots to achieve convergence in different synchrony assumptions.

5.3 Lower Bound on the Number of Faulty Robots in the ATOM[FS] Model

The following lemma shows that any convergence algorithm needs at least $2f+1$ robots in order to tolerate f byzantine robots even in the strongest model: the fully-synchronous one. This lower bound extends to weaker models with respect to atomicity and synchrony.

Theorem 5.3.1. *In uni-dimensional robot networks, byzantine-resilient convergence is impossible to solve under a fully-synchronous scheduler when $n \leq 2f$.*

Proof. Assume for contradiction that convergence is possible when $n \leq 2f$. Since the convergence of a single correct robot is trivial, we consider only the case when the network contain at least two correct robots, which leads to $n \geq 4$. Now fix a configuration in which correct robots are spread over two distinct positions A and B as follows: $\lceil \frac{n-f}{2} \rceil$ correct robots are located at A and the remaining $\lfloor \frac{n-f}{2} \rfloor$ correct robots are at B. The byzantine robots are also divided between A and B as follows: $\lfloor \frac{f}{2} \rfloor$ byzantine robots at A and $\lceil \frac{f}{2} \rceil$ byzantine robots at B. Since $n \leq 2f$, it follows that $n - f \leq f$. Hence, the total number of robots located at A is equal to

$$\begin{aligned} \lceil \tfrac{n-f}{2} \rceil + \lfloor \tfrac{f}{2} \rfloor &\leq \lceil \tfrac{f}{2} \rceil + \lfloor \tfrac{f}{2} \rfloor \\ &\leq f \end{aligned}$$

Using a similar argument we can show that B contains *at most* f robots also. This implies that both A and B contain *at least* $n - f$ robots. Assume that the scheduler never instruct byzantine robots to move. Thus, according to Lemma 5.1.1, then no correct robot will ever move and convergence is prevented. This proves our lemma.

□

5.4 Lower Bound on the Number of Faulty Robots in the ATOM[B(k)] Model

Theorem 5.4.1. *In uni-dimensional robot networks, byzantine-resilient convergence is impossible to solve under a k-bounded scheduler ($k > 1$) when $n \leq 3f$.*

Proof. Our proof is based on a particular initial setting in which we prove that no convergence algorithm is possible if a third or more of the robots are byzantine when the scheduler is 2-bounded (and hency trivially k-bounded for any $k \geq 2$). We suppose the existence of at least two correct robots since the convergence of a single correct robot is trivial. Thus $f + 2 < n \leq 3f$.

Now assume that the correct robots are spread over two distinct points A and B in a uni-dimensional space. Let P_1 be an initial configuration in which $\lceil \frac{n-f}{2} \rceil$ correct robots are located at A and the remaining $\lfloor \frac{n-f}{2} \rfloor$ correct robots are at B. Note that both A and B contain at least one correct robot each. All the byzantine robots in P_1 are located at A (refer to Figure 5.1(a)). Therefore, the total number of robots at A (whether correct or not) is $\lceil \frac{n-f}{2} \rceil + f$ which is at least equal to $n-f$ since $n \leq 3f$.

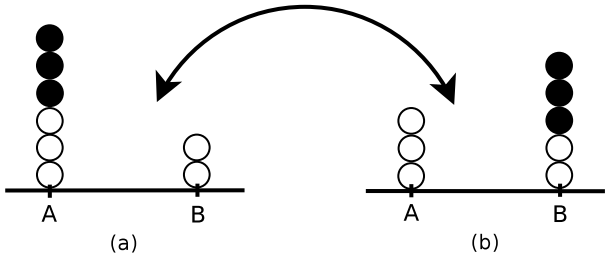


Figure 5.1: Impossibility of convergence under a k-bounded scheduler with $n \leq 3f$, black robots are byzantine. (a) Configuration P_1. (b) Configuration P_2.

Thus, according to Lemma 5.1.1, when the correct robots at A are activated they remain in their location (A) and do not move. Next, the adversary moves the byzantine robots to B which leads to the configuration P_2 (see figure 5.1(b)). Again, the total number of robots at B in P_2 is at least equal to $n-f$. Therefore, the correct robots at B do not move neither upon their activation.

So by repeatedly alternating between the two configurations P_1 and P_2, the adversary ensures that every robot is activated infinitely often in the execution yet prevents convergence at the same time since robots at A and B remain always at their initial positions and never converge.

□

5.5 Lower Bound on the Number of Faulty Robots in the ATOM[AS] model

In this section we prove the fact that, when the number of robots in the network does not exceed $5f$ (with f of those robots possibly being byzantine), the problem of byzantine resilient convergence is impossible to solve under a fully asynchronous scheduler using a **cautious** algorithm. **The result is proved for the weaker ATOM model, and thus extends to the NTOM model.**

In the case when $n \leq 3f$, we proved in the last section the impossibility of convergence for k-bounded k-bounded, so the impossibility trivially extends to asynchronous schedulers. Hence, in the following we assume that $3f < n \leq 5f$.

Let us start with some definitions. Our proof is based on a special kind of configurations, referred in the following as **trivalent**, illustrated in Figure 5.2 and described as follows. A configuration of robots is **trivalent** (Figure 5.2) iff the robots can be divided into three subsets: $SetA(P)$, $SetB(P)$ and $SetX(P)$ (or simply $SetA, SetB$ and $SetB$) located at the three positions, say A, B and X respectively with $A \neq B$ and $X \in [A, B]$ such that:

(i) $SetA$ and $SetB$ contain exactly f *correct* robots each.

(ii) Either (ii.a) $|SetA| = f$ and $||SetB| - |SetX|| \leq 1$ or (ii.b) $|SetB| = f$ and $||SetA| - |SetX|| \leq 1$. We say that the configuration is *right trivalent* in the first case and *left trivalent* in the second case.

When the points A, B, X are distinct, the trivalent configuration is **canonic**, otherwise it is **degenerate** (In this last case either $X = A$ or $X = B$). Unless stated otherwise, all trivalent configurations in this section are canonic.

If $|SetX| = 0$ then the configuration is **bivalent**. Moreover, if $||SetA| - |SetB|| \leq f$, it said **balanced**.

In the remainder of this section we prove that starting from a trivalent configuration, no cautious algorithm is able to achieve byzantine-resilient convergence in uni-dimensional networks under an asynchronous scheduler when $3f < n \leq 5f$. Our proof is by contradiction, we assume the existence of a cautious convergence algorithm $\mathscr{A}$ when the robots are activated by a fully asynchronous scheduler. Then we show that $\mathscr{A}$ satisfies certain properties when applied to trivalent configurations. These properties can be used by the adversary to prevent convergence, contradiction !

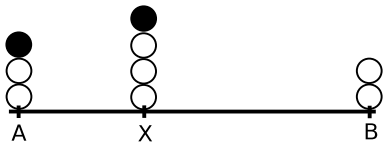


Figure 5.2: Trivalent canonic configuration for ($n = 9, f = 2$)

The following two properties are satisfied by cautious protocols when applied to trivalent configurations:

- **Property 1:** When a robot of $SetA$ (resp. $SetB$) is activated in a right (resp. left) trivalent configuration, its computed destination using a cautious algo-

rithm lies necessarily inside $[A,X]$ (resp. $[X,B]$). This property is proved in Lemma 5.5.1.

- **Property 2:** The adversary is able to move the robots of $SetX$ as close as desired to location A (resp. B). This is proved by Lemmas 5.5.2, 5.5.3 and 5.5.4.

Based on this, the adversary first moves the robots of $SetX$ very close to A (using Property 2) and then activates the robots of $SetA$ that remain in the neighborhood of A (due to Property1). Afterward, it moves the intermediate robots of $SetX$ very close to B (using Property 2) and activates the robots of $SetB$ which also remain in the neighborhood of B (due to Property1). By repeating these actions indefinitely, the adversary ensures that every robot is activated infinitely often in the execution yet prevents convergence at the same time since robots at A and B remain always arbitrarily close to their initial positions and never converge.

In the following, we prove Properties 1 and 2 by a sequence of lemmas, and then give a formal presentation of the algorithm used by the adversary to prevent any cautious protocol from achieving convergence. Let us start with Property 1:

Lemma 5.5.1. *When a robot of $SetA$ (resp. $SetB$) is activated in a right (resp. left) trivalent configuration, its computed destination using a cautious algorithm lies necessarily inside $[A,X]$ (resp. $[X,B]$).*

Proof. We consider only right trivalent configurations, the other case being symmetric.

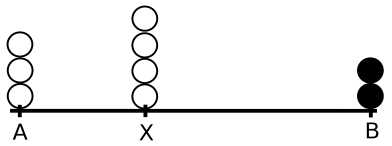


Figure 5.3: Illustration of Lemma 5.5.1, configuration P_2

Fix P_1 a right trivalent configuration and let P_2 such that $P_2 \underset{\varnothing M}{\sim} P_1$, but where the correct and byzantine robots are located differently in P_2 (see Figure 5.3): all robots located at B are byzantine (there are f such robots), and all robots inside $[A,X]$ are correct. Since the robot convergence algorithm is cautious, the diameter of correct robots in P_1 must never increase, hence all their calculated destination points must lie inside $[A,X]$. Since P_2 and P_1 are indistinguishable to individual robots of $SetA$, the Look and Compute phases give the same result in the two cases, which proves our lemma. □

Property 2 says that if the number of robots in the network is lower or equal to $5f$ then it always exists a judicious placement of the byzantine robots that permits the adversary to make the intermediate robots in X move in the direction of B up to a location that is as close as desired to B. To prove this property , we divide the analysis in two cases depending on the parity of $(n-f)$.

Case 1: $(n-f)$ is even

To push the robots of $SetX$ as close to A or B as wanted, the adversary uses algorithm $GoToBorder1$ ($G2B1$). Informally, the algorithm divides byzantine robots between position X and the target border to which the adversary wants to push the robots of $SetX$ (*e.g.* B in what follows). The aim of the adversary is to maintain the same number of robots in X and B (this is possible because $n-f$ is even). We prove that in this case, any cautious convergence algorithm makes the robots of $SetX$ move towards B. However, the distance traveled by them may be too small to bring them sufficiently close to B. Since the scheduler is fully asynchronous, it is authorized to activate the robots of $SetX$ as often as necessary to bring them close to B, as long as it does so for a finite number of times.

Algorithm 1 GoToBorder1 (G2B1)

Input: $Border$: the border towards which robots of $SetX$ move (equal to A or B).
Input: d: a distance.

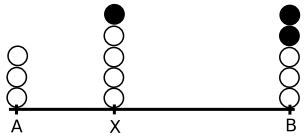
Actions:
Place $\frac{n-3f}{2}$ byzantine robots at $Border$.
Place $\frac{5f-n}{2}$ byzantine robots at X.
while $|X, Border| > d$ **do**
  Activate simultaneously all robots of $SetX$ and make them move to their computed destination D.
end while

Lemma 5.5.2. *Let $3f < n \leq 5f$ with $(n-f)$ odd. If robots run a cautious convergence algorithm then algorithm $G2B1(Border, d)$ terminates for any $d < |A, B|$ and any $Border \in \{A, B\}$.*

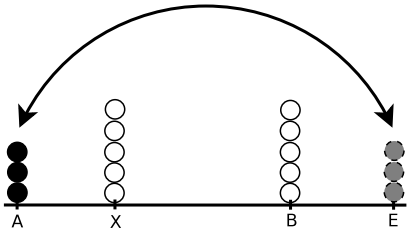
Proof. The placement of byzantine robots in $G2B1$ implies that initially, and for $3f < n \leq 5f$, the number of robots located at X and B is the same and is equal to $(n-f)/2$ as illustrated in Figure 5.4.(a). Note that this initial configuration is trivalent, let us denote it by P_1. Let us denote by P_1 this initial trivalent configuration. Assume *w.l.o.g.* that $Border = B$.

We prove the lemma by contradiction. We assume that the algorithm does not terminate for a given input distance d_0, and we prove that this leads to a contradiction. The non-termination of the algorithm implies that there exists some distance $d_1 \leq d_0$ such that robots at X and B always remain distant by at least d_1 from each other, even if robots at X are activated indefinitely.

Consider an initial configuration P_2 with $P_2 \underset{\varnothing M}{\sim} P_1$ but with a different distribution of byzantine and correct robots (see Figure 5.4.(b)): correct robots are divided equally between X and B: $\frac{n-f}{2}$ correct robots at X and $\frac{n-f}{2}$ others at B. Since the robots are endowed with a cautious convergence algorithm, they are supposed to converge to a single point located between X and B. By using the equivalence between P_1 and P_2 we prove that this never happens, contradiction !



(a) Configuration P_2 for ($n = 13, f = 3$)



(b) Configuration P_2 for ($n = 13, f = 3$)

Figure 5.4: Illustration of lemma 5.5.2 (Fact2, $(n-f)$ even)

The placement of byzantine robots and the choice of activated robots at each cycle is divided into two parts. During even cycles, byzantine robots are placed at point A and robots located at X are activated. During odd cycles, the scheduler constructs a strictly symmetrical configuration by moving byzantine robots from A to a point E with $E > B$ and $|B,E| = |A,X|$. In this case, the scheduler activates robots at B.

In these conditions, activating robots at X ensures that they always remain at a distance of at least d_1 from those located at B (as in configuration P_2). Indeed, configurations P_2 and P_1 are equivalent and completely indistinguishable to individual robots which must behave similarly in both cases (as the algorithm

is deterministic). By symmetry, the activation of robots at B during odd cycles also ensures that minimum distance of d_1 between the two groups of robots. Hence, robots at X and B remain separated by a distance of at least d_1 forever even if activated indefinitely, which prevents the convergence of the algorithm and leads to a contradiction. This proves our Lemma. □

Case 2: $(n-f)$ is odd

To prove Lemma 5.5.2, we relied on the symmetry induced by the placement of byzantine robots. This symmetry is possible only because $(n-f)$ is even. Indeed, having the same number of robots in B and X implies that convergence responsibility is delegated to both robots at X and at B: there is no asymmetry that a convergence algorithm can exploit to get one of these two groups to play a role that would be different from the other group. Robots of $SetX$ and $SetB$ have thus no other choice but to move toward each other when they are activated. The distance traveled at each activation must be large enough to ensure the eventual convergence of the algorithm.

However, the situation is quite different when $(n-f)$ is odd. Indeed, the number of robots is necessarily different in X and B, which means that one of the two points has a greater multiplicity than the other. Then in this case there is no guarantee that a cautious convergence algorithm will order the robots of $SetX$ to move toward B when they are activated (the protocol could delegate the convergence responsibility to robots of $SetB$). Nevertheless, we observe that whatever the cautious algorithm is, if it does not move the robots that are located at the greatest point of multiplicity, it must do so for those at the smallest one (and *vice versa*), otherwise no convergence is ever possible. The convergence is thus either under the responsibility of robots at the larger point of multiplicity or those at the smaller one (or both).

This observation is exploited by Algorithm $GoToBorder2$ ($G2B2$) that is presented as Algorithm 2, that tries the two possible cases to ensure its proper functioning when confronted to any cautious algorithm. The algorithm forms the larger point of multiplicity at B at one cycle, and the next cycle at X. Thus, point X will be the larger point of multiplicity one time, and the smallest one the next time. This implies that the robots of $SetX$ must move towards B at least once every two cycles. So by repeatedly alternating between the two configurations where robots of $SetX$ are successively the set of larger and smaller multiplicity, the adversary ensures that they end up moving towards B. The full asynchrony of the scheduler ensures that they are activated as many times as it takes to move them as close to B as wanted, provided that the algorithm terminates.

Algorithm 2 GoToBorder2 (G2B2)

Input: $Border$: the border towards which the robots of $SetX$ move (equal to A or B).
Input: d: a distance.

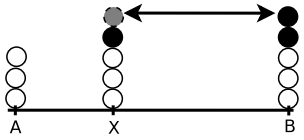
Actions:
Place $\frac{n-3f+1}{2}$ byzantine robots at $Border$.
Place $\frac{5f-n-1}{2}$ byzantine robots at X.
while $|X, Border| > d$ **do**
  Activate simultaneously all robots at X and make them move to their computed destination D.
  $X \leftarrow D$.
  Move a byzantine robot from $Border$ to X.
  Activate simultaneously all robots at X and make them move to their computed destination D.
  $X \leftarrow D$
  Move a byzantine robot from X to $Border$.
end while

Lemma 5.5.3. *Let $3f < n \leq 5f$ with $(n-f)$ odd. If robots run a cautious convergence algorithm then algorithm G2B2($Border, d$) terminates for any $d < |A, B|$ and any $Border \in \{A, B\}$.*

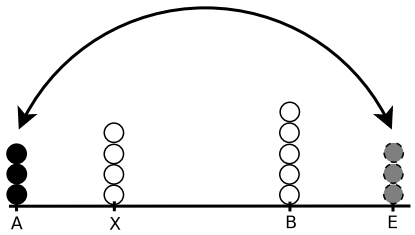
Proof. We consider in our proof only the case when $Border = B$ since the other case is symmetric. The placement of byzantine robots in $G2B2$ is such that the multiplicity of X exceeds that of B by 1 during even cycles, and lowers it by 1 during odd cycles. We denote by P_1 the initial configuration (in which the multiplicity of X is less than of B by 1 as illustrated in Figure 5.5.(a)).

We assume for the purpose of contradiction that $G2B2$ does not terminate for some input distance d_0. This means that robots of $SetX$ and $SetB$ remain always distant from each others by a distance at least equal to d_1 with d_1 being some distance $\leq d_0$. The resulting execution in this case is denoted by $E = \{P_1, P_2, P_3, P_4, ...\}$. A configuration P_{i+1} is obtained from P_i by activating robots at X, letting them execute their Move phases, and moving one byzantine robot from X to B or *vice versa*.

We construct a configuration P'_1 equivalent to P_1 ($P'_1 \underset{\diamondsuit M}{\sim} P_1$) but where correct robots are divided between X and B with $\lfloor (n-f)/2 \rfloor$ robots at X and $\lceil (n-f)/2 \rceil$ robots at B (see Figure 5.5.(b)). By definition, these robots must converge to a point between X and B since they are endowed with a cautious convergence algorithm. Byzantine robots are at A. Since P'_1 and P_1 are equivalent, the activation of robots at X and the displacement of byzantine robots to the right of B will produce a configuration P'_1 that is equivalent to P_1 by symmetry.



(a) Configuration P_1 for ($n = 12, f = 3$)



(b) Configuration P'_1 for ($n = 12, f = 3$)

Figure 5.5: Illustration of lemma 5.5.3 (Fact2, $(n-f)$ odd)

This time, activated robots are those at B. By moving them to their calculated destination points and by moving byzantine robots again to the left of X the scheduler can form a configuration P'_2 which is equivalent to P_2.

This process can be repeated: during odd cycles, byzantine robots are at the left of X and robots at X are activated. During even cycles, the situation is symmetrical: byzantine robots are to the right of B and robots at B are activated. The obtained execution $E' = \{P'_1, P'_2, P'_3, P'_4, ...\}$ is equivalent to E, and robots at X and B remain separated by a distance at least equal to d_1 forever even if they are activated indefinitely. This prevents the convergence of the convergence protocol while ensuring fairness of activations, which contradicts the assumptions and proves our Lemma. □

We are now ready to prove Property 2.

Lemma 5.5.4. *For $n \leq 5f$, $\forall d < |A,B|$, if the robots run a cautious convergence algorithm, the fully distributed scheduler is able to move the robots of $SetX$ into a position $\geq B - d$ or $\leq A + d$.*

Proof. The proof follows directly from Lemmas 5.5.2 and 5.5.3. □

Handling Degenerate Trivalent Configuration

The purpose of Algorithms $G2B1$ and $G2B2$ is to push the intermediate robots of $SetX$ as close as the adversary wants to the extremities of the network. For ease of the description, we assume in the following that the adversary wants to push them towards the extremity B. These two routines are then used by the adversary to prevent the convergence of robots. For the algorithm of the adversary to work, it is necessary to keep the robots of $SetA$, $SetB$ and $SetX$ separated from each other and to avoid for example that the robots of $SetX$ merge with those of $SetB$ and form a single point of multiplicity leading to a *degenerate* trivalent configuration. Yet, functions $G2B1$ and $G2B2$ cannot prevent such a situation to appear because the destinations are computed by the convergence algorithm which can order the robots to move exactly towards B. If the distance to travel is too small ($|X,B| \leq \Delta$), then the adversary can not stop the robots of $SetX$ before they arrive at B. The following lemma states that we can recover from this situation and transform a degenerate trivalent configuration to a canonic one having the same diameter. Its proof make use of Lemma 5.5.6 proved immediately after.

Lemma 5.5.5. *Let P be any degenerate trivalent configuration of n robots with $3f < n \leq 5f$ and $diam(P) > \Delta$, the adversary is able to form a canonic trivalent configuration P' with $diam(P') = diam(P)$ and $SetA(P') = SetA(P), SetB(P') = SetB(P)$ and $SetX(P') = SetX(P)$.*

Proof. Assume *w.l.o.g.* that P is *right* trivalent. The transformation into a canonic trivalent configuration is done by the function $Splint(Set, Border)$ described in Algorithm 3 where the parameters Set and $Border$ take in our case the values $SetX$ and B.

The adversary places the byzantine robots in such a way to obtain a balanced bivalent configuration. Note that since P is trivalent, $SetA$ and $SetB$ contain each exactly f correct robots. Hence, by putting all the f byzantine robots at A, the adversary ensures that this location will contain exactly $2f$ robots. Since $3f < n \leq 5f$, it follows that B contain between $f+1$ and $3f$ robots. Hence, the difference between the number of robots at A and B is at most f. That is, the obtained configuration is *balanced bivalent*.

Consequently, if we activate the robots of $SetX$ (which are located at B), they will necessarily move towards A according to Lemma 5.5.6. Since the initial distance between A and B is greater than Δ, the adversary is able to prevent the robots of $SetX$ from reaching A. By stopping these robots once they all travelled a distance equal to Δ or reached their destination before, we ensure that the three sets $SetA$, $SetX$ and $SetB$ are disjoint. Hence, the reached configuration P' is canonic trivalent with the desired properties.

Algorithm 3 Function Split(Set, Border)

Require: $|A,B| > Max\delta$

Variables:
Input: $Border$: is equal to A or to B.
Input: Set: the set of robots to move away from $Border$.
$OppositeBorder$: is equal to B if $Border = A$, and equal to A if $Border = B$.

Actions:
Place all byzantine robots in $OppositeBorder$.
Activate the robots of Set, and stop them at a point Δ away from $Border$.

□

The following lemma proves that if a robot of a *balanced bivalent* configuration is activated, it cannot stay in its position and moves toward the robots located in the other point.

Lemma 5.5.6. *Let P be a balanced bivalent configuration with $\mathbb{L}(P) = \{A,B\}$. The destination computed by any robot located at A (resp. B) using any cautious convergence algorithm lies necessarily inside $(A,B]$ (resp. $[A,B)$).*

Proof. Denote by a and b the number of robots located at A and B respectively. Since the configuration is balanced it holds that $|a-b| \leq f$. Observe that since we assume that the algorithm is cautious, all the computed destinations are necessarily located inside $[A,B]$. Hence, to prove the lemma it suffices to show that the destination of a robot located at A (resp. B) is distinct from A (resp. B). In the following we consider only the robots located at A since the other case is symmetric. Hence, we have to show that robots located at A move towards B upon their activation. Assume to the contrary that their destination is A and let us separate the analysis into three cases depending on the relationship between a and b. We show that each case leads to a contradiction :

1. $a > b$: Let P_2 be a configuration equivalent to P_1 ($P_2 \underset{\Diamond M}{\sim} P_1$) with the following placement of robots: At A there are f byzantine robots and $a-f$ correct ones, and at B are located b correct robots. Since Configurations P_1 and P_2 are indistinguishable to individual robots, the destinations computed in the two cases are the same. So when the robots at A are activated, they do not move. The next cycle, the adversary moves $(a-b)$ byzantine robots from A to B to obtain a configuration P_3 with $P_3 \underset{\Diamond M}{\sim} P_2$. This time, the adversary activates the robots at B which do not move neither since robots cannot

distinguish between P_2 and P_3 and must behave similarly in the two configurations as the algorithm is deterministic. Then, the adversary brings back the $(a-b)$ byzantine robots to A to get again the configuration P_2 and then activates the robots at A. The process repeats, and by placing these $(a-b)$ byzantine robots in one cycle at A and the next cycle at B, the adversary prevents the convergence of the algorithm. This is a contradiction.

2. $a < b$: The argument is symmetric to Case 1.

3. $a = b$: In this case the configuration is perfectly symmetric. Hence, if the activated robots at A do not move upon their activation, it is also the case for robots located at B. Consequently, the robots never converge towards a single location, contradiction !

□

The Lower Bound

Theorem 5.5.1. *Starting from a trivalent configuration, no cautious algorithm is able to achieve byzantine-resilient convergence in uni-dimensional networks under an asynchronous scheduler when* $3f < n \leq 5f$.

Proof. The algorithm of the adversary for the case when $3f < n \leq 5f$ is given as Algorithm 4 and it can prevent any cautious algorithm to converge.

Algorithm 4 Adversary Algorithm

Require: $|A,B| > Max\delta$

Definitions:
d_0: any distance that is strictly smaller than $|A,B|/4$, let $d_0 \leftarrow |A,B|/10$.
$G2B(Border,d) \equiv G2B1(Border,d)$ if $(n-f)$ is even, $\equiv G2B2(Border,d)$ if $(n-f)$ is odd.

Actions:
while true **do**
  $G2B(A,d_0)$.
  Activate the robots at A.
  if the robots of SetX are at A, then $Split(SetX,A)$.
  $G2B(B,d_0)$.
  Activate the robots at B.
  if the robots of $SetX$ are at B, then $Split(SetX,B)$.
  $d_0 \leftarrow d_0/2$
end while

Indeed, if for example the initial distance between robots at A and B is equal to $d = 10 \cdot d_0$, then these robots will always remain distant from each other by a distance at least equal to:

$$\begin{aligned} d - 2\sum_{k\geq 0}(d_0/2^k) &= 10d_0 - 2 * 2d_0 \\ &= 6d_0 \end{aligned}$$

The proof of algorithm 4 follows directly from Lemmas 5.5.1, 5.5.4 and 5.5.6. □

Theorem 5.5.2. *In uni-dimensional robot networks, byzantine-resilient convergence is impossible to solve under a fully asynchronous scheduler when $n \leq 5f$.*

Proof. We proved in Theorem 5.4.1 that convergence of robots is impossible in unidimensional networks in the presence of byzantine failures when $n \leq 3f$. Moreover, Theorem 5.5.1 says that no cautious algorithm is impossible when $3f < n \leq 5f$. Hence the theorem follows. □

5.6 Necessary and Sufficient Conditions for Deterministic Convergence

In this section we present the theoretical platform that we use to prove the correctness of our algorithms given in the following sections. We address the necessary and sufficient conditions to achieve convergence of robots in systems prone to byzantine failures. We define *shrinking* algorithms (algorithms that eventually decrease the range of correct robots) and prove that this condition is necessary for convergence even in *fault-free* environments. But shrinkingness alone is not sufficient for convergence. Thus, we define *cautious* algorithms (algorithms that ensure that the position of correct robots always remains inside the range of the correct robots) and show that this condition, combined with the previous one, is sufficient to reach convergence even in *faulty* systems.

5.6.1 Shrinking algorithms

By definition, convergence aims at asymptotically decreasing the range of possible positions for the correct robots. The shrinking property captures this property. An algorithm is shrinking if there exists a constant factor $\alpha \in (0,1)$ such that starting in any configuration the range of correct robots eventually decreases by a multiplicative α factor. Note that to deal with the asynchrony of the model, the diameter calculation takes into account both the positions and destinations of correct robots.

Definition 16 (Shrinking Algorithm)**.** *An algorithm is* shrinking *if and only if* $\exists \alpha \in (0,1)$ *such that* $\forall \tau, \exists \tau' > \tau$*, such that* $diam(U(tau') \cup D(\tau')) < \alpha * diam(U(\tau) \cup D(\tau))$*, where* $U(\tau)$ *and* $D(\tau)$ *are respectively the the multisets of positions and destinations of correct robots.*

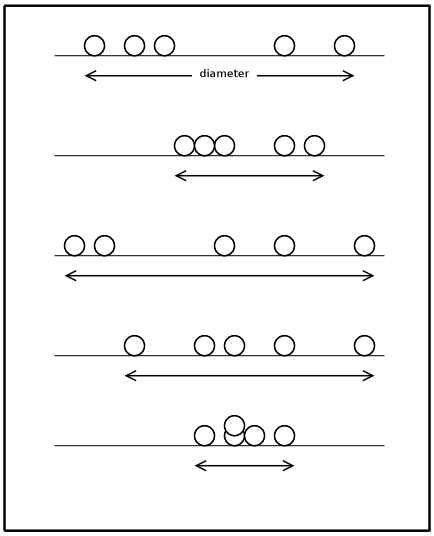


Figure 5.6: Oscillatory effect of a shrinking algorithm

Note 5.6.1. *In the atomic model, a robot position at a given time equal its computed destination in the previous cycle. Hence, it suffices for an algorithm to be shrinking in this model that at* $\tau' : diam(U(\tau')) < \alpha * diam(U(\tau))$.

Note 5.6.2. *Note that the definition does not imply that the diameter always remains smaller than* $\alpha * diam(U(\tau) \cup D(\tau))$ *after* τ' *(see Figure 5.6). Therefore, an oscillatory effect is possible: the algorithm alternates between periods where the diameter is increased and decreased. However, each increasing period is followed by a decreasing one as depicted in Figure 5.6. It follows that a shrinking algorithm is not necessarily convergent.*

Lemma 5.6.1. *Any algorithm that solves the byzantine convergence problem is necessarily shrinking.*

Proof. Assume w.l.o.g that the model is atomic. That is, for every time τ $U(\tau) = D(\tau)$.

Assume for contradiction the existence of a byzantine convergence algorithm $\mathscr{A}$ that is not shrinking. This implies the existence of constant factor $\alpha \in (0,1)$, and some time instant τ after which the diameter of correct robots running $\mathscr{A}$ is always greater than $\alpha * \text{diam}(U(\tau))$. Hence, for each point p in the uni-dimensional plane, there exists infinitely many times $\tau' \geq \tau$ at which the distance between p and some correct robot in the network is greater than $\frac{\alpha * \text{diam}(U(\tau))}{2}$. Therefore, p cannot be the convergence point of robots. Since p is arbitrary, robots never convergence using $\mathscr{A}$, contradiction ! □

5.6.2 Cautious algorithms

The following two lemmas state some properties of cautious algorithms.

Lemma 5.6.2. *In the* ATOM *model, if an algorithm is cautious then*

$$\forall\, t' > t \;\; diam(U(\tau')) \leq diam(U(\tau))$$

Proof. Assume that it is not the case *i.e.* that $\text{diam}(U(\tau')) > \text{diam}((U(\tau))$ for some $\tau' > \tau$. Then there exists two **successive** time instants (or cycles) τ_1 and τ_2 with $\tau \leq \tau_1 < \tau_2 \leq \tau'$ such that the diameter of correct robots at τ_2 is strictly greater than the diameter at τ_1 i.e. $\text{diam}(U(\tau_2)) > \text{diam}(U(\tau_1))$. Thus, there exists at least one correct robot, say r_1, that was inside $\text{diam}(U(\tau_1))$ at τ_1, and moved outside it at τ_2. We prove that this is impossible.

Since cycles are atomic in ATOM, no robot moves between τ_1 and the LOOK phase of τ_2, and the resulting snapshot of correct robots at this LOOK phase is equal to $U(\tau_1)$. Thus, the destination point calculated by r_1 at τ_2 is necessarily inside $\text{range}(U(\tau_1))$ since the algorithm is cautious. This contradicts the assumption that r_1 moves outside $\text{range}(U(\tau_1))$ at τ_2, and the lemma follows. □

Note 5.6.3. *The preceding lemma does not hold for the* NTOM *model. Due to asynchrony, the following type of scenario may occur. Assume three correct robots r_1, r_2 and r_3. Assume without loss of generality that r_1 and r_2 are collocated at the same position initially. r_1 calculates some point, say p_g, between it and r_3 as its destination point and starts moving towards it. Assume that r_1 is slow and before it reaches p_g, r_3 has been activated several times until it becomes closer to r_2 than p_g. At this moment, say τ_1, the diameter is less than $|r_2, p_g|$.Then when r_1 completes its Move phase and reaches its destination p_g, the diameter is equal to $|r_2, r_1| = |r_2, p_g|$ that is greater than the diameter at τ_1.*

So in the following lemma, we consider the range of correct robots and their destinations and we prove that this range never decreases in the NTOM model if the algorithm is cautious.

Lemma 5.6.3. *In the* NTOM *model, if an algorithm is cautious then* $\forall \tau' > \tau, diam(U(\tau') \cup D(\tau')) \leq diam(U(\tau) \cup D(\tau))$.

Proof. Fix $\tau \in \mathbb{T}$ and assume for contradiction that $\exists \tau' > \tau$ such that $\text{diam}(U(\tau') \cup D(\tau')) > \text{diam}(U(\tau) \cup D(\tau))$. This assumption implies that there exists at least one correct robot, say r_i, whose position and destination were both inside $range(U(t) \cup D(t))$ at t, and whose position or destination is outside the range at τ'. Formally, there exists some $\tau' > t$ such that $U_i(\tau') \notin \text{range}(U(\tau) \cup D(\tau))$ or $D_i(\tau') \notin \text{range}(U(\tau) \cup D(\tau))$. Assume without loss of generality that r_i is the only robot in this case between τ and τ', and distinguish the following two cases:

1. The destination point of r_i is outside $range(U(t) \cup D(t))$ at τ', that is $D_i(\tau') \notin \text{range}(U(\tau) \cup D(\tau))$. But since no other robot was outside the range after τ, and since $D_i(\tau')$ was calculated after τ using a cautious algorithm, $D_i(\tau')$ is necessarily inside the range, which leads to a contradiction.

2. The position of r_i is outside $\text{range}(U(\tau) \cup D(\tau))$ at τ', that is $U_i(\tau') \notin \text{range}(U(\tau) \cup D(\tau))$. But since the precedent position of r_i and its destination were both inside $\text{range}(U(\tau) \cup D(\tau))$, r_i can only move between these two points and stays necessarily inside $\text{range}(U(\tau) \cup D(\tau))$, a contradiction.

Both cases lead to a contradiction which proves our lemma. □

Theorem 5.6.1. *Any algorithm that is both cautious and shrinking solves the convergence problem in faulty robot networks.*

Proof. We construct our proof for the most general model (the NTOM model). Since the algorithm is **shrinking** then: $\exists \alpha \in (0,1)$ such that $\forall \tau \in \mathbb{T}$, there exists some time $\tau' > \tau$, such that $diamU(\tau') \cup D(\tau') < \alpha * \text{diam}(U(\tau) \cup D(\tau))$. And since it is **cautious**, we have by Lemma 5.6.3 that $\forall \tau' > \tau : \text{diam}(U(\tau') \cup D(\tau')) \leq \text{diam}(U(\tau) \cup D(\tau))$. So $\exists \alpha \in (0,1)$ such that $\forall \tau$, there exists some time $\tau' > \tau$, such that $\forall \tau'' > \tau' : \text{diam}(U(\tau'') \cup D(\tau'')) < \alpha * \text{diam}(U(\tau) \cup D(\tau))$.

So given any initial configuration, by repeatedly decreasing the diameter of correct robots by a factor of α, we can make them as close as we like. Formally speaking, if we refer to the initial diameter of robots by σ_0 (*i.e.* $\sigma_0 = \text{diam}(U(\tau_0) \cup D(\tau_0))$), then for every $\epsilon > 0$, there exists $k > 0$ such that $\frac{\sigma_0}{\alpha^k} < \epsilon$. So for every $\epsilon > 0$, there exists a time τ_ϵ such that $\forall \tau \geq \tau_\epsilon : \text{diam}(U(\tau) \cup D(\tau)) < \epsilon$. Define $c(\tau)$ to be the point that is in the middle of $\text{range}(U(\tau) \cup D(\tau))$ at time τ and let c be the limit of $c(\tau)$ as τ gets to infinity. Clearly, by cautiousness we have that $\forall \tau \in \mathbb{T} : c \in \text{range}(U(\tau) \cup D(\tau))$. Hence, $\forall \epsilon > 0$, there is a time τ_ϵ such that $\forall \tau \geq \tau_\epsilon$, $\forall i \leq m, distance(U_i(\tau), c) < \epsilon$. Consequently, the algorithm is convergent. □

5.7 Deterministic Convergence in ATOM[FS] Networks

In this section we propose a deterministic convergence algorithm and prove its correctness and optimality in the fully-synchronous atomic model when robots are endowed with strong multiplicity detectors $\diamondsuit M$. Our algorithm matches the upper bound on the number of faulty robots proved in Theorem 5.3.1 (Chapter 5, Section 5.3) and requires that $n > 2f$ to be correct. Algorithm 5, similarly to the approximate agreement algorithm in [24], uses two functions: $\text{trim}_f(P(\tau))$ and $mid(P(\tau))$. The former removes the f largest and f smallest values from the multiset given in parameter. The latter returns the point that is in the middle of the input range. Using Algorithm 5, each robot computes the middle point of the positions of the robots seen in its last Look phase ignoring the f largest and f smallest positions.

Algorithm 5 Byzantine Tolerant Convergence in ATOM[FS].

Functions:
$\text{trim}_f()$: removes the f largest and f smallest values from the multiset given in parameter.
$mid()$: returns the point in the middle of the range of points given in parameter.

Actions:
move towards $mid(\text{trim}_f(P(\tau)))$

In the following we prove the correctness of Algorithm 5 in fully-synchronous ATOM model. In order to show that Algorithm 5 is convergent we prove that it is both cautious and shrinking.

5.7.1 Algorithm 5 is Cautious

First we propose a set of lemmas that will be further used in the construction of the cautiousness proof of our algorithm. In the following we recall a result related to the functions $\text{trim}_f()$ and $range$ proved in [24].

Lemma 5.7.1. *[24] For $n > 2f$: $range(trim_f(P(\tau))) \subseteq range(U(\tau))$?*

A direct consequence of the above property is that Algorithm 5 is cautious for $n > 2f$.

Lemma 5.7.2. *Algorithm 5 is cautious for $n > 2f$ in the unidimensional* ATOM[FS] *model.*

Proof. The cautiousness property follows from Lemma 5.7.1: range($\text{trim}_f(P(\tau))$) $\subseteq$ range($U(\tau)$) implies that $mid(\text{trim}_f(P(\tau))) \in \text{range}(U(\tau))$. □

5.7.2 Algorithm 5 is Shrinking

The following lemma addresses the shrinkingness property of Algorithm 5 in the fully-synchronous ATOM model.

Lemma 5.7.3. *Algorithm 5 is shrinking for $n > 2f$ in the unidimensional* ATOM[FS] *model.*

Proof. Denote by d_0 the diameter of the initial configuration. At each cycle, all robots move towards the same destination by a distance of at least Δ unless they reach their destination.

If all robots are at a distance smaller than Δ from the common destination point, gathering is achieved and the diameter is null. Otherwise, the robots that are further than Δ from the destination point approach it by at least Δ so the diameter decreases by at least Δ. Overall, the diameter of robots decreases by at least factor of $\alpha = 1 - (\Delta/d_0)$ at each cycle and thus the algorithm is shrinking. □

The correctness of Algorithm 5 follows directly from Lemmas 5.7.2 and 5.7.3 and Theorem 5.6.1:

Theorem 5.7.1. *Algorithm 5 solves byzantine-resilient convergence for $n > 2f$ in fully-synchronous uni-dimensional ATOM networks.*

Actually, Algorithm 5 can do better than convergence. As explained in the proof of Lemma 5.7.3, the range of correct robots decreases at each cycle by a constant *additive* factor (Δ) until it reaches 0. It follows that Algorithm 5 achieves gathering.

Theorem 5.7.2. *Algorithm 5 solves byzantine-resilient* gathering *for $n > 2f$ in fully-synchronous uni-dimensional ATOM networks.*

5.8 Deterministic Convergence in NTOM[B(k)] Networks

In this section we propose a deterministic convergence algorithm and prove its correctness in the NTOM[B(k)] model when the robots are endowed with strong multiplicity detectors $\diamondsuit M$. This algorithm works correctly for $n > 3f$ thus matching the upper bound on the number of faulty robots under k-bounded schedulers that was proved in Section 5.4 of the precedent chapter. The idea of Algorithm 6 is as follows: each robot computes the center of the positions of the robots seen in

its last Look phase ignoring the f largest positions if they are larger than his own position and the f smallest positions if they are smaller than his own position.

Algorithm 6 uses two functions, $\text{trim}_f^i()$ and $mid()$. The choice of the function $\text{trim}_f^i()$ makes the difference between this algorithm and Algorithm 5. Indeed, in Algorithm 5 the trimming function removes the f largest and the f smallest values from the multiset given in parameter. That is, the returned multiset does not depend on the position of the calling robot. In Algorithm 6, $\text{trim}_f^i()$ removes among the f largest positions *only* those that are greater than the position of the calling robot r_i. Similarly, it removes among the f smallest positions only those that are smaller than the position of the calling robot.

Formally, let $minindex_i$ be the index of the minimum position between $P_i(\tau)$ and $P_{f+1}(\tau)$ (if $P_i(\tau) < P_{f+1}(\tau)$ then $minindex_i$ is equal to i, otherwise it is equal to $f+1$). Similarly, let $maxindex_i$ be the index of the maximum position between $P_i(\tau)$ and $P_{n-f}(\tau)$ (if $P_i(\tau) > P_{n-f}(\tau)$ then $maxindex_i$ is equal to i, otherwise it is equal to $n-f$). $\text{trim}_f^i(P(\tau))$ is the multiset consisting of positions $\{P_{minindex_i}(\tau), P_{minindex_i+1}(\tau), ..., P_{maxindex_i}(\tau)\}$. As in Algorithm 5, $mid()$ returns the center point of the input range. The two functions are illustrated in Figure 5.7)
.

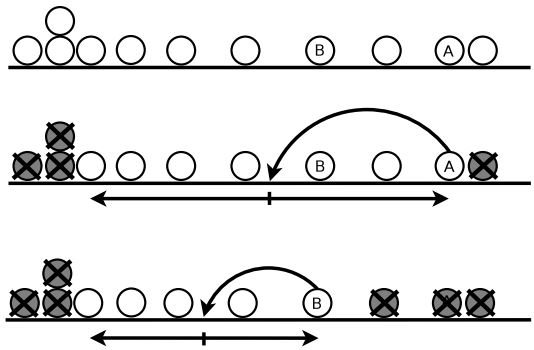


Figure 5.7: Illustration of functions $\text{trim}_f^i()$ and $mid()$ for robots A and B in a system of $n = 11$ robots with $f = 3$.

In the following we prove the correctness of Algorithm 6 in the NTOM[B(k)] model. In order to show that Algorithm 6 converges, we prove first that it is cautious then we prove that it satisfies the specification of a shrinking algorithm. Convergence then follows from Theorem 5.6.1.

Algorithm 6 Byzantine Tolerant Convergence in NTOM[B(k)].

Functions:
- $\text{trim}_f^i(P(\tau))$: removes up to f largest positions that are larger than $P_i(\tau)$ and up to f smallest positions that are smaller than $P_i(\tau)$ from the multiset $P(\tau)$ given in parameter.
- $mid()$: returns the center point of the input range.

Actions:
move towards $mid(\text{trim}_f^i(P(\tau)))$

5.8.1 Algorithm 6 is Cautious

In this section we prove that Algorithm 6 is cautious when for $n > 3f$. The following lemma states that the range of the trimmed multiset $\text{trim}_f^i(P(\tau))$ is contained in the range of correct positions.

Lemma 5.8.1. *Let r_i be a correct robot executing Algorithm 6, it holds for $n > 3f$ that*

$$range(trim_f^i(P(\tau))) \subseteq range(U(\tau))$$

Proof. We prove that for any correct robot r_i, the following conditions hold:

1. $\forall \tau \in \mathbb{T} : min(\text{trim}_f^i(P(\tau))) \in \text{range}(U(\tau))$.
2. $\forall \tau \in \mathbb{T} : max(\text{trim}_f^i(P(\tau))) \in \text{range}(U(\tau))$.

1. By definition, $min(\text{trim}_f^i(P(\tau))) = min\{P_i(\tau), \ldots, P_{f+1}(\tau)\}$. Hence proving Property (1) reduces to proving $P_i(\tau) \in \text{range}(U(\tau))$ and $P_{f+1}(\tau) \in rangeU(\tau)$. Similarly, proving property (2) reduces to proving $P_i(\tau) \in \text{range}(U(\tau))$ and $P_{n-f}(\tau) \in \text{range}(U(\tau))$.

   - $P_i(\tau) \in \text{range}(U(\tau))$ directly follows from the assumption that robot r_i is correct.
   - $P_{f+1}(\tau) \in \text{range}(U(\tau))$. Suppose the contrary: there exists some time instant τ such that $P_{f+1}(\tau) \notin \text{range}(U(\tau))$ and prove that this leads to a contradiction. If $P_{f+1}(\tau) \notin \text{range}(U(\tau))$ then either $P_{f+1}(\tau) < U_1(\tau)$ or $P_{f+1}(\tau) > U_m(\tau)$.
     - If $P_{f+1}(\tau) < U_1(\tau)$ then there are at least $f+1$ positions $P_1(\tau)$, $P_2(\tau)$, ..., $P_f(\tau)$, $P_{f+1}(\tau)$ that are smaller than $U_1(\tau)$ which is the *first* correct position in the network at time τ. This means that

there would be at least $f+1$ Byzantine robots in the system. But this contradicts the assumptions that at most f byzantine robots are present in the system.

- If $P_{f+1}(\tau) > U_m(\tau)$ then since $n > 3f$ there are more than f positions $P_f(\tau)$, $P_{f+1}(\tau)$, ..., $P_n(\tau)$ that are greater than $U_m(\tau)$, which is the *last* correct position in the system at time τ. This also leads to a contradiction.

2. The property is symmetric to the precedent one and can by proved using the same argument.

□

A direct consequence of the above property is that correct robots always compute a destination within the range of positions held by correct robots, whatever the behavior of byzantine ones. Thus, the diameter of positions held by correct robots never increases. Consequently, the algorithm is cautious. The formal proof is proposed in the following lemma.

Lemma 5.8.2. *Algorithm 6 is cautious for $n > 3f$.*

Proof. According to Lemma 5.8.1, $\text{range}(\text{trim}_f^i(P(\tau))) \subseteq \text{range}(U(\tau))$ for each correct robot r_i, thus $mid(\text{trim}_f^i(P(\tau))) \in \text{range}(U(\tau))$. It follows that all destinations computed by correct robots are located inside $\text{range}(U(\tau))$ which proves the lemma. □

5.8.2 Algorithm 6 is Shrinking

In this section we prove that Algorithm 6 is shrinking. The following lemma states that a robot can not compute a destination that is far from its current position by more than half the diameter of correct positions. More specifically, a robot located on one end of the network can not move to the other end in a single movement.

Interestingly, the property of lemma 5.8.3 is guaranteed even though robots are not able to figure out the range of correct positions nor to compute their diameter. The bound on the movements of robots is achieved by taking into account the position of the calling robot when computing the trimming function.

Lemma 5.8.3. *For every correct robot r_i, for every time $\tau \in \mathbb{T}$, if r_i computes its destination point at time τ, then $|U_i^i(\tau), D_i(\tau)| \leq \frac{diam(U^i(\tau))}{2}$.*

Proof. Suppose for contradiction that $|U_i^i(\tau), D_i(\tau)| > \frac{\text{diam}(U^i(\tau))}{2}$ for some robot r_i at time τ. Assume without loss of generality that $U_i^i(\tau) < D_i(\tau)$ (the other case is symmetric). This means that $U_i^i(\tau) > D_i(\tau) - \text{diam}(U^i(\tau))/2$. We prove that this is impossible.

Let p denotes the point $U_i^i(\tau)$. Hence $p \in \text{range}(\text{trim}_f^i(P(\tau)))$ with $p < D_i(\tau)$ and $|p, D_i(\tau)| > \text{diam}(U^i(\tau))/2$, Since $D_i(\tau)$ is the center of $\text{trim}_f^i(P(\tau))$ it follows that there must exist another point $q \in \text{trim}_f^i(P(\tau))$ with $q > D_i(\tau)$ such that $|D_i(\tau), q| > \text{diam}(U^i(\tau))/2$.

Hence,

$$\begin{aligned} |p,q| &= |p, D_i(\tau)| + |D_i(\tau), q| \\ &> \text{diam}(U^i(\tau))/2 + \text{diam}(U^i(\tau))/2 \\ &> \text{diam}(U^i(\tau)) \end{aligned}$$

Since both p and q belong to $\text{trim}_f^i(P(\tau))$, it follows that $\text{diam}(\text{trim}_f^i(P(\tau))) \geq |p,q| > \text{diam}(U^i(\tau))$. This contradicts lemma 5.8.1 which states that $\text{range}(\text{trim}_f^i(P(\tau))) \subseteq \text{range}(U(\tau))$. □

The following lemmas describe some important properties on the destination points computed by correct robots which will be used in proving the shrinkingness of Algorithm 6. These properties are satisfied whatever the positions of byzantine robots are, and thus they capture the limits of the influence of byzantine robots on the actions undertaken by correct robots.

The next lemma shows that the correct positions $\{U_{f+1}(\tau), ..., U_{m-f}(\tau)\}$ are always included in the trimmed range (the output range of the function $\text{trim}_f^i()$) regardless of the positions of byzantine robots.

Lemma 5.8.4. *It holds that* $range(trim_f^i(U(\tau))) \subseteq range(trim_f^i(P(\tau)))$.

Proof. We prove that:

1. $\forall t\ U_{f+1}(\tau) \in \text{range}(\text{trim}_f^i(P(\tau)))$.
2. $\forall t\ U_{m-f}(\tau) \in \text{range}(\text{trim}_f^i(P(\tau)))$.

1. Suppose that $U_{f+1}(\tau) \notin \text{range}(\text{trim}_f^i(P(\tau)))$. Then either

$$U_{f+1}(\tau) < \min(\text{trim}_f^i(P(\tau)))$$

or

$$U_{f+1}(\tau) > \max(\text{trim}_f^i(P(\tau)))$$

- If $U_{f+1}(\tau) < \min(trim_f(P(\tau)))$ then there are at least $f+1$ positions $\{U_1(\tau), ..., U_{f+1}(\tau)\}$ which are smaller than $\min(trim_f(P(\tau)))$. This contradicts the definition of $trim_f(P(\tau))$ (at most f among the smallest elements of $P(\tau)$ are removed).
- If $U_{f+1}(\tau) > \max(trim_f(P(\tau)))$ and since $|U(\tau)| > 2f$ (because $n > 3f$), then there are at least $f+1$ positions in $U(\tau)$ ($\{U_{f+1}(\tau), ..., U_{2f+1}(\tau)\}$) that are greater than $\max(trim_f(P(\tau)))$, which also leads to a contradiction.

2. The property is symmetric to the precedent one.

□

Let $D(\tau)$ be the set of destinations computed with Algorithm 6 in systems with $n > 3f$, and let $UD(\tau)$ be the union of $U(\tau)$ and $D(\tau)$. If a robot r_i executed its last Look phase at time $\tau' \leq \tau$, then $UD^i(\tau) = UD(\tau')$. The following lemma proves that the destination computed by each correct robot r_i is always within the range $[(min(UD^i(\tau)) + U^i_{m-f}(\tau))/2, (U^i_{f+1}(\tau) + max(UD^i(\tau)))/2]$ independently of the positions of byzantine robots.

Lemma 5.8.5. *The following properties hold:*

$\forall i$, *each destination point calculated by a correct robot* r_i *at time* τ *is (1) smaller than* $(U^i_{f+1}(\tau) + max(UD^i(\tau)))/2$ *and (2) greater than* $(min(UD^i(\tau)) + U^i_{m-f}(\tau))/2$.

Proof. Let d_1 be the distance between $U^i_{f+1}(\tau)$ and $max(UD^i(\tau))$.

1. We suppose the contrary: there exists some calculated destination point D_i by some correct robot r_i at time τ such that

$$D_i > (U^i_{f+1}(\tau) + max(UD^i(\tau)))/2$$

and we prove that this leads to a contradiction.

$$D_i > (U^i_{f+1}(\tau) + max(UD^i(\tau)))/2$$

implies that $U^i_{f+1}(\tau) < D_i - d_1/2$. And by Lemma 5.8.4, $U^i_{f+1}(\tau)$ is inside

$$\text{range}(\text{trim}^i_f(P^i(\tau)))$$

which means that there is a position inside $\text{range}(\text{trim}^i_f(P^i(\tau)))$ which is smaller than $D_i - d_1/2$. Hence there must exists a position inside

range($\text{trim}^i_f(P^i(\tau))$) say p, such that $p > D_i + d_1/2$ because D_i is the center of $\text{trim}^i_f(P^i(\tau))$. $U^i_{f+1}(\tau) < D_i - d_1/2$ and $p > D_i + d_1/2$ implies that $|U_{f+1}(\tau), p| > |U_{f+1}(\tau), max(UD^i(\tau)|$ which in turn implies that $p > max(UD^i(\tau))$. But $p \in \text{range}(\text{trim}^i_f(P^i(\tau)))$, it follows that

$$max(\text{trim}^i_f(P^i(\tau))) > max(UD^i(\tau))$$

which contradicts Lemma 5.8.1 and thereby proves our lemma.

2. Symmetric to the precedent property.

□

Lemma 5.8.6. *Let $S(\tau)$ be a multiset of $f+1$ arbitrary elements of $U(\tau)$. The following properties hold: (1) $\forall t$, $U_{f+1}(\tau) \leq \max(S(\tau))$ and (2) $\forall t$, $U_{m-f}(\tau) \geq \min(S(\tau))$*

Proof. 1. Assume the contrary: $U_{f+1}(\tau) > \max(S(\tau))$. This means that $U_{f+1}(\tau)$ is strictly greater than at least $f+1$ elements of $U(\tau)$, which leads to a contradiction ($U_{f+1}(\tau)$ is by definition the $(f+1)$-th correct position in $U(\tau)$).

2. The property is symmetric to the precedent.

□

The next lemma generalizes and extends the properties of Lemmas 5.8.4 and 5.8.5 (proven for a fixed time instant) to a time interval. It describes bounds on the destination points computed by correct robots during a time interval $[\tau_1, \tau_2]$. It states that if there is a subset of $f+1$ robots whose positions are less than S_{max} during $[\tau_1, \tau_2]$, then all destinations computed during $[\tau_1, \tau_2]$ by all correct robots in the network are necessarily smaller than $(S_{max} + Max(UD(\tau_1)))/2$.

Lemma 5.8.7. *Let a time $\tau_2 > \tau_1$ and let $S(\tau)$ be a multiset of $f+1$ arbitrary elements in $U(\tau)$. If $\forall p \in S(\tau)$ and $\forall \tau \in [\tau_1, \tau_2]$ $p \leq S_{max}$ then all calculated destination points at time interval $[\tau_1, \tau_2]$ are smaller than $(S_{max} + Max(UD(\tau_1)))/2$.*

Proof. By definition of S_{max} we have that $\forall \tau \in [\tau_1, \tau_2], max(S(\tau)) \leq S_{max}$. According to Lemma 5.8.6, $\forall \tau \in [\tau_1, \tau_2]$ $U_{f+1}(\tau) \leq max(S(\tau))$. So $\forall \tau \in [\tau_1, \tau_2]$, $U_{f+1}(\tau) \leq S_{max}$.

By Lemma 5.8.5, each calculated destination point by each correct robot r_i at time interval $[\tau_1, \tau_2]$ is smaller than $(U^i_{f+1}(\tau) + max(UD(\tau)))/2$, so because $U_{f+1}(\tau) \leq S_{max}$ these destinations points are also smaller than $(S_{max} + max(UD(\tau)))/2$. Since the algorithm is cautious, $\forall i, \forall \tau \in [\tau_1, \tau_2]$ $max(UD(\tau)) \leq max(UD(\tau_1))$ and the lemma follows. □

The next Lemma states that if some calculated destination point is in the neighborhood of one end of the network, then a majority of $m-f$ correct robots are necessarily located in the neighborhood of this end.

Lemma 5.8.8. *If some correct robot r_i executes its Look phase at time τ and then compute (in the Compute phase which immediatly follows) a destination D_i such that $D_i < min(UD(\tau))+b$ (with b any distance smaller than $diam(UD(\tau))/2$), then at t, there are at least $m-f$ correct robots whose positions are (strictly) smaller than $min(UD(\tau))+2b$.*

Proof. We prove first that at τ, $max(trimiP(\tau)) \leq min(UD(\tau))+2b$. According to Lemma 5.8.1, $min(\text{trim}^i_f(P(\tau))) \geq min(UD(\tau))$. And we have by hypothesis that $D_i < min(UD(\tau))+b$. This gives us $D_i < min(trimiP(\tau))+b$. But D_i is the center of $\text{trim}^i_f(P(\tau))$ which means that $|D_i, min(\text{trim}^i_f(P(\tau)))|$ must be equal to $|D_i, max(\text{trim}^i_f(P(\tau)))|$. Thus, $max(\text{trim}^i_f(P(\tau))) < D_i + b$. And since by hypothesis $D_i < min(UD(\tau))+b$, we have

$$max(\text{trim}^i_f(P(\tau))) < min(UD(\tau))+2b$$

which means that at τ there are at most f correct positions greater than $min(UD(\tau))+2b$, and by definition no correct position is smaller than $min(UD(\tau))$. It follows that at τ, the range $[min(UD(\tau)), min(UD(\tau))+2b)$ contains at least $m-f$ correct positions. □

We are now ready to give the proof of shrinkingness of our algorithm in the NTOM[B(k)] model. The general idea of the proof is to show that the destination points computed by correct robots are located either around the middle of the range of correct positions or/and in the neighborhood of only one end of this range.

If all computed destinations are located around the middle of the range of correct robots then the diameter of this range decreases and the algorithm is shrinking. Otherwise, if some computed destinations are located in the neighborhood of one end of the range, it is shown that there is a time at which no correct robot will be in the neighborhood of the other end of the range, which leads again to a decrease in the range of correct positions and shows that the algorithm is shrinking.

Lemma 5.8.9. *Algorithm 6 is shrinking for $n > 3f$ in the* NTOM[B(k)] *model for any $k > 0$.*

Proof. Let $U(\tau_0) = \{U_1(\tau_0), ..., U_m(\tau_0)\}$ be the configuration of correct robots at initial time τ_0 and $D(\tau_0) = \{D_1(\tau_0), ..., D_m(\tau_0)\}$ the multiset of their calculated destination points at the same time τ_0 and $UD(\tau_0)$ is the union of $U(\tau_0)$ and $D(\tau_0)$.

Let τ_1 be the first time at which all correct robots have been activated and executed their Look and Compute phase at least once since τ_0 ($U(\tau_1)$ and $D(\tau_1)$ are the corresponding multisets of positions and destinations). Assume that robots are ordered from left to right and define d_0 and d_1 as their diameters at τ_0 and τ_1 respectively. Since the model is non-atomic, the diameter calculation takes into account both the positions and the destinations of robots. So $d_0 = \text{diam}(UD(\tau_0))$ and $d_1 = \text{diam}(UD(\tau_1))$. Let b be any distance that is smaller than $d_0/4$, for example take $b = d_0/10$.

We consider the actions of correct robots after τ_1 and we separate the analysis into two cases:

- *Case A*: All calculated destinations by all correct robots after τ_1 are inside $[min(UD(\tau_0)) + b, max(UD(\tau_0)) - b]$. So when all correct robots are activated at least once, their diameter decreases by at least $\min\{2\Delta, 2b = d_0/5\}$. Thus by setting $\alpha_1 = \max\{1 - 2\Delta/d_0, 4/5\}$, the algorithm is shrinking.

- *Case B*: Let $\tau_2 > \tau_1$ be the first time when a robot, say r_i, execute a Look phase such that the Compute phase that follows compute a destination point, say D_i, that is outside $[min(UD(\tau_0)) + b, max(UD(\tau_0)) - b]$. This implies that either $(D_i < min(UD(\tau_0)) + b)$ or $(D_i > max(UD(\tau_0)) - b)$. Since the two cases are symmetric, we consider only the former which implies according to Lemma 5.8.8 that the range $[min(UD(\tau_0)), min(UD(\tau_0)) + 2b]$ must contain at least $m - f$ correct positions.

  If some robots among these $m - f$ robots are executing a Move phase, their destination points have necessarily been calculated after τ_0 (since at τ_1 each robot has been activated at least once). And we have by lemma 5.8.3 that the distance between each robot and its destination can not exceed half the diameter, so we conclude that at τ_2 the destination points of these $m - f$ robots are all inside $[min(UD(\tau_0)), min(UD(\tau_0)) + b + d_0/2]$.

  Let $S(\tau_2)$ be a submultiset of $UD(\tau_2)$ containing the positions and destinations of $f + 1$ arbitrary robots among these $m - f$ whose positions and destinations are inside

  $$[min(UD(\tau_0)), min(UD(\tau_0)) + b + d_0/2]$$

  So $\max(S(\tau_2)) \leq min(UD(\tau_0)) + b + d_0/2$. And since we chose $b < d_0/4$, we have $\max(S(\tau_2)) < max(UD(\tau_0)) - 3d_0/4$. Let $\tau_3 \geq \tau_2$ be the first time each correct robot in the system has been activated at least once since τ_2. We prove in the following that at τ_3, $\max(S(\tau_3)) < max(UD(\tau_0)) - 3d_0/2^{k(f+1)+2}$.

To this end we show that the activation of a single robot of $S(\tau)$ can not reduce the distance between the upper bound of $max(S)$ and $max(UD(\tau_0))$ by more than half its precedent value, and since the scheduler is k-bounded, we can guarantee that this distance at τ_3 is at least equal to $3d_0/2^{k(f+1)+2}$.

According to Lemma 5.8.5, if some robot r_i calculates its destination D_i at time $t \in [\tau_2, \tau_3]$, $D_i \leq (U_{f+1}(\tau) + max(UD(\tau)))/2$. But $U_{f+1}(\tau) \leq max(S(\tau))$ by Lemma 5.8.6 and $max(UD(\tau)) \leq max(UD(\tau_0))$ due to cautiousness. This gives us $D_i \leq (max(S(\tau) + max(UD(\tau_0)))/2$. Therefore, an activation of a single robot in $S(\tau)$ to execute its Compute phase can reduce the distance between $Max(UD(\tau_0))$ and $max(S(\tau))$ by at most half its previous value.

So at τ_3, after a maximum of k activations of each robot in $S(\tau)$, we have $max(S(\tau_3)) \leq Max(UD(\tau_0)) - 3d_0/2^{k(f+1)+2}$, and by Lemma 5.8.7, all calculated destinations by all correct robots between τ_2 and τ_3 are less than or equal to $Max(UD(\tau_0)) - 3d_0/2^{k(f+1)+3}$.

Since robots are guaranteed to move toward their destinations by at least a distance Δ before they can be stopped by the scheduler, after τ_3, no robot will be located beyond $Max(UD(\tau_0)) - min\{\Delta, 3d_0/2^{k(f+1)+3}\}$. Hence by setting $\alpha = \max\{\alpha_1, 1 - \Delta/d_0, 1 - 3/2^{k(f+1)+3}\}$ the lemma follows.

□

The convergence proof of Algorithm 6 directly follows from Lemmas 5.8.9 and 5.8.2 and Theorem 5.6.1.

Theorem 5.8.1. *Algorithm 6 solves the byzantine convergence problem for $n > 3f$ in the* NTOM[B(k)] *model for any $k > 0$.*

5.9 Deterministic Convergence in NTOM[AS] Networks

In this section, we propose a deterministic convergence algorithm and prove its correctness in the atomic fully asynchronous model NTOM[AS] when there are at least $5f + 1$ robots.

Algorithm Description The idea of our algorithm is based on three mechanisms: (1) a trimming function for the computation of destinations, (2) location dependency and (3) an election procedure. The purpose of the trimming function is to ignore the most extreme positions in the network when computing the destination. Robots move hence towards the center of the remaining positions.

Consequently, the effect of Byzantine robots is canceled since they cannot drag the correct robots away from the range of correct positions.

Location dependency affects the computation of the trimming function such that the returned result depends on the position of the calling robot. This leads to interesting properties on the relation between the position of a robot and its destination that are critical to convergence. The election procedure instructs to move only the robots located at the two extremes of the network. Thus, by the combined effect of these three mechanisms, as the algorithm progresses, the extreme robots come together towards the middle of the range of correct positions which ensures the eventual convergence of the algorithm.

The algorithm 7 uses three functions as follows. The trimming function $\text{trim}^i_{2f}()$ removes among the $2f$ largest positions of the multiset given in parameter *only* those that are greater than the position of the calling robot r_i. Similarly, it removes among the $2f$ smallest positions only those that are smaller than the position of the calling robot. It is clear that the output of $\text{trim}^i_{2f}()$ depends on the position of the calling robot. Formally, let $minindex_i$ be the index of the minimum position between $P_i(\tau)$ and $P_{2f+1}(\tau)$ (if $P_i(\tau) < P_{2f+1}(\tau)$ then $minindex_i$ is equal to i, otherwise it is equal to $2f+1$). Similarly, let $maxindex_i$ be the index of the maximum position between $P_i(\tau)$ and $P_{n-2f}(\tau)$ (if $P_i(\tau) > P_{n-2f}(\tau)$ then $maxindex_i$ is equal to i, otherwise it is equal to $n-2f$). $\text{trim}^i_{2f}(P(\tau))$ is the multiset consisting of positions $\{P_{minindex_i}(\tau), P_{minindex_i+1}(\tau), \ldots, P_{maxindex_i}(\tau)\}$.

The function $mid()$ simply returns the median point of the input range. The two functions are illustrated in Figure 5.8).

The election function returns true if the calling robot is allowed to move. Only the robots that are located at the extremes of the networks are allowed to move, that is those whose position is either $\leq P_{f+1}(\tau)$ or $\geq P_{n-f}(\tau)$.

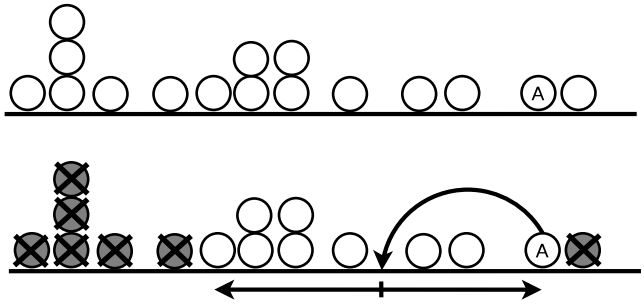


Figure 5.8: Illustration of functions $\text{trim}^i_{2f}()$ and $mid()$ in a system of ($n = 16, f = 3$) robots.

Algorithm 7 Byzantine Tolerant Convergence in NTOM[AS]

Functions:
$\text{trim}^i_{2f}(P(\tau))$: removes up to $2f$ largest positions that are larger than $P_i(\tau)$ and up to $2f$ smallest positions that are smaller than $P_i(\tau)$ from the multiset $P(\tau)$ given in parameter.
$mid()$: returns the point that is in the middle of the range of points given in parameter.
$elected() \equiv ((P_i(\tau) \leq P_{f+1}(\tau))$ or $(P_i(\tau) \geq P_{n-f}(\tau)))$. This function returns true if the calling robot is allowed to move.

Actions:
if $elected()$ move towards $mid(\text{trim}^i_{2f}(P(\tau)))$

In the following we prove the correctness of Algorithm 7 in the NTOM[AS] model by showing that it is both shrinking and cautious.

5.9.1 Algorithm 7 is Cautious

In this section we prove that Algorithm 7 is a cautious algorithm (see Definition 14) for $n > 5f$. The following lemma states that the range of the trimmed multiset $\text{trim}^i_{2f}(P(\tau))$ is contained in the range of correct positions.

Lemma 5.9.1. *Let r_i be a correct robot executing Algorithm 7, it holds that*

$$\forall \tau, range(trim^i_{2f}(P(\tau))) \subseteq range(U(\tau))$$

Proof. We prove that for any correct robot, r_i, the following conditions hold:

1. $\forall \tau,\ min(\text{trim}^i_{2f}(P(\tau))) \in \text{range}(U(\tau))$.
2. $\forall \tau,\ max(\text{trim}^i_{2f}(P(\tau))) \in \text{range}(U(\tau))$.

1. By definition, $min(\text{trim}^i_{2f}(P(\tau))) = min\{P_i(\tau), P_{2f+1}(\tau)\}$. Hence proving Property (1) reduces to proving $P_i(\tau) \in \text{range}(U(\tau))$ and $P_{2f+1}(\tau) \in \text{range}(U(\tau))$.

   a) $P_i(\tau) \in \text{range}(U(\tau))$ directly follows from the assumption that robot r_i is correct.

   b) $P_{2f+1}(\tau) \in \text{range}(U(\tau))$. Suppose the contrary: there exists some time instant τ such that $P_{2f+1}(\tau) \notin \text{range}(U(\tau))$ and prove that this leads to

a contradiction. If $P_{2f+1}(\tau) \notin \text{range}(U(\tau))$ then either $P_{2f+1}(\tau) < U_1(\tau)$ or $P_{2f+1}(\tau) > U_m(\tau)$.

i. If $P_{2f+1}(\tau) < U_1(\tau)$ then there are at least $2f+1$ positions $\{P_1(\tau), P_2(\tau), \ldots, P_{2f}(\tau), P_{2f+1}(\tau)\}$ that are smaller than $U_1(\tau)$ which is the *first* correct position in the network at time τ. This means that there would be at least $2f+1$ Byzantine robots in the system. But this contradicts the assumption that at most f Byzantine robots are present in the system.

ii. If $P_{2f+1}(\tau) > U_m(\tau)$ then since $n > 5f$ there are more than $3f$ positions $\{P_{2f+1}(\tau), ..., P_n(\tau)\}$ that are greater than $U_m(\tau)$, which is the *last* correct position in the system at time τ. This also leads to a contradiction.

2. The property is symmetric to 2) and can be proved using the same argument.

□

A direct consequence of the above property is that correct robots always compute a destination within the range of positions held by correct robots, whatever the behavior of Byzantine ones. Thus, the diameter of positions held by correct robots never increases. Consequently, the algorithm is cautious. The formal proof is proposed in the following lemma.

Lemma 5.9.2. *Algorithm 7 is cautious for $n > 5f$.*

Proof. According to Lemma 5.9.1, $\text{range}(\text{trim}^i_{2f}(P(\tau))) \subseteq \text{range}(U(\tau))$ for each correct robot r_i, thus $mid(\text{trim}^i_{2f}(P(\tau))) \in \text{range}(U(\tau))$. It follows that all destinations computed by correct robots are located inside $\text{range}(U(\tau))$ which proves the cautiousness property. □

5.9.2 Algorithm 7 is Shrinking

The following lemma proves that the only robots that can be elected are those located at the extremes of the network, namely those whose position is either less equal than $U_{f+1}(\tau)$ or greater equal than $U_{m-f}(\tau)$. The activation of these robots move them away from the extremes of the network, thereby reducing the diameter of positions held by correct robots which leads to convergence.

Lemma 5.9.3. *If some correct robot r_i is activated at time τ, then either $U_i(\tau) \leq U_{f+1}(\tau)$ or $U_i(\tau) \geq U_{m-f}(\tau)$ where m is the number of correct robots in the network and $U_i(\tau)$ denotes the position of correct robot r_i at τ;*

Proof. By definition of the algorithm, a robot is activated only if its position is either $\leq P_{f+1}(\tau)$ or $\geq P_{n-f}(\tau)$. To prove the lemma, it suffices then to show that $P_{f+1}(\tau) \leq U_{f+1}(\tau)$ and $P_{n-f}(\tau) \geq U_{m-f}(\tau)$:

To prove that $P_{f+1}(\tau) \leq U_{f+1}(\tau)$, we suppose to the contrary that $P_{f+1}(\tau) > U_{f+1}(\tau)$. In this case, $P_{f+1}(\tau)$ would be strictly greater than all the positions $\{U_1(\tau), ..., U_{f+1}(\tau)\}$, which contradicts the definition of $P_{f+1}(\tau)$ as the $(f+1)$-th position in the network. This proves that $P_{f+1}(\tau) \leq U_{f+1}(\tau)$ and the same argument is used to prove that $P_{n-f}(\tau) \geq U_{m-f}(\tau)$, since the two cases are symmetric.

□

The following lemma proves an important property on the relationship between the position of a robot and its computed destination. Indeed, knowing the position $U_i(\tau)$ held by a correct robot r_i at time τ, it is possible to give bounds on the possible value of its destination point $D_i(\tau)$. Interestingly, this bound holds irrespective of the positions of Byzantine robots and the actions of the adversary.

Formally, consider any initial configuration at time τ_0, such that $U(\tau_0)$ and $D(\tau_0)$ are respectively the multiset of positions and destinations of correct robots at time τ_0. Define $UD(\tau_0)$ to be the union of $U(\tau_0)$ and $D(\tau_0)$. By considering the cycles started by correct robots after τ_0, the following property holds:

Lemma 5.9.4. *For each correct robot r_i that starts a cycle after τ_0, the following inequalities hold:*

$$D_i(\tau) \in [\frac{U_i(\tau) + Min(UD(\tau_0))}{2}, \frac{U_i(\tau) + Max(UD(\tau_0))}{2}]$$

.

Proof. The proof is twofold. First, we show that (1) $D_i(\tau) \geq (U_i(\tau) + Min(UD(\tau_0)))/2$. Then, we prove the symmetric property (2) $D_i(\tau) \leq (U_i(\tau) + Max(UD(\tau_0)))/2$.

1. $D_i(\tau) \geq (U_i(\tau) + Min(UD(\tau_0)))/2$:

   Assume towards contradiction that for some robot r_i that start a cycle at time $\tau_1 \geq \tau_0$, there exists a time $\tau \geq \tau_1$ in this cycle such that:

$$D_i(\tau) < \frac{U_i(\tau) + min(UD(\tau_0))}{2}$$

Note that $U_i(\tau_1) \geq U_i(\tau)$ because if robot r_i moves between τ_1 and τ, it becomes closer to its destination $D_i(\tau)$. Thus:

$$D_i(\tau) < \frac{U_i(\tau_1) + min(UD(\tau_0))}{2} \ldots (1)$$

This means that $|min(UD(\tau_0)), D_i(\tau)| < |D_i(\tau), U_i(\tau_1)|$. Denote by d the distance between $U_i(\tau_1)$ and $D_i(\tau)$. Note that $D_i(\tau) < U_i(\tau_1)$.

The computation of $D_i(\tau)$ by r_i is based on the configuration of the network as last seen by robot r_i. That is, the configuration of the system at the beginning of its cycle $P(\tau_1)$. This implies that:

$$D_i(\tau) = mid(\text{trim}^i_{2f}(P(\tau_1))) \ldots (2)$$

We prove that (1) and (2) combined lead to a contradiction.

The location dependency property of the trimming function implies that $U_i(\tau_1) \in \text{trim}^i_{2f}(P(\tau_1))$.

So, up to this point we proved that there exists a point $U_i(\tau_1) \in \text{trim}^i_{2f}(P(\tau_1))$ such that $U_i(\tau_1) > D_i(\tau)$ and $|U_i(\tau_1), D_i(\tau)| = d$.

But since by (2), $D_i(\tau)$ is the center of $\text{trim}^i_{2f}(P(\tau_1))$, there must exists another point $q \in \text{trim}^i_{2f}(P(\tau_1))$, such that $q < D_i(\tau)$ and $|q, D_i(\tau)| = d$.

But we observed from (1) that $|min(UD(\tau_0)), D_i(\tau_1)| < d$, which implies that $|min(UD(\tau_0)), D_i(\tau_1)| < |q, D_i(\tau)|$. This means that $q < min(UD(\tau_0))$. But $q \in \text{trim}^i_{2f}(P(\tau_1))$, so $min(\text{trim}^i_{2f}(P(\tau_1))) < min(UD(\tau_0))$. This contradicts lemma 5.9.1, which proves the first part of our lemma.

2. **(2)** $D_i(\tau) \leq (U_i(\tau) + Max(UD(\tau_0)))/2$: The property is symmetric to (1) and can be proved using the same argument.

□

Let S be a subset of correct robots, and define $UD_S(\tau)$ to be the multiset of their positions and destinations at time τ.

Lemma 5.9.5. *If $|S| \geq m - 2f$ and there exists a time $\tau_1 \geq \tau_0$ such that for each $\tau > \tau_1$, $max(UD_S(\tau)) \leq max(UD(\tau_0)) - b$, then all computed destinations by all correct robots in cycles that start after τ_1 are $\leq max(UD(\tau_0)) - b/2$.*

Proof. Let r_i be any correct robot that computes its destination D_i in a cycle started after τ_1, say at τ. We prove in the following that $D_i \leq max(UD(\tau_0)) - b/2$:

First, observe that since $max(UD_S(\tau)) \leq max(UD(\tau_0)) - b$ and $|S| \geq m - 2f > 2f$, then

$$min(\text{trim}^i_{2f}(P(\tau)) \leq max(UD(\tau_0)) - b$$

Otherwise, $min(\text{trim}^i_{2f}(P(\tau))$ would be greater than all the positions in S ($> 2f$ positions), which contradicts the definition of $\text{trim}^i_{2f}()$ (at most the $2f$ smallest positions are removed).

According to lemma 5.9.1, we have

$$max(\text{trim}^i_{2f}(P(\tau)) \leq max(UD(\tau_0))$$

But D_i is the center of $\text{trim}^i_{2f}(P(\tau))$, which means that $|D_i, min(\text{trim}^i_{2f}(P(\tau)))|$ must be equal to $|D_i, max(\text{trim}^i_{2f}(P(\tau)))|$. Hence,

$$D_i \leq max(UD(\tau_0)) - b/2$$

□

Lemma 5.9.6. *If $|S| \geq m - f$ and at some time $\tau_1 \geq \tau_0$, $max(UD_S(\tau_1)) \leq max(UD(\tau_0)) - b$, then all computed destinations by all correct robots in cycles that start after τ_1 are less or equal to $max(UD(\tau_0)) - b/2$.*

Proof. First, we prove that after τ_1, the robots in S remains always at positions $< max(UD(\tau_0)) - b$, meaning that all their computed destinations after τ_1 are $< max(UD(\tau_0)) - b$.

Assume the contrary: Let r_i be the first robot in S that starts a cycle after τ_1 such that its computed destination in this cycle is $> max(UD(\tau_0)) - b$. This implies that $max(\text{trim}^i_{2f}(P(\tau_1))) > max(UD(\tau_0)) - b$, which means that at least $2f+1$ positions in the network at τ_1 are strictly greater than $max(UD(\tau_0)) - b$.

If we add to these $2f + 1$ positions that are greater than $max(UD(\tau_0)) - b$, the $m - f \geq n - 2f$ positions in S that are less or equal than $max(UD(\tau_0)) - b$, we get a total number of robots in the network that is strictly greater than n, which leads to contradiction. This proves that all positions and destinations of robots in S after τ_1 are less than or equal to $max(UD(\tau_0)) - b$. Thus by lemma 5.9.5, the destinations computed by *all* correct robots in the network are less than or equal to $max(UD(\tau_0)) - b/2$.

□

The next Lemma states that if some computed destination is located in the neighborhood of one extreme of the network, then a majority of correct robots (at least $m-2f$) are located in the neighborhood of this extreme.

Lemma 5.9.7. *Let D_i be a destination point computed by a correct robot r_i in a cycle started at time τ. If $D_i < min(UD(\tau)) + b$, then at least $m-2f$ correct robots are located at positions that are $< min(UD(\tau)) + 2b$ at τ.*

Proof. The computation of D_i is based on the configuration of the network as last seen by robot r_i, that is the configuration at the beginning of the cycle at τ, $P(\tau)$. So we first prove that at τ, $max(\text{trim}^i_{2f}(P(\tau))) < min(UD(\tau)) + 2b$:

By hypothesis, $D_i < min(UD(\tau)) + b$. But according to lemma 5.9.1, $min(UD(\tau)) \leq min(\text{trim}^i_{2f}(P(\tau)))$. Thus, $D_i < min(\text{trim}^i_{2f}(P(\tau))) + b$. This means that

$$|D_i, min(\text{trim}^i_{2f}(P(\tau)))| < b$$

But D_i is the center of $\text{trim}^i_{2f}(P(\tau))$, which means that $|D_i, min(\text{trim}^i_{2f}(P(\tau)))|$ must be equal to $|D_i, max(\text{trim}^i_{2f}(P(\tau)))|$. Hence,

$$max(\text{trim}^i_{2f}(P(\tau))) < D_i + b$$

But since by hypothesis $D_i < min(UD(\tau)) + b$, we have

$$max(\text{trim}^i_{2f}(P(\tau))) < min(UD(\tau)) + 2b$$

This means that at most $2f$ positions (which may be correct) are $\geq min(UD(\tau)) + 2b$ at τ. This completes the proof.

□

Let $U(\tau_0)$ and $D(\tau_0)$ be respectively the multisets of positions and destinations of correct robots at the initial time τ_0, and define $UD(\tau_0)$ to be the union of $U(\tau_0)$ and $D(\tau_0)$. Take b to be any distance $< \text{diam}(UD(\tau_0))/4$, for example $b = \text{diam}(UD(\tau_0))/10$.

The next lemma states that if a correct robot elected at $\tau > \tau_0$ is located inside the range $(min(UD(\tau_0)) + b, max(UD(\tau_0)) - b)$, then the destinations points computed by correct robots after τ are either $all \leq max(UD(\tau_0)) - b/4$ or $all \geq min(UD(\tau_0)) + b/4$. This means that the election of a robot located inside $(min(UD(\tau_0)) + b, max(UD(\tau_0)) - b)$ is a sufficient condition to convergence.

Lemma 5.9.8. *Let τ_1 be the first time at which all correct robots in the network executed a complete cycle at least once since τ_0.*

If a correct robot is elected at $\tau > \tau_1$ and is located inside $[min(UD(\tau_0)) + b, max(UD(\tau_0)) - b]$, then the destination points computed by correct robots in cycles that start after τ are either all located at positions $\leq max(UD(\tau_0)) - b/4$ or all located at positions $\geq min(UD(\tau_0)) + b/4$.

Proof. Let r_i be a correct robot that is elected at time $\tau > \tau_1$ and whose position $U_i(\tau)$ is inside $[min(UD(\tau_0)) + b, max(UD(\tau_0)) - b]$. According to lemma 5.9.3, either $U_i(\tau) \geq U_{m-f}(\tau)$ or $U_i(\tau) \leq U_{f+1}(\tau)$. Thus, we separate the analysis into two cases depending on the rank of the elected robot:

- **Case 1:** $U_i(\tau) \geq U_{m-f}(\tau)$.

  Define $S(\tau)$ to be the set of correct positions $\{U_1(\tau), ..., U_{m-f}(\tau), ..., U_i(\tau)\}$, and note that $|S(\tau)| \geq m - f$.

  By hypothesis, $U_i(\tau) \leq max(UD(\tau_0)) - b$ which implies that the positions of all robots in $S(\tau)$ are $\leq min(UD(\tau_0)) + b$. Thus by lemma 5.9.4, the destinations of all robots in $S(\tau)$ are $\leq max(UD(t_0)) - b/2$. This means that $range_S(\tau)$, the range of positions and destinations of robots in $S(\tau)$ is such that at τ, $max(range_S(\tau)) \leq max(UD(\tau_0)) - b/2$. Hence, according to lemma 5.9.6, all destinations points computed by correct robots in cycles that start after τ are $\leq max(UD(\tau_0)) - b/4$.

- **Case 2:** $U_i(\tau) \leq U_{f+1}(\tau)$.

  The case is symmetric and we prove by a similar argument to **Case 1** that all destination points computed by correct robots are $\geq min(UD(\tau_0)) + b/4$, which proves our lemma.

□

Lemma 5.9.9. *Algorithm 7 is shrinking for $n > 5f$ in the* `NTOM[AS]` *model.*

Proof. Let $U(\tau_0) = \{U_1(\tau_0), U_m(\tau_0)\}$ be the configuration of correct robots at initial time τ_0, and let $D(\tau_0) = \{D_1(\tau_0), ..., D_m(\tau_0)\}$ the multiset of their destinations at τ_0. Define $UD(\tau_0)$ to be the union of $U(\tau_0)$ and $D(\tau_0)$, and let $\text{diam}(\tau_0)$, the diameter at τ_0, be equal to $max(UD(\tau_0)) - min(UD(\tau_0))$. $U(\tau)$, $D(\tau)$, $UD(\tau)$ and $\text{diam}(\tau)$ for each $\tau > \tau_0$ are defined similarly.

Let τ_1 be the first time at which every correct robot in the network has executed a whole cycle at least once since τ_0. We consider the evolution of the network after τ_1. The aim of this is to apply lemma 5.9.4, that is, based only on the position of a correct robot, we can give bounds on its destination point which is especially interesting in the case of a robot executing a Move phase of its cycle.

We take into account all the computed destinations by correct robots after τ_1 and we distinguish between two cases: (1) the case when all destinations computed after τ_1 are inside $[min(UD(\tau_0)) + \frac{\text{diam}(\tau_0))}{10}, max(UD(\tau_0)) - \frac{\text{diam}(\tau_0)}{10}]$. and (2) the case when a computed destination after τ_1 lay outside this range. We show that in both cases, there is a time at which the diameter of correct positions decreases by a factor of at least 39/40.

- **Case 1:** All destinations computed by correct robots in cycles started after τ_1 are inside the range

  $$[min(UD(\tau_0)) + \text{diam}(\tau_0)/10, max(UD(\tau_0)) - \text{diam}(\tau_0)/10].$$

  In this case, since each robot r_i is guaranteed to move a minimal distance of δ_i before it can be stopped by the adversary, there is a time $\tau_2 \geq \tau_1$ when all correct robots are located inside $[min(UD(\tau_0)) + \text{diam}(\tau_0)/10, max(UD(\tau_0)) - \text{diam}(\tau_0)/10]$. Thus $\text{diam}(\tau_2) = \text{diam}(\tau_0) * 4/5$, and by setting $\alpha = 4/5$, our algorithm is shrinking.

- **Case 2:** There is a destination D_i, computed by a correct robot r_i in a cycle started after τ_1, that is outside the range

  $$[min(UD(\tau_0)) + \text{diam}(\tau_0)/10, max(UD(\tau_0)) - \text{diam}(\tau_0)/10].$$

  This means that either $D_i < min(UD(\tau_0)) + \text{diam}(\tau_0)/10$ or $D_i > max(UD(\tau_0)) - \text{diam}(\tau_0)/10$. Since the two cases are symmetric, there is no loss of generality to assume that $D_i < min(UD(\tau_0)) + \text{diam}(\tau_0)/10$.

  The calculation of D_i is based on the configuration of the network as seen by robot r_i at the beginning of the cycle, say at τ_2 (with $\tau_2 \geq \tau_1$). Thus, according to lemma 5.9.7, at τ_2, at least $m - 2f$ correct robots are located at positions $< min(UD(\tau_0)) + \text{diam}(\tau_0)/5$. Denote by $S(\tau_2)$ the set of these robots. By lemma 5.9.4, the destinations of robots in $S(\tau_2)$ are $< min(UD(\tau_0)) + \text{diam}(\tau_0) * (3/5)$. Thus, the positions and destinations of robots in $S(\tau_2)$ are $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 2/5$.

  We now observe the positions of elected robots whose rank is $\leq f + 1$ and which are activated after τ_2. We separate the analysis into two subcases:

  - **Subcase 2A:** There is a time $\tau > \tau_2$ at which is elected a correct robot r_i whose rank is $\leq f + 1$ and whose position $U_i(\tau)$ is $>$

$min(UD(\tau_0)) + \text{diam}(\tau_0)/10$. Notice that since $|S(\tau_2)| > m - 2f$, $U_i(\tau)$ is also $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 2/5$ which is the upper bound on the positions of robots in $S(\tau_2)$. Thus, $U_i(\tau) \in [min(UD(\tau_0)) + \text{diam}(\tau_0)/10, max(UD(\tau_0)) - \text{diam}(\tau_0)/10]$ and according to lemma 5.9.8, the diameter eventually decreases by a multiplicative factor of $1 - 1/40$. Hence, by setting $\alpha = 39/40$ the lemma follows.

– **Subcase 2B:** All elected correct robots that are activated after τ_2 and whose rank is $\leq f + 1$ are located at positions $< min(UD(\tau_0)) + \text{diam}(\tau_0)/10$. This implies, according to lemma 5.9.4, that the positions of these elected robots remain always at positions $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 9/20$. Thus, all robots in $S(\tau_2)$ remain always at positions $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 9/20$ $(\forall\, t > \tau_2)$.

  According to lemma 5.9.5, all destinations computed at cycle that start after τ_2 are $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 9/40$. And since robots are guaranteed to move toward destinations by a minimum distance before they can be stopped by the adversary, they all end up located at positions $< max(UD(\tau_0)) - \text{diam}(\tau_0) * 9/40$. Hence there is a time $\tau > \tau_2$ such that $\text{diam}(\tau) = \text{diam}(\tau_0) * (1 - 9/40)$. It suffices to set $\alpha = 31/40$ and the lemma follows.

Consequently, we set $\alpha = 39/40$ and the lemma is proved. □

CHAPTER 6

RoboCast

6.1 Introduction

In this chapter we implement the RoboCast communication abstraction in the non-oblivious NTOM model. We assume that each robot has its own local coordinate system which remains consistent during the whole execution unless it is changed by the robot. The RoboCast primitive permits robots not agreeing on a common coordinate system, to exchange their local coordinate systems. For simplicity, we first present an algorithm for a network consisting in only two robots (Section 6.2). Then, we generalize the solution for any number of robots (Section 6.3). Our algorithms rely on a local collision avoidance scheme that is presented in Section 6.4. Finally, using the RoboCast primitive, we propose in Section 6.5 algorithms for deterministic non-oblivious asynchronous gathering for two robots networks and binary information exchange (stigmergy). to implement two fundamental building blocks in robot networks: non-oblivious gathering for two robots networks and stigmergy.

6.2 RoboCasting the Local Coordinate System: Two Robots Networks

In this section we present algorithms for robocasting the local coordinate system in a two robots networks. The local coordinate system is defined by two axes (ab-

scissa and ordinate), their positive directions and the unity of measure. In order to robocast this information we use a modular approach. That is, robots invoke first the RoboCast primitive ($LineRbcast1$ hereafter) to broadcast a line representing their abscissa. Then, using a parametrized module ($LineRbcast2$), they robocast three successive lines encoding respectively their ordinate, unit of measure and the positive direction of axes. When a node broadcasts a line, without any additional knowledge, two different points have to be sent in order to uniquely identify the line at the destination. However, in the case of a coordinate system, only for the first transmitted axis nodes need to identify the two points. The transmission of the subsequent axes needs the knowledge of a unique additional point.

6.2.1 **Line** RoboCast

In robot networks the broadcast of axes is a long-studied issue. Suzuki and Yamashita [34, 35] presented an algorithm for broadcasting the axes via motion that performs in the ATOM model. However, their algorithm heavily relies on the atomicity of cycles and the observation focus on the different positions of the other robots during their Move phase.

The idea is as follows: assume two robots r and r' that want to broadcast their x-axes to each others. To do so, each of them move through the positive direction of its local x-axis each time it is activated. Hence, both move on the positive direction of their x-axes. When a robot r observes r' in at least four different positions, it can infer that r' executed at least two complete cycles [35], which means that r' observed r in at least two different positions of its x-axis. Such a claim is true only in the ATOM model, which guarantees that robots execute their cycles in a lock-step manner. The same idea has been exploited in [22] for building stigmergic systems in the ATOM model.

This type of observation and correctness argument is invalid in the asynchronous NTOM model. In this model, when a robot r moves towards its destination, another robot r' can be activated $k > 1$ times with k arbitrarily large, and thus observe r in k different positions without having any clue on the number of complete cycles executed by r. In other words, the number of different positions observed for a given robot is not an indicator on the number of complete executed cycles since in NTOM, cycles are completely uncorrelated.

Our solution thus uses a novel strategy. That is, the focus moves from observing robots in different positions to observing their change of direction: each robot changes its direction of movement when a particular stage of the algorithm is completed; this change allows the other robots to infer information about the observed robot.

Line RoboCast **Detailed Description** Let r_0 and r_1 be the two robots in the system. In the sequel, when we refer to one of these robots without specifying which one, we denote it by r_i and its peer by r_{1-i}. In this case, the operations on the indices of robots are performed modulo 2. For ease of presentation we assume that initially each robot r_i translates and rotates its local coordinate system such that its x-axis and origin coincide with the line to be broadcast and its current location respectively. We assume also that each robot is initially located in the origin of its local coordinate system.

At the end of the execution, each robot must have broadcast its own line and have received the line of its peer. A robot "receives" the line broadcast by its peer when it knows at least two distinct positions of this line. Thus, to send its line, each robot must move along it (following a scheme that will be specified later) until it is sure that it has been observed by the other robot.

The algorithm idea is simple: each robot broadcasts its line by moving along it in a certain direction (considered to be positive). Simultaneously, it observes the different positions occupied by its peer r_{1-i}. Once r_i has observed r_{1-i} in two distinct positions, it informs it that it has received its line by changing its direction of movement, that is, by moving along its line in the reverse direction (the negative direction if the first movement have been performed in the positive direction of the line). This change of direction is an acknowledgement for the reception of the peer line. A robot finishes the algorithm once it changed its direction and observed that the other robot also changed its direction. This means that both robots have sent their line and received the other's line.

The algorithm is described in detail as Algorithm 8. Each robot performs four stages referred in Algorithm 8 as states:

- state S_1: This is the initial state of the algorithm. At this state, the robot r_i stores the position of its peer in the variable pos_1 and heads towards the position (1.0) of its local coordinate system. That is, it moves along its line in the positive direction. Note that r_i stays only one cycle in this state and then goes to state S_2.

- state S_2: A this point, r_i knows only one point of its peer line (recorded in pos_1). To be able to compute the whole peer line, r_i must observe r_{1-i} in another (distinct) position of this line. Hence, each time it is activated, r_i checks if r_{1-i} is still located in pos_1 or if it has already changed its position. In the first case (line 2.a of the code), it makes no movement by selecting its current position as its destination. Otherwise (line 2.b), it saves the new position of r_{1-i} in pos_2 and delivers the line formed by pos_1 and pos_2. Then,

it initiates a change of direction by moving towards the point (-1.0) of its local coordinate system, and moves to state S_3.

- state S_3: at this point r_i knows the line of its peer locally derived from pos_1 and pos_2. Before finishing the algorithm, r_i must be sure that also r_{1-i} knows its line. Therefore, it observes r_{1-i} until it detects a change of direction (the condition of line 3.a). If this is not the case and if r_i is still in the positive part of its x-axis, then it goes to the position $(-1,0)$ of its local coordinate system (line 3.b). Otherwise (if r_i is already in the negative part of its x-axis), it performs a null movement (line 3.c). When r_i is in state S_3 one is sure, as we shall show later, that r_{1-i} knows at least one position of l_i, say p. Recall that l_i corresponds to the x-axis of r_i. It turns out that p is located in the positive part of this axis. In moving towards the negative part of its x-axis, r_i is sure that it will eventually be observed by r_{1-i} in a position distinct from p, which allows r_{1-i} to compute l_i.

- state S_4: At this stage, both r_i and r_{1-i} received the line sent by each others. That is, r_i has already changed its own direction of movement, and observed that r_{1-i} also changed its direction. But nothing guarantees that at this step r_{1-i} knows that r_i changed its direction of movement. If r_i stops now, r_{1-i} may remain stuck forever (in state S_3). To announce the end of the algorithm to its peer, r_i heads towards a position located outside l_i, That is, it will move on a line $nextl_i$ (distinct from l_i) that is given as parameter to the algorithm. During the move from l_i to $nextl_i$, r_i should avoid points outside these lines. To this end, r_i must first pass through $myIntersect$, which is the intersection of l_i and $nextl_i$ - before moving to a point located in $nextl_i$ but not on l_i (refer to lines 3.a.2, 3.a.3 and 4.a of the code).

  Note that the RoboCast of a line is usually followed by the RoboCast of other information (e.g. other lines that encode the local coordinate system). To helps this process the end of the RoboCast of l_i should mark the beginning of the next line, $nextl_i$, RoboCast. Therefore, once r_i reaches $myIntersect$, r_i rotates its local coordinate system such that its x-axis matches now with $nextl_i$, and then it moves toward the point of (1,0) of its (new) local coordinate system. When r_{1-i} observes r_i in a position that is not on l_i, it learns that r_i knows that r_{1-i} learned l_{1-i}, and so it can go to state S_4 (lines 3.a.$*$) and finish the algorithm.

In the following we prove that Algorithm 8 satisfies the specification of a RoboCast, namely validity and termination. First we introduce some notations that will be further used in the proofs of the algorithm. For each variable v, and

Algorithm 8 Line RoboCast **LineRbcast1** for two robots: Algorithm for robot r_i.

Variables:
$state$: initially S_1
pos_1, pos_2: initially $\perp$
$destination, myIntersect$: initially $\perp$

Actions:
1. State [S_1]: *%Robot r_i starts the algorithm%*

a. $pos_1 \leftarrow observe(1-i)$
b. $destination \leftarrow (1,0)_i$
c. $state \leftarrow S_2$
d. Move to destination

2. State [S_2]: *%r_i knows one position of l_{1-i}%*

a. **if** $(pos_1 = observe(1-i))$ **then** $destination \leftarrow observe(i)$
b. **else**
1. $pos_2 \leftarrow observe(1-i)$
2. $l_{1-i} \leftarrow line(pos_1, pos_2)$
3. Deliver (l_{1-i})
4. $destination \leftarrow (-1,0)_i$
5. $state \leftarrow S_3$ **endif**
c. Move to destination

3. State [S_3]: *%r_i knows the line robocast by robot r_{1-i}%*

a. **if** (pos_2 is not inside the line segment $[pos_1, observe(1-i)]$) **then**
1. $state \leftarrow S_4$
2. $myIntersect \leftarrow intersection(l_i, nextl_i)$
3. $destination \leftarrow myIntersect$
b. **else if** $(observe(i) \geq (0,0)_i)$ **then** $destination \leftarrow (0,-1)_i$
c. **else** $destination \leftarrow observe(i)$ **endif endif**
d. Move to destination

4. State [S_4]: *%r_i knows that robot r_{1-i} knows its line l_i%*

a. **if** $(observe(i) \neq myIntersect)$ **then** $destination \leftarrow myIntersect$
b. **else**
1. r_i rotates its coordinate system such that its x-axis and the origin match with
$nextl_i$ and $myIntersect$ respectively.
2. $destination \leftarrow (1,0)_i$; return **endif**
c. Move to destination

each robot r_i, we denote $r_i.v(\tau)$ the value of the variable v in the local memory of r_i at time τ. When the time information can be derived from the context, we use simply $r_i.v$.

Proof of the Validity property. We start by the validity property. For this, we first prove a series of technical lemmas. The following two lemmas state the existence of a time instant at which both robots have reached S_2 and a following time instant at which at least one of them have reached S_3.

Lemma 6.2.1. *Eventually, both robots reach state S_2.*

Proof. Thanks to the fairness assumption of the scheduler, every robot is activated infinitely often. The first time each robot is activated, it executes the lines (1.a, 1.b, 1.c) of Algorithm 8 and it reaches the state S_2. □

Lemma 6.2.2. *Eventually, at least one robot reaches state S_3.*

Proof. Let r_1 and r_2 be two robots executing Algorithm 8, and assume towards contradiction that neither of them reaches state S_3. But according to Lemma 6.2.1, they both eventually reach S_2. Consider for each robot r_i the cycle in which it reaches state S_2, and define τ_i to be the time of the end of the Look phase of this cycle. Without loss of generality, assume $\tau_1 \leq \tau_2$ (the other case is symmetric). Hence, the variable $r_1.pos_1$ describes the position of robot r_2 at τ_1 expressed in the local coordinate system of robot r_1. Let $\tau_3 > \tau_2$ be the time at which robot r_2 finishes its cycle that leads it to state S_2. Between τ_2 and τ_3, robot r_2 performed a non null movement because it moved towards the point (1.0) of its local coordinate system (line 1.b of the code). Hence, the position of r_2 at τ_3 is different from its position at $\tau_1 \leq \tau_2$, which was recorded in the variable $r_1.pos_1$. By assumption, r_2 never reaches state S_3, so each time it is activated after τ_3 it keeps executing the lines 2.a and 2.c of the code and never moves from its current position (reached at τ_3). By fairness, there is a time $\tau \geq \tau_3$ at which robot r_1 is activated again. At this time, it observes r_2 in a position different from $r_1.pos_1$. This means that for r_1 the condition of line 2.a is false. Hence, r_1 executes the else block of the condition (2.$b.*$) and reaches state S_3, which contradicts the assumption and proves the lemma. □

The next lemma expresses the fact that our algorithm exhibits some kind of synchrony in the sense that robots advance in the execution of the algorithm through the different states in unison. That is, neither of them surpasses the other by more than one stage (state).

Lemma 6.2.3. *If at some time robot r_i is in state S_j and robot r_{1-i} is in state S_k then $|j-k| \leq 1$.*

Proof. The proof of the lemma is divided into two parts:

- If robot r_i is in state S_3 and robot r_{1-i} is in state S_j then $j \geq 2$.

  proof: Since robot r_i is in state S_3, it has necessarily executed the lines $(1.a \ldots 1.d)$ and $(2.b.*)$ of the code. Hence, the value of variable $r_i.pos_1$ is different from that of $r_i.pos_2$. This means that r_i has seen r_{1-i} in at least two different positions that implies that r_{1-i} has been activated at least once. Hence, r_{1-i} has necessarily executed the lines $(1.a \ldots 1.c)$ and has reached S_2.

- If robot r_i is in state S_4 and robot r_{1-i} is in state S_j then $j \geq 3$.

  proof: For a robot r_{1-i} to reach state S_4, it must execute the line $3.a$ of the code and detect that the other robot r_{1-i} has changed its direction of movement (it moved toward the negative part of its x-axis). Thus, robot r_{1-i} has necessarily executed lines $(2.b.2 \ldots 2.c)$ of the code, which means that r_{1-i} is in state S_3. Before this, robot r_{1-i} moved only in one direction, that is, in the positive direction of its x-axis.

This proves the lemma. □

The following lemma states that each robot r_i is guaranteed to be observed by its peer at least once in a position located in the positive part of its local x-axis. The observed position is stored in $r_{1-i}.pos_1$. Expressed otherwise, this means that each robot "sends" a position located in its positive x-axis to its peer. This property is important for proving validity that corresponds to both robots eventually reaching S_3 (Lemmas 6.2.5 and 6.2.6). Indeed, since each robot r_i is guaranteed that a point located in its local x-axis was received by its peer r_{1-i}, it suffices for r_i to send its line to head toward the negative part of its x-axis and to stay there until it is observed by r_{1-i}. That is, until a position located in its negative x-axis (and thus *distinct* from $r_{1-i}.pos_1$) is received by r_{1-i}.

Lemma 6.2.4. *For each robot r_i, the variable $r_i.pos_1$ describes a position located in the positive axis of the other robot r_{1-i}.*

Proof. The value of the variable $r_i.pos_1$ is assigned for the first time in line $1.a$ when r_i is still in state S_1. At this time, according to Lemma 6.2.3, r_{1-i} is necessarily in state S_1 or S_2 (Otherwise, this would contradict Lemma 6.2.3 since we would have a time at which a robot (r_{1-i}) is in a state S_j with $j \geq 3$ concurrently with another robot (r_i) that is in state S_1). This means that the variable $r_i.pos_1$ describes

a position held by robot r_{1-i} while it was in state S_1 or S_2. But according to the algorithm, when a robot is in state S_1 or S_2, it is still located in a position of its positive x-axis (or the origin). Hence, the variable $r_i.pos_1$ describes a position located in the positive x-axis of robot r_{1-i}, which proves the lemma. □

Lemma 6.2.5. *Eventually, both robots reach state S_3.*

Proof. According to Lemmas 6.2.2 and 6.2.3, there is a time at which some robot, say r_i, reaches S_3 and the other one (r_{1-i}) is at a state S_j with $j \geq 2$. If $j \geq 3$ then the lemma holds and we are done. So we assume in what follows that r_{1-i} is at S_2 and we prove that it eventually reaches S_3. Assume for the sake of contradiction that this is not the case, that is, r_{1-i} remains always stuck in S_2. This implies, according to Lemma 6.2.3, that r_i remains also stuck in state S_3. When r_i is in state S_3, it keeps executing the line 3.b of the code until it reaches a position located in the negative part of its x-axis. Denote by τ_1 the first time at which r_i reaches its negative x-axis. Each time r_i is activated after τ_1, it executes the line 3.c of the code corresponding to a null movement and it never moves from its current position (located in the negative x-axis). This is because we assumed that r_i remains stuck in S_3 forever. By fairness, there is a time $\tau_2 \geq \tau_1$ at which r_{1-i} is activated. This time, the condition of line 2.a does not hold for robot r_{1-i} because the position returned by $observe(i)$ is located in the negative x-axis of r_i and is different from $r_{1-i}.pos_1$ that is located in the positive part of the x-axis of r_1 (as stated in Lemma 6.2.4). Hence, r_{1-i} executes the part 2.$b.*$ of the code and changes its status to S_3. □

Proof of the Validity property.

Lemma 6.2.6. *Algorithm 8 satisfies the **validity** property of the Line* RoboCast *Problem for two robots.*

Proof. Eventually both robots reach state S_3 according to Lemma 6.2.5. Each robot r_i in state S_3 has necessarily executed the blocks 1.$*$ and 2.$b.*$ of the algorithm and thus delivered the line defined by the positions $r_i.pos_1$ and $r_i.pos_2$. Now we prove that this line is well defined and that it does correspond to l_{1-i}, the line sent by r_{1-i}. $r_i.pos_1$ and $r_i.pos_2$ are well defined since they are assigned a value in lines 1.a and 1.b.1 respectively before r_i delivers the line. The assigned values correspond to two positions of r_{1-i}. Moreover, by the condition of line 2.b. we have that these two positions are distinct. It remains to prove that they belong to l_{1-i}.

The values of variables $r_i.pos_1$ and $r_i.pos_2$ are assigned when r_i is in state S_1 and S_2 respectively. Hence, according to Lemma 6.2.3, when $r_i.pos_1$ and $r_i.pos_2$

are defined, r_{1-i} did not yet reach S_4 and moved only through its x-axis. This means that $r_i.pos_1$ and $r_i.pos_2$ correspond to two distinct positions of the x-axis of r_{1-i}. Hence, r_i delivered l_{1-i}. Since both robots eventually reach S_3, both lines l_i and l_{1-i} are eventually delivered. □

Proof of the Termination property. Now we prove that the algorithm actually terminates. Before terminating, each robot r_i must be sure that its peer r_{1-i} has received its sent line, that is, r_{1-i} has reached the state S_3. As already explained, r_i can infer the transition of r_{1-i} to S_3 by detecting a change of its direction of movement. Upon this, r_i can go on to state S_4 and terminates safely. In the following two lemmas, we prove that at least one robot reaches S_4. To do this, we first prove in Lemma 6.2.7 that at least one robot, say r_{1-i}, is observed by its peer in two distinct positions located in the positive part of its x-axis. Later, when r_{1-i} moves to its negative x-axis and r_i observes it there, r_i learns that r_{1-i} changed its direction of movement that allows the transition of r_i to state S_4. This is proved in Lemma 6.2.8.

Lemma 6.2.7. *For at least one robot, say r_i, the two variables $r_i.pos_1$ and $r_i.pos_2$ describe two positions located in the positive x-axis of r_{1-i} and such that $r_i.pos_2 > r_i.pos_1$ with respect to the local coordinate system of r_{1-i}.*

Proof. Let r_i be the first robot to enter state S_3. The other robot (r_{1-i}) is in state S_2 in accordance with Lemma 6.2.3. Hence, r_{1-i} moved only through the positive direction of its x-axis, so the variables $r_i.pos_1$ and $r_i.pos_2$ correspond to two different positions in the positive x-axis of robot r_{1-i} or in its origin. But since $r_i.pos_2$ was observed after $r_i.pos_1$ and r_{1-i} moves in the positive direction of its x-axis, then $r_i.pos_2 > r_i.pos_1$ with respect to the local coordinate system of r_{1-i}. □

Lemma 6.2.8. *Eventually, at least one robot reaches state S_4.*

Proof. We assume towards contradiction that no robot ever reach S_4. But according to Lemma 6.2.5, both robots eventually reach S_3. Hence we consider a configuration in which both robots are in S_3 and we derive a contradiction by proving that at least one of them does reach S_4. Let r_i be the robot induced by Lemma 6.2.7. The variables $r_i.pos_1$ and $r_i.pos_2$ of r_i correspond to two different positions occupied by r_{1-i} while it was on the positive part of its x-axis. By assumption, r_{1-i} eventually reaches state S_3. At the end of this cycle, r_{1-i} is either located in a position of its negative x-axis or it keeps executing lines $3.b.*$ each time it is activated until it reaches such a position, let's call it p. The next cycles it is activated, r_{1-i} executes the line $3.c$ of the code because we assumed that r_{1-i} never reaches S_4. It results that r_{1-i} never quits p. Hence, r_{1-i} is guaranteed to be eventually observed

by r_i in a position that is smaller than $r_i.pos_2$ with respect to the local coordinate system of r_{1-i}. At this point, the condition of line 3.4 becomes true for robot r_i, which executes the block of the code labeled by 3.4.$*$ and sets is state to S_4. □

Lemma 6.2.9. *Eventually, both robots reach state S_4.*

Proof. According to Lemma 6.2.8 at least one robot, say r_i, eventually reaches S_4. When r_i reaches S_4, r_{1-i} is in a state S_j with $j \geq 3$ according to Lemma 6.2.3. If $j = 4$ the lemma holds trivially, so we consider in the following a configuration in which r_{1-i} is in state S_3 and we prove that it eventually joins r_i in state S_4. The variables $r_{1-i}.pos_1$ and $r_{1-i}.pos_2$ of r_{1-i} describe two distinct positions located in the x-axis of robot r_i. Let p_i describes the position of r_i at the end of the cycle in which it reaches S_4. Once in state S_4, r_i moves towards the point $myIntersect$ each time it is activated until it reaches it (lines 3.a.2 and 4.a of the code). $myIntersect$ is the point located at the intersection of l_i and $nextl_i$ and its distance from p_i is finite. Since r_i is guaranteed to move a minimal distance of δ_i at each cycle in which it is activated, it reaches $myIntersect$ after a finite number of cycles. The next cycle, r_i chooses a destination located outside l_i (4.b.2) and moves towards it before finishing the algorithm. Let τ_i be the time of the end of the Move phase of this cycle and let q_i be the position occupied by r_i at τ_i. $q_i \notin l_i$ means that $q_i \notin line(r_{1-i}.pos_1, r_{1-i}.pos_2)$. It follows that $r_{1-i}.pos_2 \notin line(r_{1-i}.pos_1, q_i)$. By fairness, there is a time $\tau > \tau_i$ at which r_{1-i} is activated again, and at which it observes r_i in the position q_i. But we showed that q_i is such that $r_{1-i}.pos_2 \notin line(r_{1-i}.pos_1, q_i)$. Hence the condition of line 3.a is true for robot r_{1-i} in τ and it reaches state S_4 in this cycle. □

Lemma 6.2.10. *Algorithm 8 satisfies the **termination** property of the Line* RoboCast *Problem for two robots.*

Proof. Eventually, both robots reach S_4 as proved by Lemma 6.2.9. Let r_i be a robot in state S_4 and let p_i be its position at the end of the cycle in which it reaches S_4, Let d_i be the distance between p_i and $myIntersect_i$. Since the scheduler is fair and a robot is allowed to move in each cycle a minimal distance of σ_i before it can be stopped by the scheduler, it follows that r_i is guaranteed to cover the distance d_i and to reach $myIntersect$ after at most d_i/σ_i cycles. The next cycle, r_i moves outside l_i and terminates. □

Theorem 6.2.1. *Algorithm 8 solves the Line* RoboCast *Problem for two robots in non-oblivious* NTOM *systems.*

Proof. Follows directly from Lemmas 6.2.6 and 6.2.10. □

6.2.2 Composing RoboCast

Line RoboCast primitive is usually used as a building block for achieving more complex tasks. For example, the RoboCast of the local coordinate system requires the transmission of four successive lines representing respectively the abscissa, the ordinate, the value of the unit measure and a forth line to determine the positive direction of axes. In stigmergic communication a robot has to transmit at least a line for each binary information it wants to send. In all these examples, the transmitted lines are dependent one of each other and therefore their successive transmission can be accelerated by directly exploiting this dependence. Indeed, the knowledge of a unique point (instead of two) is sufficient for the receiver to infer the sent line. In the following we propose modifications of the Line RoboCast primitive in order to exploit contextual information that are encoded in a set of predicates that will be detailed in the sequel.

In the case of the local coordinate system, the additional information the transmission can exploit is the fact that the abscissa is perpendicular to the ordinate. Once the abscissa is transmitted, it suffices for a robot to simply send a single position of its ordinate, say $pos1$. The other robots can then calculate the ordinate by finding the line that passes through $pos1$, which is perpendicular to the previously received abscissa. In the modified version of the Line RoboCast algorithm the predicate $isPerpendicular$ encodes this condition.

For the case of stigmergy, a robot transmits a binary information by robocasting a line whose angle to the abscissa encodes this information. The lines transmitted successively by a single robot are not perpendicular to each others. However, all these lines pass through the origin of the coordinate system of the sending robot. In this case, it suffices to transmit only one position located on this line as long as it is distinct from the origin. We say in this case that the line satisfies the predicate $passThrOrigin$.

A second change we propose relates to the asynchrony of the algorithm. In fact, even if robots execute in unison, they are not guaranteed to finish the execution of $LineRbcast1$ at the same time (by reaching S_4). A robot r_i can begin transmitting its k-th line l_i when its peer r_{1-i} is still located in its $(k-1)$-th line $ancientl_{1-i}$ that r_i has already received. r_i should ignore the positions transmitted by r_{1-i} until it leaves $ancientl_{1-i}$ for a new line. It follows that to make the module composable, the old line that the peer has already received from its peer should be supplied as an argument ($ancientl_{1-i}$) to the function. Thus, it will not consider the positions occupied by r_{1-i} until the latter leaves $ancientl_{1-i}$.

In the following, we present the code of the new Line RoboCast function that we denote by $LineRbcast2$.

6.2.3 RoboCast of the Local Coordinate System

To robocast their two axes (abscissa and ordinate), robots call LineRbcast1 to robocast the abscissa, then LineRbcast2 to robocast the ordinate. The parame-

Algorithm 9 Line RoboCast **LineRbcast2** for two robots: Algorithm for robot r_i.

Inputs:
l_i : the line to robocast
$nextl_i$: the next line to robocast after l_i
$precedentl_{1-i}$: the line robocast precedently by r_{1-i}
$predicate$: a predicate on the output l_{1-i}, for example $isPerpendicular$ and $passThrOrigin$.

Outputs:
l_{1-i} : the line robocast by r_{1-i}

Variables:
$state$: initially S_1
pos_1: initially $\perp$
$destination, myIntersect, peerIntersect$: initially $\perp$

Actions:
1. State [S_2]: *%r_i starts robocasting its line l_i%*

a. **if** ($observe(1-i) \in precedentl_{1-i}$) **then** $destination \leftarrow observe(i)$
b. **else**
1. $pos3 \leftarrow observe(1-i)$
2. $l_{1-i} \leftarrow$ the line that passes through $pos3$ and satisfies $predicate$.
3. Deliver (l_{1-i})
4. $peerIntersect \leftarrow$ intersection between l_{1-i} and $precedentl_{1-i}$
5. $destination \leftarrow (0,-1)_i$
6. $state \leftarrow S_3$ **endif**
c. Move to destination

2. State [S_3]: *%r_i knows the line robocast by robot r_{1-i}%*

a. **if** ($pos3$ is not inside the line segment $[peerIntersect, observe(1-i)]$) **then**
1. $state \leftarrow S_4$
2. $myIntersect \leftarrow intersection(l_i, nextl_i)$
3. $destination \leftarrow myIntersect$
b. **else if** ($observe(i) \geq (0,0)_i$) **then** $destination \leftarrow (0,-1)_i$
c. **else** $destination \leftarrow observe(i)$ **endif endif**
d. Move to destination

3. State [S_4]: similar to state S_4 of the $lineRbcast1$ function.

ter $\neq myOrdinate$ of $LineRbcast2$ stands for the next line to be robocast and it can be set to any line different from $myOrdinate$. The next line to robocast ($unitLine$) is a line whose angle with the x-axis encodes the unit of measure. This angle will be determined during the execution $LineRbcast2$.

1. $peerAbscissa \leftarrow LineRbcast1(myAbscissa, myOrdinate)$
2. $peerOrdinate \leftarrow$
$LineRbcast2(myOrdinate, \neq myOrdinate, peerAbscissa, isPerpendicular)$

After executing the above code, each robot knows the two axes of its peer coordinate system but not their positive directions neither their unit of measure. To robocast the unit of measure we use a technique similar to that used by [34]. The idea is simple: each robot measures the distance d_i between its origin and the peer's origin in terms of its local coordinate system. To announce the value of d_i to its peer, each robot robocast via LineRbcast2 a line, $unitLine$, which passes through its origin and whose angle with its abscissa is equal to $f(d_i)$ where for $x > 0, f(x) = (1/2x) \times 90°$ is a monotonically increasing function with range $(0°, 90°)$. The receiving robot r_{1-i} can then infer d_i from $f(d_i)$ and compute the unit measure of r_i that is equal to d_{1-i}/d_i. The choice of $(0°, 90°)$ as a range for $f(x)$ (instead of $(0°, 360°)$) is motivated by the fact that the positive directions of the two axes are not yet known to the robots. It is thus impossible to distinguish between an angle α with $\alpha \in (0°, 90°)$ and the angles $\Pi - \alpha$, $-\alpha$, and $\Pi + \alpha$. To overcome the ambiguity and to make $f(x)$ injective, we restrict the range to $(0°, 90°)$. In contrast, Suzuki and Yamashita [34] use a function $f'(x)$ slightly different from ours: $(1/2x) \times 360°$. That is, its range is equal to $(0°, 360°)$. This is because in ATOM, robots can robocast at the same time the two axis and their positive directions, for example by restricting the movement of robots to only the positive part of their axes. Since the positive directions of the two axes are known, $unitLine$ can be an oriented line whose angle $f'(x)$ can take any value in $(0°, 360°)$ without any possible ambiguity.

Positive directions of axes Once the two axes are known, determining their positive directions amounts to selecting the upper right quarter of the coordinate system that is positive for both x and y. Since the line used to robocast the unit of distance passes through two quarters (the upper right and the lower left), it remains to choose among these two travelled quarters which corresponds to the upper right one. To do this, each robot robocast just after the line encoding the unit distance another line that is perpendicular to it such that their intersection lays inside the upper right quarter.

6.3 RoboCasting the Local Coordinate System: n-Robots Networks

In this section, we describe the RoboCast Algorithm for the general case of n robots. Then we give its formal proof of correctness.

The generalization of the solution to the case of $n > 2$ robots has to use an additional mechanism to allow robots to "recognize" other robots and distinguish them from each others despite anonymity. Let us consider the case of three robots r_1, r_2, r_3. When r_1 looks the second time, r_2 and r_3 could have moved (or be moving), each according to its local coordinate system and unit measure. At this point, even with memory of past observations, r_1 may be not able to distinguish between r_2 and r_3 in their new positions given the fact that robots are anonymous. Moreover, r_2 and r_3 could even switch places and appear not to have moved. Hence, the implementation of the primitive $observe(i)$ is not trivial. For this, we use the collision avoidance techniques presented in the next section to instruct each robot to move only in the vicinity of its initial position. This way, other robots are able to recognize it by using its past positions. The technical details of this mechanism are given at the end of the next section.

We now present the details of the generalization.

Description of the Algorithm The Line RoboCast algorithm for the general case of n processes is a simple generalization of the algorithm for two robots. The code of this algorithm is in Algorithm 10. It consists in the following steps: The first time a robot r_i is activated in state S_1, it simply records the positions of all other robots in the array $pos_1[]$. Then, it moves towards the point $(1,0)$ of its local coordinate system and goes to state S_2. When r_i is in state S_2, each time it observes some robot r_j in a position different from the one recorded in $pos_1[j]$, it stores it in $pos_2[j]$. At this point, r_i can infer the line sent by r_j that passes through both $pos_1[j]$ and $pos_2[j]$. Hence, r_i delivers $line(r_i, r_j)$, which corresponds to l_j. r_i does not move from its current position until it assigns a value to all the cells of $pos_2[]$ (apart from the one associated with itself, which is meaningless). That is, until it delivers all the lines sent by its peers. Upon this, it transitions to S_3 and heads to the point $(-1,0)$. At state S_3, r_i waits until it observes that all other robots changed the direction of their movement or moved outside their sent line. Then, it moves towards a position located outside its current line l_i. In particular, it goes to a position located in $nextl_i$, the next line it will robocast. Hence r_i first passes by the intersection of l_i and $nextl_i$. Then, it moves outside l_i and terminates the algorithm.

We now prove the correctness of our algorithm by proving that it satisfies the

Algorithm 10 Line RoboCast **LineRbcast1** for n robots: Algorithm for robot r_i.

Variables:
$state$: initially S_1.
$pos_1[1 \ldots n]$: initially $\perp$
$pos_2[1 \ldots n]$: initially $\perp$
$destination, intersection$: initially $\perp$

Actions:
1. State [S_1]: *%Robot r_i starts the algorithm%*

a. **foreach** $1 \leq j \leq n$ **do** $pos_1[j] \leftarrow observe(j)$ **enddo**
b. $destination \leftarrow (1,0)_i$
c. $state \leftarrow S_2$
d. Move to destination

2. State [S_2]: *%r_i knows at least one position of the lines of all other robots%*

a. **if** $\exists j \neq i$ s.t. $(pos_2[j] = \perp)$ and $(pos_1[j] \neq observe(j))$ **then**
1. $pos_2[j] \leftarrow observe(j)$
2. Deliver $(line(pos_1[j], pos_2[j])$ **endif**
b. **if** $\exists j \neq i$ s.t. $(pos_2[j] = \perp)$ **then** $destination \leftarrow observe(i)$
c. **else**
1. $destination \leftarrow (-1,0)_i$
2. $state \leftarrow S_3$ **endif**
d. Move to destination

3. State [S_3]: *%r_i knows the lines of all other robots%*

a. **if** $\forall j \neq i$ $pos_2[j]$ is outside the line segment $[pos_1[j], observe(j)]$ **then**
1. $intersection \leftarrow l_i \cap nextl_i$
2. $destination \leftarrow intersection$
3. $state \leftarrow S_4$
b. **else if** $(observe(i) \geq (0,0)_i)$ **then** $destination \leftarrow (-1,0)$
c. **else** $destination \leftarrow observe(i)$ **endif endif**
d. Move to destination

4. State [S_4]: *%r_i knows that all robots have learned its line l_i%*
a. **if** $(observe(i) \neq intersection)$ **then** $destination \leftarrow intersection$
b. **else**
1. r_i rotates its coordinate system such that its x-axis and the origin match with
$nextl_i$ and $intersection$ respectively.
2. $destination \leftarrow (1,0)_i$
3. return **endif**
c. Move to destination

validity and termination property of the RoboCast Problem specification. The general idea of the proof is similar to that of the two robots algorithms even if it is a little more involved.

Proof of the Validity property

Lemma 6.3.1. *Eventually, all robots reach state S_2.*

Proof. Similar to the proof of Lemma 6.2.1. □

Lemma 6.3.2. *Eventually, at least one robot reaches state S_3.*

Proof. Let $\mathbb{R} = \{r_1, r_2, \ldots, r_n\}$ be a set of n robots executing Algorithm 10. We assume towards contradiction that neither of them ever reach S_3. But according to Lemma 6.3.1, all robots eventually reach state S_2. Thus we proceed in the following way: we consider a configuration in which all robots are in S_2 and we prove that at least one of them eventually reaches S_3 that leads us to a contradiction. Consider for each robot $r_i \in \mathbb{R}$ the cycle in which it reaches the state S_2, and define τ_i and τ'_i to be respectively the time of the end of the Look and the Move phases of this cycle. Let τ_k be equal to $min\{\tau_1, \tau_2, \ldots, \tau_n\}$ and let r_k be the corresponding robot. That is, at τ_k, robot r_k finishes to execute a Look phase and at the end of this cycle it reaches state S_2. This means that for robot r_k, the array $r_k.pos_1[]$ corresponds to the configuration of the network at time τ_k.

Between $\tau_i \geq \tau_k$ and τ'_i each robot executes complete Compute and Move phases. The movement performed in this phase cannot be null because robots move from the point $(0,0)$ towards the point $(1,0)$ of their local coordinate system (line $1.b$ of the code). Moreover, the scheduler cannot stop a robot before it reaches the point $(\delta_i, 0)$. Hence, the position of each robot r_i at τ'_i is different from its position at τ_i. But the position of r_i at τ_i is equal to its position at τ_k. Thus, the position of each robot at τ'_i is different from its position at τ_k that is stored in $r_k.pos_1[i]$. We have by assumption that no robot ever reaches S_3. So each time a robot r_i is activated after τ'_i, it keeps executing the lines $2.b$ and $2.d$ of the code and never moves from its current position reached at τ'_i. Define τ_{end} to be equal to $max\{\tau'_1, \ldots, \tau'_n\}$. It follows that at $\forall \tau \geq \tau_{end}$, the position of each robot r_i at τ is different from $r_k.pos_1[i]$. But by fairness, there is a time $\tau_a \geq \tau_{end}$ at which r_k is activated again. At this cycle, r_k observes that each robot r_i is located in a position different from $r_k.pos_1[i]$. Consequently, if there exists a robot r_i such that $r_k.pos_2[i]$ was equal to $\perp$ before this cycle, then r_k assigns the current observed position of r_i to $r_k.pos_2[i]$. This implies that the condition of line $2.b$ is now false for r_k. Hence r_k executes the else block of the condition and reaches state S_3. This is the required contradiction that proves the lemma. □

Lemma 6.3.3. *For each robot i, for each robot j, if $r_i.pos_1[j] \neq \perp$, then $pos_1[j]$ describes a position that is necessarily located in the positive x-axis of robot j.*

Proof. The proof follows the same lines as that of Lemma 6.2.4. □

Lemma 6.3.4. *If at some time robot r_i is in state S_j and robot r'_i is in state S_k then $|j-k| \leq 1$.*

Proof. The lemma can be proved by generalising the proof of Lemma 6.2.3 to the case of n robots. We divide the analysis into two subcases:

- If robot r_i is in state S_3 and robot r'_i is in state S_j then $j \geq 2$.

  proof: If r_i is in state S_3, this means that it observed all other robots in at least two distinct positions. This means that all other robots started a Move phase, which implies that they all finished a complete Compute phase in which they executed the lines $1.a \ldots 1.c$ of the code and reached S_2.

- If robot r_i is in state S_4 and robot r'_i is in state S_j then $j \geq 3$.

  proof: For a robot to reach S_4, it must detect a change of direction by *all* other robots in the network that is captured by the condition of line $3.a$. We prove that this condition cannot be true unless all robots have reached S_3 and no robot in the network is still in state S_2. Indeed, robots in state S_1 move in the positive direction of their x-axis and those in state S_2 does not move. So, a robot cannot change its direction before reaching state S_3. This change of direction is reflected by the choice of point $(-1,0)$ as a destination in line $2.c.1$ of the code before the transition to state S_3 in line $2.c.2$.

□

Corollary 6.3.1. *If at some time τ, $\exists i, j$ such that robots r_i and r_j are respectively in state S_k and S_{k+1} at τ with $k \in \{1,2,3\}$, then all the robots of the network are either in state S_k or S_{k+1} at τ.*

Proof. Since r_i is in state S_k, no robot in the network can be in a state S_l with $l \geq k+2$ according to Lemma 6.3.4. Similarly, the fact that r_j is in state S_{k+1} implies that no robot in the network can be in a state S_l with $l \leq k-1$. By the conjunction of the two facts, we obtain that all robots are either in state S_k or S_{k+1}. □

The following lemma proves the fact that if at time τ_a, some robots of the network are in state S_2 and others are in state S_3, then at least one robot that is in state S_2 at τ_a eventually reaches S_3.

Lemma 6.3.5. *Let $G_2(\tau), G_3(\tau)$ be the groups of robots that are respectively in state S_2 and S_3 at time τ. If at some time τ_a $\|G_2(\tau_a)\| > 0$ and $\|G_3(\tau_a)\| > 0$, then there exists a time $\tau \geq \tau_a$ at which $\|G_3(\tau)\| \geq \|G_3(\tau_a)\| + 1$.*

Proof. Since by assumption $\|G_2(\tau_a)\| > 0$ and $\|G_3(\tau_a)\| > 0$, it follows from Corollary 6.3.1 that $\mathbb{R} = G_2(\tau_a) \cup G_3(\tau_a)$. We assume toward contradiction that $\forall \tau \geq \tau_a$ $G_2(\tau) = G_2(\tau_a) = G_2$, that is, no robot that is in S_2 at τ_a ever reach S_3. But by assumption we have $\|G_2\| > 0$. Hence $\forall \tau > \tau_a$ $\|G_2(\tau)\| = \|G_2\| > 0$. This implies, in accordance with Lemma 6.3.4, that no robot of the network can reach S_4 after τ_a. Consequently, $\forall \tau' \geq t$ $G_3(\tau') = G_3(\tau) = G_3 = \mathbb{R} \setminus G_2$.

- As discussed above, we have by assumption that all robots have reached S_2 at τ_a. This means that all robots have executed the line $1.a$ of the code at τ_a. Consequently, at τ_a, $\forall r_i \in \mathbb{R}$, $\forall r_j \in \mathbb{R}$ with $j \neq i$, $r_i.pos_1[j] \neq \perp$. Moreover, according to Lemma 6.3.3 all these positions stored in the arrays $pos_1[]$ describe positions located in the positive x-axis of the corresponding robots.

- By assumption we have that $\forall \tau > t_a$ $\|G_3(\tau)\| = \|G_3\|$. This means that no robot in G_3 ever reach S_4. Hence robots of G_3 never execute the block $3.a.*$ of the code and they keep executing the line $3.b$ each time they are activated until they reach a position in their negative x-axes. Then, once a robot of $G3$ arrives to the negative part of its x-axis, it keeps executing the line $3.c$ of the code each time it is activated. As we showed above, the positions stored in the different arrays $pos_1[]$ are different from $\perp$ and correspond to points located in the positive x-axes of the corresponding robots. Since robots of G_3 eventually get to positions in their negative x-axes and stay there, they eventually get observed by each robot in the network in a position different from the one that is stored in its local variable $pos_1[]$ that corresponds to a positive x-axis position. Formally, there is a time $\tau_v > t$ at which $\forall r_i \in G_3, \forall r_j \in \mathbb{R}$ with $r_j \neq r_i$ $r_j.pos_2[i] \neq \perp$.

- Let $r_1, \ldots, r_m$ be the robots of G_2. Consider for each robot $r_i \in G_2$ the cycle in which it reaches the state S_2, and define τ_i and τ'_i to be respectively the time of the end of the Look and the Move phase and of this cycle. Let τ_k be equal to $min\{\tau_1, \tau_2, \ldots, \tau_m\}$ and let $r_k \in G_2$ be the corresponding robot. That is, at τ_k, robot r_k finishes to execute a Look phase and at the end of this cycle it reaches state S_2. This means that for robot r_k, $r_k.pos_1[]$ describes the configuration of the network at time τ_k. Following the lines of the proof of Lemma 6.3.2 we obtain that there exist a time $\tau_u > t$ at which $\forall r_j \in G_2 \setminus \{r_k\}, r_k.pos_2[j] \neq \perp$.

Now, let $\tau_x = max\{\tau_v, \tau_u\}$. From the discussion above it results that at time τ_x, for robot $r_k \in G_2$ it holds that $\forall r_j \in G_3, r_k.pos_2[j] \neq \perp$ and $\forall r_j \in G_2 \setminus \{r_k\}, r_k.pos_2[j] \neq \perp$. Hence, at τ_x, $\forall j \in \mathbb{R} \setminus \{r_k\}, r_k.pos_2[j] \neq \perp$. This means that the condition of line 2.a is false for r_k at τ_x, so r_k executes the *else* block of this condition when activated after τ_x and reaches state S_3, which contradicts the assumption that $\forall \tau > \tau_a \, \|G_2(\tau)\| = \|G_2\|$. □

Lemma 6.3.6. *Eventually, all robots of the network reach state S_3*

Proof. Follows from Lemmas 6.3.1, 6.3.2 and 6.3.5. □

Lemma 6.3.7. *Algorithm 10 satisfies the **validity** property.*

Proof. The idea of the proof is similar to Lemma 6.2.6. According to Lemma 6.3.6, all robots eventually reach state S_3. Each robot that reach state S_3 has necessarily executed the block 1.$*$ and the line 2.c of Algorithm 10. Hence, this robot has its two arrays $pos_1[]$ and $pos_2[]$ well defined and according to the way the elements of $pos_2[]$ are defined (refer to line 2.a of the code), we conclude that $\forall 1 \leq j \leq n$, $pos_2[j] \neq pos_1[j]$. Moreover, since robots move only through their x-axes, $\forall j, pos_2[j]$ and $pos_1[j]$ correspond to two positions of the x-axis of robot j. Hence, each robot in state S_3 can infer the x-axes of its peers from $pos_1[]$ and $pos_2[]$, which proves the lemma. □

Proof of the Termination property

Lemma 6.3.8. *Eventually, at least one robot reaches state S_4*

Proof. We assume for the sake of contradiction that no robot ever reach S_4. However, according to Lemma 6.3.6, all robots eventually reach state S_3. Hence we consider a configuration in which all robots are in state S_3 and we prove that at least one of them eventually reaches S_4 that leads us to a contradiction. The idea of the proof is similar to that of Lemma 6.3.2: we consider the first robot r_k that executes a Look phase of a cycle leading it from S_2 to S_3. Let τ_k be the time of the end of this Look phase. Clearly, $\forall r_i \in \mathbb{R} \setminus \{r_k\}$, $r_k.pos_1[i]$ and $r_k.pos_2[i]$ describe two positions of r_i located in its positive x-axis. This is because these two positions were observed by r_k before r_i reaches S_3 and changes its direction of movement towards its negative x-axis. Moreover, $r_k.pos_2[i] > r_k.pos_1[i]$ with respect to the local coordinate system of r_i since r_i was observed in $r_k.pos_1[i]$ and then in $r_k.pos_2[i]$ while it was moving along the positive direction of its x-axis. The claim can be proved formally as in Lemma 6.2.7. After τ_k, all other robots of the network perform a transition from S_2 to S_3. Then, they head towards the negative part of their local x-axes (lines 2.c.1 and 3.b of the code) and stay there (line 3.c)

since they cannot reach S_4 by assumption. Each robot r_i that reaches the negative part of its x-axis is located in a position p_i such that $r_k.pos_2[i]$ is outside the line segment $[r_k.pos_2[i],\ p_i]$. Hence the condition of line 3.a eventually becomes true for robot r_k, and it reaches S_4 after executing the block 3.$a.*$ of the code. This is the required contradiction. □

Lemma 6.3.9. *Eventually, all robots of the network reach S_4.*

Proof. The proof is similar to that of Lemma 6.2.9. The intuition behind it is as follows: we proved in Lemma 6.3.8 that at least one robot, say r_i, eventually reaches S_4. After reaching S_4, and after a finite number of executed cycles, r_i quits l_i (line 4.b.2). When they observe r_i outside l_i, the other robots transition to state S_4. □

Lemma 6.3.10. *Algorithm 10 satisfies the **termination** property.*

Proof. The proof is similar to that of Lemma 6.2.10 □

Theorem 6.3.1. *Algorithm 8 solves the Line* RoboCast *Problem for n robots in un-oblivious* NTOM *systems.*

Proof. Follows directly from Lemmas 6.3.7 and 6.3.10. □

Movement Complexity Now we show that the total number of robot moves in the coordinate system RoboCast is upper bounded. For the sake of presentation, we assume for now that the scheduler does not interrupt robots execution before they reach their planned destination. Each robot is initially located at the origin of its local coordinate system. To robocast each axis, a robot must visit two distinct positions: one located in the positive part of this axis and the other one located in its negative part. For example, to robocast its x-axis, a robot has first to move from its origin to the position $(1.0)_i$, then from $(1.0)_i$ to the $(-1,0)_i$. Then, before initiating a RoboCast for the other axis, the robot must first return back to its origin. Hence, at most 3 movements are needed to robocast each axis. This implies that to robocast the whole local coordinate system, at most 12 movements have to be performed by a particular robot.

In the general NTOM model, the scheduler is allowed to stop robots before they reach their destination, as long as a minimal distance of δ_i has been traversed. In this case, the number of necessary movements is equal to at most $8*(1+1/\delta_i)$. This worst case is obtained when a robot is *not* stopped by the scheduler when moving from its origin towards another position (thus letting it go the farthest possible), but stopped whenever possible when returning back from this (far) position to the origin.

This contrasts with [34] and [22] where the number of positions visited by each robot to robocast a line is unbounded (but finite). This is due to the fact that in both approaches, robots are required to make a non null movement whenever activated until they know that their line has been received. Managing an arbitrary large number of movements in a restricted space to prevent collisions yields severe requirements in [22]: either robots are allowed to perform infinitely small movements (and such movements can be seen by other robots with infinite precision), or the scheduler is restricted in its choices for activating robots (no robot can be activated more than k times, for a given k, between any two activations of another robot) and yields to a setting that is not fully asynchronous. Our solution does not require any such hypothesis.

6.4 Avoinding Collisions

In this section we enhance the algorithms proposed in Section 6.2 with the collision-free feature. In this section we propose novel techniques for collision avoidance that cope with the system asynchrony.

Our solution is based on the same principle of locality as the Voronoi Diagram based schemes. However, acceptable moves for a robot use a different geometric area. This area is defined for each robot r_i as a local *zone of movement* and is denoted by ZoM_i. We require that each robot r_i moves only inside ZoM_i. The intersection of different ZoM_i must remain empty at all times to ensure collision avoidance. We now present three possible definitions for the zone of movement: ZoM_i^1, ZoM_i^2 and ZoM_i^3. All three ensure collision avoidance in `NTOM`, but only the third one can be computed in a model where robots do not know the initial position of their peers.

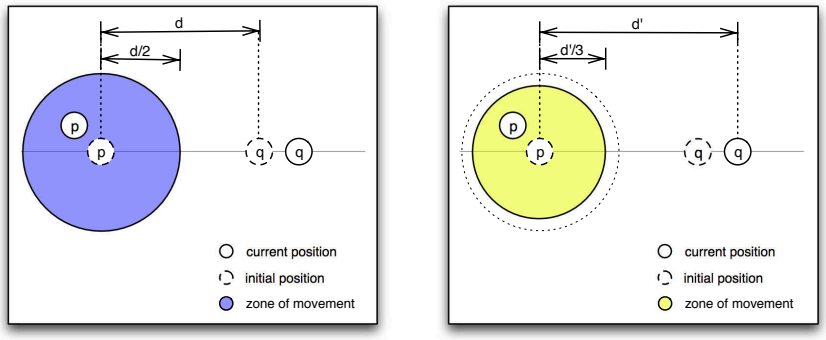
Let $P(\tau) = \{p_1(\tau), p_2(\tau) \ldots, p_n(\tau)\}$ be the configuration of the network at time τ, such that $p_i(\tau)$ denotes the position of robot r_i at time τ expressed in a global coordinate system. This global coordinate system is unknown to individual robots and is only used to ease the presentation and the proofs. Note that $P(\tau_0)$ describes the initial configuration of the network.

Definition 17. *(**Voronoi Diagram**) The Voronoi diagram of a set of points $P = \{p_1, p_2, \ldots, p_n\}$ is a subdivision of the plane into n cells, one for each point in P. The cells have the property that a point q belongs to the Voronoi cell of point p_i iff for any other point $p_j \in P$, $|q, p_i| < |q, p_j|$. In particular, the strict inequality means that points located on the boundary of the Voronoi diagram do not belong to any Voronoi cell.*

Definition 18. *(ZoM_i^1) Let $DV(\tau_0)$ be the Voronoi diagram of the initial configuration $P(\tau_0)$. For each robot r_i, the zone of movement of r_i at time τ, $ZoM_i^1(\tau)$, is the Voronoi cell of point $p_i(\tau_0)$ in $DV(\tau_0)$.*

Definition 19. *(ZoM_i^2) For each robot r_i, define the distance $d_i = min\{|p_i(\tau_0), p_j(\tau_0)| \text{ with } r_j \neq r_i\}$. The zone of movement of r_i at time τ, $ZoM_i^2(\tau)$, is the circle centered in $p_i(0)$ and whose diameter is equal to $d_i/2$. A point q belongs to $ZoM_i^2(\tau)$ iff $|q, p_i(\tau_0)| < d_i/2$.*

Definition 20. *(ZoM_i^3) For each robot r_i, define the distance $d_i(\tau) = min\{|p_i(\tau_0), p_j(\tau)| \text{ with } r_j \neq r_i\}$ at time τ. The zone of r_i at time τ, $ZoM_i^3(\tau)$, is the circle centered in $p_i(\tau_0)$ and whose diameter is equal to $d_i(\tau)/3$. A point q belongs to $ZoM_i^3(\tau)$ iff $|q, p_i(\tau_0)| < d_i(\tau)/3$.*



(a) ZoM_p^2 (b) ZoM_p^3

Figure 6.1: Example zones of movement: The network is formed of two robots: p and q. d is the distance between the initial positions of p and q (dashed circles), d' is the distance between the initial position of p and the current position of q. The diameter of ZoM_p^2 (blue) is $d/2$ and that of ZoM_p^3 (yellow) is $d'/3$.

Note that ZoM^1 and ZoM^2 are defined using information about the initial configuration $P(\tau_0)$, and thus cannot be used with the hypotheses of Algorithm 9. In contrast, robot r_i only needs to know its *own* initial position and the *current* positions of other robots to compute ZoM_i^3. As there is no need for r_i to know the *initial* positions of other robots, ZoM_i^3 can be used with Algorithm 9. It remains to prove that ZoM_i^3 guarantees collision avoidance. We first prove that ZoM_i^1 does, which is almost trivial because its definition does not depend on time. Then, it suffices to prove that $ZoM_i^3 \subseteq ZoM_i^2 \subseteq ZoM_i^1$. Besides helping us in the proof, ZoM_i^2

can be interesting in its own as a cheap collision avoidance scheme in the ATOM model, as computing a cycle of radius half the distance to the nearest neighbor is much easier that computing a full blown Voronoi diagram.

Lemma 6.4.1. *If $\forall t$, for each robot r_i, the destination point computed by r_i at τ remains inside $ZoM_i^1(\tau)$, then collisions are avoided.*

Proof. By definition of Voronoi diagram, different Voronoi cells do not overlap. Moreover, for a given i, ZoM_i^1 is static and does not change over time. Hence, $\forall i, j \in \Pi$, $\forall t, \tau'$, $ZoM_i^1(\tau) \cap ZoM_j^1(\tau') = \emptyset$. □

Clearly, $ZoM_i^2 \subseteq ZoM_i^1$, which means that ZoM_i^2 ensures also collision avoidance.

Lemma 6.4.2. *If $\forall t$, for each robot r_i, the destination point computed by r_i at τ always remains inside $ZoM_i^2(\tau)$, then collisions are avoided.*

Lemma 6.4.3. $\forall t, ZoM_i^3(\tau) \subseteq ZoM_i^2(\tau)$.

Proof. Fix some robot r_i and let r_j be the closest robot from r_i at time τ_0. Let d_0 denote the initial distance between r_i and r_j, that is, $d_0 = |p_i(\tau_0), p_j(\tau_0)|$. We assume that all robots move only inside their ZoM_i^3 computed as explained in Definition 20. Let $\tau_1 \geq \tau_0$ be the first time at which a robot in $\{r_i, r_j\}$, say *e.g.* r_i, finishes a Look phase after τ_0. The destination computed by r_i in this cycle is located inside $ZoM_i^3(\tau_1)$, which is a circle centered at $p_i(\tau_0)$ and whose diameter is $\leq d_0/3$. Hence, the destination computed by r_i is distant from $p_i(0)$ by at most $d_0/3$. Let $\tau_2 \geq \tau_1$ be the first time after τ_1 at which a robot, say r_j, finishes a Look phase. Between τ_1 and τ_2, r_i may have finished its Move phase or not. In any case, the observed configuration by r_j at τ_2 is such that r_i is distant from $p_j(\tau_0)$ by at most $d_0 + d_0/3$. This implies that $ZoM_j^3(\tau_2)$ has a diameter of at most $(d_0 + d_0/3)/3$, which implies that the destination point computed by r_j in this cycle is distant from $p_j(\tau_0)$ by at most $d_0 + d_0/3 + d_0/9$. Repeating the argument, we get that $\forall t$, $ZoM_i^3(\tau)$ has a diameter $\leq \sum_{i=1}^{\infty} d_0/3^i$. Reducing the formula, we obtain that $ZoM_i^3(\tau)$ is always $\leq d_0/2$, which implies that $ZoM_i^3(\tau) \subseteq ZoM_i^2(\tau)$. □

Ensuring Collision-freedom in Line RoboCast **Algorithms** To make LineRbcast1 and LineRbcast2 collision-free, it is expected that any destination computed by a robot r_i at τ be located within its $ZoM_i^3(\tau)$. The computation of destinations is modified as follows: Let $dest_i(\tau)$ be the destination computed by a robot r_i at time τ. Based on $dest_i(\tau)$, r_i computes a new destination $dest_i'(\tau)$ that ensures collision avoidance. $dest_i'(\tau)$ can be set to any point located in

$[p_i(\tau_0), dest_i(\tau)] \cap ZoM_i^3(\tau)$. For example, we can take $dest'_i(\tau)$ to be equal to the point located in the line segment $[p_i(\tau_0), dest_i(\tau)]$ and distant from $p_i(\tau_0)$ by a distance of $d_i(\tau)/2$ with $d_i(\tau)$ computed as explained in Definition 20.

This modification of the destination computation method does not impact algorithms correctness since it does not depend on the exact value of computed destinations, but on the relationship between the successive positions occupied by each robot. The algorithms remain correct as long as robots keep the capability to freely change their direction of movement and to move in both the positive and the negative part of each such direction. This capability is not altered by the collision avoidance scheme since the origin of the coordinate system of each robot - corresponding to its original position - is strictly included in its zone of movement, be it defined by ZoM^1, ZoM^2 or ZoM^3.

Generalisation of the Collision-avoidance Protocols to n Robots As explained at the end of Section 6.2, the generalisation of our algorithms to the case of n robots has to deal with the issue of distinguishing robots from each others despite their anonymity. The solution we use is to instruct each robot to move in the close neighbourhood of its original position. Thus, other robots can recognize it by comparing its current position with past ones. For this solution to work, it is necessary that each robot always remains the closest one to all the positions it has previously occupied. Formally speaking, we define the zone of movement ZoM^4 in a similar way as ZoM^3 except that the diameter is this time equal to $d_i(\tau)/6$ (vs. $d_i(\tau)/3$). We now show that ZoM^4 provides the required properties. Let r_i and r_j be an arbitrary pair of robots and Let d_{ij} denotes the distance between their initial positions. It can easily shown, using the same arguments as the proof of Lemma 6.4.3, that:

1. Neither of the two robots moves away from its initial position by a distance greater than $d_{ij}/4$. This implies that each robot remains always at a distance strictly smaller than $d_{ij}/2$ from all the positions it has previously held.

2. The distance between r_i (resp. r_j) and all the positions held by r_j (r_i) is strictly greater than $d_{ij}/2$.

Hence, r_i can never be closer than r_j to a position that was occupied by r_j, and vice versa. This implies that it is always possible to recognize a robot by associating it with the position which is closest to it in some previously observed configuration.

6.5 Applications

In this section we present two possible application of our RoboCast scheme, namely the gathering and stygmergy problems.

6.5.1 Asynchronous Deterministic 2-Gathering

Given a set of n robots with arbitrary initial locations and no agreement on a global coordinate system, n-Gathering requires that all robots eventually reach the same unknown beforehand location. n-Gathering was already solved when $n > 2$ in both ATOM [34] and NTOM [15] oblivious models. The problem is impossible to solve for $n = 2$ even in ATOM, except if robots are endowed with persistent memory [34]. In this section we present an algorithm that uses our RoboCast primitive to solve 2-Gathering in the non-oblivious NTOM model.

A first "naive" solution is for each robot to robocast its abscissa and ordinate axes and to meet the other robot at the midpoint m of their initial positions. RoboCasting the two axes is done using our Line RoboCast function described above in conjunction with the ZoM^3−based collision avoidance scheme.

A second possible solution is to refine Algorithm $\psi_{f-point(2)}$ of [34, 35] by using our Line RoboCast function to "send" lines instead of the one used by the authors. The idea of this algorithm is that each robot that is activated for the first time translates and rotates its coordinate system such that the other robot is on its positive y-axis, and then it robocasts its (new) x-axis to the other robot using our Line RoboCast function. In [34], the authors give a method that allows each robot to compute the initial position of one's peer by comparing their two robocast x-axes defined above. Then each robot moves toward the midpoint of their initial positions. Our Line RoboCast routine combined with the above idea achieves gathering in asynchronous systems within a bounded (*vs.* finite in [34]) number of movements of robots and using only two (*vs.* four) variables in their persistent memory.

Theorem 6.5.1. *There is an algorithm for solving deterministic gathering for two robots in non-oblivious asynchronous* NTOM *networks.*

6.5.2 Asynchronous Stigmergy

Stigmergy [22] is the ability of a group of robots that communicate only through vision to exchange binary information. Stigmergy comes to encode bits in the movements of robots. Solving this problem becomes trivial when using our RoboCast primitive. First, robots exchange their local coordinate system as explained in Section 6.2. Then, each robot that has a binary packet to transmit robocasts a line to

its peers whose angle with respect to its abscissa encodes the binary information. Theoretically, as the precision of visual sensors is assumed to be infinite, robots are able to observe the exact angle of this transmitted line, hence the size of exchanged messages may be infinite also. However, in a more realistic environment in which sensor accuracy and calculations have a margin of error, it is wiser to discretize the measuring space. For this, we divide the space around the robot in several sectors such that all the points located in the same sector encode the same binary information (to tolerate errors of coding). For instance, to send binary packets of 8 bits, each sector should have an angle equal to $u = 360^\circ/2^8$. Hence, when a robot moves through a line whose angle with respect to the abscissa is equal to α, the corresponding binary information is equal to $\lfloor \alpha/n \rfloor$. Thus, our solution works in asynchronous networks, uses a bounded number of movements and also allows robots to send binary packets and not only single bits as in [22].

CHAPTER 7

CONCLUSION

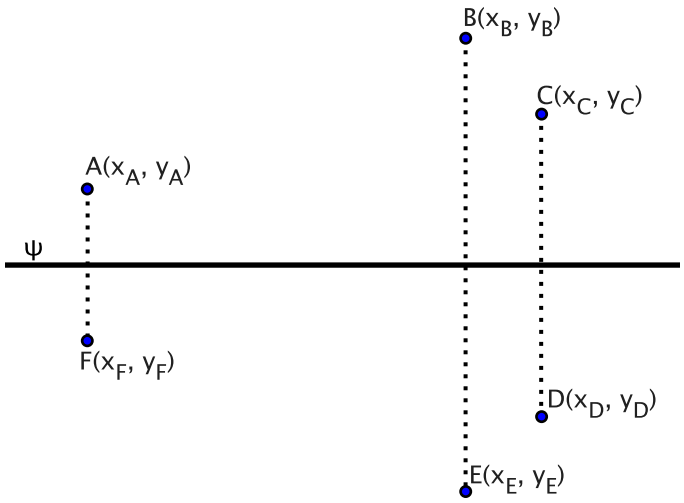


Figure 7.1: Axial Symmetric Configuration

This chapter summarizes our contributions and presents some directions for future research.

In this thesis, we studied several problems in distributed robot networks. The research for an effective protocol for a given problem in robot networks often translates into the quest for a good spacial invariant that allows the design of simple and "natural" algorithms. The most famous example is the Weber point which, when computed, provides a simple solution to the gathering problem : move towards the Weber point of the current configuration. However, the Weber point is known to be impossible to solve for general configurations, and algorithms that compute Weber points are known for only few specific configurations. We proposed in this thesis an algorithm that computes the Weber point for a large class of configurations that exhibit some kind of rotational symmetry around a point. Our method exploits this symmetry to find the Weber point.

We are currently investigating whether it is possible to use similar techniques for configurations that are axial (but not rotational) symmetric. Let us illustrate this with the configuration P of Figure 7.1 that is symmetric with respect to the axis ψ. Since the configuration is not linear, its Weber point WP(P) is unique, hence it lies necessarily on the line ψ. That is,

$$\begin{aligned} WP(P) &= \operatorname*{argmin}_{x\in\psi} \textstyle\sum_{p\in\{A,B,C\}} |x,p| \\ &= \operatorname*{argmin}_{x\in\mathbb{R}} f(x) = \textstyle\sum_{p\in\{A,B,C\}} \sqrt{(x-x_p)^2+y_p^2} \end{aligned}$$

Therefore, WP(P) is the solution of the equation

$$0 = f'(x) = \sum_{p\in\{A,B,C\}} \frac{x-x_p}{\sqrt{(x-x_p)^2+y_p^2}}$$

While this equation can be solved numerically, we could not find a general formula for x.

Using our Weber point computation primitive, we designed a deterministic algorithm that achieves gathering even if an arbitrary number of robots crash. Our protocol assumes that robots share a common notion of handedness. It works roughly as follows. If the configuration is assymetric, robots elect a robot as leader and its location is used as the gathering point. Otherwise, if the configuration exhibits some symmetry, robots move towards the Weber point which we know how to compute in this case. The problem rises when the configuration is axial symmetric. In this case, we can neither elect a leader nor know how to compute the Weber point. Adding the assumption of the common notion of clockwise direction resolves this difficulty by allowing pairs of robots that are symmetric with respect

to the axis to distinguish between themselves. For example, in Figure 7.1, robot A can be distinguished from D using the common notion of *clockwise* orientation: A is the clockwise neighbor of D in the convex hull of P. It would be interesting to investigate whether it is possible to solve the problem while removing this assumption of common chirality, maybe by finding a way to compute Weber points for axial symmetric configurations.

Then, we studied the convergence problem in uni-dimensional systems in which robots may incur byzantine failures. In this context, we proved several necessary and sufficient conditions for the problem to be solvable in terms of the synchrony of the model and its resilience. An avenue for future research is to generalize these results for multi-dimensional robot networks.

Finally, we presented RoboCast, an all-to-all communication primitive for robot networks, that can be implemented in fully asynchronous and non-atomic networks if robots are endowed with memory. Our scheme has the additional properties of being motion, memory, and computation efficient. For this purpose, we presented a collision avoidance for non-atomic networks that can be used for protocols in which robots remain in the vicinity of their initial positions during the whole protocol execution. A collision-avoidance scheme that could be used with all classes of protocols in non-atomic networks is a challenging issue. Using the RoboCast primitive, a swarm of *failure-free* robots can simulate any algorithm that works in anonymous synchronous message passing systems. The question is still open if we can obtain such a result with faulty (crash, byzantine) robots.

BIBLIOGRAPHY

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36(1):56–82, 2006.

[2] L. Anderegg, M. Cieliebak, and G. Prencipe. The weber point can be found in linear time for points in biangular configuration. Technical Report TR-03-01, Department of Informatics, University of Pisa, 2003.

[3] L. Anderegg, M. Cieliebak, and G. Prencipe. Efficient algorithms for detecting regular point configurations. *Theoretical Computer Science*, pages 23–35, 2005.

[4] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on*, 15(5):818–828, 1999.

[5] Alberto Apostolico, Martin Farach, and Costas S Iliopoulos. Optimal superprimitivity testing for strings. *Information Processing Letters*, 39(1):17–20, 1991.

[6] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(1):177–191, 1988.

[7] Z. Bouzid, S. Das, and S. Tixeuil. Gathering of mobile robots tolerating multiple crash faults. In *ICDCS*, 2013.

[8] Z. Bouzid, S. Dolev, M. Potop-Butucaru, and S. Tixeuil. Robocast: Asynchronous communication in robot networks. *Principles of Distributed Systems*, pages 16–31, 2010.

[9] Z. Bouzid, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Byzantine convergence in robot networks: The price of asynchrony. *Principles of Distributed Systems*, pages 54–70, 2009.

[10] Z. Bouzid, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Byzantine-resilient convergence in oblivious robot networks. In *ICDCN*, pages 275–280, 2009.

[11] Z. Bouzid, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Optimal byzantine resilient convergence in asynchronous robots networks. In *SSS*, pages 165–179, 2009.

[12] Z. Bouzid, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Optimal byzantine-resilient convergence in uni-dimensional robot networks. *Theor. Comput. Sci.*, 411(34-36), 2010.

[13] Z. Bouzid and A. Lamani. Robot networks with homonyms: the case of patterns formation. *Stabilization, Safety, and Security of Distributed Systems*, pages 92–107, 2011.

[14] R. Chandrasekaran and A. Tamir. Algebraic optimization: the fermat-weber location problem. *Mathematical Programming*, 46(1):219–224, 1990.

[15] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the robots gathering problem. *Automata, Languages and Programming*, pages 192–192, 2003.

[16] R. Cohen and D. Peleg. Robot convergence via center-of-gravity algorithms. *Proc. of the 11th Int. Colloquium on Structural Information and Communication Complexity*, pages 79–88, 2004.

[17] R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.

[18] R. Cohen and D. Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. In B. Durand and W. Thomas, editors, *23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS'06)*, volume 3884 of *LNCS*, pages 549–560, Marseille, France, February 2006. Springer.

[19] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6):481–499, 2009.

[20] S. Das, P. Flocchini, N. Santoro, and M. Yamashita. On the computational power of oblivious robots: forming a series of geometric patterns. In *PODC*, pages 267–276. ACM, 2010.

[21] X. Defago, M. Gradinariu, S. Messika, and P.R. Parvedy. Fault-tolerant and self-stabilizing mobile robots gathering. *DISC06, the 20th International Conference on Distributed Computing. LNCS*, 3274:46–60, 2006.

[22] Y. Dieudonné, S. Dolev, F. Petit, and M. Segal. Deaf, dumb, and chatting asynchronous robots. In *OPODIS*, pages 71–85, 2009.

[23] Y. Dieudonné and F. Petit. Self-stabilizing deterministic gathering. *Algorithmic Aspects of Wireless Sensor Networks*, pages 230–241, 2009.

[24] D. Dolev, N.A. Lynch, S.S. Pinter, E.W. Stark, and W.E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[25] P. Flocchini, D. Ilcinkas, A. Pelc, and N. Santoro. Remembering without memory: Tree exploration by asynchronous oblivious robots. *Theoretical Computer Science*, 411(14-15):1583–1598, 2010.

[26] P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3(2):1–185, 2012.

[27] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous mobile robots with limited visibility. *Theoretical Computer Science*, 337:147–168, 2005.

[28] Taisuke Izumi, Samia Souissi, Yoshiaki Katayama, Nobuhiro Inuzuka, Xavier Défago, Koichi Wada, and Masafumi Yamashita. The gathering problem for two oblivious robots with unreliable compasses. *SIAM J. Comput.*, 41(1):26–46, 2012.

[29] B. Katreniak. Biangular circle formation by asynchronous mobile robots. *Structural Information and Communication Complexity*, pages 185–199, 2005.

[30] B. Katreniak. Convergence with limited visibility by asynchronous mobile robots. *Structural Information and Communication Complexity*, pages 125–137, 2011.

[31] David G Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM journal on computing*, 15(1):287–299, 1986.

[32] G. Prencipe. On the feasibility of gathering by autonomous mobile robots. In A. Pelc and M. Raynal, editors, *Proc. Structural Information and Communication Complexity, 12th Intl Coll., SIROCCO 2005*, volume 3499 of *LNCS*, pages 246–261, Mont Saint-Michel, France, May 2005. Springer.

[33] Giuseppe Prencipe. Instantaneous actions vs. full asynchronicity : Controlling and coordinating a set of autonomous mobile robots. In *ICTCS*, pages 154–171, 2001.

[34] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.

[35] I. Suzuki and M. Yamashita. Erratum: Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. of Computing*, 36(1):279–280, 2006.

Zohir BOUZID

Modèles et Algorithmes pour les Systèmes Émergents

Résumé

Les réseaux de robots autonomes sont des entités mobiles qui communiquent uniquement à travers leurs mouvements et l'observation de leurs positions respectives. Ils sont anonymes, sans mémoire et sans système de coordonnées global, ni une notion commune de la distance. Nous nous concentrons sur l'étude algorithmique des problèmes de rassemblement et de convergence des robots quand ils sont sujets à des pannes. Notre première contribution est de nature géométrique. Nous fournissons un protocole pour calculer le point Weber d'une grande classe de configurations qui ont une symétrie rotationnelle. Se basant sur cette cette primitive, nous présentons un algorithme qui résout le problème du rassemblement en présence de n'importe quel nombre de pannes franches. Ensuite, nous abordons le problème de convergence quand les robots peuvent subir des pannes byzantines qui sont plus difficiles à traiter que les pannes franches. Nous fournissons plusieurs bornes optimales qui relient le degré de synchronie du système à sa résilience. Enfin, nous étudions les robots qui sont dotées de mémoire et nous montrons que ce modèle est plus fort que le modèle de passage de messages.

Mots clés : réseaux de robots; point Weber; tolérance aux pannes; systèmes répartis; rassemblement; byzantins

Résumé en anglais

Networks of autonomous robots are mobile entities that communicate only through their movements and the observation of their respective positions. They are anonymous, without memory and without a global coordinate system nor a common notion of distance. We focus on the algorithmic study of the problems of gathering and convergence of robots when some of them may be subject to failures. Our first contribution is of geometric nature. We provide a protocol to compute the Weber point of a large class of rotational symmetric configurations. Based on this primitive, we present an algorithm that solves the gathering problem in presence of any number of crash failures. Then, we address the convergence problem when robots may incur byzantine failures which are harder to handle than crash failures. We provide several tight bounds relating the degree of synchronicity of the system to its resiliency. Finally, we study robots that are endowed with memory and we show that this model is stronger than the message passing model.

Keywords : robot networks; Weber point ; fault tolerance ; distributed systems ; gathering ; byzantine