Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: TD-4/02

Distributed Coordination of a Set of Autonomous Mobile Robots

Giuseppe Prencipe

April 2002

Thesis supervisors:

Prof. Linda Pagli, Prof. Nicola Santoro

Abstract

The distributed coordination and control of a set of autonomous mobile robots is a problem widely studied in a variety of fields, such as engineering, artificial intelligence, artificial life, robotics. Generally, in this areas the problem is studied mostly from an empirical point of view. In contrast, in this thesis we aim to understand the fundamental algorithmic limitations on what a set of autonomous mobile robots can or cannot achieve. Therefore, we first present CORDA, a model whose main aim is to describe the possible interactions that can happen in an asynchronous environment populated by autonomous mobile robots. By *robot* we mean a physical device equipped with *motorial capabilities*, that allow it to move on a two dimensional plane, and with *sensorial capabilities*, that allow it to sense the presence of other robots in the plane. Then, we analyze what kind of tasks these units can accomplish, and under which conditions. In particular, we propose the study of a set of problems selected from and motivated by the areas cited above, with the robots operating under several assumptions which enable us to better comprehend the computability and complexity of such problems in a distributed setting.

You cannot control the universe,
but you can control how you react to it.
Anonymous.

He who esteems trifles for themselves is a trifler;
he who esteems them for the conclusions to be drawn from them,
or the advantage to which they can be put, is a philosopher.
Edward Bulwer-Lytton

Acknowledgments

This thesis has been written partly in Canada, at *Carleton University* in Ottawa, and partly in Italy, at the *Dipartimento di Informatica* of Pisa. Therefore, I need to thank many people from both continents, and since this is the only part of the thesis in which I do not have to follow any particular rules, some lines of these acknowledgments will be in my native language - Italian.

First, my Italian supervisor, Linda Pagli, who always supported and encouraged me, giving me the freedom to work on the topic I liked best. Then, my Canadian supervisor, Nicola Santoro, who first suggested the topic of this thesis to me. He and Paola Flocchini always animated me and were fantastic people to work with, and I will never forget all the hours spent in their kitchen talking about "stupid" robots. Very important to the final writing of the thesis were also all the people that I met during my brief but important stay at ETH Institute of Zürich. In particular, Peter Widmayer and all his family; Mark, Zsuzsanna, and Konrad, with whom I had several *energetic* discussions on the model discussed in the thesis. A particular thank you goes also to the referees of this thesis, Peter Widmayer and Masafumi Yamashita, and to Maurizio Bonuccelli and Pierpaolo Degano for all their helpful comments.

My stay in Ottawa was essential for my Ph.D. studies, and also very important during that period were all the people I met, who helped me in making me feel like I was at home there. In particular, Gwen, my first landlady - I have never known so energetic and lively a woman, and Faouzi. Then all the people I met at St. Anthony's Italian Parish in Ottawa: the "big" choir and all the girls from "Coro Arcobaleno", Father Marcel, Rose Marie, Valeria, Nick, Maddalena and Pasquale, Luciano and Claire, who several times hosted me and my family in their home during my brief periods there, and the Petosa's, who welcomed me like a son, and that helped me in not missing (almost) all the Italian cuisine. A special thanks goes to Olga, who shared with me two years of her life and all my moments of sadness and happiness. All the people from Carleton University: Frank, Peter, Luis and all the guys from the *soccer under the snow* team, and all the administrative staff - in particular Nicole and Maureen. Finally Tatiana and Brian, my two Canadian piano

teachers, who kept alive in me my great passion for the piano, and Giovanni, who generously helped me develop my new passion for the saxophone.

Even if I spent a lot of time in Canada, several people helped me from Italy. *Un ringraziemento particolare va alla mia famiglia, che ha sempre sostenuto le mie scelte fino in fondo dandomi la forza di proseguire anche nei momenti di maggiore difficoltà.* A very important person was "papà" Vincenzo: a colleague, my housemate, a real friend; he always had the right word at the right time, helping me to smile in grey moments (will I ever forget the paper we wrote while I was in Canada and he was in Australia?). An especially warm thank you goes to him and Enzo who have proven to be true friends over the past few months, through their selfless generosity, their useful advise, and their willingness to go out of their way to support me during a particularly difficult time of my life.

All my friends from Pisa, that have been particularly close to me during these last months: my "fratellozzo" Andrea, Stefano for proofreading part of the thesis, Mimma for all her support, Massimilano and Angela, Laura and Antonio, Michele, Rossella, and Angela for all her funny jokes. Finally, the colleagues and friends of my office: Vincenzo, Valentina, Nadia and Paolo; they resisted the temptation to occupy my desk and my computer during my absence, and defended them with strength and valor.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

A robot may not act unless its actions are subject to the Laws of Robotics.

The Meta-Law – Isaac Asimov's Laws of Robotics (1940–1985)

Abstract

The study of the coordination and control of a set of agents that can freely and independently move on a plane in order to achieve a common goal is a widely studied topic, especially in robotics and engineering. In the majority of these studies, algorithmic aspects were somehow implicitly an issue, but clearly not a major concern.

In contrast, in this thesis we approach the problem from a **computational** point of view. In this chapter, we give a general and informal overview of the framework and of the problem under study. Moreover, a brief description of the related work done in this area and of the main contributions of this thesis are presented as well. The chapter ends with the outline of the thesis.

1.1 Framework

The goal of this thesis is to study a *distributed system* whose *entities* are devices equipped with both *motorial capabilities*, that allow them to *freely* move on a two dimensional plane, and *sensorial capabilities*, that allow them to sense the presence of fellow entities in the plane where they operate. Because of the particular nature of these entities, in the following we will refer to them as *agents*, *robots*, *vehicles*, or *units*. The aim of the entities is to solve some assigned problem all together, and in finite time. For instance, they can be asked to form a circle, to gather in a point of the plane, or to move an heavy load from one place of the plane to another.

The study of a set of mobile agents that can independently move on a plane in order to accomplish some task in cooperation, has gained much attention in the last decade. One of the main reason for this interest, is that in current robotics research, both from engineering and behavioral viewpoints, the trend has been to move away from the design and deployment of *few problem-specific robots* which are *powerful* enough to solve the problem at hand; the main reason is that such robots are clearly *complex*, *difficult* to program and to construct, and, thus, usually expensive. The interest has shifted instead towards the design and use of a *large number of generic robots* which have *minimal* physical capabilities, exceedingly *simple* behavior but are *easy* (and, thus, relatively inexpensive) to design, construct and deploy.

In fact, such a system is preferable to one made up of just one large and powerful unit for many reasons: low costs in changing faulty units; ability to solve tasks otherwise unsolvable by a single unit [32, 66] (e.g., robots that have to move big objects [64]); advantages that can arise from a distributed and parallel solution to the given problems, such as a faster computation [6] (e.g., robots that are asked to clean a room [52]); for fault tolerance considerations; the decreased cost through simpler individual unit design. In particular, these robots are only capable of sensing their immediate surrounding, performing simple computations on the sensed data, and moving towards the computed destination; their behavior is a simple cycle of sensing, computing, moving and being inactive (e.g., see [13, 53, 57, 62]). In spite of their limitations, the robots should be able, together, of performing rather complex tasks.

One major question that arises is: how is it possible to coordinate well these group of mobile robots, so that they can accomplish together what they are asked to do? Another basic set of questions refers to determining *minimal robot capabilities*; that is, how "simple" the robots can be to perform the required task [35]? In computational terms, this question is to identify the factors that influence solvability of a given problem (the task) by a set of mobile robots. The research is still focusing on basic tasks such as *gathering*, *flocking*, *pattern formation*, *scattering*, etc. [9, 36, 57, 78, 52].

In the majority of the studies, the problem is approached from an experimental point of view: algorithms are designed using mainly heuristics, and then tested either by means of computer simulations or on real robots. No proof of correctness and finiteness of the proposed solutions is given. Nor analysis of the relationship between the problem to be solved, the capabilities of the units employed, and their knowledge of the environment where they interact is generally done. In other words, given a task, which conditions must be satisfied so that they can solve it? What is it necessary they must be able to do in order to complete it? What is the knowledge of the environment they must share in order to successfully reach their goal?

In this thesis, we aim to provide a theoretical framework to model such simple units, hence to formally analyze their behavior. In particular, we want to gain a better understanding of which set of "minimal capabilities" and knowledge of the world the agents must have in order to complete the given task correctly and in finite time.

1.2 The Problem

We study the problem of coordinating a set of autonomous mobile agents in a totally distributed environment. Each agent is assumed to occupy a point of the two dimensional plane where they operate, and is modeled as a unit equipped with sensors that allow it to sense the world, and the other agents around it; it then processes the data it sensed: in particular, it computes a destination point in the plane, according to its algorithm; it finally moves to the point it just computed. The *life* of each agent is thus constituted by a sequence of cycles of waiting, sensing, computing, and moving. Furthermore, the agents are

1. *homogeneous* in terms of hardware and software, that is they all have the same algorithm for solving a given task;
2. *autonomous*, in the sense that they do not depend on some central coordination mechanism;
3. *asynchronous*, in that there is no global clock;
4. *mobile*, since they are allowed to move on the plane;
5. *anonymous*, meaning that an agent does not have an identity that it can use during the computation;
6. *memoryless*, that is they cannot remember the past, and what an agent computes in a particular moment of the execution of an algorithm depends only on the positions of the others at that moment;
7. finally, they cannot *communicate* explicitly with each other: communication is based exclusively on the position of the agents in the plane, and thus on how the environment in which the agents operate changes during the computation[1].

Each agent has locally a coordinate system, an unit measure, and the algorithm to execute. In general the agents do not agree on the orientation of the axes of their

[1]In biology, this type of communication is called *stigmergy* [12, 41, 49]. A brief description of it will be given in the next chapter.

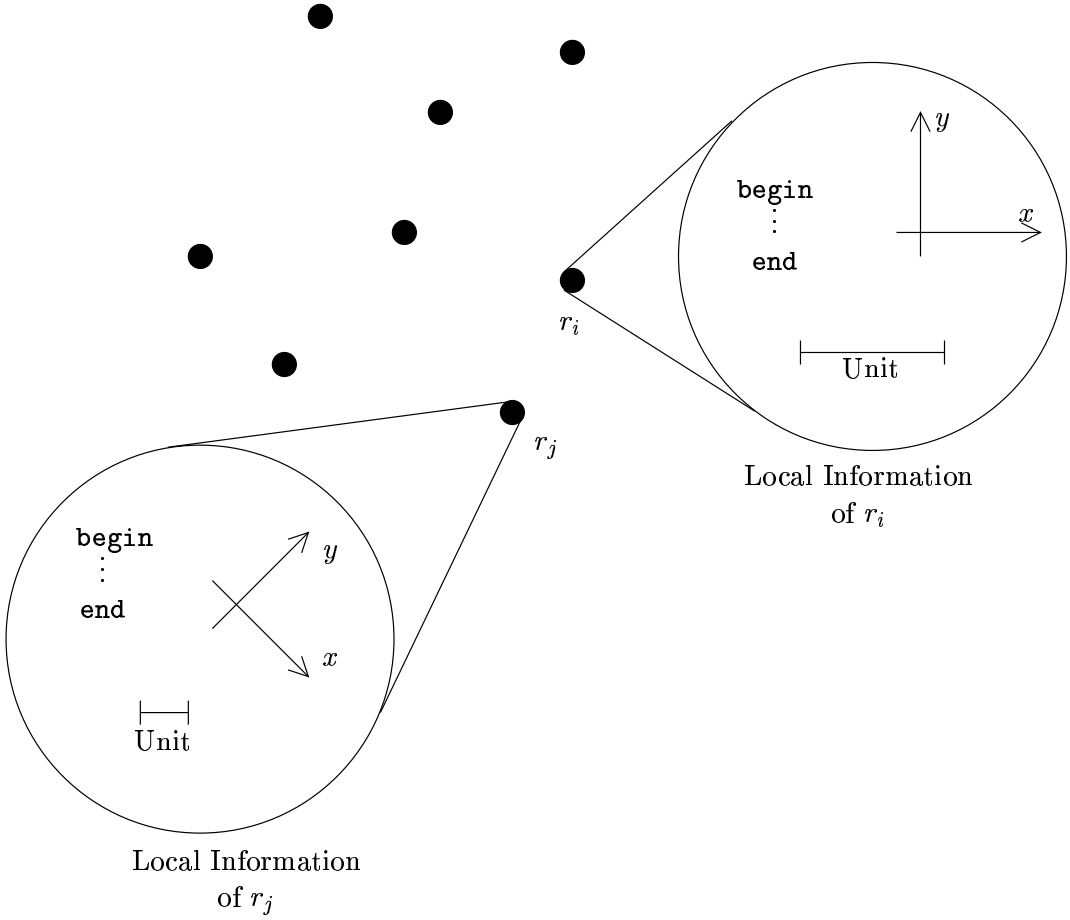


Figure 1.1: The local information that each unit has is constituted by a coordinate system, an unit measure, and the algorithm to execute.

local coordinate systems, nor on the unit measure. They do, however, share the same algorithm (see Figure 1.1).

These assumptions make our agents simple and rather "weak" in many aspects, hence they could seem unnecessarily exaggerated and extreme, especially if we think of the agents as robots and we consider the state of the art in engineering technology. But in this study we approach the problem differently: from a **computational** point of view. We therefore intentionally make these extreme assumptions so we can better understand what kind of basic functionality a set of agents must have in order to accomplish a given task in a distributed fashion. By assuming the "weakest" agents, we can analyze in greater detail the strengths and weaknesses of distributed control; furthermore, this approach allows us to highlight the set of agents' capabilities that are necessary to accomplish a certain task. We study several problems in this model, describing the level of capabilities and knowledge of the world the agents have to

share in order to complete the given task correctly and in finite time. This approach is rare in literature, and the only work that shares similar ideas with ours is that of Suzuki *et al.*, that will be briefly outlined in Section 1.3 and analyzed in more detail in Chapter 3.

1.3 Related Work

The problem of controlling a set of autonomous, mobile robots in a distributed fashion has been studied extensively, but almost exclusively from an engineering and from an artificial intelligence point of view. In a number of remarkable studies (on social interaction leading to group behavior [57], on selfish behavior of cooperative agents in animal societies [66], on primitive animal behavior in pattern formation [8], to pick just a few), algorithmic aspects were somehow implicitly an issue, but clearly not a major concern, let alone the focus, of the study.

In particular, several studies approached this problem by employing simple units programmed with basic behaviors, and studying what kind of complex behaviors could emerge when they operate in group. An example of this approach is that of Matarić, who first programmed basic behaviors such as *homing*, *collision avoidance*, *aggregation*; then, she compounded these behaviors, and realized that more complex behavior emerged from the group, such as *flocking* and *foraging*. This supported her theory that complex group behavior result from basic and elementary interactions among the individuals of the group (a more detailed survey of the literature on these topics will be presented in the next chapter).

Our work starts from these ideas to support the choice of modeling the agents as "weak" and "simple" units. Our main concern is, however, to identify the algorithmic limitations of what autonomous, mobile robots can or cannot do. An investigation with this flavor has been undertaken within the AI community by Durfee [35], who argues in favor of limiting the knowledge that an intelligent agent must possess in order to be able to coordinate its behavior with others. The work of Suzuki and Yamashita [76, 77, 78], however, is closest to our study (and, with this focus, a rarity in the mobile robots literature); it gives a nice and systematic account on the algorithmics of pattern formation for robots, under several assumptions on the power of the individual robot. The model that we use differs from that of [76, 77, 78] in the fact that our robots are as weak as possible in every single aspect of their behavior. The reason is that we want to identify the role of the robots' common knowledge of the world for performing a task. In contrast with [76, 77, 78], we do not assume that on a move, we know ahead of time the limited, but nonzero distance that a robot travels. We do not assume that the distance that a robot may

travel in one step is so short that no other robot can see it *while* it is moving.

The most radical deviation from previous models may, however, be our assumption of *asynchronicity within one step*. In contrast, [76, 77, 78] assume the *atomicity of a step*: a robot moves immediately after it has observed the current situation, with all awake robots moving at the same clock tick (some robots may be asleep). This difference influences the power of the system of robots so drastically that in general, algorithms that make use of atomicity within one step do not work in our model; in particular, this is true for the work in [76, 77, 78]. The main differences between our model and the one proposed by Suzuki *et al.* will be discussed in more detail in Chapter 3.

1.4 Contributions

In this thesis we describe a theoretical framework to model a set of autonomous, asynchronous, oblivious robots that can freely move on a plane in order to reach a common goal. The approach we adopt to study the coordination and control of these units differs from that usually adopted in robotics. In fact, as already pointed out, the solutions that can be found in literature are generally based on heuristics, not giving any evidence of correctness or finiteness, nor describing the proposed solution and the power of the employed units; that is, what they must know or share in order to successfully accomplish the common task.

Our model differs also from that of Suzuki *at al.*, the only other work, to our knowledge, that shares the same perspective as ours. In fact, the model they present differs in several aspects from ours, especially in the way the asynchronicity of the units is modeled.

In conclusion, the main contribution of this thesis is to analyze the problem of coordinating and controlling a set of totally independent mobile units from a different perspective: from a **computational** point of view. This allows us to gain a better understanding of the power of the agents, in relation to the problem they are asked to solve. In fact, the model we propose allows us to study and analyze some of the problems that are generally experimented in robotics, pointing out the key factors that play a crucial role in their solvability. In particular, the following problems are analyzed.

Arbitrary Pattern Formation, assuming that the sensors the agents have allow them to sense the presence of all the other agents in the system (we say they have *Unlimited Visibility*). In this problem, the agents have in input the same pattern, described as a set of coordinates with respect to their local coordinate system. Their task is to *form* the given pattern; that is, they have to place

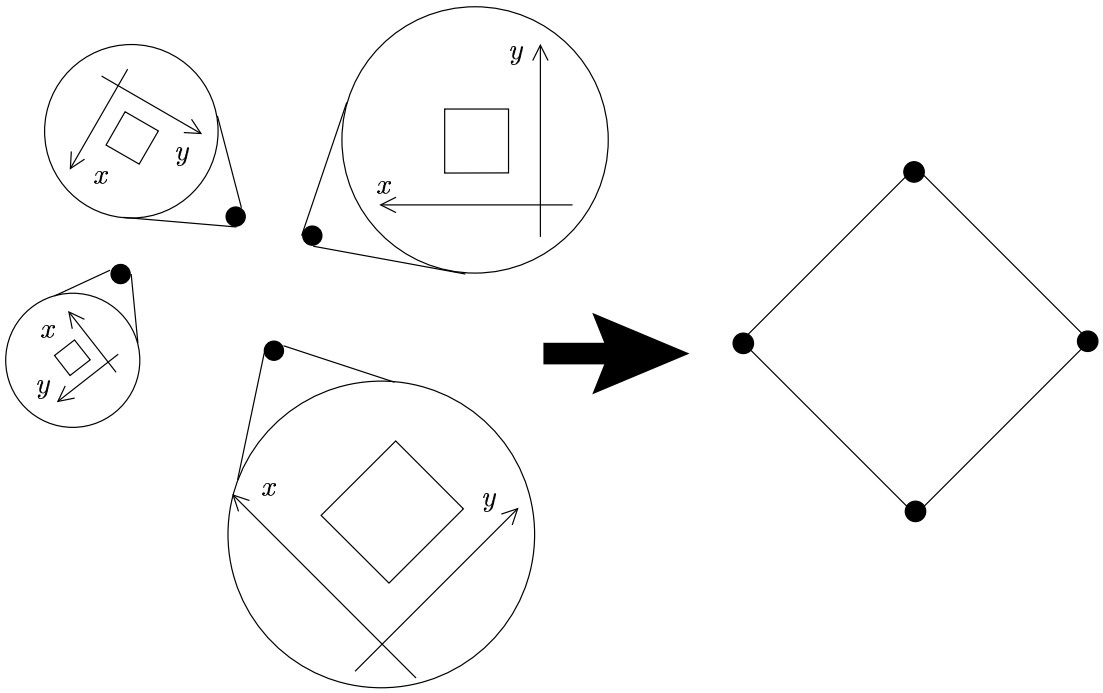


Figure 1.2: The Pattern Formation Problem. In this example, the robots are required to form a square.

themselves in the plane in such a way that their final positions form a pattern *similar* to the one in input (see Figure 1.2).

In order to do so, they have to find an agreement on the unit measure and on the orientation of the coordinate system, so that they can agree on where and how to form the pattern in the plane. We study under which conditions the agents are able to form the given pattern; in particular, we investigate the impact of the different level of common knowledge of the orientation of the axes on the computability of the problem.

Gathering. The agents are required to meet in a point, not determined in advance (Figure 1.3). We first study this problem in the case when the robots can only sense a limited portion of the plane (we say they have *Limited Visibility*). We provide an algorithm that requires the robots to agree on the orientation and directions of both the axes of their local coordinate systems (e.g., they must have a compass).

Then, we analyze what happens if the robots have unlimited visibility. In this case, this problem has a very simple solution: the robots can meet in the *Weber Point* [81]. The Weber point WP of $r_1, \ldots, r_n$ points in the plane

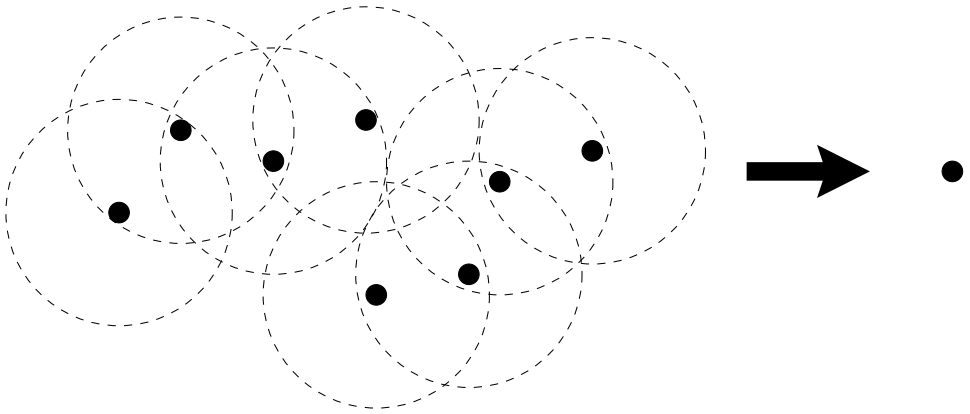


Figure 1.3: The Gathering Problem. The dotted circles represent the radius of visibility of each robot, in the limited visibility case.

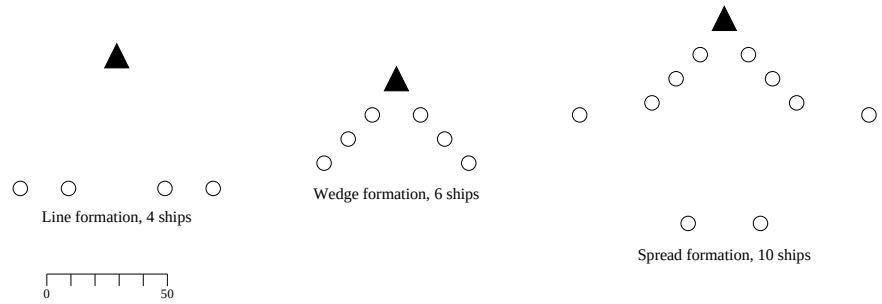


Figure 1.4: Fleet formations used in the simulations. The triangle represents the leader, and the circles the followers.

is defined as follows: $WP = \arg\min_p \Sigma_i dist(p, r_i)$, where $dist(p, r_i)$ is the Euclidean distance between a point p and r_i. An interesting property of WP is that, moving any r_i on the line between the initial position of r_i and WP, the Weber point does not change. Hence, the robots can simply gather in WP. Unfortunately, it has been also shown that the Weber point is not computable [24]. Therefore, we present a different strategy and provide an algorithm that lets the robots gather in finite time: when the robots can sense all the environment, the problem is solvable even with no common knowledge on the orientation of the coordinate systems.

Flocking, in the *Unlimited Visibility* setting. In this case, there are two different kind of robots: a *leader*, and the *followers*. While the leader follows an arbitrary path, the followers are required to follow it, while keeping a formation they are given as input. Once again, we provide a solution that does not require the robots to have any knowledge on the orientation of the axes of their local coordinate systems. Moreover, we also test our solution by computer

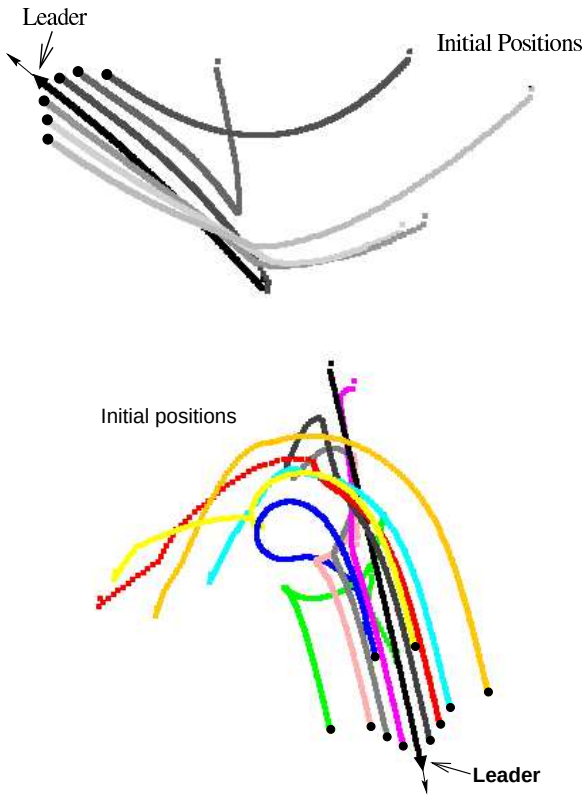


Figure 1.5: Trace of the robots while forming and keeping a wedge shaped formation (above) and the spread formation (below).

simulations. In particular, we test a *line* formation with 4 robots, a *wedge* shaped formation with 6 robots, and a *spread* formation with 10 robots (see Figure 1.4). In Figure 1.5, the paths traced by the robots that are following the leader while keeping the wedge and the spread formation, are reported.

1.5 Thesis Organization

This thesis can be divided in two parts. In the first part, we present the reasons that mainly prompted this study, and we formally introduce the model under study.

In Chapter 2, we give an overview of the most common approaches found in literature to solve similar problems. In particular, studies from robotics and the *emergent behaviors* approach, and from the *self-stabilization* area are presented.

In Chapter 3, we formally introduce the model under study. Moreover, we highlight the main differences with the other model (the only one to our knowledge) that approaches the problem of studying mobile robots from the same angle, giving evidence that the solutions proposed in that model do not work in ours.

In the second part, we present algorithms that solve some of the most common problems generally studied in robotics. In particular, in Chapter 4 we study the *arbitrary pattern formation* problem, where the robots are required to form patterns that are given in input.

In Chapters 5 and 6, the *gathering* problem is analyzed, where the agents are required to gather in a point not fixed in advance. This problem is studied both in the *unlimited visibility* (the agents can sense all the world) and *limited visibility* (the agents can sense only a limited portion of the world) setting.

In Chapter 7, we present an algorithm that lets the robots to form and maintain a formation while following a leader: the *flocking* problem.

Finally, in Chapter 8, some conclusions are drawn, and ideas for possible expansion of this work are proposed.

Chapter 2

Autonomous Mobile Robots

A robot may not injure humanity, or, through inaction, allow humanity to come to harm.

The Zeroth Law Of Robotics – Isaac Asimov

A robot may not injure a human being, or, through inaction, allow a human being to come to harm, unless this would violate a higher-order Law.

The First Law Of Robotics – Isaac Asimov

Abstract

A system consisting of a set of totally distributed robots is generally preferable to one made up of just one powerful robot for several reasons: certain problems can be solved quicker; the ability to perform tasks unsolvable by a single unit; increase in fault tolerance; decrease of design's cost.

The choice of the "weak" features of the robots and of the assumptions briefly introduced in the previous chapter is further motivated by some studies done in other fields besides that of distributed coordination and control of multi-agent teams. In this chapter these studies will be discussed, and we will outline the main approaches that have been adopted in several fields to let the agents in the system efficiently interact in order to reach a common goal.

2.1 Introduction

In this thesis we study a distributed environment inhabited by rather "weak" agents: they act independentely from each other, without any central control to coordinate their actions, and with no global concept of time; they look all identical to an external observer; they do not have any kind of device that allows them to communicate explicitly (e.g., radio frequencies, infrared signals, etc.).

In this chapter we will outline some of the research modeled on sets of totally distributed autonomous agents – research which prompted us to start operating, in our own model, under the basic assumptions delineated above. Among the meanings assigned to the term "agent" is that of "robot". It is, therefore, not surprising that robotics is one of the big fields which concerns itself with extensively studying the problem of controlling a set of autonomous mobile robots in a distributed fashion. Hence, the first part of this chapter is dedicated to a brief overview of the main works done in this area.

Other significant work which has informed and motivated us in our study comes from the research in the area of self-stabilization, to which the second part of this chapter is dedicated.

2.2 Robotics and Behavior-Oriented Systems

In a system consisting of a set of totally distributed agents the goal is generally to exploit the multiplicity of the elements in the system so that the execution of a certain number of predetermined tasks occurs in a coordinated and distributed way. Such a system is preferable to one made up of just one powerful robot for many reasons: the advantages that can arise from a distributed and parallel solution to the given problems [6], such as a faster computation; the ability to perform tasks which are unable to be executed by a single agent [32, 66]; for fault tolerance considerations; the decreased cost through simpler individual robot design.

The way an agent re-acts when it moves in the environment, the regularity in the interaction dynamics between an agent and its environment [73], the control law for reaching and maintaining a particular goal [58], are some of the definitions of the *behavior* of an agent that can be found in the literature. When agents in a set move, they exploit a behavior generically denoted as *collective behavior*. If the agents cooperate to solve the assigned task, the behavior they exploit is called *cooperative behavior* [19]. In the literature there are several definitions for this kind of behavior: "joint collaborative behavior that is directed toward some goal in which there is a common interest or reward" [11], "a form of interaction, usually based on communication" [58], "[joining] together for doing something that creates a progressive result such as increasing performance or saving time" [68].

An agent's behavior can or cannot be explicitly connected to the goals the set of agents wants to achieve. The former approach is typical of classical artificial intelligence (AI), whereas the latter, so called *behavior-based* approach, is a relatively new approach to the AI which emerged in the mid-eighties. An interesting discussion on the main features of the two approaches can be found in [18, 73]. The following

is an overview of the main ideas of the newer, behavior-oriented approach to AI.

2.2.1 Emergent Behavior

This field of studies on collaborative behaviors analyzes *emergent behaviors* arising from the coordination and cooperation of the set of robots in the system, and it is characterized by goals that are not "explicitly programmed in, but result from local interactions between a system's component" [57, 58]. In other words, the functionality of a machine usually arises from its component and it is part of the project of its designer, whereas the emergent functionality (behavior), on the other hand, arises from the interaction of its components which are not directly programmed with a particular function in mind. The only things, therefore, that are programmed in the robots are the *behaviors*: a set of laws that guides the robot to react to environmental stimuli, with the property that there is no explicit goal programmed in. Hence, although the agents are working together from an observer's viewpoint, they are not from the agents' viewpoint. The cooperation and the goals simply *emerge* (externally) as the computation goes on [34]. This feature renders this approach environment-independent, allowing quick reaction to changes in the environment. These kind of systems are also called *self-organizing*, because of their capacity to exploit behavior that are not directly designed.

One of the first studies conducted in this direction is by R. A. Brooks. In [14] he describes an architecture, the *subsumption architecture*, composed by a set of layers, each describing a behavior of the robot associated to increasing level of competence: from the lowest level describing the collision avoidance behavior of the robot, to the last level that uses visual observation to select places in the environment to go to. Brooks has other interesting works in this field [15, 16, 17, 18], where he strongly motivates the behavior-oriented approach to the problem of coordinate and controlling multi-robot teams.

Another noteworthy study in this field, the one which has most motivated the use of the "weak" assumptions of our model, is that of Matarić [58]. In this study she tries to understand which kind of simple interactions (behaviors) are necessary to produce complex group behavior in a fully distributed multi-agent system. The set of agents under study has no *a priori* leader, hence all the agents are homogeneous. There is no explicit one-to-one communication, and all communication is based on changes in the environment that the agent sense (implicit communication). There is no central coordination.

The simplicity of the adopted agents goes against a school of thought in cognitive psychology which believes that social interaction need the presence of a *theory of mind*; that is, agents need to have models of the other agents in the environment

in order to start meaningful interactions [23, 26]. Moreover, the necessity of internal (mental) representation of the world is also claimed as a necessity by classical artificial intelligence [61]. But not all scientists agree on these ideas as they are not supported by studies done both in developmental psychology and ethology. In fact, research on young children has shown that despite the fact they do not have an adult theory of mind, they nevertheless can engage in complex interactions with others [1, 7, 10]. Furthermore, studies of other primates (apes, monkeys) have also produced the same results [23, 61]. Therefore, in order to prove that the presence of a theory of mind is unnecessary in robotics, Matarić chooses a bottom-up approach, using agents with simple capabilities (hence, with no explicit models of each other), with the purpose of studying whether any kind of meaningful interactions are possible.

Another main idea in Matarić work is that "interactions between individual agents need not to be complex to produce complex global consequences" [58]. This idea finds its motivation in sociology and ethology, where in order to understand an individual intelligence, it is observed and analyzed within its social and cultural contest rather than in isolation. Moreover, the culture of a society arises from the local interactions of its components, and many are the examples of complex interaction that emerge from simple local interactions: from the subatomic, to the semantic, to the social.

Based on these ideas, Matarić's aim is to study the kind of complex group behavior that can be achieved when individuals do not have too much power and strong means at their disposal. At first a set of simple interactions between agents is described, then ways for compounding these basic interactions in order to obtain more complex group behavior are successively illustrated. The following is the set of simple interactions that can occur between agents:

Collision Avoidance The ability of an agent to avoid colliding with anything in the world;

Following The ability to follow another agent without colliding;

Dispersion The ability of a group of agents to spread out over an area in such a way that between each pair of agents there is at least a predetermined distance $\delta_{dispersion}$;

Aggregation The ability of a group of agents to gather in such a way that between each pair of agents there is at most a predetermined distance $\delta_{aggregation}$;

Homing The ability of one or a group of agents to reach a goal region or location.

Compounding these simple interaction, it is possible to implement more complex behaviors. Examples of experimented behaviors include *flocking*, that is, the ability of the agents of moving in a flock towards a goal and *foraging*, consisting of finding pucks in the environment, picking them up and delivering them to the home region. The author's more recent work includes studies on robot learning and the use of behavior-based primitives for controlling articulated bodies [58, 59, 60].

Another study that explores these ideas is by Beckers, Holland and Deneubourg, [12]. The authors analyze the possibility to apply common behaviors that can be observed in several natural systems – such as ants, termites, wasps, and bees – within the field of robotics. In particular, the principle they want to exploit is that of *stigmergy*, first named by the French biologist Grassé [49]. Stigmergy refers to the ability of these social insects to produce complex tasks (such as build nests) by locally modifing the environment and without the need of direct communication. Beckers *et al.* experimented this principle using simple robots that operate completely autonomously and independently. The task consists of collecting pucks spread over the environment (a square arena) in a single cluster. The robots are equipped with grippers to collect the pucks. The robots are simple in the sense that they act following only three behaviors, and only one is active at any time: if no sensor is active, go straight until an obstacle or a puck is detected; if an obstacle is detected, the robot avoids it, retaining pucks it has in its grippers (if any); if the gripper pushes three or more pucks, a microswitch is activated that releases the pucks. Although the time of the completion depended on the number of robots involved, the experimental results demonstrate that the robots always completed the assigned task. Therefore, despite the simplicity of the robots and the kind of implicit communication adopted, potentially useful tasks can be performed.

L. E. Parker, [66, 67], extends the behavior-based approach defining an architecture, ALLIANCE, where several "behavior sets" are defined, but with only one set active at any point in time (the others are in a hibernation state). The activation of a set depends on some *motivational* behaviors (a set increases its priority if its corresponding task is not being accomplished) and on *sensory filters* that influences the activities of a behavior set once it has been activated. He uses this adaptive action selection strategy to simulate a set of robots that have to clean a room that is unfamiliar to the robots at the beginning of the task.

T. Balch and C. Arkin studied formation and navigation problems in multi-robot teams. In particular in [5] the problem of specifying the behavior for the navigation of a mobile unit is analyzed, and results of both computer simulation and real experimentation are reported. In [8] and [9] the approach is extended to multi-robot teams that navigate the environment maintaining particular formations: in particular the case of a line, column, diamond and wedge are examined.

David Jung, Gordon Cheng and Alexander Zelinsky conducted some experiments in cooperative cleaning behavior between autonomous mobile robots. They approach the problem from different viewpoints: with implicit (emergent) cooperation, explicit cooperation and implicit communication by passive observation and explicit communication between the robots [52].

Our work arises from emergent behavior studies, in particular that of Matarič, as they motivate the simplicity of the agents under analysis. As mentioned earlier, this simplicity allows us to formally highlight from an algorithmic and computational viewpoint the minimal capabilities the agents must have in order to accomplish basic tasks and produce interesting interactions. Furthermore, it allows us to better understand the limitations of the distributed control in an environment inhabited by mobile agents, hence to formally prove what can not be achieved under the "weakness" assumptions of our model, that will be described later in more detail.

2.2.2 Other Approaches to Robotics

Another kind of approach to the problem of studying multi-robot systems, is that of Fukuda, Nadagawa, Kawauchi and Buss [42, 43, 44, 45, 54], who have studied the CEBOT (Cellular Robotic System). CEBOT is a dynamically reconfigurable robotic system, consisting of several simple fundamental components, called "cells". Cells can physically combine and detach autonomously, so that the system can self-reorganize its total shape and software according to the assigned task, giving to the system higher flexibility and adaptability.

In this same perspective is the work of Walter *et al.*, that study metamorphic robotic system composed by two dimensional hexagonal modules. In particular, in [51], they study how to reach a specific configuration starting from a given one.

Noreils [63, 64] deals with the problem of developing an architecture for cooperative, autonomous mobile robots. The architecture he describes is planner-based, in the sense that the global task to solve is given to a robot, that becomes the leader. The main task of the leader is to decompose the global task into sub-tasks, which it then allocates to set of robots in the system, thus creating the team that will solve the global task. The author implements this idea on the task of box-pushing with two robots. The same problem is solved by Matarić, Nilsson and Simsarian [55] adopting the strategies of the emergent behavior described in the previous section.

The common feature of all these approaches is that they do not deal with formal correctness and they are only analyzed empirically. Algorithmic aspects were somehow implicitly an issue, but clearly not a major concern - let alone the focus - of the study. We aim to identify the algorithmic limitations of what autonomous, mobile robots can do. An investigation with this flavor has been undertaken within

the AI community by Durfee [35], who argues in favor of limiting the knowledge that an intelligent agent must possess in order to be able to coordinate its behavior with others.

Another work similar to the one we propose is that of B. R. Donald, J. Jennings and D. Rus [31, 32, 33]. The authors' goal is to investigate the information requirements for robots tasks, in particular they try to understand "what information is needed by a particular robot to accomplish a particular task", "how may the robot acquire such information", "what properties of the world have a great effect on the fragility of a robot program/plan" and "what are the capabilities of a given robot in a specific environment or class of environments". Their aim is to formally measure the intrinsic amount of information required to perform a given task, developing a notion of information invariants for characterizing sensors, tasks, and the complexity of robotics operations. In particular they depict the sensori-computational system as a "circuit" (labelled graph), whose vertices represent sensori-computational components of the system and whose edges correspond to "data paths" through which information passes. This provides a way of automatically deciding whether a protocol that solve a problem is "reducible" to some other protocol that solves the same problem (that is if there exists a permutation of the graph's components associated with the first protocol that leads to the graph associated with the second protocol). They provide the example of two robots involved in a box-pushing task, giving three different protocols that solve the problem, each with different ways of collecting the necessary informations (direct or indirect communication). Our work differs from that of Donald et al., since we do not deal with the circuitry that implements the algorithms that the agents use during the computation; instead, we want to study the intrinsic limitation of the distributed control, pointing out the minimal capabilities the agents must have in order to accomplish the given tasks.

2.3 Self-stabilization

The term *self-stabilization* was first introduced by Dijkstra, who called a distributed system self-stabilizing "if and only if, regardless of the initial state [...], the system is guaranteed to find itself in a legitimate state after a finite number of moves" [28]. From his definition, we have that such a system does not need to be initialized and can recover from transient failures caused by moves that brought the system in an illegitimate state, where *transient failures* are events that may change the state of the system by corrupting the local state of a processor (for an example its local memory or program counter). Due to these features, recently this notion has been intensely studied as an approach to fault tolerance, since a self-stabilizing system

can recover from inconsistent system states that could occur without any outside intervention [29, 71].

One of the features of our agents is that they are *oblivious*, that is they do not need to have memory of the past in order to accomplish the assigned tasks. This clearly gives the self-stability property to the algorithms designed for our model, since the decision an agent takes at some time in the computation does not depend on what happened in the system previously, therefore from what could be stored in its local memory. But there is one main difference between our self-stabilizing algorithms and the ones presented in the literature: our system is not modeled on a graph. In fact, the main work done on self-stabilization models distributed systems as graphs whose nodes are the processors in the system and whose edge represent the connection between the processors, hence the topology of the system [4, 20, 21, 30, 48]. In contrast, our model allows the agents (the processors in the distributed system) to move on the plane, without any restriction in the moves they can do.

A study by Debest points out the importance of better understanding self-stabilization in systems built from several components that are moving independently from each other, but that are cooperating in order to reach a common goal [25]. He studies the problem of the circle formation by a set of autonomous mobile robots, describing informally an algorithm and analyzing the reaction of the system to "unexpected events", such as the changing of position of one or more robot, failure of one robots and program modification or erroneous behavior by only one robot. In his paper he concludes that "in an environment where software applications are growing more and more distributed [...], a good understanding of how the self-stabilizing state of a complex system is influenced by the intrinsic behavior of its individual components is a prerequisite to designs working well [...]".

The same problem of studying the relationship between the circle formation problem and self-stabilization was first mentioned in [74], and successively investigated in [76], where the proposed algorithm may form a Reuleaux's triangle. However, the idea of studying self-stabilizing concepts in systems composed by cooperating mobile agents is new and the only works to our knowledge dealing with this problem are the ones just mentioned. Therefore, we find that our work can be useful in contributing to a better understanding of self-stabilization in "self-organizing" systems.

2.4 The Algorithmic Approach

In the previous sections, we discussed some of the studies dealing with the control and coordination of autonomous mobile agents, in order to support the "weak"

feature that characterize the entities of the distributed environment under study in this thesis.

In the majority of these studies, the approach is empirical. In fact, no sketch of correctness of the approaches adopted to solve a given coordination problem are in general given. In some cases, the solutions are not even proven to converge.

In contrast, in our study, we aim at an *algorithmic* approach. Namely, given a problem that the agents are asked to solve, our first goal is to understand when the agents can solve it. In other words, we aim to highlight the relationships between capabilities of the robots, shared knowledge of the world, and solvability of the problem. For instance, we answer questions like: is this problem solvable if the agents can sense only a portion of the plane? or if there is no common understanding among the robots on where is North? or if, given a point p in the plane, they cannot detect how many robots are on p?

Second, we provide a deterministic algorithm that allows the agents to accomplish the given task, given that they have the capabilities proved to be necessary in order to solve the problem.

Finally, we show that the provided solution is correct and finite. That is, the assigned task is accomplished in finite time independently from the positions in the plane the agents occupy at the beginning of the computation, and independently from the time each agent elapses in each cycle of sensing, computing, and moving it executes during its *life*, given that each of the sensing, computing and moving activity lasts a finite amount of time.

The only other work, to our knowledge, that approaches the problem of studying the coordination and control of asynchronous agents that can freely move on a two dimensional plane from an algorithmic and computational perspective, is that of Suzuki *et. al* [76, 77, 78] (and, with this focus, a rarity in the mobile robots literature). It gives a nice and systematic account of the algorithmics of pattern formation problem and of gathering under several assumptions on the power of the individual robot.

The model that we use differs, however, from those of [76, 77, 78] in that our agents are as weak as possible in every single aspect of their behavior. In particular, the two approaches drastically deviate in the way the asynchronous behavior of the agents is modeled. In fact, Suzuki *et al.* assume that each robot execute the sensing, computing, and moving activities of each cycle *atomically*. In contrast, we model them in a *fully asynchronous* fashion, meaning that each one of these activities can last an unpredictable but finite amount of time. A detailed description of the main differences between the two approaches will be given in Chapter 3.

2.5 Conclusions

In this chapter we presented the main motivations that prompted us to study a distributed systems whose entities are "weak" mobile agents. In particular, we described some of the studies that can be found in robotics and AI literature that deal with the coordination and control of a set of autonomous mobile robots. Most of these studies approach the problem from an empirical point of view, and very few works approach the problem from the point of view adopted in this thesis: an *algorithmic* point of view.

Chapter 3

Modeling Autonomous Mobile Robots

A robot must obey orders given it by human beings, except where such orders would conflict with a higher-order Law.
The Second Law of Robotics – Isaac Asimov

A robot must protect its own existence as long as such protection does not conflict with a higher-order Law.
The Third Law of Robotics – Isaac Asimov

Abstract

In the first part of this chapter, we formally present the features of the model under study. In the second part, we compare this model with the only other, to our knowledge, that approaches the problem under study from a computational point of view. In particular, we compare the classes of problems that are solvable in the two models, highlighting the main aspects that render the two models deeply different.

3.1 The CORDA Model

We consider a distributed system populated by a set of autonomous mobile robots, that are modeled as devices with computational capabilities, which are able to freely move on a two-dimensional plane. Each robot is viewed as a point, and it is equipped with sensors that let it observe the positions of the other robots in the plane, and form its *local view* of the world.

The behavior of a robot is a cycle of sensing, computing, moving and being inactive. In particular, each robot is capable of sensing its immediate surrounding, performing local computations on the sensed data, and moving towards the

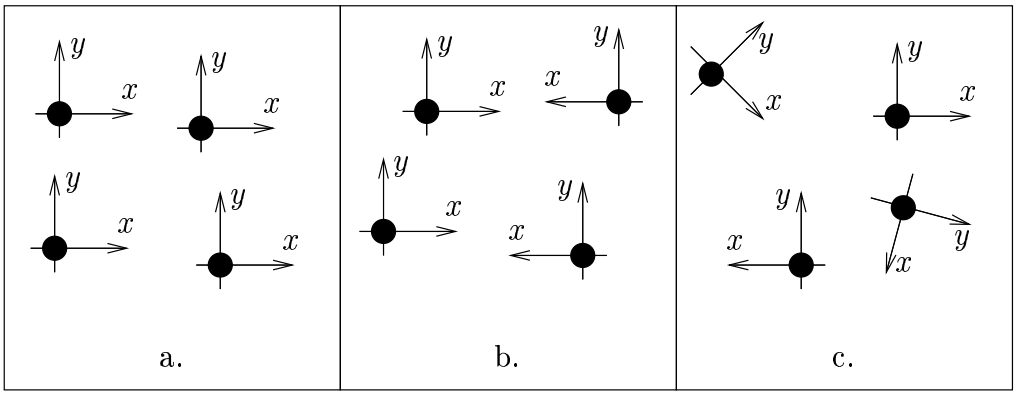


Figure 3.1: (a) *Total agreement* on the local coordinate systems. (b) *Partial agreement* on one axis direction and orientation. (c) *No agreement* on the local coordinate systems.

computed destination. The local computation is done according to a deterministic algorithm which takes in input the sensed data (i.e., the robots' positions), and returns a destination point towards which the executing robot moves. All the robots execute the same algorithm.

The local view of each agent includes a unit of length, an origin, and a Cartesian coordinate system defined by the *directions* of two coordinate axes, identified as the x and y axis, together with their *orientations*, identified as the positive and negative sides of the axes.

Different settings arise from different assumptions that are done on the robots' capabilities, and on the amount of information that they share and use during the accomplishment of the assigned task. In particular,

i. The robots are able to sense all the plane or just a portion of it. We will refer to the first case as the *Unlimited Visibility* setting. In contrast, if each robot can sense only up to at most a distance $V > 0$ from it, we are in the *Limited Visibility* setting. In the following, we will say also that the robots have unlimited/limited visibility.

ii. The robots do not necessarily share the same $x - y$ coordinate system, and do not necessarily agree on the location of the origin (that we can assume, without loss of generality, to be placed in the current position of the robot), or on the unit distance. In other words, in general there is no agreement among the robots on the chirality of the local coordinate systems (i.e., in general they do not share the same concept of where North, East, South, and West are). In the most favorable scenario, the robots agree on the direction and

orientation of both axes, hence they agree on where North, South, East and West are (see Figure 3.1.a). Knowing the direction of the x axis means that all robots know and use the fact that all the lines identifying their individual x axes are parallel. Similarly, knowing the orientation of an axis means that the positive side of that axis in the local coordinate system coincides for all robots. As an example for an axis whose direction and orientation is known, each robot may have a built-in compass needle that points North, with all compass needles parallel, consistently for all robots. In this case, we will talk of *total agreement* on the local coordinate systems. Note that knowledge of the directions and orientations of both axes does not imply knowledge of the origin or the unit of length. An alternative scenario is when the robots agree only on the direction and orientation of one axis, say y, hence they agree on the orientation of the East-West axis, but there is no general agreement on where is East or West (see Figure 3.1.b): we will talk of *one axis direction and orientation agreement* (or shortly *partial agreement*). In the worst case, the robots do not have any kind of agreement on the orientation of the local coordinate systems (see Figure 3.1.c): we will refer to this scenario as *no agreement* on the local coordinate systems.

iii. The algorithm the robots execute can access different amount of information regarding the robots' positions. In particular, if the algorithm takes in input the robots' positions retrieved only in the last observation, we have an *oblivious* algorithm. In contrast, if the algorithm takes in input all the positions retrieved since the beginning of the computation, we have a *non oblivious* algorithm. We will also refer to the robots as *oblivious* or *non oblivious*, because of this feature of the algorithms they execute.

Note that, *the robots have always common knowledge [50] of the conditions under which they operate.* For instance, having limited visibility, obliviousness, and total agreement on the local coordinate systems, means the robots have common knowledge of these operating conditions.

The robots are completely *autonomous*: no central control is needed. Furthermore they are *anonymous*, meaning that they are a priori indistinguishable by their appearance, and they do not (need to) have any kind of identifiers that can be used during the computation.

Moreover, there are no explicit direct means of communication: any communication occurs in a totally implicit manner. Specifically, it happens by means of observing the change of robots' positions in the plane while they execute the algorithm. In other words, the only means for a robot to send information to some other robot is to move and let the others observe (reminiscent of bees in a bee dance). For

oblivious robots, even this sending of information is impossible, since they will not remember previous positions.

The robots are *fully asynchronous*: the amount of time spent in observation[1], in computation, in movement, and in inaction is finite but otherwise unpredictable. In particular, the robots do not (need to) have a common notion of time. Each robot executes its actions at unpredictable time instants.

Clearly, these basic features render the modeled robots simple and rather "weak", especially considering the current engineering technology. But, as already noted, the main interest of this study is to approach the problem of coordinating and controlling a set of mobile units from a *computational* point of view. The robots are modeled as "weak robots" because in this way it is possible to formally analyze the strengths and weaknesses of the distributed control. Furthermore, this simplicity also leads to some advantages. For example, avoiding the ability to remember what has been computed in the past (obliviousness) gives the system the nice property of self-stabilization [37, 78] (see Section 2.3).

Summarizing, the robots are totally asynchronous, oblivious, anonymous, and in general they do not share a sense of direction (e.g., a compass).

3.1.1 The Computational Cycle

During its life, each robot cyclically executes four *states*: (i) it is inactive, (ii) it observes the positions of the others in the world, (iii) it computes its next destination point by executing the algorithm every robot has, and (iv) it moves towards the point it just computed. As already stated, the robots execute these states *asynchronously*, without any central control: in this feature our model drastically differ from previous ones (see Section 3.2). Formally,

Definition 3.1.1 (State). A *state* for a robot r is a 4-tuple $< s, t_s, t_e, pos >$, where $s \in \{Wait, Look, Compute, Move\}$, t_s and t_e are two time instants, with $t_s \leq t_e$, and pos is an absolute[2] position in the plane. Given a state st, the four fields that define s will be denoted by $st.s$, $st.t_s$, $st.t_e$, and $st.pos$, respectively. ◁▷

In the above definition, t_s and t_e are the time instants when robot r starts and finishes being in state s, respectively; furthermore, pos is the position in the plane that r occupies at time t_e.

A robot is initially in a *waiting* state (*Wait*); asynchronously and independently from the other robots, it *observes* the environment in its area of visibility (*Look*); it *calculates* its destination point by executing the deterministic algorithm it has

[1] i.e., activating the sensors and receiving their data.

[2] i.e., with respect to an inertial reference frame.

(*Compute*); it then *moves* towards that point (*Move*); after the move it goes back to a waiting state.

The sequence: *Wait* - *Look* - *Compute* - *Move* will be called a *computation cycle* (or briefly *cycle*) of a robot. The operations performed by each robot r in each state will be now described in more details.

1. **Wait** The robot is idle. A robot cannot stay indefinitely idle (see Assumption A1 below). At the beginning all the robots are in *Wait*.

2. **Look** The robot r observes the world by activating its sensors which will return a *snapshot* of the positions of all other robots with respect to its local coordinate system. Each robot is viewed as a point, hence its position in the plane is given by its coordinates, and the result of the snapshot (hence, of the observation) is just a set of coordinates: this set form the *view of the world* of r. More formally, the view of the world of r at time t, denoted by $\mathbb{V}_r(t)$, is defined as the last snapshot of all robots' positions retrieved during a *Look* state whose execution started at a time smaller than or equal to t.

   In addition, a robot cannot in general detect whether there is more than one fellow robot on any of the observed points, included the position where the observing robot is. We say it cannot detect *multiplicity*.

3. **Compute** The robot performs a *local computation* according to a deterministic algorithm $\mathcal{A}$ (we also say that the robot *executes* $\mathcal{A}$). The algorithm is the same for all the robots, and the result of the *Compute* state is a *destination point*. If $\mathcal{A}$ is oblivious, then this point is returned by executing $\mathcal{A}$ with input the set of robots' positions retrieved during the last *Look* ($\mathcal{A}$ can possibly have in input also data that are specific to the problem being solved); otherwise, $\mathcal{A}$ can access the information relative to all the positions of the robots observed since the beginning of the computation.

4. **Move** If the point computed in the previous state is the current location of r, we say that r performs a *null movement*, and it does not move: we will denote this situation in code by `do_nothing()`; otherwise it moves towards the point computed in the previous state. The robot moves towards the computed destination of an unpredictable amount of space, which is assumed neither infinite, nor infinitesimally small (see Assumption A2 below). Hence, the robot can only go towards its goal, but it cannot know how far it will go in the current cycle, because it can stop anytime during its movement[3]. The amount

[3]That is, a robot can stop before reaching its destination point, e.g. because of limits to the robot's motorial autonomy.

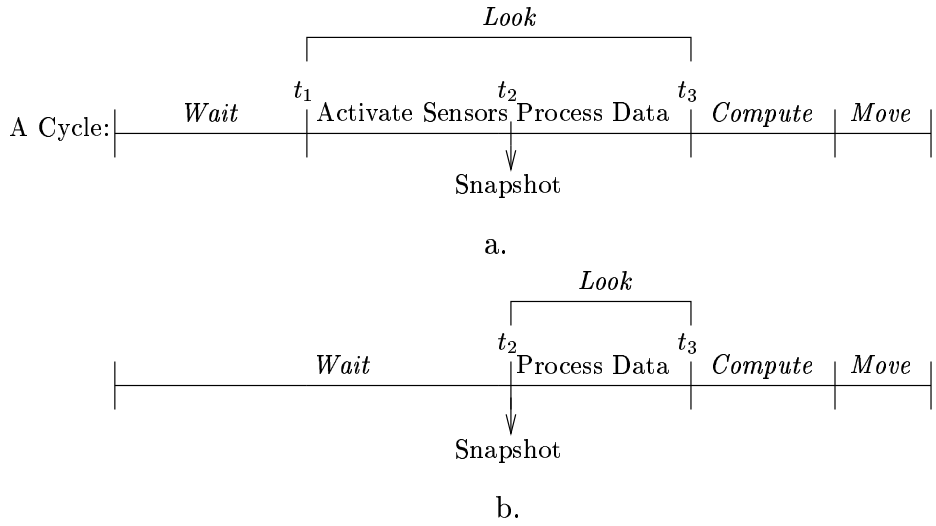


Figure 3.2: (a) An example of a cycle. In particular, the three parts that constitute the *Look* state (between time t_1 and t_3) are put in evidence. (b) Without loss of generality, we can assume that the *Look* state starts with the snapshot, since the time spent in activating the sensors can be thought as part of the previous *Wait*.

of space traveled by a robot during this state is also called the *length* of the move.

The (global) time that passes between two successive states of the same robot is finite. In addition, we do not make any timing assumptions within a state. For instance, this implies that the time that passes after the robot starts observing the positions of all others and before it starts moving is arbitrary, but finite. That is, the actual move of a robot may be based on a situation that was observed arbitrarily far in the past, and therefore it may be totally different from the current situation. We feel that this assumption of *asynchronicity within a state* is important in a totally asynchronous environment, since we want to give each robot enough time to perform its local computation; furthermore, in this way it is possible to model also different motorial speed of the robots.

Only one remark regarding the *Look* state. As already stated, the result of this state is a set of positions retrieved at one time instant, i.e. at the time when the snapshot of the world was done. That is, each *Look* can be split in three parts: in the first part the sensors are activated; in the second part the actual snapshot is performed; in the last part, the data captured by the sensors are sent away in order to be processed. For instance, referring to the cycle depicted in Figure 3.2.a, the first part of the *Look* is executed between time t_1 and t_2, the snapshot is done at time t_2, and the third part is executed between time t_2 and t_3. In the following, we

will assume that the first part has null length. This is not a loss of generality: in fact, the first part can be thought to be part of the previous *Wait* state (as shown in Figure 3.2.b). Therefore, each *Look* starts with the snapshot. According to this assumption, if r is executing a *Look* at time t, then $\mathbb{V}_r(t)$ is the snapshot retrieved during the *Look* being executed at t.

In the model, there are only two limiting assumptions about time and space. The first refers to the length of a computational cycle.

Assumption A1(Computational Cycle) *The amount of time required by a robot r to complete a computational cycle is neither infinite nor infinitesimally small.*

In particular, there exists a constant $\epsilon_r > 0$ such that the cycle will require at least ϵ_r time.

As no other assumption on time exists, the resulting system is *fully asynchronous* and the duration of each activity (or inactivity) is unpredictable. As a result, the robots do not have a common notion of time, robots can be seen while moving, and computations can be made based on obsolete observations.

The second assumption in the model refers to space; namely, the distance traveled by a robot during a computational cycle.

Assumption A2(Distance) *The distance traveled by a robot r in a move is neither infinite nor infitesimally small.*

In particular, there exists an arbitrarily small constant $\delta_r > 0$, such that if the destination point is closer than δ_r, r will reach it; otherwise, r will move towards it of at least δ_r.

As no other assumptions on space exists, the distance traveled by a robot in a cycle is unpredictable. In the following, we shall use $\delta = \min_r \delta_r$.

3.1.2 Basic Definitions and Notations

In the rest of the thesis, the following notation will be used. We assume to have n robots in the system, denoted by $r_1, \ldots, r_n$; when no confusion arises, however, r_i indicates also the position occupied in the plane by robot r_i. x and y indicate the two axes in the local coordinate system of each robot; given a robot r, its coordinates will be denoted by $r.x$ and $r.y$. An arbitrary point in the plane is indicated by small or capital letters (e.g., A, p).

A *configuration* is a set of distinct points in the plane. In particular, a *configuration of the robots* at time t is defined as the set of robots' positions at time t. We say that r is *allowed to move at time t* by an algorithm $\mathcal{A}$, if the algorithm executed

on robot r with input the configuration of the robots at time t does not return a *null movement*.

Capital calligraphic letters (e.g., $\mathcal{C}$) indicate closed regions in the plane; the letter $\mathcal{A}$, however, is reserved to denote an arbitrary algorithm executed in Corda. Given a region $\mathcal{Z}$, the number of robots in that region is denoted by $|\mathcal{Z}|$. Furthermore, $p \in \mathcal{Z}$ denotes the fact that point p belongs to $\mathcal{Z}$ (i.e., p is inside $\mathcal{Z}$ or on its border); by $p \tilde{\in} \mathcal{Z}$ we indicate that p is *strictly* inside $\mathcal{Z}$ (i.e., p is not on its border). In particular, $\mathcal{C}$ denotes a circle and $Rad(\mathcal{C})$ its radius. We say that a point p is *in* (or *inside*) $\mathcal{C}$, if $dist(p, c) < Rad(\mathcal{C})$, with c the center of $\mathcal{C}$ and $dist(a, b)$ the Euclidean distance between points a and b in the plane; we say that p is *on* $\mathcal{C}$, if $dist(p, c) = Rad(\mathcal{C})$ (see Figure 3.3.a). Moreover, two points on the circle are said to be *opposite* if they are the end points of a diameter of $\mathcal{C}$. The triangle having as vertices points a, b and c is denoted by $\triangle(a, b, c)$.

Lines, half-lines and segments will be denoted by capital greek letters (e.g., Ψ, Ξ). In particular, given two distinct points a and b, $[ab)$ denotes the half-line that starts in a and passes through b; and $[ab]$ the segment between a and b. Given two distinct parallel lines Γ and Γ', and a line Γ'' orthogonal to Γ and Γ', we define the *horizontal distance* between Γ and Γ', denoted by $\overline{\Gamma\Gamma'}$, as the length of the segment $[qq']$, with $q = \Gamma \cap \Gamma''$ and $q' = \Gamma' \cap \Gamma''$ (see Figure 3.3.b). Furthermore, given a point p, we define the horizontal distance of p from Γ, denoted by $\overline{p\Gamma}$, as the horizontal distance between Γ and the line passing through p and parallel to Γ.

Given two half-lines $[Va)$ and $[Vb)$, we denote by $\alpha = a\widehat{V}b$ the convex angle centered in V and with sides $[Va)$ and $[Vb)$ (see Figure 3.3.c); the concave angle will be denoted by $a\check{V}b$ (see Figure 3.3.d). We recall that, given a circle $\mathcal{C}$ centered in c, any angle centered in c is called *angle at the center*; moreover, an angle whose center is on $\mathcal{C}$ and whose sides are both secant the circumference is called *angle at the circumference*. The intersection between the circumference of $\mathcal{C}$ and an angle at the center α is denoted by $arc(\alpha)$ (see Figure 3.3.e); the intersection between α and $\mathcal{C}$ is denoted by $sector(\alpha)$ (see Figure 3.3.f); and the area of the circle delimited by $[ab]$ and $arc(\alpha)$ is denoted by $segm(\alpha)$, with a and b the endpoints of $arc(\alpha)$ (see Figure 3.3.g).

Double lined capital letters (e.g., $\mathbb{E}$) will denote set of points, set of robots' positions, and set of robots. In particular, $\mathbb{V}_r(t)$ denotes the view of the world of r at time t. Moreover,

1. $\mathbb{W}(t)$ denotes the set of robots that are in *Wait* at time t;

2. $\mathbb{L}(t) = \mathbb{L}_\emptyset(t) \cup \mathbb{L}_+(t)$, is the set of robots that at time t are in state *Look*. The set $\mathbb{L}_\emptyset(t)$ contains those robots whose computation's result in their next *Compute* state is a *null movement*, while $\mathbb{L}_+(t)$ contains those robots whose

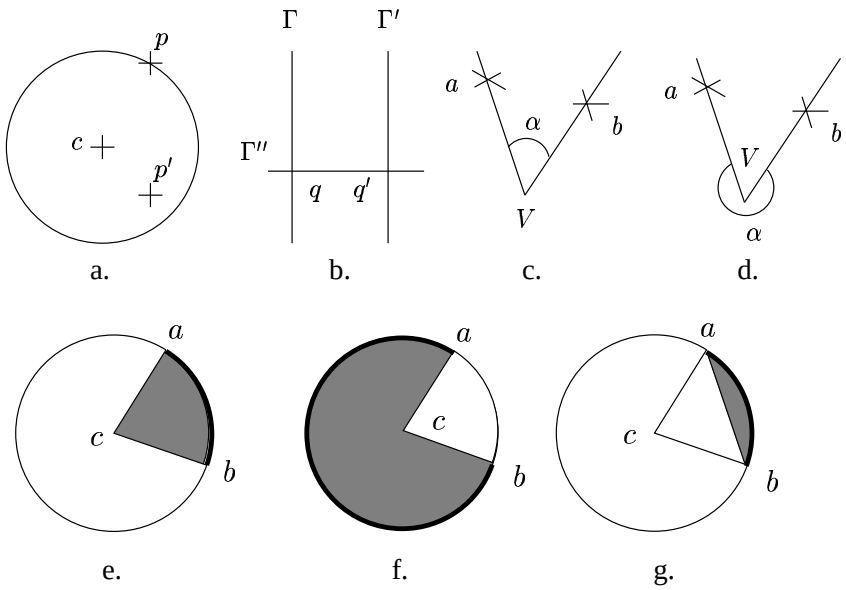


Figure 3.3: (a) Point p is *on* the circle centered in c, while p' in *inside* it. (b) Definition of *horizontal distance* between Γ and Γ'. In (c) and (d) the angles $a\widehat{V}b$ and $a\check{V}b$ are pictured, respectively. In (e) and (f), the arc (thick line) and sector (grey part) defined by $a\widehat{cb}$ and $a\check{c}b$ are shown, respectively. (g) The grey area represents $segm(a\widehat{cb})$.

computation's result in their next *Compute* is some destination point different from the position where the observing robot is (we say that they will execute a *real movement*). Furthermore, we denote by $\mathbb{L}_S(t)$ the set of robots that start a *Look* state at time t.

3. $\mathbb{C}(t) = \mathbb{C}_\emptyset(t) \cup \mathbb{C}_+(t)$, is the set of all the robots that at time t are in state *Compute*. The set $\mathbb{C}_\emptyset(t)$ contains those robots whose computation's result is a *null movement*, while $\mathbb{C}_+(t)$ contains those robots whose computation's result is a *real movement*.

4. $\mathbb{M}(t) = \mathbb{M}_\emptyset(t) \cup \mathbb{M}_+(t)$ is the set of all the robots that at time t are executing a movement. The set $\mathbb{M}_\emptyset(t)$ contains the robots executing a *null movement* (they stay still); $\mathbb{M}_+(t)$ contains those executing a *real movement* (they are effectively moving towards a destination).

Finally, we define a particular set of robots, $\mathbb{SD}(t)$, that will be useful in order to analyze the behavior of the robots while executing the algorithms studied in the following. Namely, $\mathbb{SD}(t) = \mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{L}_S(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$.

3.1.3 Activation Schedules in Corda

In this section, we will define two different way to *schedule* robots in Corda. The first one schedules the robots in a completely asynchronous fashion, according to the definition of Corda. Informally, in an *asynchronous activation schedule*, there are no constraints on how long lasts each of the states a robot can be in during its life, given that Assumptions A1 and A2 are satisfied.

Then, we define a particular asynchronous schedule, in which all the robots execute their cycles in a perfectly synchronized fashion: the *synchronous activation schedule*. This particular scheduling will be useful in the following sections, in order to compare Corda with the only other model, to our knowledge, that approaches the problem addressed in this thesis with our same perspective.

Asynchronous Activation Schedule in Corda

In this section, we aim to describe the possible ways in which the robots in Corda can be scheduled while executing a given algorithm, reflecting the full asynchronicity of the model. According to the description of Corda given above, during its life each robots executes a sequence of cycles, each constituted by four states. Furthermore, each robot can elapse an unpredictable but finite amount of time to complete its operations during its life. Therefore, we define

Definition 3.1.2 (Asynchronous Activation Sequence). Given an algorithm $\mathcal{A}$, an *asynchronous activation sequence* $Async\mathcal{S}_i$ in Corda for a robot r_i is a sequence of states $st_1, st_2, \ldots$, such that

a. if $st_j =< Wait, t_w, t', pos >$, then $st_{j+1} =< Look, t', t'', pos >$;

b. if $st_j =< Look, t, t', pos >$, then $st_{j+1} =< Compute, t', t'', pos >$;

c. if $st_j =< Compute, t, t', pos >$, then $st_{j+1} =< Move, t', t'', pos' >$, with pos' the result of the *Compute* state executed between time t and t', such that Assumption A2 is satisfied;

d. if $st_j =< Move, t, t', pos >$, then $st_{j+1} =< Wait, t', t'_w, pos >$,

with $t_w \leq t' < t''$, and $t < t' \leq t'_w$, such that Assumption A1 is satisfied. By $Async\mathcal{S}_i[j]$ we denote the $j-th$ state in $Async\mathcal{S}_i$. ◁▷

If all the n robots execute the given algorithm according to asynchronous sequences, we have the definition of an asynchronous activation schedule.

Definition 3.1.3 (Asynchronous Activation Schedule). An *asynchronous activation schedule* $Async\mathcal{F}$ for an algorithm $\mathcal{A}$ is a set of n asynchronous activation sequences $Async\mathcal{S}_1, \ldots, Async\mathcal{S}_n$ such that $Async\mathcal{S}_1[1].t_s = \ldots = Async\mathcal{S}_n[1].t_s = t^*$, and $Async\mathcal{S}_1[1].s = \ldots = Async\mathcal{S}_n[1].s = Wait$. We say that $Async\mathcal{F}$ starts at time t^*. ◁▷

In the following, we denote by $Async\mathcal{F}[i]$ the $i-th$ activation sequence in $Async\mathcal{F}$ (hence, $Async\mathcal{F}[i][j]$ will indicate the $j-th$ state in the $i-th$ asynchronous activation sequence of $Async\mathcal{F}$).

Definition 3.1.4. Given an algorithm $\mathcal{A}$ and an asynchronous activation schedule $Async\mathcal{F}$ for $\mathcal{A}$, we say that *the computation is done according to* $Async\mathcal{F}$ if the following holds, for all $1 \leq i \leq n$ and $j \geq 1$. If $Async\mathcal{F}[i][j] =< s, t, t', pos >$, then r_i is in s for all $t \leq t < t'$, with $s \in \{Wait, Look, Compute\}$, while occupying position pos on the plane. If $Async\mathcal{F}[i][j] =< Move, t, t', pos' >$, then r_i is in *Move* for all $t \leq t < t'$; furthermore, it occupies position $Async\mathcal{F}[i][j-1].pos$ at time t, position pos' at time t', and for all $t \leq t < t'$ it moves on the segment $[pos, pos']$. Alternatively, we say that $\mathcal{A}$ *is executed according to* $Async\mathcal{F}$. ◁▷

The aim of the robots is to solve a given problem $\mathcal{P}$, that defines a task the robots have to accomplish in finite time.

Definition 3.1.5. We say that an algorithm $\mathcal{A}$ *solves* a problem $\mathcal{P}$ if all the robots accomplish the task defined by $\mathcal{P}$ in a finite number of cycles, by executing $\mathcal{A}$ according to any asynchronous activation schedule and starting from any initial configuration considered to be *valid* for $\mathcal{P}$. ◁▷

Synchronous Activation Schedule in Corda

In this section, we introduce some definitions regarding a specific way the robots in Corda can be scheduled. In particular, we introduce the definition of a *synchronous activation schedule* in Corda, where all the robots execute their cycles in a perfectly synchronized fashion. Let $\rho \geq \min_r \epsilon_r$, with ϵ_r the constant introduced in Assumption A1 above.

Definition 3.1.6 (Long Wait). A *long wait* for a robot r_i is the state $LW(t_s, pos) = < Wait, t_s, t_s + 4\rho, pos >$. We will say that r_i starts a long wait at time t_s, and that *pos* is the absolute position in the plane of r_i between time t_s and $t_s + 4\rho$. ◁▷

The next definition introduces the idea of executing a cycle that lasts exactly 4ρ, where the time elapsed to execute each state in this cycle is ρ.

Definition 3.1.7 (Normal Activation). A *normal activation* of robot r_i in Corda is a cycle defined as follows: $NA(t_s, pos, pos') = \{< Wait, t_s, t_s + \rho, pos >, < Look, t_s + \rho, t_s + 2\rho, pos >, < Compute, t_s + 2\rho, t_s + 3\rho, pos >, < Move, t_s + 3\rho, t_s + 4\rho, pos' >\}$. We say that a normal activation starts at time t_s, and that *pos* and *pos'* are the absolute positions in the plane of r_i at time t_s and $t_s + 4\rho$, respectively. ◁▷

Now, we are ready to define a *synchronous activation sequence* for a robot r_i in Corda: r_i can be either normal activated or execute long waits during its life.

Definition 3.1.8 (Synchronous Activation Sequence). Given an algorithm $\mathcal{A}$, a *synchronous activation sequence* $Sync\mathcal{S}_i$ in Corda for a robot r_i is a sequence of *elements*, where each element is either $NA(\cdot, \cdot, \cdot)$ or $LW(\cdot, \cdot)$. In particular,

1. any normal activation $NA(t, pos, pos')$ is followed either by $NA(t+4\rho, pos', pos'')$ or by $LW(t + 4\rho, pos')$, where *pos''* is the result of the *Compute* state executed at time $t + 6\rho$; and

2. any long wait $LW(t, pos)$ is followed either by $NA(t + 4\rho, pos, pos')$ or by $LW(t + 4\rho, pos)$, where *pos'* is the result of the *Compute* state executed at time $t + 6\rho$.

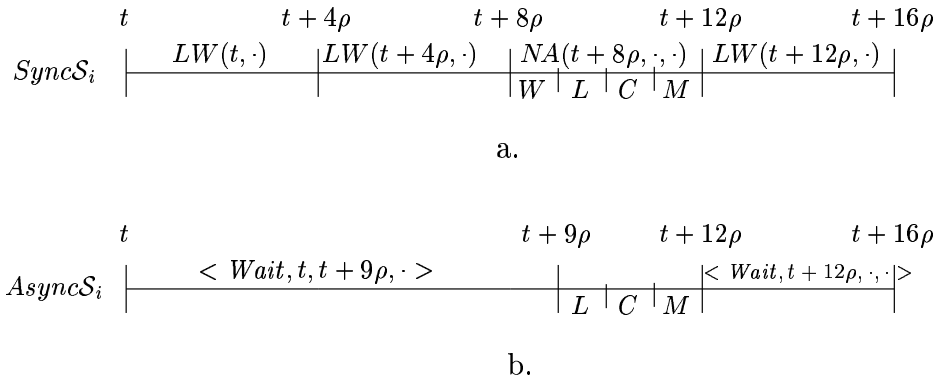


Figure 3.4: (a) A synchronous activation sequence, and (b) its corresponding asynchronous activation schedule (see Note 3.1.1).

Furthermore, between any two consecutive normal activations, or between the last normal activation and the end of $Sync\mathcal{S}_i$, there is a finite number of consecutive long waits. ◁▷

A pictorial example of a synchronous activation sequence is depicted in Figure 3.4. In the following, we denote by $Sync\mathcal{S}_i[j]$, $j \geq 1$, the $j-th$ element in $Sync\mathcal{S}_i$, and by $Last$ the index of the last element in a synchronous activation sequence. Furthermore, $Sync\mathcal{S}_i[j].t_s$ denotes the time when the $j-th$ element in $Sync\mathcal{S}_i$ starts; $Sync\mathcal{S}_i[j].pos_s$ and $Sync\mathcal{S}_i[j].pos_e$ denote the absolute positions of r_i at time $Sync\mathcal{S}_i[j].t_s$ and $Sync\mathcal{S}_i[j].t_s + 4\rho$, respectively. In other words, $Sync\mathcal{S}_i[j].t_s$ and $Sync\mathcal{S}_i[j].pos_e$ are the first and last parameter of $Sync\mathcal{S}_i[j]$, respectively; $Sync\mathcal{S}_i[j].pos_s$ is the second parameter of $Sync\mathcal{S}_i[j]$.

Given two non-empty synchronous activation sequences $Sync\mathcal{S}$ and $Sync\mathcal{S}'$ such that $Sync\mathcal{S}'[1].t_s = Sync\mathcal{S}[Last].t_s + 4\rho$ and $Sync\mathcal{S}'[1].pos_s = Sync\mathcal{S}[Last].pos_e$, we denote by $Sync\mathcal{S} \circ Sync\mathcal{S}'$ the synchronous activation sequence obtained by appending the elements in $Sync\mathcal{S}'$ to the elements in $Sync\mathcal{S}$.

If all the robots execute their cycles in a synchronous fashion during the computation, we have a *synchronous activation schedule*.

Definition 3.1.9 (Synchronous Activation Schedule). A *synchronous activation schedule* $Sync\mathcal{F}$ for an algorithm $\mathcal{A}$ is a set of n synchronous activation sequences $Sync\mathcal{S}_1, \ldots, Sync\mathcal{S}_n$ such that $Sync\mathcal{S}_1[1].t_s = \ldots = Sync\mathcal{S}_n[1].t_s = t^*$. We say that $Sync\mathcal{F}$ starts at time t^*. ◁▷

In the following, we denote by $Sync\mathcal{F}[i]$ the $i-th$ synchronous activation sequence in $Sync\mathcal{F}$ (hence, $Sync\mathcal{F}[i][j]$ will indicate the $j-th$ element in the $i-th$ synchronous activation sequence of $Sync\mathcal{F}$).

Given an algorithm $\mathcal{A}$ and a synchronous activation schedule $Sync\mathcal{F}$ for $\mathcal{A}$, we say that $Sync\mathcal{F}$ is *extended* with a normal activation of r_i, if a normal activation is appended to $Sync\mathcal{F}[i]$; that is, $Sync\mathcal{F}[i] := Sync\mathcal{F}[i] \circ NA(t, pos^*, pos^{**})$, where $t = Sync\mathcal{F}[i][Last].t_s + 4\rho$; $pos^* = Sync\mathcal{F}[i][Last].pos_e$, and pos^{**} is the result of the *Compute* state started at time $t + 2\rho$. Similarly, we say that $Sync\mathcal{F}$ is *extended* with a long wait of r_i, if a long wait is appended to $Sync\mathcal{F}[i]$; that is, $Sync\mathcal{F}[i] := Sync\mathcal{F}[i] \circ LW(t, pos^*)$, where $t = Sync\mathcal{F}[i][Last].t_s + 4\rho$; and $pos^* = Sync\mathcal{F}[i][Last].pos_e$. Alternatively, in the following we will say that r_i is normally activated at time t, and that a long wait of r_i starts at time t. Note that, by extending a synchronous activation schedule, a synchronous activation schedule is obtained.

Definition 3.1.10. Given an algorithm $\mathcal{A}$ and a synchronous activation schedule $Sync\mathcal{F}$ for $\mathcal{A}$, we say that *the computation is done according to* $Sync\mathcal{F}$ if the following holds, for all $1 \leq i \leq n$ and $j \geq 1$. If $Sync\mathcal{F}[i][j] = NA(t_s, pos, pos')$, then r_i is in *Wait* for all $t_s \leq t < t_s + \rho$; in *Look* for all $t_s + \rho \leq t < t_s + 2\rho$; in *Compute* for all $t_s + 2\rho \leq t < t_s + 3\rho$; and in *Move* for all $t_s + 3\rho \leq t < t_s + 4\rho$; furthermore, r_i is on pos at time t_s, on pos' at time $t_s + 4\rho$, and for all $t_s \leq t \leq t_s + 4\rho$ it moves on the segment $[pos, pos']$. If $Sync\mathcal{F}[i][j] = LW(t_s, pos)$, then r_i is in *Wait* for all $t_s \leq t < t_s + 4\rho$. Alternatively, we say that $\mathcal{A}$ *is executed according to* $Sync\mathcal{F}$. ◁▷

Note 3.1.1. It is easy to see that a synchronous activation schedule is an asynchronous activation schedule. In fact, given a synhronous activation schedule $Sync\mathcal{F}$, in order to obtain the corresponding asynchronous schedule it is sufficient to merge all the sequences of LW present in $Sync\mathcal{F}$ into only one state st, with $st.s = Wait$, as depicted in Figure 3.4.b. Therefore, if an algorithm $\mathcal{A}$ lets the robots solve a problem $\mathcal{P}$, then $\mathcal{P}$ is solved by $\mathcal{A}$ also if the computation is done according to a synchronous activation schedule. ⋆

3.2 A Different Approach: *Instantaneous Actions*

As highlighted in the previous section, one important feature of Corda is that the agents in the environment act following a *fully asynchronous* behavior. The only other study, to our knowledge, on the problem of coordinating and controlling a set of autonomous and mobile entities from a computational point of view has been done by Suzuki *et al.* [3, 78]. They presented a model, that we will refer in the following as SYm, whose main purpose is, similarly to Corda, to study the possible interactions that can happen among the robots.

The main aspect where SYm and CORDA drastically differ is in the way the robots interacts, specifically in the way the asynchronicity is modeled. In fact in SYm, like in CORDA, the cycle of life of each robot is (i) to observe the positions of the others, (ii) compute its next destination point according to a deterministic algorithm (shared by all the robots), and (iii) move towards this destination.

In particular, the movement of the robots is modeled in SYm [3, 78] as follows. The authors assume discrete time $0, 1, 2, \ldots$. At each time instant t, every robot r_i is either *active* or *inactive*. At least one robot is active at every time instant, and every robot becomes active at infinitely many unpredictable time instants. A special case is when every robot is active at every time instant; in this case the robots are *synchronized*. By A_t the authors denote the set of active robots at t; furthermore, they call the sequence $A_0, A_1, \ldots$ an *activation schedule*. In the following, we denote this sequence by ActSYm. Therefore, we can say that the computation is done according to ActSYm (or that $\mathcal{A}$ is executed according to ActSYm) if $r_i \in A_t$ implies that r_i is active at time t, for all $1 \leq i \leq n$ and $t \geq 0$.

Let $p_i(t)$ indicate the position of robot r_i at time instant t, and ψ the algorithm every robot uses. Since the robots are viewed as points, in SYm it is assumed that two robots can occupy the same position simultaneously and never collide. ψ is a function that, given the positions of the robots at time t (or, in the non oblivious case, all the positions the robots have occupied since the beginning of the computation[4]), returns a new destination point p. For any $t \geq 0$, if r_i is inactive, then $p_i(t+1) = p_i(t)$; otherwise

$$p_i(t+1) = p, \tag{3.1}$$

where p is the point returned by ψ.

The maximum distance that r_i can move in one step is bounded by a distance[5] $\sigma_i > 0$ (this implies that every robot is than capable of traveling at least a distance $\sigma = \min\{\sigma_1, \ldots, \sigma_n\} > 0$). The reason for such a constant is to simulate a continuous monitoring of the world by the robots. In CORDA there is no assumption on the maximum distance a robot can travel before observing again (apart from the bound given from the destination point that has to be reached), while in SYm an active robot r_i always travels at most a distance σ_i in each step. The only assumption in CORDA is that there is a lower bound on such distance: when a robot r moves, it moves at least some positive, small constant δ_r. The reason for this constant is to better model reality: it is not realistic to allow the robots to move an infinitesimally small distance.

[4]Note that the non obliviousness feature does not imply the possibility for a robot to find out which robot corresponds to which position it stored, since the robots are anonymous.

[5]In [3, 78], this constant is denoted by ϵ_i.

Thus, from (3.1), r_i executes the three states (i)–(iii) *instantaneously*, in the sense that a robot that is active and observes at t, has already reached its destination point p at $t+1$. Therefore, we have that a robot takes a certain amount of time to move (the time elapsed between t and $t+1$), but no fellow robot can see it *while* it is moving (or, alternatively, the movement is *instantaneous*).

Therefore, the main difference between the two models is, as stated before, in the way the asynchronicity is regarded. In Corda the environment is *fully asynchronous*, in the sense that there is no common notion of time, and a robot observes the environment at unpredictable time instants. Moreover, no assumptions on the cycle time of each robot, and on the time each robot elapses to execute each state of a given cycle are made. It is only assumed that each cycle is completed in finite time, and that the distance traveled in a cycle is finite. Thus, each robot can take its own time to compute, or to move towards some point in the plane: in this way, it is possible to model different computational and motorial speeds of the units. Moreover, every robot can be seen *while* it is moving by other robots that are observing[6]. This feature renders more difficult the design of an algorithm to control and coordinate the robots. For example, when a robot starts a *Move* state, it is possible that the movement it will perform will not be "coherent" with what it observed, since, during the *Compute* state, other robots can have moved. On the other hand, we feel that the full asynchronicity of Corda better models the way a set of autonomous, mobile, and *asynchronous* robots interact and coordinate in order to accomplish some given task.

3.3 *Instantaneous Action* vs. *Full Asynchronicity*

In this section, we study the relationship between Corda and SYm. In particular, we first show that any algorithm designed in Corda to solve some problem $\mathcal{P}$ can be used in SYm to let the robots accomplish the task defined by $\mathcal{P}$. The vice versa is not true. In fact, we will give strong evidences that the differences pointed out in the previous sections, in particular the way in which the asynchronicity is modeled, render the two models *really* different, both in the oblivious and non oblivious case, and that an algorithm designed in SYm to solve a given problem does not solve the same problem when executed in Corda.

The first question we address is: can an algorithm designed in SYm be *executed* in Corda? The difference between an algorithm designed in SYm and one for

[6]Note that this does not mean that the observing robot can distinguish a moving robot from a still one.

Corda relies in the amount of distance that each robot is allowed to travel in each cycle. In fact, to our knowledge, in all the algorithms designed in SYm there is an explicit knowledge by the robots of the maximum amount of distance that can be traveled in one cycle. Namely, if the destination point computed at a given cycle by robot r_i is further than σ_i, than the algorithm explicitly returns a point at most at distance σ_i (e.g., Algorithm 2 in Appendix A.2). In contrast, such a knowledge is not required in Corda. In order to establish a relationship between the kind of algorithms designed in SYm and Corda, we introduce the following

Definition 3.3.1. A *bounded step* algorithm is an algorithm that, when executed by r_i, returns a destination point at distance at most Upp_i from the current position of r_i, with $Upp_i > 0$ a constant given as input to the algorithm. Upp_i are called the *steps* of the algorithm. ◁▷

Clearly, any algorithm designed in SYm is a bounded step algorithm where $Upp_i = \sigma_i$, for all $1 \leq i \leq n$. Viceversa, any bounded step algorithm where $Upp_i = \sigma_i$, for all $1 \leq i \leq n$, is an algorithm that can be executed in SYm.

On the other hand, given an algorithm $\mathcal{A}$ designed in Corda, it is easy to transform it into a bounded step algorithm $\mathcal{A}_b$ with steps Upp_i, hence into an algorithm that works also in SYm. In fact, it is sufficient to give in input to the algorithm also the maximum amount of distance each robot r_i can travel at each cycle, Upp_i. Let us fix $\delta_{r_i} \leq Upp_i$, for all $1 \leq i \leq n$. Moreover, let p_i be the position of r_i when it executes $\mathcal{A}$ at a given time, and $dest_i$ be the destination point returned by this computation. Then, the transformed algorithm $\mathcal{A}_b$ returns a point p on the segment $[p_i, dest_i]$ such that $dist(p_i, p) = Upp_i$. Viceversa, by fixing $\delta_{r_i} \leq Upp_i$, for all $1 \leq i \leq n$, it is trivial to show that a bounded step algorithm can be executed in Corda too. Thus,

Fact 3.3.1. Let $\mathcal{A}_b$ be a bounded step algorithm obtained by transforming an algorithm $\mathcal{A}$ that solves a problem $\mathcal{P}$ in Corda as described above. If $Upp_i \geq \delta_{r_i}$, for all $1 \leq i \leq n$, then $\mathcal{A}_b$ still solves $\mathcal{P}$ in Corda. ⋈

In the following lemma, we answer to a further question. Namely, we highlight the relationship between an activation schedule in SYm and a synchronous activation schedule in Corda.

Lemma 3.3.1. *Let $\mathcal{A}$ be a deterministic bounded step algorithm, with $Upp_i = \sigma_i \geq \delta_{r_i}$, for all $1 \leq i \leq n$; and let* ActSYm $= A_0, A_1, \ldots$ *be an activation schedule for $\mathcal{A}$ in* SYm. ActSYm *can be transformed into a synchronous activation schedule Sync$\mathcal{F}$ for $\mathcal{A}$ in* Corda *such that, a configuration $\mathbb{G}$ is reached at time t by executing*

$\mathcal{A}$ in SYm *according to* ActSYm *(the* SYm*'s execution) if and only if the same configuration is reached at time $t4\rho$ by executing $\mathcal{A}$ in* Corda *according to $Sync\mathcal{F}$ (the* Corda*'s execution), if the robots start from the same initial configuration.*

Proof. At the beginning $Sync\mathcal{F}$ is empty, and $t = 0$. Let $start_1, \ldots, start_n$ be the positions of the robots at the beginning. The transformation is obtained by applying the following rules for all the robots in the system:

1. If $r_i \in A_0$, then $Sync\mathcal{F}[i][1] := NA(0, start_i, pos')$, with pos' the destination point computed in the *Compute* state executed at time 2ρ. Otherwise, $Sync\mathcal{F}[i][1] := LW(0, start_i)$. Set $t := 1$.

2. If $r_i \in A_t$, then $Sync\mathcal{F}$ is extended with a normal activation of r_i; otherwise (r_i is inactive in ActSYm at time t), $Sync\mathcal{F}$ is extended with a long wait of r_i. Set $t = t + 1$, and go to 2.

Let us execute $\mathcal{A}$ in SYm and Corda according to ActSYm and $Sync\mathcal{F}$, respectively. Moreover, let $\mathbb{G}$ be the configuration reached at time t by executing $\mathcal{A}$ in SYm according to ActSYm. In the following we will prove by induction that the same configuration is reached at time $t4\rho$ by executing $\mathcal{A}$ in Corda according to $Sync\mathcal{F}$. Since in the two executions the robots start from the same initial configuration, the statement is trivially true at the beginning ($t = 0$). Let us assume it is true for $t \geq 0$, and let us prove it for $t+1$. In the SYm's execution the robots are in $\mathbb{G}$ at time t; let $\mathbb{G}'$ be the configuration they reach at time $t + 1$. By inductive hypothesis, the robots are in $\mathbb{G}$ at time $t4\rho$ in Corda's execution too. Let us consider all the robots in A_t. It follows from the definition of the above transformation that all these robots are normally activated at time $t4\rho$ in $Sync\mathcal{F}$, while all the others start a long wait. Since the same algorithm is executed in both SYm and Corda at every cycle, and all these robots execute their states in $Sync\mathcal{F}$ between time $t4\rho$ and $(t + 1)4\rho$ perfectly synchronized, it follows that all these robots will compute the destination points computed in SYm's execution at time t; hence $\mathbb{G}'$ is reached at time $t + 1$ in SYm and at time $t4\rho + 4\rho$ in Corda, and the statement follows.

Similarly, it can be proven by induction that, if $\mathbb{G}$ is a configuration reached at time $t4\rho$ in Corda's execution, then the same configuration is reached at time t in SYm's execution, if the robots start from the same initial configuration. □

Let us denote by $\mathfrak{C}$ and $\mathfrak{Z}$ the class of problem that are solvable in Corda and SYm, respectively. We are now ready to show that SYm is at least as powerful as Corda, that is $\mathfrak{C} \subseteq \mathfrak{Z}$.

Theorem 3.3.1. *Any algorithm that correctly solves a problem $\mathcal{P}$ in* Corda*, correctly solves $\mathcal{P}$ also in* SYm*.*

Proof. Let $\mathcal{A}$ be an algorithm that solves a given problem $\mathcal{P}$ in CORDA, and let us fix $\delta_{r_i} \leq \sigma_i$, for all $1 \leq i \leq n$. Then, using the technique observed at the beginning of this section, $\mathcal{A}$ can be transformed into a bounded step algorithm $\mathcal{A}_b$, with $Upp_i = \sigma_i$, for all $1 \leq i \leq n$. By Fact 3.3.1, $\mathcal{A}_b$ still solves the problem in CORDA; furthermore, it can be executed in SYm too. Let ActSYm $= A_0, A_1, \ldots$ be an activation schedule for $\mathcal{A}_b$ in SYm. Following the transformation described in Lemma 3.3.1, ActSYm can be transformed into a synchronous activation schedule $Sync\mathcal{F}$ for $\mathcal{A}$ in CORDA.

Since $\mathcal{A}_b$ solves $\mathcal{P}$ in CORDA, it follows that $\mathcal{P}$ is solved in CORDA when executing $\mathcal{A}_b$ according to $Sync\mathcal{F}$; that is, in finite time the robots reach a configuration such that the task defined by $\mathcal{P}$ is accomplished. By Lemma 3.3.1, if the robots are in a certain configuration in CORDA's execution (according to $Sync\mathcal{F}$) at time $t4\rho$, then the robots are in the same configuration at time t in the SYm's execution. That is, all the configurations in CORDA's execution every 4ρ time happen also in SYm's execution, and there is no configuration in SYm's execution that is not in CORDA's execution. Hence, the theorem follows. □

Corollary 3.3.1. *Any problem that can be solved in* CORDA*, can be solved in* SYm*; hence* $\mathfrak{C} \subseteq \mathfrak{Z}$.

The viceversa is not in general true. In fact, the proof of Theorem 3.3.1 relies on the fact that an activation schedule in SYm can be transformed into a synchronous activation schedule in CORDA. In general, however, the executions in CORDA do not happen in a synchronized fashion; hence, knowing that a problem is solvable in SYm does not help.

To prove that the inclusion is strict, we place ourselves in the *non oblivious* setting: the robots have an unlimited amount of memory, hence they can remember the positions of all the other robots since the beginning of the execution, and they can use this information in their computations.

Definition 3.3.2 (Movement Awareness). The *Movement Awareness* problem $\mathcal{MA}$ is divided in two subtasks $\mathcal{T}_1$ and $\mathcal{T}_2$. In $\mathcal{T}_1$, robot r_i, $1 \leq i \leq n$, simply moves along a direction it chooses arbitrarily; r_i can start $\mathcal{T}_2$ only after it observed r_j in at least three different positions, and after r_j observed r_i in at least three different positions, for all $j \neq i$. ◁▷

Theorem 3.3.2. *There exists no algorithm that solves* $\mathcal{MA}$ *in* CORDA *in the non oblivious setting.*

Proof. By contradiction, let us assume that there exists an algorithm $\mathcal{A}$ that correctly solves $\mathcal{MA}$ in CORDA. The generic robot r starts its execution by moving

along the direction it chooses. By hypothesis, it will eventually and within a finite number of cycles start the second subtask. Let t be the time when r decides to switch to $\mathcal{T}_2$. Since the robots operate in full asynchronicity, there can exist a robot r' that started its first *Move* state at time $t' < t$, and is still moving at time t (that is r' is still executing its first cycle). Then $\mathcal{MA}$ is not correctly solved, since r' has not started its second cycle at time t yet, hence r' has not observed r in at least three different positions yet, having a contradiction. □

An algorithm similar to the one used in [78] to discover the initial configuration ("distribution") of the robots in the system, can be used to solve in SYm $\mathcal{MA}$. Namely, each robot starts moving along the direction it locally chooses, e.g. the direction of its local y axis. When a robot r observes another robot r' in at least three different positions, r moved at least twice. Moreover, since in SYm the actions are instantaneous, r can correctly deduce that r' observed at least twice, hence that r' observed r in at least three different positions. Therefore, r can correctly start $\mathcal{T}_2$ when it observes all $r' \neq r$ in at least three different positions. Hence, we can state the following

Theorem 3.3.3. *$\mathcal{MA}$ is solvable in* SYm*, in the non oblivious setting.*

Corollary 3.3.2. $\mathfrak{C} \subset \mathfrak{Z}$.

A question that arises is: what does it happen in the oblivious case? Unfortunately, we do not yet have an answer. Our conjecture, however, is that the result stated in Corollary 3.3.2 holds also in the oblivious case. In the non oblivious setting, the fact that in Corda a robot can be seen by its fellows *while* it is moving is crucial to prove $\mathfrak{C} \subset \mathfrak{Z}$. This is not the case in the oblivious setting. In fact, since the robots have no memory of robots' positions observed in the past, every time a robot r observes another robot r', r can not tell if r' moved since last cycle or not, and every observation is like the first one (that is every time r observes, is like the execution begins). Hence, we believe that the key to prove $\mathfrak{C} \subset \mathfrak{Z}$ in the oblivious case is related to the fact that in Corda the positions of the robots between a *Look* and a *Compute* can change, hence the computation can be done on "outdated" data. In other words, if r executes the *Look* at time t and the *Compute* at time $t' > t$, the set of robots' positions at t and at t' can be clearly different; hence r computes its destination point on the old data sensed at time t, implying that the movement will not be "choerent" with what it observed at time t. This clearly does not happen in SYm, where the possible states a robot can be in are executed instantaneously.

3.4 Case Study: Oblivious Gathering

In this section, we will give evidence that the algorithms designed in SYm in the oblivious setting do not work in general in CORDA.

The problem we consider is the *gathering problem*: the robots are asked to gather in a not predetermined point in the plane in a finite number of cycles, call it $\mathfrak{p}$. An algorithm is said *to solve* the gathering problem if it lets the robots gather in a point, given any *valid* initial configuration. A valid initial configuration for this problem is the set of robots' positions when the computation starts, one position per robot, with no position occupied by more than one robot. This is the only problem, to our knowledge, solved with an oblivious algorithm in SYm [3, 78]. In the following, we will analyze both the unlimited and limited visibility setting.

3.4.1 The Unlimited Visibility Setting

A deterministic and oblivious algorithm for solving the gathering problem in SYm in the unlimited visibility setting (called Algorithm 1 in Appendix A.1) is presented in [78]. The idea is as follows. Starting from distinct initial positions, the robots are moved in such a way that eventually there will be exactly one position $\mathfrak{p}$ that two or more robots occupy. Once such a situation has been reached, all the robots move towards $\mathfrak{p}$. It is clear that such a strategy works only if the robots in the system have the ability to detect multiplicity. In SYm this capability is never mentioned, but it is clearly used implicitly.

Theorem 3.4.1. *Algorithm 1 does not solve the gathering problem in* CORDA*, in the unlimited visibility setting.*

Proof. In the following we show that Algorithm 1 does not solve the gathering problem in CORDA. Specifically, we give an initial configuration of the robots and describe an activation schedule that leads to having two points in the plane with multiplicity greater than two, thus violating the invariant proven for Algorithm 1, that "eventually there will be exactly one position that two or more robots occupy" [78]. As noted in Section 3.3, Algorithm 1 is a bounded step algorithm; hence it can be executed in CORDA by fixing $\delta_{r_i} \leq \sigma_i$, for all $1 \leq i \leq n$.

Let us suppose to have 4 robots r_i, $i = 1, 2, 3, 4$, that at the beginning are on a circle C, with r_2 and r_4 that occupy the ending points of a diameter of C (as pictured in Figure 3.5, Cycle 1). In the following, the positions of the robots are indicated by p_i, $i = 1, 2, 3, 4$. Executing Algorithm 1, but assuming the features of CORDA, a possible run is described in the following.

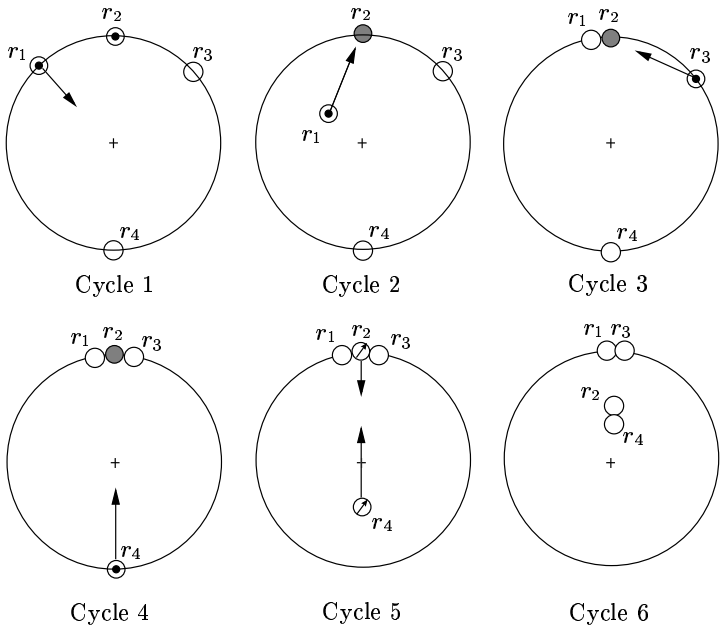


Figure 3.5: Proof of Theorem 3.4.1. The dotted circles indicate the robots in the *Look* state; the grey ones the robots in the *Compute* state; the circle with an arrow inside are the robots that are moving; the white circles represent the robots in *Wait*. The arrows indicate the direction of the movement computed in the *Compute* state.

Cycle 1 At the beginning the four robots are in distinct positions, on a circle C. r_1 and r_2 enter the *Look* state, while the others are in *Wait*. After having observed, both of them enter the *Compute* state, and let us assume that r_2 is computationally very slow (or, alternatively, that r_1 is very fast). Therefore, r_1 decides to move towards the center of C (part 2.3 of Algorithm 1), while r_2 is stuck in its *Compute* state. r_1 starts moving towards the center, while r_2 is still in *Compute*, and r_3 and r_4 are in *Wait*.

Cycle 2 r_1 is inside C, while the other robots are still on C. Now r_1 observes again (already in its second cycle) and, according to part 2.1 of the algorithm, decides to move toward a robot that is on the circle, say r_2. Moreover, r_2 is still in the *Compute* state of its first cycle, and r_3 and r_4 are in *Wait*.

Cycle 3 r_1 reaches r_2 and enters the *Wait* of its third cycle: at this point, there is one position in the plane with two robots, namely $p = p_1 = p_2$. Now, r_3 enters its first *Look* state, looks at the situation and, according to the algorithm, decides to move towards p, that is the only point in the plane with more than ore robots on it. r_2 is still in its first *Compute*, and r_4 in *Wait*.

Cycle 4 r_3 reaches r_1 and r_2 on p, and it starts waiting. r_1 is in *Wait*, r_2 still in its first *Compute* state, and r_4 starts its first *Look* state, decides to move towards p, and starts moving.

Cycle 5 While r_4 is on its way towards p, r_2 ends its first *Compute* state. Since the computation is done according to what it observed in its previous *Look* state (Cycle 1), it decides to move towards the center of C (part 2.3 of the algorithm). r_2 starts moving towards the center of C after r_4 passes over the center of C, and while r_4 is still moving towards p; r_1 is in *Wait*.

Cycle 6 r_2 and r_4 are moving in opposite directions on the same diameter of C, and they stop exactly on the same point p' (in CORDA a robot can stop before reaching its final destination). There are two points in the plane, namely p and p' with $p \neq p'$, with two robots on each. Therefore, the invariant proven for Algorithm 1, that "eventually there will be exactly one position that two or more robots occupy" [78], is violated. □

Remark 3.4.1. We note that in Cycle 6 we made use of the possibility that a robot stops before reaching the destination point it computed. The proof, however, works even if we do not assume this; that is, if r_2 and r_4 do not stop before reaching their respective destination points. In fact, if we assume, as in SYm, that the robots simply cross each other without stopping, if (i) the crossing happens in a point $p' \neq p$, and (ii) r_1 enters its *Look* state exactly when the crossing happens, we have that r_1 sees two points in the plane with two robots on each, namely p and p', and does not know what to do, since this possibility is not mentioned in SYm's algorithm. Therefore, Theorem 3.4.1 still holds.

Remarks on Multiplicity Detection

We stress again that in the solution proposed in [78] the ability of the robots to detect multiplicity (i.e., if on a given point there is more than one robot) is used implicitily. It is possible, however, to prove that such a capability is indeed necessary to solve the problem, in both CORDA and SYm. In the remaining of this section, we will prove that no oblivious algorithm can be designed to solve the gathering problem when the robots cannot detect multiplicity, in the unlimited visibility setting. We first prove that multiplicity detection is necessary in order to solve the gathering problem in CORDA, in the oblivious setting. Then, we will extend the result to SYm.

In the following we assume that the n robots in the system cannot detect multiplicity, and that they execute only deterministic and oblivious algorithms. In

particular, we denote by $\mathcal{A}$ a generic deterministic and oblivious algorithm, and by $\mathcal{A}_g$ an oblivious deterministic algorithm that correctly solves the gathering problem in Corda. Recall that $\mathcal{A}_g$ solves the gathering problem if, starting from any valid initial configuration, it lets the robots gather on the same point $\mathfrak{p}$ in finite time. Furthermore, the robots must gather regardless their local unit measures, and the local orientation of their axes. Moreover, let $\mathbb{H}$ be a set of robots that at time t lie all togheter on the same point on the plane: in the following, we indicate such a position by $p^t_{\mathbb{H}}$, and by $|\mathbb{H}|$ the number of robots in $\mathbb{H}$.

Let us start with two general lemmas, that are not specific to the gathering problem. The first one stresses out the fact that, if a set of robots that at a given time instant t lie on the same position of the plane are normally activated all togheter at time t, then they will behave like they were one robot.

Lemma 3.4.1. *Let $\mathbb{H}$ be a set of robots that at time t lie all on the same point $p^t_{\mathbb{H}}$. If all the robots in $\mathbb{H}$ are normally activated at time t, then at time $t + 4\rho$ all the robots in $\mathbb{H}$ will again lie on the same position (possibly different from $p^t_{\mathbb{H}}$).*

Proof. All the robots in $\mathbb{H}$ start a *Look* state at time $t + \rho$. Since at this time they all occupy the same position $p^t_{\mathbb{H}}$, clearly they all have the *same* view of the world. Let $r \in \mathbb{H}$, and let p' be the destination point it computes after the execution of the *Compute* state it starts at time $t + 2\rho$. The lemma follows from the fact that all the robots are normally activated at the same time, and that they execute the same oblivious and deterministic algorithm at time $t + 2\rho$, whose result is based only on the observation done in the *Look* state started at time $t + \rho$. □

The following lemma points out that, if all the robots in the system take the decision to move towards a point p at the same time instant t, then, even if a subset of them is blocked, all the others will still move towards p.

Lemma 3.4.2. *Let us assume that normally activating all the robots at time t they gather on the same point p at time $t+4\rho$, and let $\mathbb{H}$, with $1 \leq |\mathbb{H}| < n$, be a subset of robots that are not on p at t. If all the robots not in $\mathbb{H}$ are still normallly activated at t, while all the robots in $\mathbb{H}$ start a long wait at t, then all the r_i, $r_i \notin \mathbb{H}$, are on p at $t + 4\rho$, while all the robots in $\mathbb{H}$ are not.*

Proof. Let us consider the two scenarios:

(I) All the robots are normally activated at time t. Let $\mathbb{G}$ be the configuration of the robots at time $t + 2\rho$, that is when all the robots just finished observing, and they are starting the *Compute* state.

(II) All the robots not in $\mathbb{H}$ are normally activated at time t, while all the robots in $\mathbb{H}$ start a long wait at time t. Let $\mathbb{G}'$ be the configuration of the robots at time $t + 2\rho$.

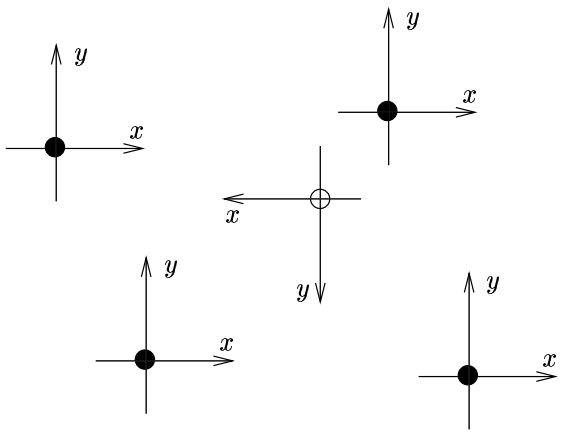


Figure 3.6: Orientation of the axes of the black robots and of the white robot, in Assum3.

Since the robots execute an oblivious algorithm, the result of the *Compute* state started at time $t+2\rho$ in (I) depends on $\mathbb{G}$; similarly, the result of the *Compute* state started at time $t+2\rho$ in (II) depends on $\mathbb{G}'$. In both scenarios, no robot moves before time $t+3\rho$ (when the *Move* starts); hence $\mathbb{G}' = \mathbb{G}$. Therefore, since by hypothesis in scenario (I) all the robots decide to move towards p at time $t+3\rho$, and the robots in $\mathbb{H}$ start a long wait at t, also in scenario (II) all $r_i \notin \mathbb{H}$ decide to move towards p, reaching it at time $t+4\rho$, while all the robots in $\mathbb{H}$ are still on the position they occupied at time t (they start a long wait at t). Hence, the lemma follows. □

The general idea to prove that the multiplicity detection is necessary in order to solve the gathering problem is as follows. First, we define a scenario that we will use to defeat any possible $\mathcal{A}_g$. In particular, in this scenario

Assum1. all the robots have the same unit distance;

Assum2. $\delta = \delta_1 = \ldots = \delta_n$ (see Assumption A2 in Section 3.1.1);

Assum3. robots $r_1, \ldots, r_{n-1}$, from now on the *black* robots, have the same orientation and direction of the local coordinate system, while r_n, from now on the *white* robot, has a local coordinate system where both axes have the same direction but opposite orientation with respect to the coordinate system of the black robots (see Figure 3.6). In the following, we denote by p_w^t the position of the white robot at time t. The black and white coloring is used only for the sake of presentation, and this information is not used by the robots during the computation.

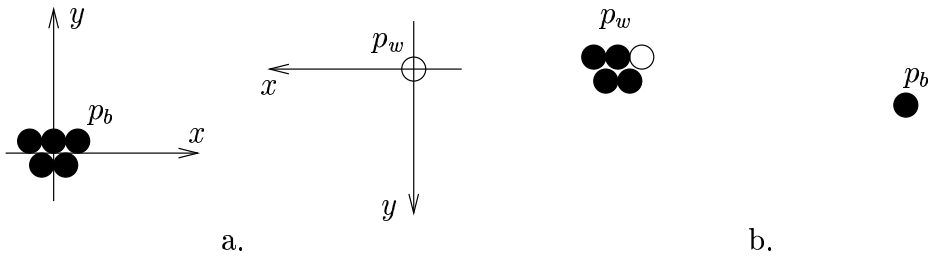


Figure 3.7: In (a) an $\mathbb{E}_1$-configuration is depicted, while in (b) an $\mathbb{E}_2$-configuration. By Assum3. and since the robots cannot detect multiplicity, in both configurations (and in general in any $\mathbb{E}$-configuration) the white robot has the *same* view of the world as the robots in $\mathbb{B}$.

We want to stress out, however, that Assum1.-Assum3. are not known to the robots; hence they cannot use these information in their computations. We will study the multiplicity detection matter under these three assumptions because, if there exists $\mathcal{A}_g$, then it must work also in a scenario described by Assum1-Assum3. Otherwise, $\mathcal{A}_g$ is not an algorithm that correctly solves the gathering problem.

Second, we indeed show that there exists no $\mathcal{A}_g$ that can be executed in such a scenario according to a synchronous activation schedule and that allows the robots to gather in a point in finite time, when they cannot detect multiplicity. Therefore, since any algorithm that solves the problem should work also under Assum1.-Assum3., and when executed according to a synchronous activation schedule, we can conclude that the multiplicity detection is necessary in order to solve the gathering problem. More specifically, we first show that, given $\mathcal{A}_g$, there exists always a synchronous activation schedule that brings the robots in a particular configuration, called $\mathbb{E}$-configuration, in a finite number of cycles. Then, we prove that $\mathcal{A}_g$, starting from an $\mathbb{E}$-configuration, lets the robots loop between $\mathbb{E}$-configurations, always avoiding the gathering.

Definition 3.4.1 ($\mathbb{E}$-configuration). An $\mathbb{E}$-configuration is a configuration of the robots where (i) the black robots are partitioned in two groups $\mathbb{B}$ and $\mathbb{B}'$, with $\mathbb{B}'$ possibly empty; (ii) the robots in $\mathbb{B}'$ and the white robot r_n lie on the same position p_w, and (iii) the robots in $\mathbb{B}$ lie on a position $p_{\mathbb{B}} \neq p_w$. Moreover, $\mathbb{E}_1$-comfiguration (shortly $\mathbb{E}_1$) is the $\mathbb{E}$-configuration where $\mathbb{B}' = \emptyset$ (see Figure 3.7.a), and $\mathbb{E}_2$-configuration (shortly $\mathbb{E}_2$) is the $\mathbb{E}$-configuration where $|\mathbb{B}| = 1$ and $|\mathbb{B}'| = n - 2$ (see Figure 3.7.b). ◁▷

Before showing that an $\mathbb{E}$-configuration can be reached by executing any $\mathcal{A}_g$ according to a particular synchronous activation schedule, we want to highlight an

interesting property of the $\mathbb{E}$-configuration. Namely, if normally activating $r \in \mathbb{B}$ at the beginning of the computation it does not change position after 4ρ time, than no robots will ever move. The same thing can be simmetrically proven for the white robot. Formally, let $SyncGen(t^*, t^{**}, pos)$ be a function that, given two time instants t^* and t^{**}, with $t^{**} \geq t^* + 4\rho$, and an absolute position in the plane, returns a random synchronous activation sequence $\mathcal{S}$ such that $\mathcal{S}[1].pos_s = pos$, $\mathcal{S}[1].t_s = t^*$, and $\mathcal{S}[Last].t_s \leq t^{**} \leq \mathcal{S}[Last].t_s + 4\rho$. If $t^* = t^{**}$, then $SyncGen(t^*, t^{**}, pos)$ returns the empty synchronous activation schedule.

Lemma 3.4.3. *Let us assume that all the robots in the system can not detect multiplicity, that they are executing the same oblivious deterministic algorithm $\mathcal{A}$, and that at a given time t they are in a $\mathbb{E}$-configuration. Let $t' = t + 4k\rho$, with $k \geq 1$ a positive integer.*

Part a. *If after executing $\mathcal{A}$ according to $Sync\mathcal{F}$, with*

$$Sync\mathcal{F}[i] = \begin{cases} NA(t, p^t_{\mathbb{B}}, pos') \circ SyncGen(t + 4\rho, t', pos') & \text{if } r_i \in \mathbb{B} \\ SyncGen(t, t', p^t_w) & \text{if } i = n \\ \bigcirc_{i=0}^{k-1} LW(t + 4i\rho, p^t_w) & \text{if } r_i \in \mathbb{B}' \end{cases}$$

and pos' the result of the Compute state executed by $r_i \in \mathbb{B}$ at time $t + 2\rho$, all the robots in $\mathbb{B}$ are still on $p^t_{\mathbb{B}}$ at time $t + 4\rho$, then all the robots never change position between time t and t' (by $\bigcirc_{i=0}^{k} LW(t + 4i\rho, p^t_w)$ we denote the activation sequence $LW(t, p^t_w) \circ LW(t + 4\rho, p^t_w) \circ \ldots \circ LW(t' - 4\rho, p^t_w)$).

Part b. *If after executing $\mathcal{A}$ according to $Sync\mathcal{F}$, with*

$$Sync\mathcal{F}[i] = \begin{cases} NA(t, p^t_w, pos') \circ SyncGen(t + 4\rho, t', pos') & \text{if } i = n \\ SyncGen(t, t', p^t_{\mathbb{B}}) & \text{if } r_i \in \mathbb{B} \\ \bigcirc_{i=0}^{k-1} LW(t + 4i\rho, p^t_w) & \text{if } r_i \in \mathbb{B}' \end{cases}$$

and pos' the result of the Compute state executed by r_n at time $t+2\rho$, the white robot is still on p^t_w at time $t + 4\rho$, then all the robots never change position between time t and t'.

Proof. In the following we will prove by induction on k the Part a. of the lemma (Part b. can be proven symmetrically). For the basis of the induction we need to show that, if the robots in $\mathbb{B}$ are still on $p^t_{\mathbb{B}}$ at time $t + 4\rho$, no robot changes position between time t and time $t + 4\rho$. By definition of $Sync\mathcal{F}$, all the robots in $\mathbb{B}'$ start a long wait at time t, hence they cannot move. The white robot can either be normally activated or can start a long wait at t. We distinguish the two cases.

Base1. If the white robot is normally activated at t, then both r_n and the robots in $\mathbb{B}$ start a *Look* state at time $t+\rho$: since by hypothesis all the robots can not detect multiplicity, it follows by Assum1. and Assum3. above that at this time they have the *same* view of the world (see Figure 3.7). As a consequence, since $\mathcal{A}$ is oblivious and deterministic, the result of the computation that the white robot starts at time $t+2\rho$ is the same as the result of the computation that the robots in $\mathbb{B}$ start executing at the same time. Hence, since by hypothesis the robots in $\mathbb{B}$ do not change position after the first normal activation, the white robot must decide not to move, and it will still be on p_w^t at time $t+4\rho$, and the basis of the induction follows.

Base2. If the white robot starts a long wait at t, then between time t and $t+4\rho$ it cannot change position, hence the basis of the induction follows.

Now, let us assume as inductive hypothesis that no robot changes position between t and $t'' = t + 4k'\rho$, with $1 \leq k' < k$; that is at time t'' the robots in $\mathbb{B}$ are still on $p_{\mathbb{B}}^t$, and the robots in $\mathbb{B}'$ and the white robot on p_w^t; hence, all the robots are still in a $\mathbb{E}$-configuration. In order to prove the lemma, we need to show that no robot changes position between time t'' and time $t''+4\rho$. By definition of $Sync\mathcal{F}$, all the robots in $\mathbb{B}'$ start a long wait at time t, hence they cannot move. On the other hand, at time t'' the robots in $\mathbb{B}$ and the white robot can either be normally activated or start a long wait. We distinguish the possible cases.

1. If both the white robot and the robots in $\mathbb{B}$ are normally activated at t'', we can use an argument similar to the one used in case Base1. above, and the lemma follows.

2. If both the white robot and the robots in $\mathbb{B}$ starts a long wait a t'', then clearly no robots can change position between time t'' and $t''+4\rho$, and the lemma follows.

3. If the white robot starts a long wait at t'', while the robots in $\mathbb{B}$ are normally activated at t'', then the white robot is clearly still on p_w^t at time $t''+4\rho$. Furthermore, at time $t''+\rho$ all the robots in $\mathbb{B}$ start a *Look* state: by inductive hypothesis and since a robot can change position only after it executes a *Move*, at this time they have the *same* view of the world they had at time $t+\rho$ (when they executed their first *Look* state after time t). It follows that, since $\mathcal{A}$ is oblivious and deterministic, the result of the *Compute* state at time $t''+2\rho$ must be the same as the result of the *Compute* state at time $t+2\rho$, hence they will still be on $p_{\mathbb{B}}^t$ at time $t''+4\rho$, and the lemma follows.

	t_s	$t_s + 4\rho$	$\cdots$	$t_{\mathbb{E}} - 4\rho$	$t_{\mathbb{E}}$	$t_{\mathbb{E}} + 4\rho$
$Sync\mathcal{S}_1$	LW	NA	$\cdots$	NA	LW	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$Sync\mathcal{S}_k$	LW	NA	$\cdots$	LW	LW	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$Sync\mathcal{S}_n$	LW	NA	$\cdots$	NA	LW	

Figure 3.8: The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}}$ described in Lemma 3.4.4.

4. If the white robot is normally activated at t, while the robots in $\mathbb{B}$ start a long wait at t'', then at time $t'' + 4\rho$, the robots in $\mathbb{B}$ are clearly still on $p^t_{\mathbb{B}}$. Furthermore, by inductive hypothesis and by Assum1. and Assum3. above, the view of the world of the white robot at time $t'' + \rho$ is the *same* as the view of the world of the robots in $\mathbb{B}$ at time $t + \rho$ (see Figure 3.7). Therefore, since $\mathcal{A}$ is oblivious and deterministic, the white robot can not decide to change position, hence it will still be on p^t_w at time $t'' + 4\rho$, and the lemma follows.□

Now, we are ready to show that an $\mathbb{E}$-configuration can be reached by executing $\mathcal{A}_g$ according to a specific synchronous activation schedule.

Lemma 3.4.4. *Given $\mathcal{A}_g$, there exists a synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}}$ for $\mathcal{A}_g$, and a time $t_{\mathbb{E}} > 0$ such that, if the robots do not all occupy the same position on the plane when the execution of $\mathcal{A}_g$ starts, they are in $\mathbb{E}_1$ or $\mathbb{E}_2$ at time $t_{\mathbb{E}}$, if the computation is done according to $Sync\mathcal{F}_{\mathbb{E}}$. That is, there exists an absolute position on the plane $\tilde{p}$ such that*

$$\exists\ r_k \mid \forall\ 1 \leq i \neq k \leq n, Sync\mathcal{F}[i][z_{\mathbb{E}}].pos_s = \tilde{p} \wedge Sync\mathcal{F}[k][z_{\mathbb{E}}].pos_s \neq \tilde{p},$$

with $z_{\mathbb{E}} \geq 1$ the index such that $Sync\mathcal{F}[i][z_{\mathbb{E}}].t_s = t_{\mathbb{E}}$, for all $1 \leq i \leq n$. Moreover, at $t_{\mathbb{E}}$ all the robots are executing a long wait.

Proof. Let t_s be the time when the computation starts, and $pos_1, \ldots, pos_n$ be the positions occupied by the robots at this time. By hypothesis, there exist at least two positions pos_i and pos_j, $i \neq j$, such that $pos_i \neq pos_j$. $Sync\mathcal{F}_{\mathbb{E}}$ is built according to the following routine (refer to Figure 3.8 for a pictorial representation):

Build$_{\mathbb{E}}(t_s, pos_1, \ldots, pos_n)$:

Init. $Sync\mathcal{F}_{\mathbb{E}}[i] := LW(t_s, pos_i)$, for all $1 \leq i \leq n$. Set $t = t_s + 4\rho$, and go to Rule1.

Rule1. If normally activating all the robots at time t, they are not on the same point $\tilde{p}$ at time $t + 4\rho$, then in $Sync\mathcal{F}_{\mathbb{E}}$ all r_i are normally activated at t. Set $t = t + 4\rho$, and go to Rule1. Otherwise,

	t_s Init.	B1.	W1.		B2.1	B2.2	B2.3	W1.
$Sync\mathcal{S}_1$	LW	NA	LW	$\cdots$	NA	LW	LW	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$
$Sync\mathcal{S}_{n-2}$	LW	NA	LW	$\cdots$	NA	LW	LW	$\cdots$
$Sync\mathcal{S}_{n-1}$	LW	NA	LW	$\cdots$	LW	LW	NA	$\cdots$
$Sync\mathcal{S}_n$	LW	LW	NA	$\cdots$	LW	NA	LW	$\cdots$

Figure 3.9: The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_1}$ described in Lemma 3.4.5. Here is depicted the case when B2. is invoked first.

Rule2. let r_k be a robot that is not on $\tilde{p}$ at time t. Then, in $Sync\mathcal{F}_{\mathbb{E}}$ all r_i, $i \neq k$, are normally activated at t, while r_k starts a long wait at time t. At time $t_{\mathbb{E}} = t + 4\rho$, all the robots start a long wait.

In the following, we will show that, starting the execution of $\mathcal{A}_g$ at time t_s according to $Sync\mathcal{F}_{\mathbb{E}}$, all the robots are in a $\mathbb{E}_1$-configuration or $\mathbb{E}_2$-configuration at time $t_{\mathbb{E}} > t_s$. In fact, since by hypothesis $\mathcal{A}_g$ solves the problem, after finite time Rule2. is executed; hence $Sync\mathcal{F}_{\mathbb{E}}$ is obtained by a finite number of extensions and $t_{\mathbb{E}}$ is finite. Moreover, until $t_{\mathbb{E}} - 4\rho$ all the robots are normal activated; after this time, each robot starts at most two long waits (in particular, r_k two, and all the others one). It follows that $Sync\mathcal{F}_{\mathbb{E}}$ is a synchronous activation schedule. By construction, $t_{\mathbb{E}}$ is the first time such that, if all the robots were normally activated at time $t_{\mathbb{E}} - 4\rho$, they would be on the same position $\tilde{p}$ at time $t_{\mathbb{E}}$. Therefore, since there exists at least two positions pos_i and pos_j at time t_s such that $pos_i \neq pos_j$, there must exist at least one robot r_k that is not on $\tilde{p}$ at time $t_{\mathbb{E}} - 4\rho$. According to Rule2., r_k starts a long wait at time $t_{\mathbb{E}} - 4\rho$. By Lemma 3.4.2, at time $t_{\mathbb{E}}$ all the robots r_i, $i \neq k$ are on $\tilde{p}$, and r_k is on a position different from $\tilde{p}$. Furthermore, all the robots start a long wait at time $t_{\mathbb{E}}$, hence the lemma follows. □

In the following two lemmas, we show that there is no algorithm that, starting from $\mathbb{E}_1$ or $\mathbb{E}_2$, lets the robots gather in a point.

Lemma 3.4.5. *In* Corda *there exists no deterministic oblivious algorithm that, starting from a $\mathbb{E}_1$-configuration, solves the gathering problem in a finite number of cycles for a set of $n \geq 3$ robots that can not detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}_g$ be a deterministic oblivious algorithm that, starting from a $\mathbb{E}_1$-configuration, lets the robots gather in a point in finite time when they cannot detect multiplicity. Let t_s be the time when the computation starts. In the following, we will describe a synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_1}$ for $\mathcal{A}_g$ such that, if the robots are at the beginning in a $\mathbb{E}_1$-configuration and the

computation is done according to $Sync\mathcal{F}_{\mathbb{E}_1}$, the robots never gather in the same point $\mathfrak{p}$.

It follows from the definition of $\mathbb{E}_1$ that at the beginning $p_{\mathbb{B}}^{t_s} \neq p_w^{t_s}$. $Sync\mathcal{F}_{\mathbb{E}_1}$ moves alternatively the black robots (as a group) and the white robot, until at time t either the black robots compute as destination point p_w^t, or the white robot computes as destination point $p_{\mathbb{B}}^t$. When this happens, the black robots and the white robot are forced to switch their positions, avoiding the gathering. In particular, let $pos_1 = \ldots = pos_{n-1} = p_{\mathbb{B}}^{t_s}$, and $pos_n = p_w^{t_s}$. Then, $Sync\mathcal{F}_{\mathbb{E}_1}$ is built according to the following routine (refer to Figure 3.9 for a pictorial representation).

Build$_{\mathbb{E}_1}(t_s, pos_1, \ldots, pos_n)$:

Init. $Sync\mathcal{F}_{\mathbb{E}_1}[i] := LW(t_s, pos_i)$, for all $1 \leq i \leq n$. Set $t = t_s + 4\rho$, and go to B1.

B1. If normally activating one of the black robots at time t, it is not on p_w^t at time $t + 4\rho$, then in $Sync\mathcal{F}_{\mathbb{E}_1}$ all the black robots are normally activated at t and moved to the destination point they compute. The white robot starts a long wait at t. Set $t = t + 4\rho$, and go to W1.

B2. Otherwise,

B2.1 in $Sync\mathcal{F}_{\mathbb{E}_1}$ the black robots $r_1, \ldots, r_{n-2}$ are normally activated at t and moved to the destination point they compute. The black robot r_{n-1} and the white robot r_n start a long wait at t. Set $t = t + 4\rho$.

B2.2 In $Sync\mathcal{F}_{\mathbb{E}_1}$ the white robot is normally activated at t and moved to the destination point it computes. All the black robots start a long wait at t. Set $t = t + 4\rho$.

B2.3 In $Sync\mathcal{F}_{\mathbb{E}_1}$ the black robot r_{n-1} is normally activated at t and moved to the destination point it computes. The black robots $r_1, \ldots, r_{n-2}$ and the white robot r_n start a long wait at t. Set $t = t + 4\rho$, and go to W1.

W1. If normally activating the white robot at time t, it is not on $p_{\mathbb{B}}^t$ at time $t + 4\rho$, then in $Sync\mathcal{F}_{\mathbb{E}_1}$ the white robot is normally activated at t and moved to the destination point it computes. The black robots start a long wait at t. Set $t = t + 4\rho$, and go to B1.

W2. Otherwise,

W2.1 As in B2.1.

W2.2 As in B2.2.

W2.3 As in B2.3, except that at the end of this step go to B1.

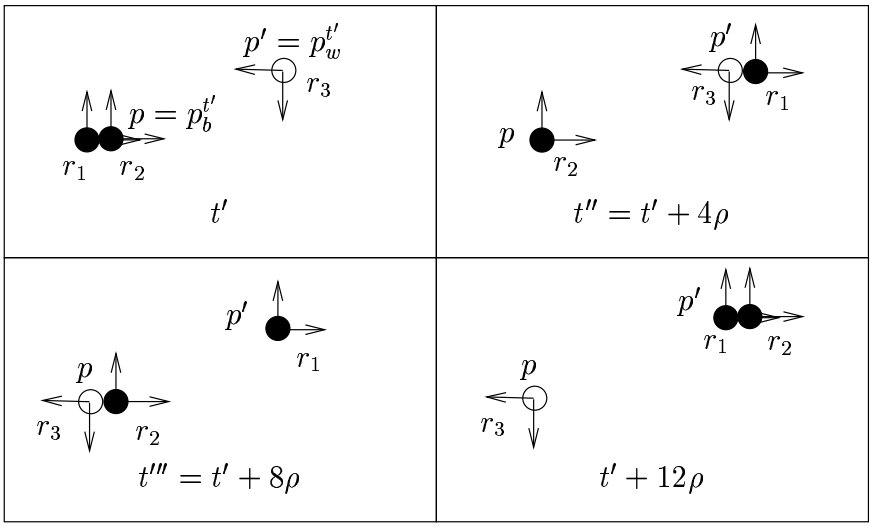


Figure 3.10: Execution of Rule B2. in routine $\mathbf{Build}_{\mathbb{E}_1}()$ in Lemma 3.4.5, with $n = 3$. At time t' each robot sees only one other robot; in particular, r_1 and r_2 see one robot on the point of coordinate (z, z') (with respect to their local coordinate system), and r_3 sees one robot on the point of coordinate (z, z') (with respect to its local coordinate system). That is, all the robots have the *same* view of the world. This view of the world is observed also by r_3 at time t'', and by r_2 at time t'''.

Now, we are ready to show that starting the execution of $\mathcal{A}_g$ at time t_s from the $\mathbb{E}_1$-configuration and according to $Sync\mathcal{F}_{\mathbb{E}_1}$, the robots never gather on the same point. First we note that, after every execution of B1. all the black robots must change position, and move *all together* towards the new destination (different from the position occupied by the white robot). In fact, let $t^* \geq t_s$ be a time instant when B1. starts being executed, and such that all the robots in $\mathbb{B}$ are on the same position $p_{\mathbb{B}}^{t^*} \neq p_w^{t^*}$. It follows from the definition of $Sync\mathcal{F}_{\mathbb{E}_1}$ that all the black robots are normally activated. If all these robots are still on $p_{\mathbb{B}}^{t^*}$ at the end of B1. (that is at time $t^* + 4\rho$), then by Lemma 3.4.3 no robot would ever move, hence the robots would never gather on the same point. Therefore, the robots in $\mathbb{B}$ cannot be on $p_{\mathbb{B}}^{t^*}$ at the end of B1. Furthermore, since there is a black robot that, if normally activated at t^*, would reach a position $p \neq p_w^{t^*}$ at time $t^* + 4\rho$, by Lemma 3.4.1 they will reach all together p at time $t^* + 4\rho$, with $p \neq p_w^{t^*}$ and $p \neq p_{\mathbb{B}}^{t^*}$. Note that, between time t^* and $t^* + 4\rho$ the white robot is in *Wait* on $p_w^{t^*}$, hence it can not see the black ones while they move. Symmetrically, it can be proven that, if W1. starts at time t^*, the white robot will be on a position $p \neq p_w^{t^*}$ and $p \neq p_{\mathbb{B}}^{t^*}$ at time $t^* + 4\rho$, while all the black robots execute a long wait (hence they are still on $p_{\mathbb{B}}^{t^*}$ at time $t^* + 4\rho$). Therefore, as long as B1. or W1. are executed, the robots are in $\mathbb{E}_1$-configurations.

Since by hypothesis $\mathcal{A}_g$ solves the problem, after a finite number of cycles either B2. or W2. is executed. Without loss of generality, let us assume that B2. is executed first, say at time $t' > t_s$ (the case when W2. is executed first can be handled similarly). Thus, according to $Sync\mathcal{F}_{\mathbb{E}_1}$, $n-2$ black robots are normally activated all together at time t', while r_{n-1} and r_n start a long wait. This rule is chosen because there is a black robot that, if normally activated at t', would compute $p_w^{t'}$ as destination point. Hence, by Lemma 3.4.1, the $n-2$ normally activated black robots will leave $p = p_{\mathbb{B}}^{t'}$ and reach $p' = p_w^{t'}$ (while they are moving, everybody else is in *Wait*, therefore they can not be seen while moving; refer to Figure 3.10).

At this point, B2.2 is invoked at time $t'' = t' + 4\rho$: the white robot is normally activated at t'', while all the black robots start a long wait. At time $t'' + \rho$, r_n starts a *Look*: by Assum1–Assum3 and since multiplicity cannot be detected, it has the same view of the world that the black robots that moved in B2.1 had at time $t' + \rho$ (refer to Figure 3.10); specifically, the white robot sees only one robot, that is the last black robot r_{n-1} that at this time is still on p (r_{n-1} started a long wait at t' and t''). As a consequence, since $\mathcal{A}_g$ is oblivious and deterministic, the result of the *Compute* state that r_n starts at time $t'' + 2\rho$ is the same as the result of the *Compute* state that the black robots performed at time $t' + 2\rho$ (in B2.1): that is, r_n decides to reach the only other robot it sees (r_{n-1}), hence r_n computes p as destination point. Therefore, at time $t'' + 4\rho$ the white robot reaches r_{n-1} on p.

Finally, B2.3 is started at time $t''' = t'' + 4\rho$: the last black robot r_{n-1} (still on p) is normally activated at t''', while all the other black robots (at this time on p') and r_n (on p) start a long wait. At time $t''' + \rho$, r_{n-1} starts a *Look*: r_{n-1} has the same view of the world that the black robots that moved in $B2.1$ had at time $t' + \rho$ (refer to Figure 3.10); specifically, since it can not distinguish multiplicity, it sees all the other black robots (on p') as one robot. Therefore it computes p' as destination point, and reaches all the other black robots at time $t''' + 4\rho$.

In conclusion, if B2.1 is started at time t', at time $t''' + 4\rho = t' + 12\rho$ all the black robots are on p', and the white robot is on p. That is, the black and white robots simply switched positions, and at time $t' + 12\rho$ they are again in a $\mathbb{E}_1$-configuration. Therefore, by executing $\mathcal{A}_g$ according to $Sync\mathcal{F}_{\mathbb{E}_1}$, the robots will never gather on the same point. This leads to a contradiction, and the lemma follows. □

Lemma 3.4.6. *In* CORDA *there exists no deterministic oblivious algorithm that, starting from a $\mathbb{E}_2$-configuration, solves the gathering problem in a finite number of cycles for a set of $n \geq 3$ robots that can not detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}_g$ be a deterministic oblivious algorithm that, starting from a $\mathbb{E}_2$-configuration, lets the robots gather in a point in finite time when they cannot detect multiplicity. Let t_s be the time when the computation

	t_s Init.	Rule1.	Rule1.		Rule2.1	B1.
$Sync\mathcal{S}_1$	LW	NA	NA	$\cdots$	NA	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\cdots$
$Sync\mathcal{S}_{n-2}$	LW	NA	NA	$\cdots$	NA	$\cdots$
$Sync\mathcal{S}_{n-1}$	LW	NA	NA	$\cdots$	NA	$\cdots$
$Sync\mathcal{S}_n$	LW	NA	NA	$\cdots$	LW	$\cdots$

Figure 3.11: The synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_2}$ described in Lemma 3.4.6. The case when Rule2.1 is executed first is depicted.

starts. Similarly to the previous lemma, we will describe a synchronous activation schedule $Sync\mathcal{F}_{\mathbb{E}_2}$ for $\mathcal{A}_g$ such that, if the robots are at the beginning in a $\mathbb{E}_2$-configuration and the computation is done according to $Sync\mathcal{F}_{\mathbb{E}_2}$, the robots never gather in the same point $\mathfrak{p}$.

It follows from the definition of $\mathbb{E}_2$ that at the beginning $p_{\mathbb{B}}^{t_s} \neq p_w^{t_s}$. Without loss of generality, let us assume that $r_1, \ldots, r_{n-2}$ are the black robots in $\mathbb{B}'$ (at the beginning they lie on $p_w^{t_s}$), and that r_{n-1} is the only robot in $\mathbb{B}$.

$Sync\mathcal{F}_{\mathbb{E}_2}$ moves all the robots until they decide to gather on the same point (eventually this happens, since by hypothesis $\mathcal{A}_g$ solves the problem); in particular, all the robots in $\mathbb{B}'$ are forced to move togheter, hence to lie always on the same point. When this happens, the robots are forced to be in a $\mathbb{E}_1$-configuration. At this point, $Sync\mathcal{F}_{\mathbb{E}_2}$ behaves exactly like $Sync\mathcal{F}_{\mathbb{E}_1}$ described in the previous lemma, hence it avoids the gathering. Let $pos_1, \ldots, pos_{n-2}, pos_n = p_w^{t_s}$, and $pos_{n-1} = p_{\mathbb{B}}^{t_s}$. Then, $Sync\mathcal{F}_{\mathbb{E}_2}$ is built according to the following routine (refer to Figure 3.11 for a pictorial representation).

Build$_{\mathbb{E}_2}(t_s, pos_1, \ldots, pos_n)$:

Init. $Sync\mathcal{F}_{\mathbb{E}_2}[i] := LW(t_s, pos_i)$, for all $1 \leq i \leq n$. Set $t = t_s + 4\rho$, and go to Rule1.

Rule1. If normally activating all the robots at time t, they are not on the same position $\tilde{p}$ at time $t + 4\rho$, then in $\mathcal{F}_{\mathbb{E}_2}$ all the robots are normally activated. Set $t = t + 4\rho$, and go to Rule1.

Rule2. Otherwise,

Rule2.1 If no robot is on $\tilde{p}$ at time t, then in $Sync\mathcal{F}_{\mathbb{E}_2}$ all the robots in $\mathbb{B}'$ and r_{n-1} are normally activated at t and moved to the destination point they compute. The white robot r_n starts a long wait at t. Set $t = t + 4\rho$, and go to B1. defined in Lemma 3.4.5.

Rule2.2 If r_n is on $\tilde{p}$ at time t, then all the robots in $\mathbb{B}'$ are normally activated at t, while all the robots in $\mathbb{B}$ and r_n start a long wait at t. Set $t = t + 4\rho$, and go to Rule1.

By Lemma 3.4.1, as long as Rule1. is executed, all the robots in $\mathbb{B}'$ move always all togheter; hence, at any time, they always occupy the same position on the plane. Since by hypothesis $\mathcal{A}_g$ solves the problem, after a finite number of cycles Rule2. is executed, say at time t'. First, note that it is impossible that at time t' the robots in $\mathbb{B}'$ and r_n are already on $\tilde{p}$, while the robots in $\mathbb{B}$ are not. In fact, let us assume that r_n and the robots in $\mathbb{B}'$ are already on $\tilde{p}$ at time t'. Then the robots are in a $\mathbb{E}$-configuration at t'. Rule2. is executed at t' because, if all the robots were normally activated at t', they would be on $\tilde{p}$ at time $t' + 4\rho$; hence, since by hypothesis r_n and the robots in $\tilde{p}$ are already on $\tilde{p}$ at time t', these robots would not move between time t' and $t' + 4\rho$. Therefore, is like the robots in $\mathbb{B}'$ start a long wait at t'. Hence, by Lemma 3.4.3, no robot would change position between time t' and $t' + 4\rho$, hence they would not gather on $\tilde{p}$ at time $t' + 4\rho$, and Rule2. would not have been executed at time t'. Similarly, it can be proven that

it is impossible that at time t' the robots in $\mathbb{B}$ and r_n are already on $\tilde{p}$, while the robots in $\mathbb{B}'$ are not (it is sufficient to switch the roles of $\mathbb{B}$ and $\mathbb{B}'$ in Lemma 3.4.3); and

it is impossible that at time t' the robots in $\mathbb{B}$ and those in $\mathbb{B}'$ are already on $\tilde{p}$, while r_n is not.

Moreover, since by hypothesis t' is the first time such that, by normally activating all the robots, they would gather on the same point, it can not be that all the robots are already on $\tilde{p}$ at t'. In the following, we analyze the possible cases.

1. *No robot is on $\tilde{p}$ at time t'.* In this case, Rule2.1 is executed, and r_n starts a long wait at time t'. Hence, by Lemma 3.4.2, at time $t' + 4\rho$ all the robots but r_n are on $\tilde{p}$; that is, the robots are in a $\mathbb{E}_1$-configuration.

2. *Only r_n is already on $\tilde{p}$ at time t'.* In this case, Rule2.2 is executed, and the robots in $\mathbb{B}'$ are normally activated at t', while all the others start a long wait at t'. Hence, by Lemma 3.4.2, at time $t' + 4\rho$ all the robots in $\mathbb{B}'$ and r_n are on $\tilde{p}$, while the robots in $\mathbb{B}$ are not. That is, the robots do not gather in $\tilde{p}$ at $t' + 4\rho$, and they are again in a $\mathbb{E}_2$-configuration.

In conclusion, at time $t' + 4\rho$, either the robots are in a $\mathbb{E}_1$-configuration or in a $\mathbb{E}_2$-configuration. In the first case, the lemma follows by Lemma 3.4.5. In the

second case, either Rule2.2 is never executed again after $t' + 4\rho$, or every time it is executed the robots are once again in a $\mathbb{E}_2$-configuration. In both cases, the lemma follows. □

Finally, we can state that

Theorem 3.4.2. *In* Corda, *there exists no deterministic oblivious algorithm that solves the gathering problem in a finite number of cycles for a set of $n \geq 3$ robots that cannot detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}_g$ be an oblivious deterministic algorithm that solves the gathering problem when the robots cannot detect multiplicity. We show that, for any valid initial configuration, there exists a synchronous activation schedule such that, by executing $\mathcal{A}_g$ according to it, the robots never gather in a point. Let $start_1, \ldots, start_n$ be the positions of the robots at the beginning, with $start_i \neq start_j$ for all $1 \leq i \neq j \leq n$. Let $Sync\mathcal{F}$ be the synchronous activation schedule returned by $\mathbf{Build}_{\mathbb{E}}(0, start_1, \ldots, start_n)$ defined in Lemma 3.4.4; $pos_i = Sync\mathcal{F}[i][Last].pos_e$, for all $1 \leq i \leq n$; and $t^* = Sync\mathcal{F}[1][Last].t_s + 4\rho$. Moreover, let $Sync\mathcal{F}'$ be the synchronous activation schedule returned by $\mathbf{Build}_{\mathbb{E}_1}(t^*, pos_1, \ldots, pos_n)$; and $Sync\mathcal{F}''$ the synchronous activation schedule returned by $\mathbf{Build}_{\mathbb{E}_2}(t^*, pos_1, \ldots, pos_n)$. Finally, let us define $Sync\mathcal{F}^*$ as

$$Sync\mathcal{F}^*[i] = Sync\mathcal{F}[i] \circ Sync\mathcal{F}'[i], \tag{3.2}$$

for all $1 \leq i \leq n$; and $Sync\mathcal{F}^{**}$ as

$$Sync\mathcal{F}^{**}[i] = Sync\mathcal{F}[i] \circ Sync\mathcal{F}''[i], \tag{3.3}$$

for all $1 \leq i \leq n$. It follows from Lemmas 3.4.4–3.4.6 that by executing $\mathcal{A}_g$ according either to $Sync\mathcal{F}^*$ or to $Sync\mathcal{F}^{**}$, the robots will never gather in a point, having a contradiction. □

Now, we are ready to extend the same result to SYm.

Theorem 3.4.3. *In* SYm, *there exists no deterministic oblivious algorithm that solves the gathering problem in finite time for a set of $n > 3$ robots*[7] *that cannot detect multiplicity.*

Proof. By contradiction, let $\mathcal{A}$ be a deterministic oblivious algorithm that solves the problem in SYm. Clearly, $\mathcal{A}$ is a bounded step algorithm with $Upp_i = \sigma_i$, for

[7] In [78], it is proven that there exists no oblivious algorithm that solves the problem when $n = 2$, under the assumption that two robots never collide.

all $1 \leq i \leq n$. Let ActSYm be an activation schedule in SYm for $\mathcal{A}$. By hypothesis, when executing $\mathcal{A}$ in SYm according to ActSYm, the robots are in a finite number of steps in a configuration $\mathbb{G}$ such that they occupy all the same position. Let us say this happens at time t, and let us fix in CORDA $\delta_{r_i} \leq \sigma_i$, for all $1 \leq i \leq n$.

By what observed at the beginning of Section 3.3, $\mathcal{A}$ is also an algorithm that can be executed in CORDA. By Lemma 3.3.1, ActSYm can be transformed into a synchronous activation schedule $Sync\mathcal{F}$ for $\mathcal{A}$ in CORDA. Furthermore, when executing $\mathcal{A}$ in CORDA according to $Sync\mathcal{F}$, $\mathbb{G}$ is reached by the robots at time $t4\rho$. Hence, $\mathcal{A}$ is an oblivious deterministic algorithm that solves the gathering problem in CORDA, when the robots cannot detect multiplicity. By Theorem 3.4.2, this is a contradiction; hence the theorem follows. □

3.4.2 The Limited Visibility Setting

In [3], an algorithm to solve the gathering problem in SYm in the limited visibility setting (called Algorithm 2 in Appendix A.2) is presented. We recall that, in this setting a robot can see only whatever is at distance V from it. In the following we shortly describe it.

Let us denote by $r_i(t)$ the position of robot r_i at time instant t. The set $P(t) = \{r_1(t), \ldots, r_n(t)\}$ then denotes the set of the robots' positions at t. Define $G(t) = (R, E(t))$, called the *Proximity Graph* at time t, by $(r_i, r_j) \in E(t) \leftrightarrow dist(r_i(t), r_j(t)) \leq V$, where $dist(p, q)$ denotes the Euclidean distance between points p and q. It can be proven that, if the proximity graph is not connected at the beginning, the robots can not gather in a point [3] (*form* a point, in SYm language).

Let $S_i(t)$ denote the set of robots that are within distance V from r_i at time t; that is, the set of robots that are visible from r_i (note that $r_i \in S_i(t)$). $C_i(t)$ denotes the smallest enclosing circle of the set $\{r_j(t) | r_j \in S_i(t)\}$ of the positions of the robots in $S_i(t)$ at t. The center of $C_i(t)$ is denoted $c_i(t)$.

Every time a robot r_i becomes active, the algorithm moves r_i toward $c_i(t)$, but only over a certain distance $MOVE$. Specifically, if r_i does not see any robot other than itself, then r_i does not move. Otherwise, the algorithm chooses x to be the point on the segment $\overline{r_i(t)c_i(t)}$ that is closest to $c_i(t)$ and that satisfies the following conditions:

1. $dist(r_i(t), x) \leq \sigma$. An arbitrary small constant $\sigma > 0$ is fixed, and it is assumed that the distance a robot can travel in one state is bounded by σ.

2. For every robot $r_j \in S_i(t)$, x lies in the disk D_j whose center is the midpoint m_j of $r_i(t)$ and $r_j(t)$, and whose radius is $V/2$. This condition ensures that

r_i and r_j will still be visible after the movement of r_i (and possibly of r_j, see Figure 3.12.a).

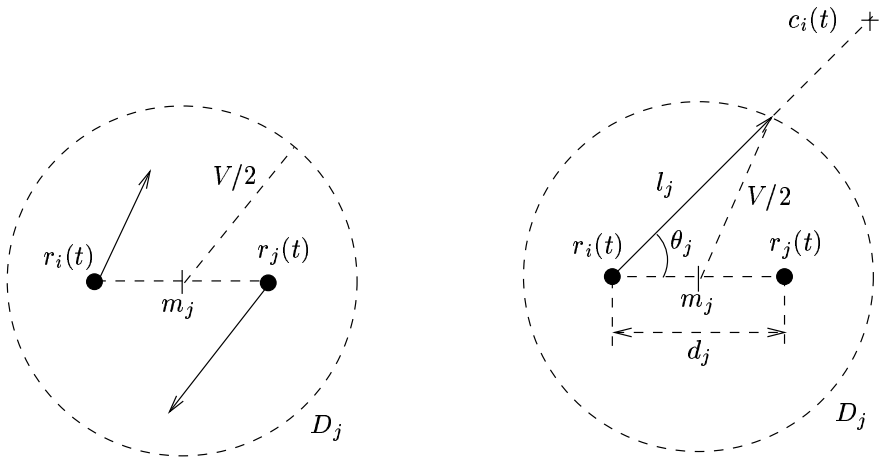


Figure 3.12: The algorithm for the gathering problem in SYm, limited visibility setting.

We note that, since by condition 1. the algorithm uses the constant σ to compute the destination point of a robot, all the robots must agree on the value of this constant, and thus it must be a priori known.

In [3] it is proven that, executing Algorithm 2, two robots that are connected in $G(t)$, will be connected in $G(t+1)$. In the following theorem we prove that it does not solve the gathering problem in Corda, in the limited visibility setting. Specifically, we give an initial configuration of the robots and describe a possible run of the algorithm that leads to partitioning the proximity graph: two robots that were visible until time t, are not visible any more at $t+1$, contradicting the result proven in [3].

Theorem 3.4.4. *The algorithm presented in [3] does not solve the gathering problem in* Corda*, in the unlimited visibility setting.*

Proof. In the following we show that Algorithm 2 does not solve the gathering problem in Corda. Specifically, we give an initial configuration of the robots and describe a possible run of the algorithm that leads to partitioning the proximity graph: two robots that were visible until time t, are not visible any more at $t+1$, contradicting the result proven in [3]. As noted in Section 3.3, Algorithm 2 is a bounded step algorithm; hence it can be executed in Corda, by fixing $\delta_{r_i} \leq \sigma_i$, for all $1 \leq i \leq n$.

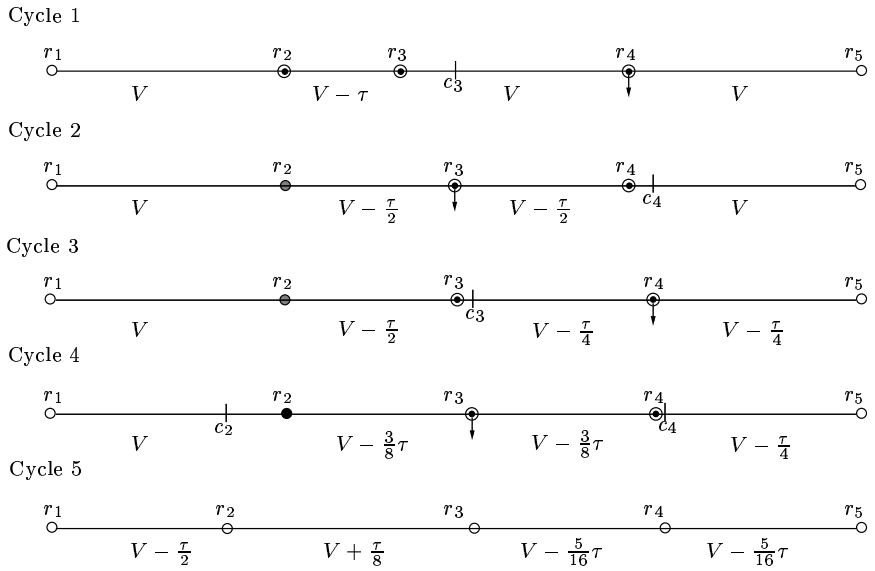


Figure 3.13: Proof of Theorem 3.4.4. The symbols used for the robots are the same as in Figure 3.5. A vertical arrow means that a robot decided not to move ($Move = 0$). A robot r_i moves always towards the center c_i of the smallest enclosing circle of all the robots it can see.

Let us suppose to have at the beginning 5 robots on a straight line, as shown in Figure 3.13. Moreover, let τ be a constant such that $\tau \leq \sigma$ and $\delta = \delta_{r_i} = \tau/16$, for all $1 \leq i \leq n$, where δ and σ are the constants introduced in the Assumption A2 in Section 3.1, and in Section 3.2, respectively.

At the beginning, we have the following *visibility situation*: r_1 can see r_2 ($dist(r_1, r_2) = V$), r_2 can see r_1 and r_3 ($dist(r_2, r_3) = V - \tau$), r_3 can see r_2 and r_4 ($dist(r_3, r_4) = V$), r_4 can see r_3 and r_5 ($dist(r_4, r_5) = V$), r_5 can see r_4. We recall that a robot r_i always move towards the center c_i of the smallest circle enclosing all the robots it can see. Executing Algorithm 2, but assuming the features of CORDA, a possible run is described in the following.

Cycle 1 All the robots, except r_1 and r_5 (that we assume in *Wait*), execute their first *Look*, and start the *Compute* state. Let us suppose that r_3 and r_4 are faster than r_2 in computing. The values they compute are:

r_3: $\begin{cases} Goal = dist(r_3, c_3) = \left|\frac{V-\tau+V}{2} - V + \tau\right| = \frac{\tau}{2} \\ Limit = \min\{-\frac{V-\tau}{2} + \frac{V}{2}, V\} = \frac{\tau}{2} \end{cases} \Rightarrow Move = \frac{\tau}{2}$

r_4: $Goal = 0 \Rightarrow Move = 0$

Moreover, r_3 and r_4 also start moving while r_2 is still computing; r_1 and r_5 are in *Wait*.

Cycle 2 After r_3 and r_4 move, the visibility situation is the same as it was in the beginning. r_3 and r_4 *Look* and *Compute* again, as follows:

r_3: $\quad Goal = 0 \;\Rightarrow Move = 0$

r_4: $\quad \begin{cases} Goal = dist(r_4, c_4) = \left| \frac{V+V-\frac{\tau}{2}}{2} - V + \frac{\tau}{2} \right| = \frac{\tau}{4} \\ Limit = \min\{-\frac{V-\frac{\tau}{2}}{2} + \frac{V}{2}, V\} = \frac{\tau}{4} \end{cases} \Rightarrow Move = \frac{\tau}{4}$

r_3 and r_4 move again, while r_2 is still in its first *Compute* state, and r_1 and r_5 in their first *Wait*.

Cycle 3 After the movement of the previous cycle, the visibility situation is still unchanged, that is, the proximity graph is still connected. r_3 and r_4 enter their third *Look* and *Compute* states.

r_3: $\quad \begin{cases} Goal = dist(r_3, c_3) = \left| \frac{V-\frac{\tau}{2}+V-\frac{\tau}{4}}{2} - V + \frac{\tau}{2} \right| = \frac{\tau}{8} \\ Limit = \min\{-\frac{V-\frac{\tau}{2}}{2} + \frac{V}{2}, \frac{V-\frac{\tau}{4}}{2} + \frac{V}{2}\} = \frac{\tau}{4} \end{cases} \Rightarrow Move = \frac{\tau}{8}$

r_4: $\quad Goal = 0 \;\Rightarrow Move = 0$

r_3 and r_4 move again. The other robots are in the same states as in the previous cycle.

Cycle 4 The proximity graph is still connected. r_3 and r_4 *Look* and *Compute* again (this is their fourth cycle).

r_3: $\quad Goal = 0 \;\Rightarrow Move = 0$

r_4: $\quad \begin{cases} Goal = dist(r_4, c_4) = \left| \frac{V-\frac{3}{8}\tau+V-\frac{\tau}{4}}{2} - V + \frac{3}{8}\tau \right| = \frac{\tau}{16}\tau \\ Limit = \min\{-\frac{V+\frac{3}{8}\tau}{2} + \frac{V}{2}, \frac{V-\frac{\tau}{4}}{2} + \frac{V}{2}\} = \frac{3}{16}\tau \end{cases} \Rightarrow Move = \frac{\tau}{16}$

r_3 and r_4 enter the *Move* state. Meanwhile, r_2 finishes its first *Compute*. The values it computes refer to what was the situation when it observed, in Cycle 1.

r_2: $\quad \begin{cases} Goal = dist(r_2, c_2) = \left| \frac{V+V-tau}{2} - V \right| = \frac{\tau}{2} \\ Limit = \min\{V, -\frac{V-\tau}{2} + \frac{V}{2}\} = \frac{\tau}{2} \end{cases} \Rightarrow Move = \frac{\tau}{2}$

r_2 starts moving according to the destination point it just computed (it enters it first *Move* state).

Cycle 5 The distance between r_2 and r_3 is $V + \tau/8 > V$; so r_2 and r_3 can not see each other anymore, breaking the proximity graph connectivity that we had at the beginning of the cycle. So, the invariant that "robots that are mutually visible at t remain within distance V of each other thereafter" asserted in [3] is violated. Therefore, the theorem follows. □

3.5 Conclusions

In this chapter we presented CORDA, the model we will use in the second part of the thesis to analyze some coordination problems for a set of autonomous mobile robots.

Furthermore, we compared it with SYm [3, 78], the only other model, to our knowledge, that approaches the problem addressed by this thesis from our perspective. We showed that the different way in which the asynchronicity is modeled in SYm and CORDA, is the key feature that renders the two models different: in SYm the robots operate executing *instantaneous actions*, while in CORDA *full asynchronicity* is modeled, and the robots elapses finite, but otherwise unpredictable, amount of time to execute their states. In particular, we showed that $\mathfrak{C} \subset \mathfrak{Z}$ in the non oblivious setting. Therefore, one open issue is to prove this result also in the oblivious setting.

Finally, we presented a case study: the oblivious gathering, in both the limited and unlimited visibility setting. We showed that the algorithmic solutions presented in SYm do not work in the asynchronous environment modeled by CORDA. Hence, since we feel that the approach used in CORDA better describes the way a set of independently-moving units operate in a totally asynchronous environment, the motivation to investigate coordination problems in a distributed, asynchronous environment using the *fully asynchronous* approach. Some of these investigations will be reported in the second part of this thesis.

Chapter 4

The Arbitrary Pattern Formation Problem

What was troubling the robot was what roboticists called an equipotential of contradiction on the second level. Obedience was the Second Law and [the robot] was suffering from two roughly equal and contradictory orders. Robot-block was what the general population called it or, more frequently, roblock for short . . . [or] 'mental freeze- out.' No matter how subtle and intricate a brain might be, there is always some way of setting up a contradiction. This is a fundamental truth of mathematics.

I. Asimov, The Rest of the Robots, 1968, pp. 102–120

Abstract

In this chapter, we concentrate on the particular coordination problem that requires the robots to form a specific geometric pattern, the *arbitrary pattern formation* problem.

We show that the ability of the robots to form arbitrary patterns depends on the amount of common knowledge they have on the orientation of the local coordinate systems.

4.1 Introduction

The *pattern formation* problem has been investigated extensively in the literature. It is interesting algorithmically, because if the robots can form any pattern, they can agree on their respective roles in a subsequent, coordinated action. We study this problem for *arbitrary* geometric patterns, where a pattern is a set of points (given by their Cartesian coordinates) in the plane. The pattern is known initially by all robots

in the system. For instance, we might require the robots to place themselves on the circumference of a circle, with equal spacing between any two adjacent robots, just like kids in the kindergarten are sometimes requested to do. We do not prescribe the position of the circle in the world, and we do not prescribe the size of the circle, just because the robots do not have a notion of the world coordinate system's origin or unit of length. The robots are said to *form the pattern*, if the actual positions of the robots coincide with the points of the pattern, where the pattern may be *translated*, *rotated*, *scaled*, and *flipped* into its mirror position in each local coordinate system. Initially, the robots are in arbitrary positions, with the only requirement that no two robots are in the same position, and that of course the number of points prescribed in the pattern and the number of robots are the same.

The pattern formation problem for *arbitrary* patterns is quite a general member in the class of problems that are of interest for autonomous, mobile robots. It includes as special cases many coordination problems, such as leader election: we just define the pattern in such a way that the leader is represented uniquely by one point in the pattern.

Experimentally, this problem has been investigated mostly as an initial step that gets the robots together and then lets them proceed in the desired formation [9, 79] (just like a flock of birds or a troupe of soldiers).

Theoretically, it has been studied by Suzuki *et al.* in SYm. They analyze the problem when the robots do not have any kind of agreement on the orientation of the local coordinate systems; the robots, however, have an unbounded amount of memory, that is they are non-oblivious. In this setting, they presented an algorithm that allows the robots to form regular n-gons [76, 78].

In contrast, in an attempt to understand the power of common knowledge for the coordination of robots, we study the pattern formation problem under several assumptions. We give a complete characterization of what can and what cannot be achieved, reflecting the general direction of the investigation in this thesis: what coordination problems can be solved, and under what conditions? First, we study the extreme cases of total agreement and no agreement on the local coordinate systems. In particular, we show that for an arbitrary number of robots with total agreement on the local coordinate systems, the pattern formation problem can be solved. In order to solve the pattern formation problem, the robots, however, need to agree also on both, origin and unit length. We propose a simple algorithm that achieves this agreement. In contrast, with no agreement on the local coordinate systems, the problem is not solvable in general. Second, we study the case of the robots with one axis direction and orientation agreement. We show that the pattern formation problem can be solved whenever the number of robots is odd, and that it is in general unsolvable when the number of robots is even. Third, we study the

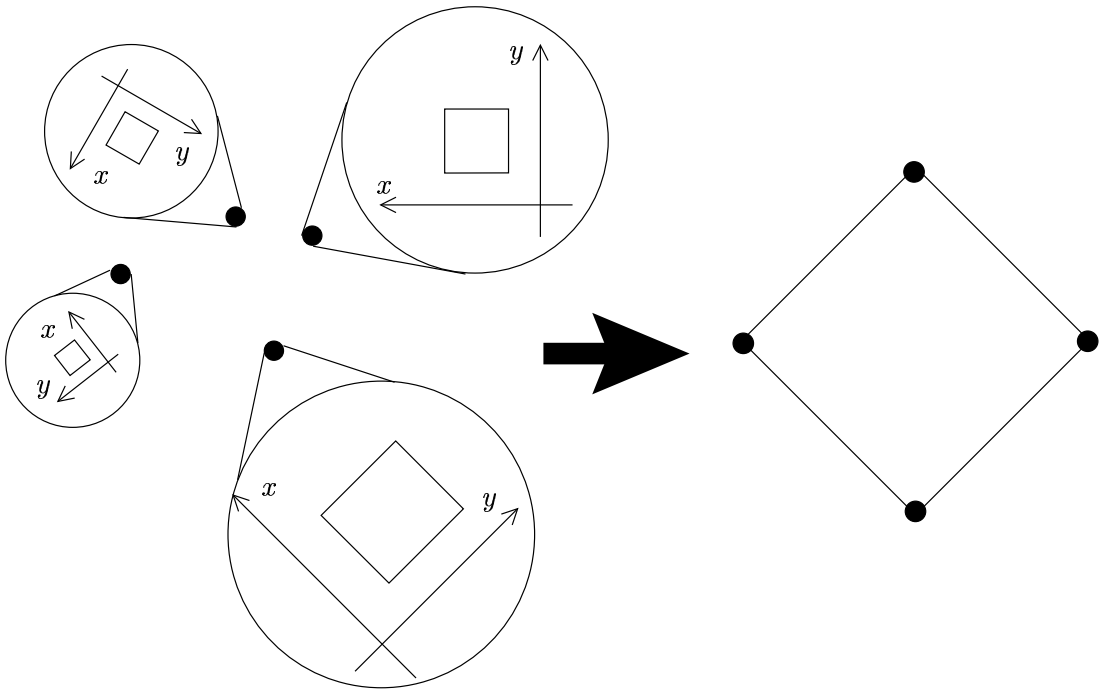


Figure 4.1: An example of the task assigned to the robots when they are asked to solve the Arbitrary Pattern Formation Problem.

pattern that can be achieved when the system is populated with an even number of robots. All the algorithms we present, do not need to store the observations done in the past; that is, they are totally *oblivious*. Hence, our study is at the extreme end in two ways: the problem is extremely hard, and the robots are extremely weak.

4.2 The Arbitrary Pattern Formation Problem

We study the pattern formation problem for arbitrary geometric patterns, where a pattern $\mathbb{P}$ is a set of points $p_0, \ldots, p_n$ (given by their Cartesian coordinates) in the plane. The pattern is known initially by all robots in the system. Initially, the robots are in arbitrary positions, with the only requirement that no two robots be in the same position, and that, of course, the number of points prescribed in the pattern and the number of robots are the same. This defines a *valid* initial configuration for this problem.

Given a pattern $\mathbb{P}$ and a configuration $\mathbb{G}$, the robots are said to *have formed* $\mathbb{P}$, if there exists a transformation $\mathcal{T}$, where $\mathcal{T}$ can be an arbitrary sequence of *translation*, *rotation*, *scaling*, or *flipping* into mirror position, such that at time t, $\mathcal{T}(\mathbb{V}_{r_i}(t)) = \mathbb{P}$, $\forall r_i$. In other words, the final positions of the robots must coincide with the points

of the input pattern, where the formed pattern may be *translated*, *rotated*, *scaled*, and *flipped* into its mirror position with respect to the input pattern $\mathbb{P}$ in each local coordinate system. For instance, in the example depicted in Figure 4.1, all the robots are given in input the same square, that each robot sees according to its local concept of unit distance and orientation of the x and y axis. They are asked, in a finite number of cycles, to find an agreement on both the unit distance, and the direction and orientation of the x and y axis, so that the desired square can be formed on the plane.

Given a pattern $\mathbb{P}$ and a configuration $\mathbb{G}$, a *final configuration* for $\mathbb{P}$ is a configuration of the robots in which the robots form the desired pattern $\mathbb{P}$. Given an arbitrary valid initial configuration and an arbitrary pattern $\mathbb{P}$, a *pattern formation algorithm* is an oblivious deterministic algorithm that brings the robots in the system to a final configuration for $\mathbb{P}$ in a finite number of cycles. We say that a pattern formation algorithm is *collision-free*, if, at any point in time t, there are no two robots that occupy the same position in the plane.

Another problem that we will refer to in the following and strictly related to the arbitrary pattern formation problem is the *leader election* problem: the robots in the system are said to elect a leader if, after a finite number of cycles, all the robots deterministically agree on (choose) the same robot L, called the leader. An oblivious deterministic algorithm that lets the robots in the system elect a leader in a finite number of cycles, given an arbitrary initial configuration, is called a *leader election algorithm*. The relationship between the pattern formation problem and the leader election problem, is stated in the following

Theorem 4.2.1. *If it is possible to solve the pattern formation problem for $n \geq 3$ robots, then the leader election problem is solvable too.*

Proof. Let $\mathcal{A}$ be a pattern formation algorithm. Let $\mathbb{P}$ be a pattern defined in the following way:

1. All the robots but one are evenly placed on the same line Ψ; the distance between two adjacent robots is d; and

2. the last robot is on Ψ, but the distance from its unique adjacent robot is $2d$.

After all the robots execute $\mathcal{A}$ to form $\mathbb{P}$, the unique robot whose distance from its neighbor is $2d$ is identified as the leader. □

We note that, since $\mathcal{T}$ can be also a rotation, two robots always form the desired pattern. Therefore, in the following we will assume to have at least 3 robots in the system.

We study the arbitrary pattern formation problem by assuming different level of knowledge among the robots. First, we analyze the two extreme cases when total agreement, and no agreement on the local coordinate systems exists. Second, we analyze the case when the robots have only partial agreement on the coordinate systems, namely they agree only on the direction and orientation of one axis.

4.3 Total and No Agreement

In this section, we consider the two extreme cases when the robots share the coordinate system (*total agreement*) and when their local coordinate systems are unrelated and possibly different (*no agreement*).

4.3.1 Total Agreement

For the case in which the directions and orientations of both axes are common knowledge, the robots can form an arbitrary given pattern as follows (Algorithm 1). First, each robot for itself identifies the first and second point of the input pattern, in lexicographic order with respect to the coordinates in which the pattern is given (Figure 4.2.a). Second, each robot identifies the first and second robot position in lexicographic order in the set of robots' positions retrieved in the last *Look* (the algorithm is oblivious). This can be done since all robots have the same notion of lexicographic order, as a result of their knowledge of both axes directions and orientations (Figure 4.2.b). Third, the first and second robots put themselves in positions matching the first and second pattern points, where the final pattern the robots form might be translated and scaled, but not rotated or flipped with respect to the one they have in input. This can be done by moving the first robot only, in such a way that it always remains first (Figure 4.2.c and d). Now, the first two robots' positions identify the translation and scaling of the pattern, and they make this visible for all the robots (Figure 4.2.e). Fourth, all the other robots go to points of the pattern. This can be done by moving one robot after the other step by step to a pattern's point. The sequence of robots is chosen to guarantee that after one robot has made even only a small move towards its destination, no other robot will move before that one has reached its destination (Figure 4.2.f). We note that the final positions of the robots are not rotated w.r.t. the input positions; in other words the algorithm keeps the "orientation" given by the input pattern. Moreover, in this case Theorem 4.2.1 holds also for $n = 2$, since the rightmost and topmost robot in the system can always be identified as the leader.

Algorithm 1 calls routines `Angle()`, `Sort()`, `Go_Into_Position()`, `Find_Final_Positions()`, and `Go_To_Points()`, whose behavior is described in the following.

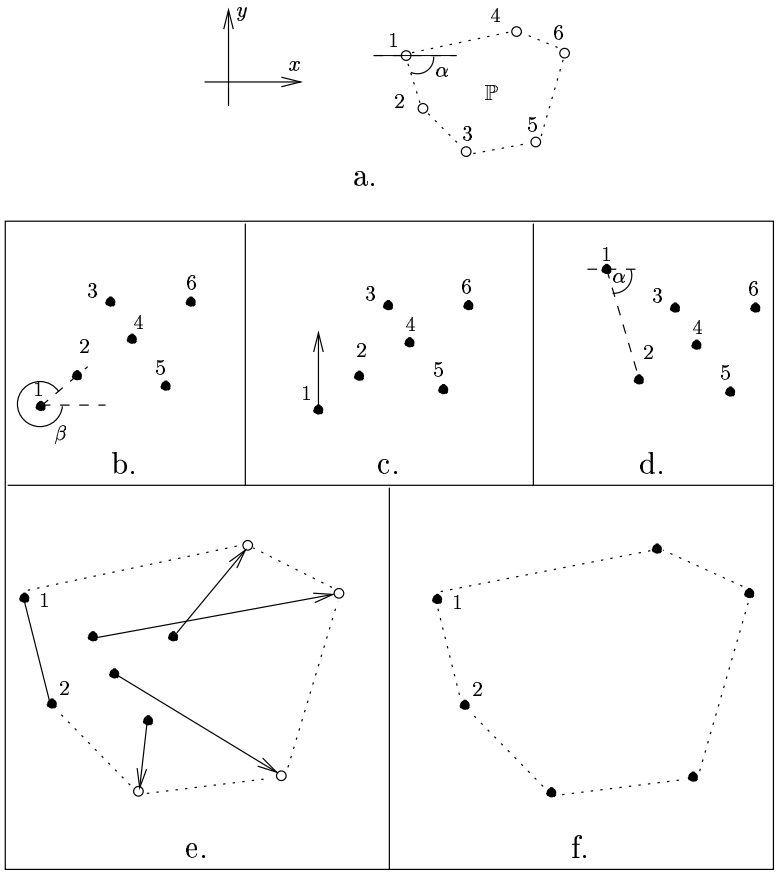


Figure 4.2: Algorithm 1. (a) The input pattern $\mathbb{P}$. The robots have total agreement on the local coordinate systems. The numbers represent the lexicographical ordering the robots give to the point of $\mathbb{P}$, and α = `Angle`(p_1, p_2). (b) The robots sort the robots' positions retrieved in the last *Look* state, and compute β =`Angle`(r_1, r_2). (c) r_1 moves in such a way that `Angle`(r_1, r_2)= α, according to routine `Go_Into_Positions()`. (d) The relative positions of r_1 and r_2 are such that `Angle`(r_1, r_2)= α. (e) At this point, all the robots can translate and scale the input pattern according to $[r_1 r_2]$. Then, all the robots, one at a time, reach the final positions of the pattern to form. (f) The final configuration.

We recall that `do_nothing()` terminates the local computation of the calling robot, and the result of its *Compute* is a *null movement* (Section 3.1).

`Angle`(p, q) computes the clockwise angle between the horizontal axis passing through p and the segment $[pq]$.

`Sort()` gives a lexicographic order to all the positions of the robots in the system observed in the last *Look*, including the robot calling the procedure, say from left to right and from bottom to top; it returns the sorted sequence.

`Go_Into_Position`(r_1, r_2, α) orders r_1 to move so as to achieve angle α with r_2 while staying lexicographically first.

`Find_Final_Positions`$(r_1, r_2, Unit)$ figures out the final positions of the robots according to the given pattern and to the positions of r_1 and r_2. In particular, p_1 is translated onto the position of r_1, and p_2 onto the position of r_2. The common scaling of the input pattern is defined by the common unit distance $Unit$.

`Go_To_Points`$(Free\_Robots, Free\_Points, \Gamma, \Gamma')$ chooses the robot r in $Free\_Robots$ that is "closest" to a point in $Free\_Points$, say p, and moves r towards p in such a way that r stays always inside the region of the plane delimited by the vertical lines[1] Γ and Γ', while avoiding collisions. In fact, if r were to move following a straight line towards its destination p, it might collide with a robot on this path. Should this be the case, the routine appropriately chooses an intermediate destination point maintaining the invariant that r is the robot in $Free\_Robots$ "closest" to a point in $Free\_Points$. The routine is described in Figure 4.3. `Minimum`$(Free\_Robots, Free\_Points)$ finds the robot r in $Free\_Robots$ that has the minimum Euclidean distance from one of the $Free\_Points$, say p. That is, it returns the unique pair (r, p) such that

$$dist(r, p) = \min_{\substack{f_r \in Free\_Robots \\ f_p \in Free\_Points}} dist(f_r, f_p). \tag{4.1}$$

Any tie is broken by choosing the pair with the topmost rightmost robot, and, in case of another tie, the topmost rightmost point. `Move`(p) terminates the local computation of the calling robot and starts its *Move* state: the robot moves towards p. In Figure 4.4, a pictorial representation of some steps of this routine are reported.

Before proving that Algorithm 1 lets the robots correctly form the input pattern, we first show some properties of routine `Go_To_Points()`.

Routine `Go_To_Points()`

In this paragraph, we aim to show that if (r, p) is the pair that satisfies Equation 4.1 at time t with respect to a set of robots $Free\_Robots$

[1] By *vertical line*, we mean a line that is parallel to the direction of y.

Algorithm 1 Arbitrary Pattern Formation Problem With Total Agreement

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order, with the ordering given left-right, bottom-up. There is total agreement on the local coordination systems.

$\alpha :=$ `Angle`(p_1, p_2);
$Sorted\_Robots :=$ `Sort()`;
$r_1 :=$ First robot in $Sorted\_Robots$;
$r_2 :=$ Second robot in $Sorted\_Robots$;
5: $r^* :=$ Last Robot in $Sorted\_Robots$;
$\beta :=$ `Angle`(r_1, r_2);
Case I Am

- r_2:
  `do_nothing()`.
10: - r_1:
  If $\alpha = \beta$ **Then** `do_nothing()`.
  Else `Go_Into_Position`(r_1, r_2, α).
- Default: %I am neither r_1 nor r_2%
  If $\alpha = \beta$ **Then**
15: $Unit := dist(r_1, r_2)$; %all the robots agree on a common unit distance%
  $Final\_Positions :=$ `Find_Final_Positions`$(r_1, r_2, Unit)$;
  $Ext :=$ Rightmost Point In $Final\_Positions$;
  If I Am On One Of The $Final\_Positions$ **Then**
    `do_nothing()`.
20: **Else**
    $Free\_Robots :=$ {*robots not on one of the* $Final\_Positions$};
    $Free\_Points :=$ {$Final\_Positions$ *with no robots on them*};
    $\Gamma :=$ Vertical Line Through r_2;
    $\Gamma^* :=$ Vertical Line Through r^*;
25: $\Gamma_{Ext} :=$ Vertical Line Through Ext;
    $\Gamma' :=$ Rightmost Vertical Line Among Γ^* and Γ_{Ext};
    `Go_To_Points`$(Free\_Robots, Free\_Points, \Gamma, \Gamma')$.
  Else `Do_nothing()`.

```
Go_To_Points(Free_Robots, Free_Points, Γ, Γ')
    (r, p) := Minimum(Free_Robots, Free_Points);
    If I Am Not r Then
        do_nothing().
    Else %I Am r%
 5:     If No Robots Is On The Line Passing Through r And p Then
            Move(p).
        Else
            Ψ := Line Passing Through r and p;
            Ψ' := Line Orthogonal To Ψ;
10:         C_p := Circle Centered In p Having Radius [rp];
            C'_p := Half Of C_p Delimited By Ψ' That Contains r;
            Avoid := {Robots' Positions Inside C'_p And Not On [rp]};
            Intersections := ∅;
            For All p' ∈ Avoid Do
15:             h := Point Between p' And p At Distance dist(p, p')/2 From p;
                Ψ'' := Line Passing Through r And h;
                s := Intersection Between Ψ' And Ψ'';
                Intersections := Intersections ∪ {s};
            End For
20:         Intersections' := ∅;
            For All f_p ∈ Free_Points \ {p} Do
                h := Point Between p And f_p At Distance dist(p, f_p)/4 From p;
                Ψ'' := Line Passing Through r And h;
                s := Intersection Between Ψ' And Ψ'';
25:             Intersections' := {Intersections'} ∪ {s};
            End For
            If {Intersections} ∪ {Intersections'} = ∅ Then
                p_cl :=Nil;
            Else %Ties broken by choosing the rightmost topmost intersection, according to the local coord. system%
30:             p_cl := Point In {Intersections} ∪ {Intersections'} Closest To p;
            If p_cl Not In C_p Or p_cl =Nil Then %p_in is chosen on Ψ' and inside c_p%
                p_in := Point On Ψ' At Distance dist(r, p)/2 From p;
            Else p_in := p_cl;
            p_in_s := Point Symmetric To p_in w.r.t. Ψ;
35:         V := Vertical Stripe With Borders Γ and Γ';
            Φ := Ψ' ∩ V;
            Λ := [p_in p_in_s] ∩ Φ;
            p* := Point Inside Λ Different From p;
            Move(p*).
```

Figure 4.3: Routine `Go_To_Points`$(Free\_Robots, Free\_Points, \Gamma, \Gamma')$ called in Algorithm 1

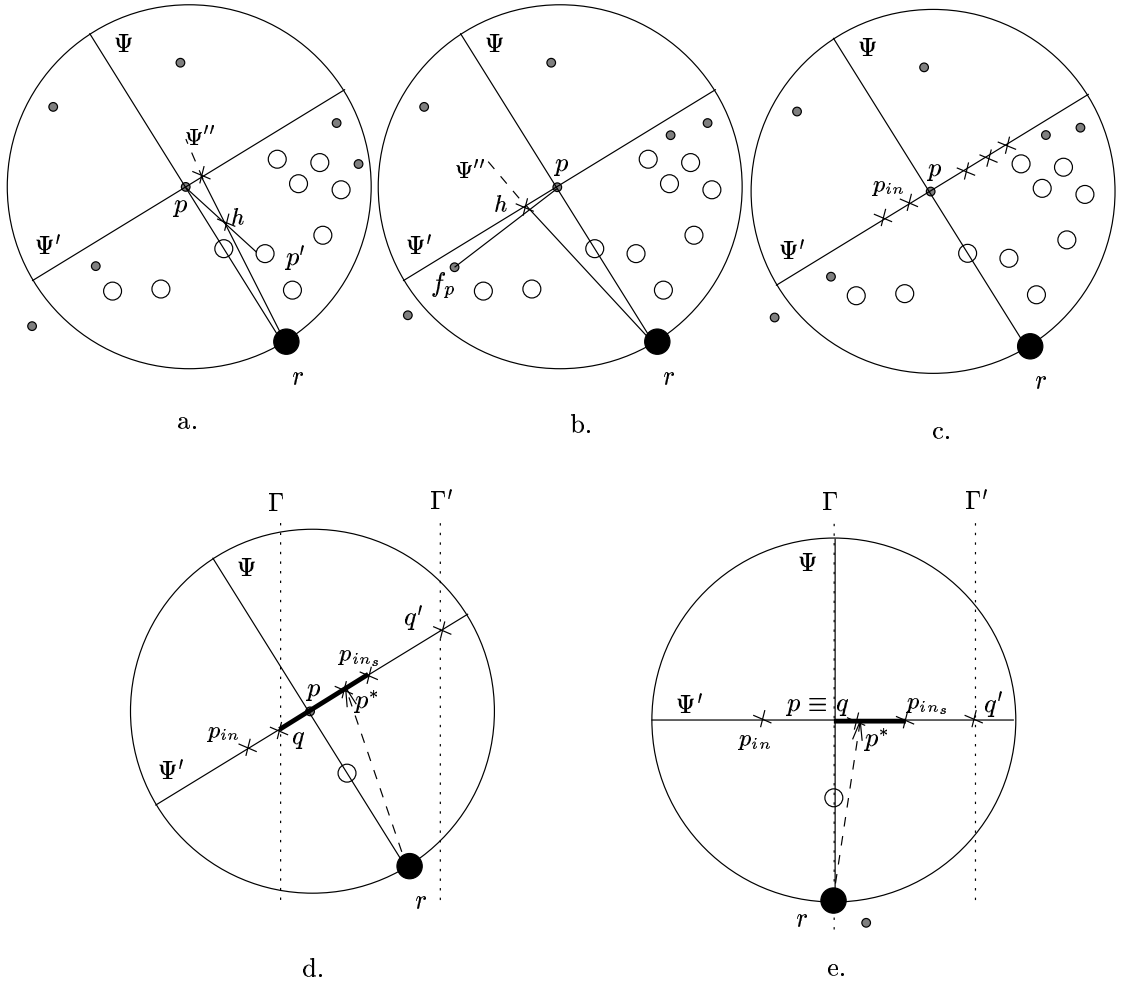


Figure 4.4: Rotine `Go_To_Points(`*Free_Robots*, *Free_Points*`)`: the white circles represent robots' positions; the filled ones represent points in *Free_Points*. (a) Computation of *Intersections*. (b) Computation of *Intersections'*. (c) Computation of p_{in} in Lines 32–34. In (d)-(e) the computation of destination point p^* is pictured. It is always chosen to be inside the vertical stripe delimited by Γ and Γ'. The thick line is the segment Λ computed in Line 38, with q and q' the end points of Φ computed in Line 36. In (e), $[p_r p] \subseteq \Gamma$.

and a set of points in the plane *Free_Points*, then the execution of routine `Go_To_Points(`$Free\_Robots, Free\_Points, \Gamma, \Gamma'$`)` at time t lets only one robot in *Free_Robots* move towards one of the points in *Free_Points*, while all the other robots do not move.

A *vertical stripe* is a region of the plane delimited by two distinct vertical *borders*. A border can be a vertical line, a vertical half-line, or a segment, and it belongs to the vertical stripe it delimits. If a point p is between or on one of the two borders, then p is said to be *inside* the vertical stripe. We say that a point p is *strictly inside* a vertical stripe, if p is between the two borders, but not on one of them. If all the positions occupied by a robot r during a *Move* are always inside the stripe, we say that r moves inside it; similarly, if the positions are strictly inside the stripe, we say that r moves strictly inside it.

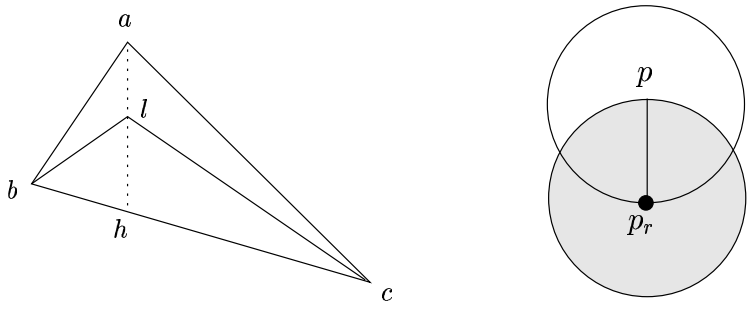
Let $\mathcal{V}$ be a vertical stripe with borders Γ and Γ', and $\mathbb{K}(\tilde{t})$ be the set of robots that at time $\tilde{t}$ are inside $\mathcal{V}$. Furthermore, let $FreeR \subseteq \mathbb{K}(\tilde{t})$ be a set of robots and *FreeP* a set of points inside $\mathcal{V}$, with $|FreeR| = |FreeP|$, such that no robot in *FreeR* occupies a point in *FreeP*, and no point in *FreeP* is occupied by a robot in $\mathbb{K}(\tilde{t})$. We denote by $(r, p)_{\tilde{t}}$ the pair that satisfies Equation 4.1 at time $\tilde{t}$ with respect to the robots in *FreeR* and to the points in *FreeP*, and by p_r the position of r at time $\tilde{t}$. Moreover, let Ψ, Ψ', $\mathcal{C}_p$, $\mathcal{C}'_p$, *Intersections*, *Intersections'*, be as defined in routine `Go_To_Points()`; let p_I be the point in *Intersections* closest to p, and p_{I_s} be the point symmetric to p_I with respect to Ψ; and let p'_I be the point in *Intersections'* closest to p, and p'_{I_s} be the point symmetric to p'_I with respect to Ψ. Finally, let p_{cl} be the point among p_I and p'_I that is closest to p, and p_{cl_s} be the point symmetric to p_{cl} with respect to Ψ. Note that Ψ and Ψ' partition $\mathcal{C}_p$ in four parts, that we will refer to as the *zones of* $\mathcal{C}_p$.

Fact 4.3.1. Let $\triangle(a, b, c)$ be a triangle in the plane such that $dist(a, b) \leq dist(a, c)$, and h be a point on $[bc]$ such that $dist(h, b) < dist(b, c)/2$. Then, for any point l on the segment line $[ah]$, $dist(l, b) < dist(l, c)$ (see Figure 4.5.a). ⋈

Moreover, it follows by Equation (4.1) that (see Figure 4.5.b)

Observation 4.3.1. In the circle $\mathcal{C}_p$ centered in p and having radius $[p_r p]$ there are no robots in *FreeR*, and in the circle $\mathcal{C}_r$ centered in p_r and having radius $[p_r p]$ there are no points in *FreeP*. ⋄

The following lemma locates a region of the plane such that all the points inside it are closer to p than to any point in $FreeP \setminus \{p\}$.



a. b.

Figure 4.5: (a) Fact 4.3.1. (b) Observation 4.3.1.

Lemma 4.3.1. *Let us assume that* $Intersections' \neq \emptyset$*, and that* p'_I *is inside* $\mathcal{C}_p$*. Then, given any point* $f_p \in FreeP \setminus \{p\}$*,*

$$dist(l,p) < dist(l,f_p),$$

for all $l \in \triangle(p_r, p'_I, p'_{I_s})$*,*

Proof. Let $f_p \in FreeP \setminus \{p\}$, h be the point between p and f_p at distance $dist(p,f_p)/4$ from p, and s be the intersection between Ψ' and the line passing through p_r and h (refer to Figure 4.6). By (4.1),

$$dist(p_r,p) \leq dist(p_r,f_p). \tag{4.2}$$

Moreover, let s' be the points symmetric to s with respect to Ψ. We first show that

$$dist(l,p) < dist(l,f_p), \quad \forall\ l \in \triangle(p_r,s,s'). \tag{4.3}$$

We distinguish two cases.

1. *f_p is in the half-plane delimited by Ψ' that does not contain p_r, or $f_p \in \Psi'$ (see Figure 4.6.a).* If $f_p \in \Psi$, then $s \equiv p$, $l \in [p_r p]$, and Equation (4.3) follows. Otherwise, let $l \in \triangle(p_r,p,s)$, Ξ be the line passing through p_r and l, and h' be the intersection between Ξ and $[pf_p]$. By construction, $dist(p,h') < dist(p,f_p)/2$; moreover, $l \in \triangle(p_r,p,f_p)$, hence by (4.2) and by Fact 4.3.1 applied on $\triangle(p_r,p,f_p)$, we have that, for all $l \in \triangle(p_r,p,s)$, $dist(l,p) < dist(l,f_p)$.

   Now, let us consider a point $l' \in \triangle(p_r,p,s')$, and the circle $\mathcal{C}_{l'}$ centered in l' and having radius $[l'p]$. By construction, no point in the zone crossed by $[pf_p]$ is inside $\mathcal{C}_{l'}$, besides p; hence $dist(l',p) < dist(l',f_p)$, and Equation (4.3) follows.

2. *f_p is in the half-plane delimited by Ψ' that contains p_r, with $f_p \notin \Psi'$ (refer to Figure 4.6.b).* If $f_p \in \Psi$, then $s \equiv p$, $l \in [p_r p]$, and Equation (4.3) follows by the fact that $f_p \notin [p_r p]$ (see Observation 4.3.1). Otherwise, let Ξ be the line passing through s and parallel to Ψ, and $\mathcal{R}$ be the rectangle with vertices p, p_r, z, and s, with z the point on Ξ such that $dist(p_r, z) = dist(p, s)$ and $dist(s, z) = dist(p_r, p)$. Let k be the intersection between $[pf_p]$ and $[sz]$. By construction, $dist(p, k) \leq 2dist(p, h) = dist(p, f_p)/2$; hence f_p is outside $\mathcal{R}$. Furthermore, the triangle $\triangle(p_r, p, s)$ can be partitioned in two triangles: $\triangle(p_r, p, h)$, and $\triangle(p, h, s)$. We distinguish the two possible cases.

   (a) Let $l \in \triangle(p_r, p, h)$, and $h' = [p_r l) \cap [pf_p]$. By construction, $l \in \triangle(p_r, p, f_p)$, and $dist(p, h') \leq dist(p, h) < dist(p, f_p)/2$; hence by (4.2) and by Fact 4.3.1 applied on $\triangle(p_r, p, f_p)$, we have that $dist(l, p) < dist(l, f_p)$.

   (b) Let $l \in \triangle(p, h, s)$. Since $dist(p, s) < dist(p, k) \leq dist(p, f_p)/2$, we have that $dist(p, s) < dist(s, f_p)$. Let $h' = [p_r l) \cap [pf_p]$. By construction, $l \in \triangle(p, s, f_p)$ and $dist(p, h') < dist(p, f_p)/2$. Hence, by applying Fact 4.3.1 to triangle $\triangle(p, s, f_p)$, we have that $dist(l, p) < dist(l, f_p)$.

   Therefore, for all $l \in \triangle(p_r, p, s)$, we have that $dist(l, p) < dist(l, f_p)$.

   Let us now consider a point $l' \in \triangle(p_r, p, s')$, and its symmetric l with respect to Ψ. Since by construction $l \in \triangle(p_r, p, s)$, we have that $dist(l, p) < dist(l, f_p)$. Let $\mathcal{C}_{f_p}$ be the circle centered in f_p and having radius $[f_p l]$. By construction l' is not inside $\mathcal{C}_{f_p}$, hence $dist(l', f_p) > dist(l, f_p)$. Therefore, $dist(l', p) = dist(l, p) < dist(l, f_p) < dist(l', f_p)$, and Equation (4.3) follows.

Therefore, we proved that Equation (4.3) holds for all $l \in \triangle(p_r, s, s')$. Since, by definition, $p'_I \in [s, s']$ and $p'_{I_s} \in [s, s']$, then $\triangle(p_r, p'_I, p'_{I_s}) \subseteq \triangle(p_r, s, s')$, and the lemma follows from (4.3). □

In other words, Lemma 4.3.1 implies that once a free point p is assigned to a free robot r as its destination, no other free point will be subsequently assigned to r as its target, as long as r stays inside the triangle having as vertices p_r, p'_I, and p'_{I_s}. Routine `Go_To_Points()` forces indeed r to stay inside this region until it reaches p.

Lemma 4.3.2. *Let us assume $Intersections \neq \emptyset$, $Intersections' \neq \emptyset$, and that p_{cl} is inside $\mathcal{C}_p$. Then, for all $l \in \triangle(p_r, p_{cl}, p_{cl_s})$,*

$$dist(l, p) < dist(l, f_p), \tag{4.4}$$

$$dist(p, l) < dist(p, f_r), \tag{4.5}$$

with $f_p \in FreeP \setminus \{p\}$, and $f_r \in FreeR \setminus \{r\}$. Furthermore, there is no robot r' such that $r' \in \triangle(p_r, p_{cl}, p_{cl_s}) \wedge r' \notin [p_r p]$, and if $r' \in [p_r p]$ then $r' \notin FreeR$.

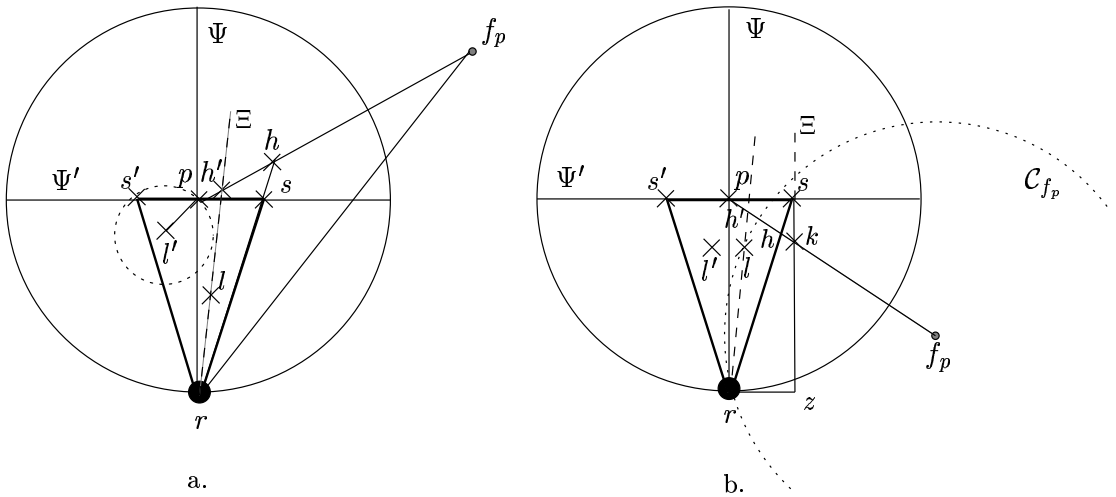


Figure 4.6: Lemma 4.3.1.

Proof. Equation (4.4) follows by Lemma 4.3.1 and by definition of p_{cl}. By Observation 4.3.1 no robot in $FreeR$ is inside $\mathcal{C}_p$; moreover, by construction $\triangle(p_r, p_{cl}, p_{cl_s}) \subset \mathcal{C}_p$, hence Equation (4.5) follows.

Furthermore, by definition of *Intersections* (see Figure 4.4.a) no robot r' is such that $r' \in \triangle(p_r, p_I, p_{I_s}) \wedge r' \notin [p_r p]$; hence, by definition of p_{cl}, no robot r' is such that $r' \in \triangle(p_r, p_{cl}, p_{cl_s}) \wedge r' \notin [p_r p]$.

Finally, the last statement of the lemma follows by Observation 4.3.1. □

Until now we proved geometrical properties of the pair $(r, p)_{\tilde{t}}$. Let us see what happens if r moves towards p. In particular, let us assume that the following conditions hold:

C1. All the robots in $\mathbb{K}(\tilde{t})$ are in[2] $\mathbb{SD}(\tilde{t})$;

C2. for any $t' \geq \tilde{t}$ such that r is not on p at t', if $r' \in FreeR$ is in a *Compute* state, then it executes the routine `Go_To_Points`$(FreeR, FreeP, \Gamma, \Gamma')$ (recall that $r \in FreeR$ at time $\tilde{t}$);

C3. for any $t' \geq \tilde{t}$ such that r is not on p at t', all robots in $\mathbb{K}(\tilde{t}) \setminus FreeR$ compute *null movements*.

Moreover, let $t_l \geq \tilde{t}$ be the first time when r starts a *Look* at a time greater or equal than $\tilde{t}$; $t_c > t_l$ be the first time when r starts a *Compute* after time t_l; and $t_m > t_c$ be the first time when r starts a *Move* after time t_c.

[2]recall that $\mathbb{SD}(t) = \mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{L}_S(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$, defined in Section 3.1.2.

Lemma 4.3.3. *Let conditions C1–C3 hold. Then, r does not move between time $\tilde{t}$ and t_m, and all the other robots in $\mathbb{K}(\tilde{t})$ are in $\mathbb{SD}(t)$, for all $\tilde{t} \leq t \leq t_m$. Furthermore, r starts executing* `Go_To_Points()` *at time t_c, $r \in \mathbb{C}_+(t_c)$, and $r \in \mathbb{M}_+(t_m)$.*

Proof. Since $t_m \geq \tilde{t}$ is the *first time* when r starts a *Move* after t_l, if r moved (i.e., changed position) before t_m then either $r \in (\mathbb{L}_+(\tilde{t}) \setminus \mathbb{L}_S(\tilde{t})) \cup \mathbb{M}_+(\tilde{t}) \cup \mathbb{C}_+(\tilde{t})$; or r completed a cycle between time $\tilde{t}$ and t_l, having a contradiction in both cases; and the first statement of the lemma follows.

In order to prove the second statement of the lemma, we will show that there exists no robot different from r in $\mathbb{L}_+(t) \cup \mathbb{C}_+(t) \cup \mathbb{M}_+(t)$, for all $\tilde{t} \leq t \leq t_m$. By contradiction, let t^*, $\tilde{t} \leq t^* \leq t_m$, be the first time such that there exists a robot $r^* \in \mathbb{L}_+(t^*) \cup \mathbb{C}_+(t^*) \cup \mathbb{M}_+(t^*)$. By C3 above, $r^* \in FreeR$; hence, by C2, it executes routine `Go_To_Points()`.

If $r^* \in \mathbb{L}_+(t^*)$, by the way t^* has been chosen, $r \in \mathbb{L}_S(t^*)$ too. Moreover, the first time r^* executes `Go_To_Points()`, it will not compute a *null movement* (by definition of $\mathbb{L}_+(\cdot)$ and $\mathbb{L}_S(\cdot)$). Hence, by Lines 1–3 of `Go_To_Points()`, $(r^*, \cdot)$ satisfies Equation (4.1) at time t^*. Since r does not move before t_m, and by definition of t^*, no other robot moves between $\tilde{t}$ and t^*. Hence, $(r^*, \cdot)$ satisfies Equation (4.1) at time $\tilde{t}$ too, contradicting the hypothesis that $(r, p)_{\tilde{t}}$ satisfies (4.1) at time $\tilde{t}$.

If $r^* \in \mathbb{C}_+(t^*) \cup \mathbb{M}_+(t^*)$, then $r^* \in (\mathbb{L}_+(\tilde{t}) \setminus \mathbb{L}_S(\tilde{t})) \cup \mathbb{C}_+(\tilde{t}) \cup \mathbb{M}_+(\tilde{t})$, having a contradiction. Therefore, since all the robots different from r are in $\mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$, for all $\tilde{t} \leq t \leq t_m$, the second statement of the lemma follows.

By definition of t_l and by C2, r starts the execution of `Go_To_Points()` at time t_c. Since all the other robots are in $\mathbb{SD}(\cdot)$ between time $\tilde{t}$ and t_m, (r, p) is the pair that satisfies Equation (4.1) at time t_l with respect to *FreeP* and *FreeR*. Therefore, r is allowed to move by `Go_To_Points()`; hence $r \in \mathbb{C}_+(t_c)$ and $r \in \mathbb{M}_+(t_m)$, and the lemma follows. □

Therefore, according to the previous lemma r is the only robot in $FreeR$ allowed to move at time t_m. The following theorem states that, if no robot is between r and p, then r reaches p in a finite number of cycles, while all the other robots do not move.

Theorem 4.3.1. *Let conditions C1–C3 hold, and no robot be between p_r and p on $[p_r p]$. Then, in a finite number of cycles, say at time $t' > t_m$, r will reach p avoiding collisions. If $[p_r p] \subseteq \Gamma$ or $[p_r p] \subseteq \Gamma'$, then r moves always on one of the two borders of $\mathcal{V}$ between time t_m and t'; otherwise, r moves always strictly inside $\mathcal{V}$ between time t_m and t'. Furthermore, all the robots in $\mathbb{K}(\tilde{t}) \setminus \{r\}$ are in $\mathbb{SD}(t)$, for all $t_m \leq t \leq t'$, and $r \in \mathbb{SD}(t')$.*

Proof. By Lemma 4.3.3, no robot moves until time t_m, and r is the only

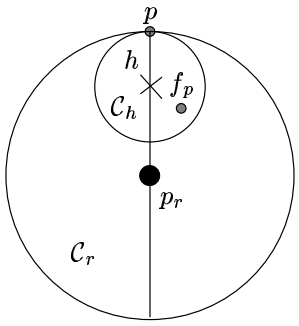


Figure 4.7: Lemma 4.3.1.

robot allowed to move. Moreover, r starts a *Compute* state (where it executes `Go_To_Point()`) at time t_c.

According to `Go_To_Points()`, r moves straight towards p, and since no robot is on its way, any collision is avoided. During this movement, it follows by Equation (4.1) that r remains the closest robot to p. Furthermore, during its movement towards p, there is no other point in $FreeP$ that becomes closer to r than p. In fact, by contradiction, let f_p be such a point, and let h be a point on $[p_r p]$ such that $dist(h,p) > dist(h,f_p)$ (see Figure 4.7). Then, f_p is clearly inside the circle $\mathcal{C}_h$ centered in h and having radius $[hp]$. But, by construction, $\mathcal{C}_h$ is completely inside $\mathcal{C}_r$, that is f_p is inside $\mathcal{C}_r$, having a contradiction by Observation 4.3.1. Therefore, while r approaches p, r remains the robot in $FreeR$ closest to p, and p remains the point in $FreeP$ closest to r; hence, by C2–C3 (r,p) is the only pair that satisfies (4.1), and r is the only robot allowed to move until it reaches p. Thus, by Assumptions A1 and A2, in a finite number of cycles, say at t', r reaches p avoiding collisions, and, until this happens, all the other robots compute only *null movements*. Moreover, $r \in \mathbb{W}(t') \cup \mathbb{L}_S(t')$, and the theorem follows. □

The following theorem deals with the case in which there is some robot between r and p on $[rp]$. Also in this case, no robot is allowed to move until r reaches p, and r reaches p in a finite number of cycles while avoiding collisions.

Theorem 4.3.2. *Let conditions C1–C3 hold, and a robot not in FreeR be between p_r and p on $[p_r p]$. Then, in a finite number of cycles, say at time $t' > t_m$, r will reach p, avoiding collisions and moving always strictly inside $\mathcal{V}$. Furthermore, all the robots in $\mathbb{K}(\tilde{t}) \setminus \{r\}$ are in $\mathbb{SD}(t)$, for all $t_m \leq t \leq t'$, and $r \in \mathbb{SD}(t')$.*

Proof. By Lemma 4.3.3, no robot moves until time t_m, and r is the only robot allowed to move. Hence, $\mathcal{C}_p$, *Intersections*, *Intersections'*, p_I, p'_I, and p_{cl} do not change between time $\tilde{t}$ and t_m. Moreover, r starts a *Compute* state (where it executes

`Go_To_Point()`) at time t_c. Let p_{in} and p_{in_s} as computed in Lines 31–34 of the routine.

If p_{cl} is inside $\mathcal{C}_p$, then $p_{in} = p_{cl}$ is the point among p_I and $p_{I'}$ that is closest to p. Hence, equations (4.4) and (4.5) hold also in $\triangle(p_r, p_{in}, p_{in_s})$. If p_{cl} is not inside $\mathcal{C}_p$ or p_{cl} =`Nil` (i.e., $Intersections \cup Intersection' = \emptyset$), p_{in} is chosen as the point on Ψ' at distance $dist(r,p)/2$ from p (hence, p_{in} is inside $\mathcal{C}_p$, being $[p_r p]$ the radius of $\mathcal{C}_p$), and (4.4) and (4.5) hold for $\triangle(p_r, p_{in}, p_{in_s})$ in this case too.

Then, point p^* is computed. Specifically, let Φ be the result of the intersection between Ψ' and $\mathcal{V}$. Since $p \in \Psi'$ and p is inside $\mathcal{V}$, then $\Phi \neq \emptyset$. Moreover, since $\Gamma \neq \Gamma'$, then $\Phi \subseteq \Psi'$; that is Φ does not contain only p (i.e., it is either a segment or a line). Let Λ be the intersection between $[p_{in}p_{in_s}]$ and Φ. Since $p \in \Phi$ and $p \in [p_{in}p_{in_s}]$, then $\Lambda \neq \emptyset$. Moreover, since by construction p is the median point on $[p_{in}p_{in_s}] \subseteq \Psi'$, Λ does not contain only p; hence $\Lambda \subseteq \Psi'$, $p^* \in [p_{in}p_{in_s}]$, and $[p_r p^*]$ is inside $\triangle(p_r, p_{in}, p_{in_s})$ (see Figure 4.4.d and e).

At time t_m, r starts moving towards p^*. By what observed above, while r moves towards p^*, r is always inside $\triangle(p_r, p_{in}, p_{in_s})$; hence, by (4.4), p remains the point in $FreeP$ closest to r, by (4.5) r remains the robot in $FreeR$ closest to p, and by C2–C3 (r,p) is the only pair that satisfies (4.1) (that is, r is the only robot in $FreeR$ allowed to move until it reaches p^*). By Assumptions A1 and A2, a finite time after r starts moving towards p^*, r reaches a point l on the segment $[p_r p^*]$ (l can be p^*), while all the other robots compute only *null movements* (either because they are not robots in $FreeR$, or because routine `Go_To_Points()` forces them to not move since r is the only one allowed to move). When this happens, r is still in $FreeR$, and by (4.4) and (4.5) (r,p) is returned again by routine `Minimum()`. Since $[lp]$ is inside $\triangle(p_r, p_{in}, p_{in_s}) \subseteq \triangle(p_r, p_{cl}, p_{cl_s})$, by Lemma 4.3.2 there is no robot on the segment connecting l to p. Therefore, r chooses p as its destination point. The theorem follows by applying an argument similar to the one adopted in Theorem 4.3.1. □

In other words, Theorem 4.3.2 implies that the robots move sequentially towards their $Final\_Positions$.

Correctness of Algorithm 1

Now, we are ready to show that Algorithm 1 lets the robots correctly form the input pattern. Let an *agreement configuration* of the robots be a configuration in which, giving to the robots' positions a lexicographic order according to routine `Sort()`, the first two robots r_1 and r_2 are such that `Angle`$(r_1, r_2) = \alpha$, with α the angle computed in Line 1 of the algorithm.

Lemma 4.3.4. *If the robots are not in a final configuration, then in a finite number of cycles, say at time t_α, and avoiding collisions, they are in an agreement config-*

uration, by executing Algorithm 1. Furthermore, all the robots but r_1 *are in* $\mathbb{SD}(t)$*, for all* $0 \leq t \leq t_\alpha$*, and* $r_1 \in \mathbb{SD}(t_\alpha)$*.*

Proof. Let $\mathbb{G}$ be the initial configuration of the robots, and let us give it a lexicographic order according to routine `Sort()`. Let r_1 and r_2 be the first two robots in this ordering. If `Angle`$(r_1, r_2) = \alpha$, then the lemma clearly follows, and $t_\alpha = 0$. Otherwise, according to the algorithm, the only robot that is allowed to move at the beginning is r_1 (Lines 9, 12, and 28), and it executes `Go_Into_Position`(r_1, r_2, α). During this movements, r_1 stays lexicographically first; hence, as long as `Angle`$(r_1, r_2) \neq \alpha$, it is the only robot allowed to move, and all the others compute only *null movements*. In a finite number of cycles, r_1 reaches a position such that `Angle`$(r_1, r_2) = \alpha$, say at time t_α. Since until t_α no robot but r_1 is allowed to move, the lemma follows. □

Finally, we can state that

Theorem 4.3.3. *Algorithm 1 lets the robots correctly form the input pattern* $\mathbb{P}$ *in a finite number of cycles, while avoiding collisions.*

Proof. According to the previous lemma, at time t_α the robots are in an agreement configuration. From now on, r_1 and r_2 never move again (Lines 9 and 11). At this point, the distance $Unit = dist(r_1, r_2)$ is used in routine `Find_Final_Positions`$(r_1, r_2, Unit)$ to compute the positions the robots have to reach in order to correctly form the input pattern (Line 16). By definition of this routine, and since at t_α all the robots are in $\mathbb{SD}(t_\alpha)$, from now on all the robots will agree on the set *Final_Positions* (i.e., in subsequent computations, they will all compute the same set of points, up to the translations due to the different origins of the local coordinate systems). Moreover, by the way these positions have been computed, r_1 and r_2 are already on their final targets. Let Γ and Γ' be as defined in Lines 23 and 26 of the algorithm.

If a robot is on a final position, it never moves again (Line 19). Otherwise, it will call at every cycle routine `Go_To_Points()` (Line 27). Let (r, p) be the pair that satisfies Equation (4.1) at time t_α with respect to the sets *Free_Robots* and *Free_Points* as defined in Lines 21 and 22 of the algorithm. According to the algorithm, r is the only robot allowed to move at t_α, and at t_α all the robots are in $\mathbb{SD}(t_\alpha)$ (by Lemma 4.3.4); furthermore, all the robots but r_1 are inside the region of the plane delimited by Γ and Γ'. Therefore, since $|Free\_Robots| = |Free\_Points|$ at t_α , by Theorems 4.3.1 and 4.3.2, r reaches in a finite number of cycles p while avoiding collisions, say at $t' > t_\alpha$, while all the other robots compute only *null movements*. Furthermore, r do not trespass the region of the plane delimited by Γ and Γ'. By the way the input pattern has been scaled, no final positions can be on Γ below

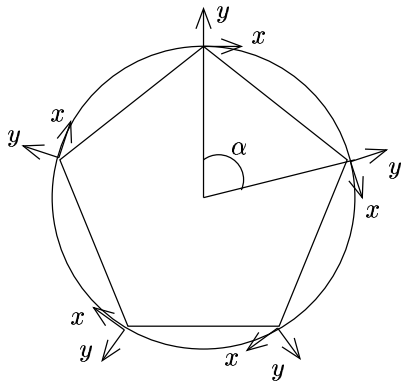


Figure 4.8: Theorem 2: the unbreakable symmetry of a 5-gon.

r_2. Furthermore, by the way the robots' positions have been sorted by `Sort()`, no robot can be on Γ below r_2. Hence r_1 and r_2 stay the first two lexicographical robots in the system. Moreover, at t' the cardinality of sets *Free_Robots* and *Free_Points* decreases by one. Hence, since the number of points in the pattern coincides with the number of robots in the system, and by iterating the above argument, eventually each robot will occupy a pattern point position, while avoiding collisions. □

Result 1. *With total agreement on the local coordinate systems, a set of autonomous, anonymous, oblivious, mobile robots can form an arbitrary given pattern.*

4.3.2 No Agreement

We will now show that giving up the common knowledge on at least one axis direction leads to the inability of the system to form arbitrary patterns.

Result 2. *With no agreement on the local coordinate systems, a set of autonomous, anonymous, oblivious, mobile robots cannot form an arbitrary given pattern.*

Proof. By contradiction, let $\mathcal{A}$ be a deterministic algorithm for solving the pattern formation problem with no common knowledge of the coordinate system. We show that there are input patterns, initial configurations of the robots, and a scheduling of the actions of the robots, such that the robots never can form the input patterns. Consider any pattern different from a regular n-gon or a single point, and let the initial positions be such that the robots form a regular n-gon. Let $\alpha = 360°/n$ be the characteristic angle of the n-gon, and let the local coordinate system of each robot be rotated by α with respect to its neighbor on the polygon (see Figure 4.8). In this situation, all the robots have the same view of the world. Now, for any move

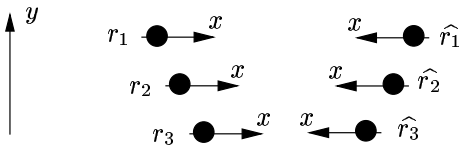


Figure 4.9: Specular configuration for the proof of Theorem 4.4.1, where each r_i has the same view of the world of $\widehat{r}_i$, for $i = 1, 2, 3$.

that any one robot can make in its local coordinate system by executing algorithm $\mathcal{A}$, we know that each robot can make the same move in its local coordinate system. If all of them move in the exact same way at the same time (i.e., they execute $\mathcal{A}$ according to a synchronous activation schedule), they again end up in a regular n-gon or a single point. Therefore, by letting all the robots move at the same time in the same way, we always proceed from one regular n-gon or single point to the next. Hence, the desired pattern cannot be formed. □

4.4 Partial Agreement: Basic Limitations

Let us now look at the case when only one axis direction and orientation is common knowledge of the robots. Without loss of generality, in the following we will assume that the robots agree on the direction and orientation of y. As an aside, note that this case would trivially coincide with the case with total agreement, if the robots would have a common handedness (or sense of orientation, as Suzuki and Yamashita call it [76, 78]).

We first show that in general, the arbitrary pattern formation problem is deterministically unsolvable. We will then show that for the special case of an odd number of robots, any symmetry can be broken and an arbitrary pattern can be formed.

The following lemma states the unsolvability of the leader election problem under the assumption of an even number of robots in the system.

Theorem 4.4.1. *There exists no deterministic algorithm that solves the leader election problem, when n is even.*

Proof. By contradiction, let $\mathcal{A}$ be a leader election algorithm. Consider now an initial placement of the robots in which each robot r has a *specular partner* $\widehat{r}$: the directions of the x axis of r and the x axis of $\widehat{r}$ are opposite, and the view of the world is the same for r and $\widehat{r}$ (see Figure 4.9). Furthermore, consider a scheduler which, for any move that r makes in its local coordinate system by executing algorithm

$\mathcal{A}$, will have its partner $\widehat{r}$ make the same move in its local coordinate system at the same speed. In this way they again end up in specular positions. Hence, each robot will always have a specular partner which, due to the lack of agreement on the direction and orientation of the x axis, is indistinguishable from it. Hence a leader cannot be elected. □

Corollary 4.4.1. *In a system with $n > 2$ anonymous robots that agree only on one axis direction and orientation, the arbitrary pattern formation problem is unsolvable when n is even.*

Proof. It follows from Theorems 4.2.1 and 4.4.1. □

In the following sections, we approach separately the case when the number of robots in the system is odd or even.

4.5 Partial Agreement: Odd Number of Robots

We now show that for breaking the symmetry, it is sufficient to know that the number n of robots is odd. We make use of the fact that either the robots are in a symmetric initial situation, in which there is a unique middle robot that will move in order to break the symmetry, or the situation is not symmetric in the very beginning, and hence the asymmetry can be used to identify an orientation of the x axis. We feel that this technique of symmetry breaking for mobile robots may have other applications, and hence it may be of independent interest. Also in this algorithm there is no use of the possibility to rotate the final positions w.r.t the input pattern.

First, given a pattern $\mathbb{P}$ and a direction for the x axis, we define some *reference points* for $\mathbb{P}$ (see Figure 4.10.a and b.). We denote by p_m, the median point in $\mathbb{P}$; the ordering is given left-right, top-down according to the local orientation of the axes; moreover, Υ_m denotes the vertical line passing through the median point. Let Υ and Υ' be the two vertical lines that are tangent to the convex hull of $\mathbb{P}$, with Υ to the right of Υ' (according to the local orientation of the axes), and p^* and p^{**} the topmost points on Υ and Υ', respectively; then

$$p_o = \begin{cases} p^* & \text{if } \overline{\Upsilon_m \Upsilon} \geq \overline{\Upsilon_m \Upsilon'} \\ p^{**} & \text{otherwise,} \end{cases}$$

and

$$p'_o = \begin{cases} p^* & \text{if } \overline{\Upsilon_m \Upsilon} < \overline{\Upsilon_m \Upsilon'} \\ p^{**} & \text{otherwise;} \end{cases}$$

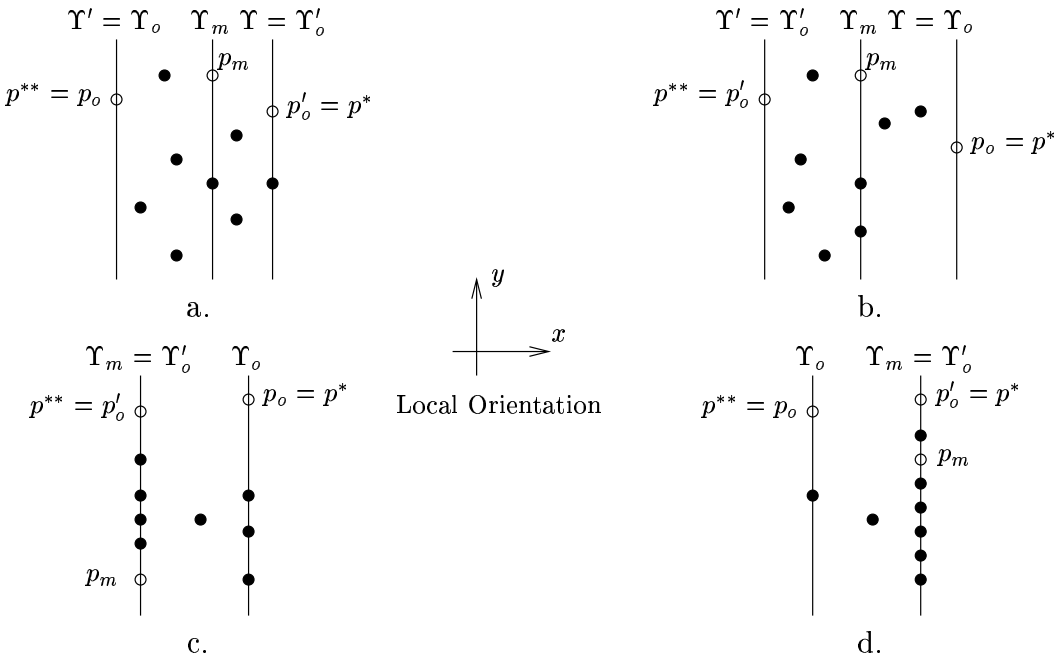


Figure 4.10: (a) Computing the *reference points* of the input pattern. In this case, $\overline{\Upsilon_m \Upsilon} \neq \overline{\Upsilon_m \Upsilon'}$, while in (b) $\overline{\Upsilon_m \Upsilon} = \overline{\Upsilon_m \Upsilon'}$. In (c) and (d) the two possible scenarios when $\Upsilon'_o \equiv \Upsilon_m \wedge \Upsilon'_o \not\equiv \Upsilon_o$ are depicted.

we will call p_o and p'_o also the *first* and the *second* outermost points in $\mathbb{P}$. Furthermore, Υ_o (resp. Υ'_o) denotes the vertical line through p_o (resp. p'_o). From the above definitions, it follows that

Observation 4.5.1. Given a pattern $\mathbb{P}$, if Υ_o coincides with Υ_m, then all the points in $\mathbb{P}$ lie on the same vertical line. If Υ'_o is distinct from Υ_m, then Υ'_o, Υ_m, and Υ_o are three distinct vertical lines. ◇

In order to solve the arbitrary pattern formation problem with an odd number of robots, we distinguish the three possible cases:

Case a. Υ'_o is distinct from Υ_m ($\Upsilon'_o \not\equiv \Upsilon_m$, see Figure 4.10.a and b.);

Case b. Υ'_o coincides with Υ_m, and Υ'_o is distinct from Υ_o ($\Upsilon'_o \equiv \Upsilon_m \wedge \Upsilon'_o \not\equiv \Upsilon_o$, see Figure 4.10.c and d.);

Case c. Υ_o coincides with Υ_m ($\Upsilon_o \equiv \Upsilon_m$).

We discuss the three cases separately.

4.5.1 Case a.: $\Upsilon'_o \not\equiv \Upsilon_m$

In this section we present an algorithm that deals with Case a.: Υ'_o is distinct from Υ_m in $\mathbb{P}$ (Algorithm 2). Its correctness will be proven under an extra assumption on the model. Namely,

Assumption A3(Observability) *A Move of length δ_r is not observable.*

> In particular, if a robot r starts at time t a *Move* of length δ_r, and it arrives at its destination point at time $t' > t$, then a robot $r' \neq r$ that executes a *Look* between time t and t' observes r either on its starting point (i.e., the position occupied by r at time t) or on its destination point (i.e., the position where r is at time t').

The parts of Algorithm 2 that make use of Assumtpion A3 will be explicitly stated.

The first thing the algorithm does is to let the robots agree on a *global coordinate system* (*GCS*), so that the patterns they have in input can be translated and scaled with respect to this system, and the pattern correctly formed. In order to do so, the algorithm computes first the reference points of the pattern it has in input.

After the reference points in $\mathbb{P}$ have been computed, the points in the view of the world corresponding to p_m, Υ_m, p_o and p'_o are located. In particular, `Median_Robot_Line()` returns the vertical line Γ_m through the median of the robots'

Algorithm 2 Arbitrary Pattern Formation, Partial Agreement, n odd, Case a.

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order, such that Υ'_o is distinct from Υ_m. The direction and orientation of the y axis is common knowledge.

$(p_m, p_o, p'_o, \Upsilon_o, \Upsilon'_o)$:= Reference Points In $\mathbb{P}$;
Γ_m := `Median_Robot_Line()`; %through the median robot position%
If $|\Gamma_m| = n$ **Or** $|\Gamma_m| = n-1$ **Then** `Same_Vertical_Axis(`Γ_m`)`.
If We Are In A *Final Configuration* **Then** `do_nothing()`.
5: (*Outer_Robot*, *Outer_Robot'*, *Asym_Side*):= `Outermost_Robot_Position(`Γ_m`)`;
Γ_o := Vertical Line Through *Outer_Robot*;
Γ'_o := Vertical Line Through *Outer_Robot'*;
If Γ'_o Coincides With Γ_m **Then**
    If I Am *Outer_Robot'* **Then** `Move_Away(`$\delta_{Outer\_Robot'}$`)`.
10: **Else** `do_nothing()`.
GCSx := `Orient_X(`p_o, *Outer_Robot*`)`;
Median_Robot := `Find_Median_Robot(`Γ_m`)`;
If I Am *Median_Robot* **Then** `do_nothing()`.
Final_Positions := `Find_Final_Positions(`*Median_Robot*, p_m, Γ'_o, $\Upsilon_{o'}$, *GCSx*`)`;
15: *External* := `Farthest(`*Final_Positions*, Γ_m, *Asym_Side*`)`;
If I Am Not *Outer_Robot* **Then**
    Final_Positions := {*Final_Positions*} $\setminus$ *External*;
(hd_o, hd'_o, hd_e) :=`Horiz_Dist(`*Outer_Robot*, *Outer_Robot'*, *External*, Γ_m`)`;
If $hd_o \leq hd_e \vee hd_o = hd'_o$ **Then**
20: **If** I Am *Outer_Robot* **Then**
        p := Point In *Asym_Side* Such That $\overline{p\Gamma_m} > \max(hd_e, hd'_o)$;
        `Move(`p`)`.
    Else `do_nothing()`.
Not_Asym_Side := Side Opposite To *Asym_Side*;
25: *External'* := `Farthest(`*Final_Positions*, Γ_m, *Not_Asym_Side*`)`;
If No Robot Is On *External'* **Then**
    If I Am The Robot On Γ'_o Closest To *External'* **Then**
        `Move(`*External'*`)`.
    Else `do_nothing()`.
30: **If** I Am On One Of The *Final_Positions* **Then** `do_nothing()`.
If I Am *Outer_Robot* **Then**
    Available := {*Positions In Final_Positions With No Robots On Them*};
    If *Available* = {*External*} **Then** `Move(`*External*`)`.
    Else `do_nothing()`.
35: $\mathcal{M}ySide$:= `My_Side(`Γ_m`)`;
Free_Points := {*Final_Positions In* $\mathcal{M}ySide$ *With No Robots On Them* };
Free_Robots := {*Robots In* $\mathcal{M}ySide$ *Not On Final_Positions*} $\setminus$ {*Outer_Robot*};
Γ^*_o := Vertical Line Among Γ_o And Γ'_o That Is In $\mathcal{M}ySide$;
`Go_To_Points(`*Free_Robots*, *Free_Points*, Γ_m, Γ^*_o`)`.

positions of the configuration retrieved in the last *Look* state; in particular, the ordering is given left-right, top-down, according to the local orientation of the coordinate system. Since there is no agreement on the direction and orientation of x, it is possible that this ordering may not be the same for all the robots. Note that, however, the position of Γ_m does not depend on the local orientations of the x axes of the robots, hence all the robots agree on it (see Observation 4.5.2 in Section "Correctness" below). Γ_m represents the vertical axis of the *GCS* the robots are trying to achieve, and it splits the plane in two *half-planes* (it is the line corresponding to Υ_m). At this point all the robots agree also on the orientation of Γ_m (since they agree by hypothesis on the orientation of the y axis); hence, they also agree on the direction of the horizontal axis of the *GCS* (i.e., orthogonal to Γ_m), but not on its orientation yet.

Then, the robots check if they all lie on Γ_m. In this case, since by hypothesis the input pattern can not be a vertical line ($\Upsilon'_o \not\equiv \Upsilon_m$), the robots put themselves on two distinct vertical lines, by calling routine `Same_Vertical_Line(`Γ_m`)`. In particular, this routine lets the second topmost robot on Γ_m move away from this line.

```
Same_Vertical_Axis(Ξ)
    d :=dist(top(Ξ),bottom(Ξ));
    Case |Ξ|
        • n
            If I Am The Second Topmost Robot On Ξ Then
 5:             p := Point To My Right At Horizontal Distance d From Ξ;
                Move(p).
            Else
                do_nothing().
        • n − 1
10:         r := Robot Not On Ξ;
            If I Am r Then
                If I Am Not At Horizontal Distance d from Ξ Then
                    p := Closest Point To Me At Horizontal Distance d From Ξ;
                    Move(p).
15:             Return
            Else
                If r Is Not At Horizontal Distance d From Ξ Then
                    do_nothing.
                Return
```

where `Move(`p`)` terminates the local computation of the calling robot and moves it towards the point p, using a straight movement. In other words, the distance

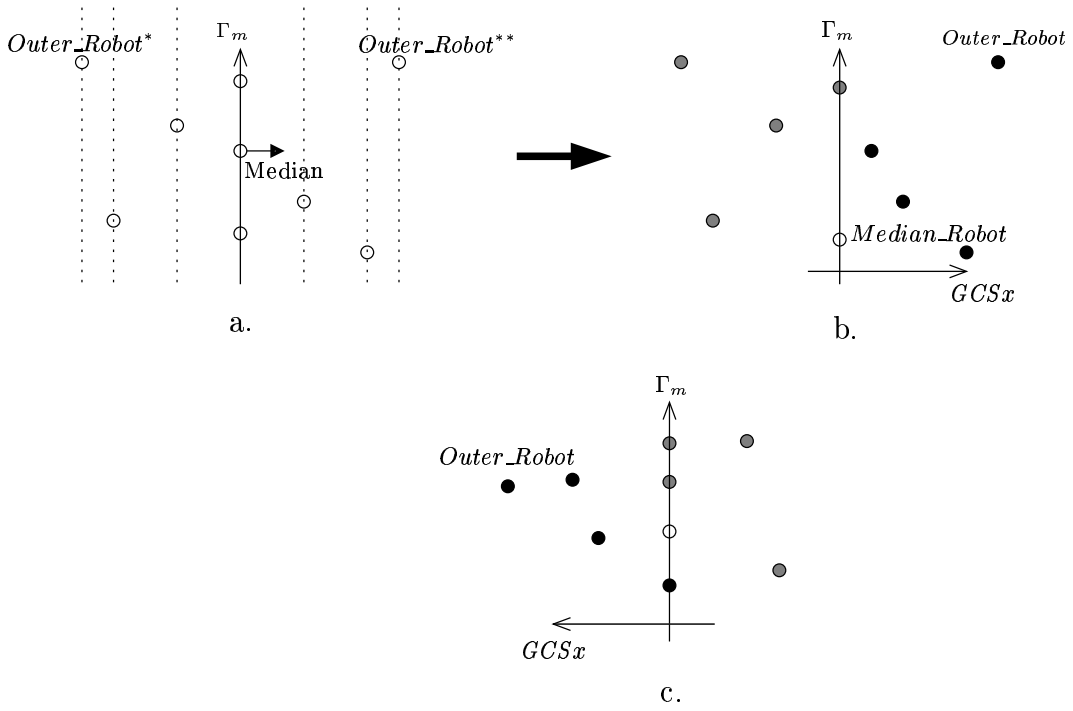


Figure 4.11: Breaking Symmetry from (a) to (b): in this example we have that $GCSx$ coincides with the $Asym\_Side$; (c) defining the sides: in this example, the black robots belong to one side, and the grey ones to the opposite side; also in this case, $GCSx$ coincides with the $Asym\_Side$.

d between the topmost (returned by `top(`Ξ`)`) and the bottommost (returned by `bottom(`Ξ`)`) robot on Ξ is computed by `dist(top(`Ξ`),bottom(`Ξ`))`; if there are exactly n robots on Ξ, the second topmost robot r on Ξ moves to its right until it is at an horizontal distance d from Ξ (note that, while r is moving, the case $|\Xi| = n-1$ forces all the other robots to stay still until r is at distance d from Ξ).

Next, the robots check if they are already forming the desired pattern. They do so by mapping Υ_m onto Γ_m, and the two outermost topmost robot's positions, say r and r', onto p_o and p'_o. They try the two possible mappings: p_o and p'_o onto r and r'; and p'_o and p_o onto r and r'. If neither of the attempts work, they are not forming the desired pattern.

`Outermost_Robot_Position(`Γ_m`)` locates the robots corresponding to p_o and p'_o. In particular, it uniquely determines a robot that is outermost with respect to Γ_m. It does so by breaking symmetry of the configuration, if necessary, in the following way (see Figure 4.11.a and b):

`Outermost_Robot_Position(`Γ_m`)`

($Outer\_Robot^*$, $Outer\_Robot^{**}$) := `Outer_Two_Robots(`Γ_m`)`;
If `Equidistant(`Γ_m`)` **Then**
    r := Median Robot On Γ_m;
    If I Am r **Then** `Move(`*to my right by* δ_r`)`.
    Else `do_nothing()`.
$Asym\_Side$:= `Outermost_Asymmetry(`Γ_m`)`;
$Outer\_Robot$:= Choose Between $Outer\_Robot^*$ And $Outer\_Robot^{**}$ The One That Lies In The $Asym\_Side$;
$Outer\_Robot'$:= $\{Outer\_Robot^*, Outer\_Robot^{**}\} \setminus \{Outer\_Robot\}$;
Return ($Outer\_Robot$, $Outer\_Robot'$, $Asym\_Side$).

`Outer_Two_Robots(`Γ_m`)` returns the two topmost robots that lie on the two farthest vertical lines from Γ_m, one in each of the half-planes determined by Γ_m.

`Equidistant(`Γ_m`)` returns *True* if the current configuration of the robots in the system is "equidistant" with respect to Γ_m, in the following sense. The configuration is called *equidistant with respect to* Γ_m, if it is possible to partition all the robots not on Γ_m in subsets $s = (r, r')$, $r \neq r'$, such that r and r' lie on two distinct vertical lines with the same horizontal distance from Γ_m (see Figure 4.11.a). Clearly, given a configuration that is not equidistant with respect to Γ_m, there exists at least one robot r not on Γ_m that can not be placed in any of the subsets. `Outermost_Asymmetry(`Γ_m`)` identifies, for a not equidistant configuration, the unique half-plane with respect to Γ_m which contains the outermost (w.r.t. Γ_m) of such robots (see Figure 4.11.b). We will call this half-plane the *asymmetric side* in the configuration.

Hence, after the execution of `Outermost_Robot_Position(`Γ_m`)`, two outermost robots with respect to Γ_m have been determined: $Outer\_Robot$ and $Outer\_Robot'$, corresponding to p_o and p'_o, respectively. It is still possible, however, that Γ'_o (vertical line through $Outer\_Robot'$) coincides with Γ_m. Since line Υ'_o does not coincide with Υ_m in $\mathbb{P}$, this scenario is broken by moving of a distance δ_r the topmost robot r on Γ'_o (i.e., $Outer\_Robot'$) outside the vertical stripe with borders Γ_m and Γ_o. This is accomplished by `Move_Away(`δ_r`)`[3]. In this way, the three lines Γ_o, Γ'_o and Γ_m will be distinct (similarly to what happens in $\mathbb{P}$). In the following, since $\overline{\Gamma_m\Gamma_o} \geq \overline{\Gamma_m\Gamma'_o}$, we will also refer to $Outer\_Robot$ as the *first outermost* robot, and to $Outer\_Robot'$ as the *second outermost* robot.

The next step of the algorithm is to give the robots a common sense of what is to the right of Γ_m, and what is to its left. This is achieved by routine `Orient_X(`p_o`,` $Outer\_Robot$`)`, that returns the orientation of the x axis in *GCS* (refer to Figure 4.12). In particular, the routine maps the half-plane that (locally) contains p_o to the half-plane that (globally) contains $Outer\_Robot$, that is $Asym\_Side$. Thus, if

[3]Note that in this case, Assumption A3 of the model is used.

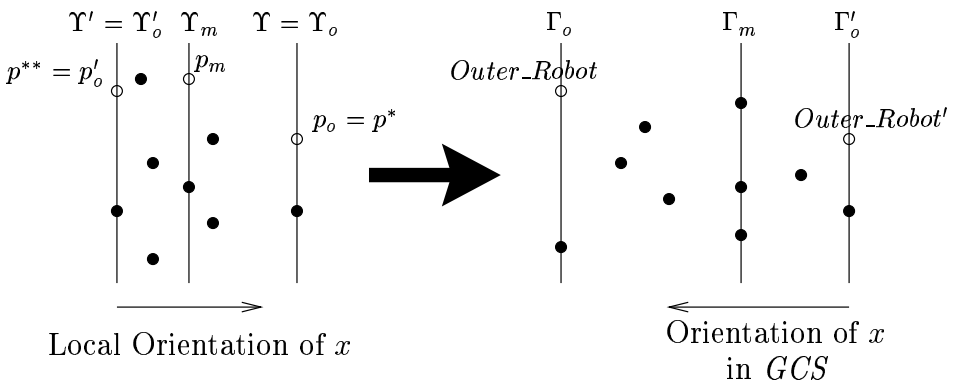


Figure 4.12: Computing the orientation of the x axis in GCS, in routine `Orient_X(`$p_o, Outer\_Robot$`)`. To the left, the input pattern $\mathbb{P}$ is depicted, with its reference points. To the right, the mapping of these points onto the robots' positions as done by `Orient_X()` is showed.

in $\mathbb{P}$ we have Υ'_o to the left of Υ_o (according to the local orientation of the x axis), then the x axis of GCS is oriented in such a way that Γ'_o is (globally) to the left of Γ_o, and viceversa. In the following we assume without loss of generality that the first case applies (Υ'_o locally to the left of Υ_o); hence, we assume that in GCS the x axis is oriented from Γ'_o towards Γ_o. The other case can be approached symmetrically. Thus, at this point, all the robots agree also on the orientation of the horizontal axis of the GCS, and the information on the orientation of the x axis in GCS is stored in $GCSx$.

At this point, `Find_Median_Robot(`Γ_m`)` finds the median robot position in the current configuration of the robots, according to the global coordinate system. In particular, the ordering is given left-right, top-down. Note that, since at this point the robots agree on GCS, they agree also on such a robot. This median robot's position splits the robots' positions into two equal-sized subsets, of size $(n-1)/2$, defined as follows. In one subset there are the robots in $Asym\_Side$, including the robots on Γ_m that are *below* the median robot position, and in the other subset there are the robots in the other half-plane defined by Γ_m, including the robots on Γ_m that are *above* the median robot's position[4]; from now on, we will call these subsets the two *sides* (see Figure 4.11.c). According to this definition, the median robot position is unique, and each robot (even if it lies on Γ_m) can decide to which side it belongs. In particular, the routine `My_Side(`Γ_m`)` returns the side where the calling robot currently lies.

[4]In the case when $GCSx$ is oriented from Γ_o towards Γ'_o, the two subsets are computed by substituting *above* with *below*, and viceversa.

Routine `Find_Final_Positions`($Median\_Robot, p_m, \Gamma'_o, \Upsilon_{o'}, GCSx$), returns the set of positions the robots have to reach in order to correctly form the given pattern, based on the agreement on *Median_Robot* and on $GCSx$. That is, Υ_m is mapped onto Γ_m, hence p_m onto the position of *Median_Robot*; $\Upsilon_{o'}$ onto Γ'_o; and the orientation of the pattern to form is chosen according to $GCSx$. The common scaling of the input pattern is defined by identifying $\overline{\Upsilon_m \Upsilon_{o'}}$ with $\overline{\Gamma_m \Gamma'_o}$.

Routine `Farthest`($Final\_Positions, \Gamma_m, Asym\_Side$) returns the topmost point in *Final_Positions* that is in *Asym_Side* with greatest horizontal distance from Γ_m; this point is stored in *External*. This is the position that will be taken by *Outer_Robot*. Routine `Horiz_Dist()` returns hd_o, hd'_o, and hd_e, that are the horizontal distances of *Outer_Robot*, *Outer_Robot'*, and *External* from Γ_m, respectively. In order to be sure that the agreement on $GCSx$ does not change while the robots approach the *Final_Positions*, *Outer_Robot* is forced to move to a position that has horizontal distance from Γ_m greater than $\max(hd_e, hd'_o)$. In this way, since no point in *Final_Positions* has horizontal distance from Γ_m greater than hd_e, and since all robots aim to reach a point in *Final_Positions*, no robot can never have horizontal distance from Γ_m greater than hd_o.

Since the scaling of the pattern is done with respect to $\overline{\Gamma_m \Gamma'_o}$, we want to preserve the agreement on Γ'_o while the robots approach their final targets. To this aim, the algorithm forces *External'* (the point in *Final_Positions* that is in *Not_Asym_Side* with greatest horizontal distance from Γ_m) to be reached first. At this point, all the robots that already are on a point in *Final_Positions* (i.e., they are on one of the vertices of the pattern they are forming), never move again.

Routine `Go_To_Points`($Free\_Robots, Free\_Points, \Phi, \Phi'$) is similar to the one described in Figure 4.3. The only difference is that in this case the routine is executed independently in both sides, since $Free\_Robots$ and $Free\_Points$ are computed with respect to $\mathcal{M}ySide$ (Lines 36 and 37). Moreover, routine `Minimum()` breaks ties by choosing the pair (r, p) such that r is the topmost robot that is closest to Γ_m (in the side where the calling robot is) and, in case of another tie, p the topmost closest point to Γ_m.

Correctness of Algorithm 2

In this section we show that Algorithm 2 solves the pattern formation problem for an arbitrary pattern, if $\Upsilon'_o \not\equiv \Upsilon_m$.

Definition 4.5.1. Let $\mathbb{G}$ be a configuration of the robots, *Pred* be a predicate on $\mathbb{G}$, and *Func* be a function that, executed on $\mathbb{G}$, returns a subset of $\mathbb{R}^2$ (i.e., a line, a point, etc.). We say that *Pred* is *uniquely satisfiable* if it returns the same value independently from the orientation of the x axis of the robot that executes *Pred*.

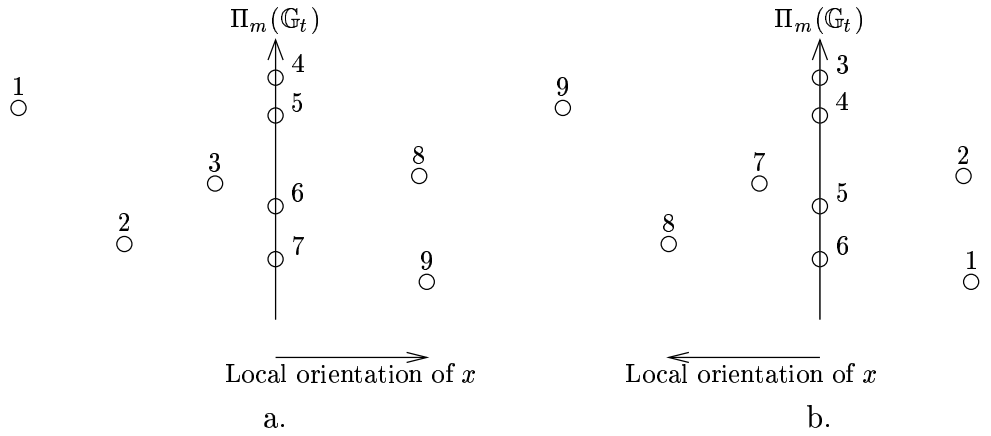


Figure 4.13: The position of the vertical axis $\Pi_m(\mathbb{G}_t)$ through the median robot in $\mathbb{G}_t$ does not depend on the local orientation of the x axis. In both (a) and (b) the median robot is the 5^{th}, but in both cases it lies on the same vertical axis.

Similarly, we say that the result of *Func* is *uniquely identifiable* if *Func* returns the same subset independently from the orientation of the x axis of the robot that executes *Func*. ◁▷

Let $\mathbb{G}_t$ be the configuration of the robots at time t (in particular, $\mathbb{G}_0$ denotes the configuration of the robots at the beginning), and let $\Pi_m(\mathbb{G}_t)$ be the vertical line returned by routine `Median_Robot_Line()` when executed on $\mathbb{G}_t$ (i.e., it is the vertical line through the median robots' positions, with the ordering given left-right, top-down, according to the local orientation of the coordinate system). It is easy to see that

Observation 4.5.2. $\Pi_m(\mathbb{G}_t)$ is uniquely identifiable (see Figure 4.13). ◇

Furthermore, it follows by the definition of `Equidistant(·)`, `Outermost_Asymmetry(`$\Pi_m(\mathbb{G}_t)$`)`, and by previous observation that

Observation 4.5.3. `Equidistant(`$\Pi_m(\mathbb{G}_t)$`)` is a uniquely satisfiable predicate on $\mathbb{G}_t$. Moreover, if $\mathbb{G}_t$ is not equidistant with respect to $\Pi_m(\mathbb{G}_t)$, then the result of `Outermost_Robot_Position(`$\Pi_m(\mathbb{G}_t)$`)` is uniquely identifiable, when this routine is executed on $\mathbb{G}_t$. ◇

Moreover, let $\Pi(\mathbb{G}_t)$ and $\Pi'(\mathbb{G}_t)$ be the two outermost vertical lines, that is the lines that are tangent to the convex hull of the points in $\mathbb{G}_t$. We call the *outermost robots* in $\mathbb{G}_t$, the topmost robot on $\Pi(\mathbb{G}_t)$ and the topmost robot on $\Pi'(\mathbb{G}_t)$. In

particular, the *first* outermost robot is the outermost robot with greatest horizontal distance from $\Pi_m(\mathbb{G}_t)$; the other one is the *second* outermost robot. Analogously, we call the first (resp. second) outermost vertical axis, the vertical axis through the first (resp. second) outermost robot. Note that, according to this definition, if both outermost robots have the same horizontal distance from $\Pi_m(\mathbb{G}_t)$, then the choice of which one is the first and which one is the second depends on the orientation of the x axis in the local coordinate systems of the robots.

We call a configuration of the robots $\mathbb{G}$ where $\Pi(\mathbb{G}) \not\equiv \Pi_m(\mathbb{G}) \not\equiv \Pi'(\mathbb{G})$ a *distinct* configuration of the robots. By this definition, and by Observations 4.5.2 and 4.5.3, it follows that

Observation 4.5.4. If a configuration $\mathbb{G}$ is not a distinct configuration, then it is not equidistant from $\Pi_m(\mathbb{G})$. ◇

Since $\Upsilon'_o \not\equiv \Upsilon_m$, then the final configuration must be distinct. The following lemma shows that, if $\mathbb{G}_0$ is not distinct, then Algorithm 2 brings the robots in a distinct configuration in a finite number of cycles.

Lemma 4.5.1. *If $\mathbb{G}_0$ is not a distinct configuration, then in a finite number of cycles, say at time t_{dis}, the robots are in a distinct configuration, by executing Algorithm 2. Furthermore, until time t_{dis} any collision is avoided, and at t_{dis} all the robot are in $\mathbb{SC}(t_{dis})$.*

Proof. Let $\Gamma_m = \Pi_m(\mathbb{G}_0)$ (by Observation 4.5.2, Γ_m is uniquely identifiable). We distinguish the possible cases.

1. If $\Pi'(\mathbb{G}_0) \equiv \Pi_m(\mathbb{G}_0) \wedge \Pi'(\mathbb{G}_0) \not\equiv \Pi(\mathbb{G}_0)$, then clearly $|\Pi_m| < n$.
   (a) If $|\Pi_m(\mathbb{G}_0)| = n - 1$, then there is only one robot r not on Γ_m. In this configuration, the robots can only call routine `Same_Vertical_Axis`(Γ_m) (Line 3). Let d be the distance between the topmost and the bottommost robot on Γ_m in $\mathbb{G}_0$.
      i. If r is at horizontal distance d from Γ, then routine `Same_Vertical_Axis`(Γ_m) simply returns. Moreover, since by hypothesis $\Upsilon'_o \not\equiv \Upsilon_m$ in $\mathbb{P}$, the robots can not be in a final configuration, the test of Line 4 fails, and routine `Outermost_Robot_Position`(Γ_m) is executed. By Observations 4.5.3 and 4.5.4, the asymmetric side in $\mathbb{G}_0$ is uniquely identifiable. Furthermore, in $\mathbb{G}_0$, the first outermost robot (*Outer_Robot*) is r, while the second outermost robot (*Outer_Robot'*) is the topmost robot on Γ_m. Therefore, $\Gamma'_o = \Pi'(\mathbb{G}_0)$ coincides with Γ_m, and according to Line 9 the only robot allowed

to move is *Outer_Robot'*, that moves of $\delta_{Outer\_Robot'}$ outside the vertical stripe with borders Γ_m and $\Gamma_o = \Pi(\mathbb{G}_0)$. Until this happens, say at time t_{dis}, no other robot is allowed to move (Line 10), and any collision is avoided. Since $n \geq 3$, at time t_{dis} we still have that $\Gamma_m = \Pi_m(\mathbb{G}_{t_{dis}})$, and the configuration of the robots is such that one robot is on $\Pi(\mathbb{G}_{t_{dis}})$, another one is on $\Pi'(\mathbb{G}_{t_{dis}})$, and all the others on $\Gamma_m = \Pi_m(\mathbb{G}_{t_{dis}})$. Furthermore, by Assumption A3, at time t_{dis} all robots $r \neq Outer\_Robot'$ are in $\mathbb{SD}(t_{dis})$, while $Outer\_Robot' \in \mathbb{W}(t_{dis}) \cup \mathbb{L}_S(t_{dis}) \subseteq \mathbb{SD}(t_{dis})$, and the lemma follows.

ii. If r is not at horizontal distance d from Γ_m, by definition of `Same_Vertical_Axis(`Γ_m`)`, no robot is allowed to move until r reaches a point at horizontal distance d from Γ_m. By Assumptions A1 and A2 on the model, r reaches such a point in a finite number of cycles, say at time $t > 0$. Since $\Gamma_m = \Pi_m(\mathbb{G}_t)$, the lemma follows as in previous case i., by substituting $\mathbb{G}_0$ with $\mathbb{G}_t$.

(b) If $|\Pi_m(\mathbb{G}_0)| < n - 1$ then, since by hypothesis $\Upsilon'_o \not\equiv \Upsilon_m$ in $\mathbb{P}$, test in Line 4 fails; hence, in this configuration the robots can only execute routine `Outermost_Robot_Position(`Γ_m`)`. The lemma follows by using an argument similar to the one adopted in previous Case 1.(a).i.

2. If $\Pi(\mathbb{G}_0) \equiv \Pi'(\mathbb{G}_0)$, then $|\Pi_m(\mathbb{G}_0)| = n$. In this configuration the robots can only call routine `Same_Vertical_Axis(`Γ_m`)` (Line 3). According to this routine, the second topmost robot r on $\Pi_m(\mathbb{G}_0)$ is the only robot allowed to move: it moves to a point p at horizontal distance d from $\Pi_m(\mathbb{G}_0)$, with d the distance between the topmost and the bottommost robot on $\Pi_m(\mathbb{G}_0)$. Let t_1 be the first time when r leaves Γ_m. Since $n \geq 3$, $\Pi_m(\mathbb{G}_0) = \Pi_m(\mathbb{G}_{t_1}) = \Gamma_m$. If r is observed when it is not on Γ_m (i.e., when $|\Gamma_m| = n - 1$), the second case of the routine is called: all the robots are forced to not move as long as r is not at horizontal distance d from Γ_m, that is on p. Therefore, until this happens, all the other robots compute only *null movements*, and any collision is avoided. Let $t_2 \geq t_1$ be the first time when r is at horizontal distance d from Γ_m (t_2 is finite, by Assumptions A1 and A2 of the model). The configuration of the robots at this time is such that $n - 1$ robots are on Γ_m, and only one is not, say r'. The lemma follows as in previous Case 1.(a).i. □

In the following lemma, we show that, by executing Algorithm 2, all the robots reach in a finite number of cycles an *odd agreement configuration*. An odd agreement configuration is a distinct configuration $\mathbb{G}$ that is not equidistant with respect to $\Pi_m(\mathbb{G})$.

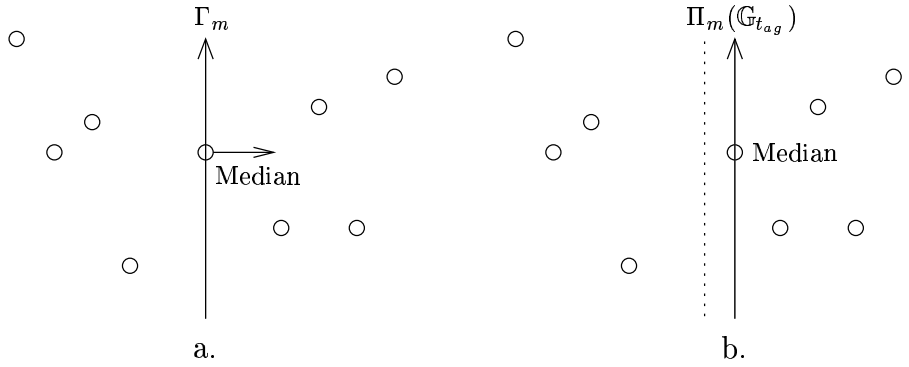


Figure 4.14: Lemma 4.5.2: at time t_{ag} (part (b) of the figure) the position of $\Pi_m(\mathbb{G}_{t_{ag}})$ can be different from the position of $\Gamma_m = \Pi_m(\mathbb{G}_{t_{dis}})$ (part (a) of the figure): this happens if at time t_{dis} there is only one robot on Γ_m, and $\mathbb{G}_{t_{dis}}$ is equidistant with respect to Γ_m. The dashed lines represents the position of Γ_m in $\mathbb{G}_{t_{ag}}$.

Lemma 4.5.2. *If the robots at the beginning are not in a final configuration, they will reach an odd agreement configuration within a finite number of cycles and avoiding collisions, by executing Algorithm 2. Let t_{ag} the first time when this happens. At time t_{ag}, all the robots are in $\mathbb{SD}(t_{ag})$. Furthermore, if the robots are at the beginning in a distinct configuration, the odd agreement configuration is reached within one move of one robot.*

Proof. We distinguish two cases:

1. If the robots are not in a distinct configuration at the beginning, by previous lemma at time t_{dis} such a configuration is reached, with all the robots in $\mathbb{SD}(t_{dis})$. By Observation 4.5.2, independently from the local orientations of the x axes, in $\mathbb{G}_{t_{dis}}$ a vertical axis $\Gamma_m = \Pi_m(\mathbb{G}_{t_{dis}})$ is uniquely identifiable (Line 2).

   If $\mathbb{G}_{t_{dis}}$ is equidistant with respect to Γ_m, then there exists an unique median robot, say r, with r on Γ_m (see Figure 4.11.a). According to the algorithm, it is the only robot allowed to move. Until it decides to move, all robots $r' \neq r$ can compute only *null movements*. Its movement is towards its local concept of right, and the distance it moves is δ_r (see Assumption A3 in Section 3.1). Let t be the time when it starts this movement and t_{ag} the time when it stops. Note that at this time the position of $\Pi_m(\mathbb{G}_{t_{ag}})$ can be different from the position of Γ_m: this happens if at t_{dis} there was only r on Γ_m (see Figure 4.14). In any case, at t_{ag} the robots are clearly not any more equidistant with respect to

$\Pi_m(\mathbb{G}_{t_{ag}})$. By Assumption A3, until t_{ag} all the robots but r computed only *null movements*, while $r \in \mathbb{W}(t_{ag}) \cup \mathbb{L}_S(t_{ag}) \subseteq \mathbb{SD}(t_{ag})$; hence, the lemma follows.

If $\mathbb{G}_{t_{dis}}$ is not equidistant with respect to Γ_m, then the robots are already in an odd agreement configuration (see Figure 4.11.c), and, since all the robots are in $\mathbb{SD}(t_{dis})$, the lemma follows.

2. If the robots are in a distinct configuration at the beginning, then $t_{dis} = 0$. If $\mathbb{G}_0$ is not equidistant with respect to Γ_m, then $\mathbb{G}_0$ is already an odd agreement configuration. Otherwise, one move of robot r (as defined in the previous case) lets the robots be in an odd agreement configuration, and the lemma follows. □

Note that in this algorithm Assumption A3 of the model is necessary. In fact, let us assume the robots are equidistant with respect to Γ_m, and that the median robot is the only robot on Γ_m. If it could be seen while it moves towards its right in routine `Outermost_Robot_Position(`Γ_m`)` (in order to break symmetry), all the other robots might compute different positions for Γ_m, hence making all subsequent computations wrong. Since A3 ensures that the median can be observed either when it starts or after it moved of a distance δ_r, all the robots will always agree on where the median axis is. Beside Line 9, this is the only place in this algorithm where Assumption A3 is used.

According to the previous lemma and to Observation 4.5.3, at time t_{ag} the robots are in a configuration in which it is possible to uniquely identify an asymmetric side, as defined in Line 6 of routine `Outermost_Robot_Position()`. Therefore, a first outermost robot and a second outermost robot are uniquely identifiable in $\mathbb{G}_{t_{ag}}$: in particular, the algorithm lets the robots choose as first outermost robot the topmost robot in the asymmetric side of $\mathbb{G}_{t_{ag}}$ with greatest horizontal distance from $\Pi_m(\mathbb{G}_{t_{ag}})$ (see Figure 4.11.b). Moreover, $GCSx$ returned by `Orient_X()` is uniquely identifiable: we say that at this time all the robots *agree* on a *GCS*. In particular, the y axis of *GCS* is $\Pi_m(\mathbb{G}_{t_{ag}})$, and it is directed as the y axis of the local coordinate systems (by hypothesis, all the robots have common knowledge of it); the direction of the x axis of *GCS* is orthogonal to the direction of y, and its orientation as defined by $GCSx$.

In the following, we will denote by *O* and *O'* the first and the second outermost robot in $\mathbb{G}_{t_{ag}}$, respectively; by Π_o and Π'_o the vertical lines through *O* and *O'*, respectively; by *M* the median robot in $\mathbb{G}_{t_{ag}}$, where the ordering of the robots is given left-right and top-down, according to *GCS*; by Π_m the vertical line through *M* (it is the y axis of *GCS*, and $\Pi_m = \Pi_m(\mathbb{G}_{t_{ag}})$); by *AsSide* and *NotAsSide* the side where *O* and *O'* lie, respectively; by *posX* the orientation of the x axis in *GCS* (recall that we are assuming that it is oriented from the position of

the second outermost robot towards the position of the first outermost robot, see description of `Orient_X()` in Algorithm 2); by FP the set of position as computed by routine `Find_Final_Positions`$(M, p_m, \Pi'_o, \Upsilon'_o, posX)$, with Π'_o the vertical line through O'; by Ext and Ext' the points returned by `Farthest`$(FP, \Pi_m, AsSide)$ and `Farthest`$(FP, \Pi_m, NotAsSide)$, respectively; and by (hd_o, hd'_o, hd_e) the points returned by `Horiz_Dist`(O, O', Ext, Π_m). Furthermore, at time t_{ag} it is possible to split the robots in two equal-sized subsets: one contains the robots in $AsSide$ and those that are on Π_m and below M, call it $\mathbb{K}$; and the other contains the robots in $NotAsSide$ and those that are on Π_m and above M, call it $\mathbb{K}'$.

Lemma 4.5.3. *After an odd agreement configuration has been reached, in a finite number of cycles the position of O becomes such that $hd_o > \overline{p\Pi_m}$, for all $p \in FP \cup (\mathbb{G}_{t_{ag}} \setminus \{O\})$. Let t_{out} be the first time when this happens. Until t_{out} any collision is avoided, and all the robots $r' \neq O$ compute* null movements. *Moreover, $\mathbb{G}_{t_{out}}$ is still an odd agreement configuration; M, O and O' are still the median, the first and the second outermost robot in $\mathbb{G}_{t_{out}}$, respectively; and Π_m, M, and Π'_o do not change position between time t_{ag} and t_{out}. Furthermore, at time t_{out}, $r' \in \mathbb{SD}(t_{out})$, for all $r' \neq O$.*

Proof. By previous lemma, at time t_{ag} the robots are in an odd agreement configuration. Furthermore, by definition of first outermost robot, $hd_o \geq hd'_o$. If $hd_o > \max(hd'_o, hd_e)$ at t_{ag}, by definition of Υ_o, Υ'_o, O, O', and Ext, the lemma follows.

If $hd_o \leq hd_e$ at t_{ag}, then according to the algorithm O is the only robot allowed to move, and all the other robots compute *null movements* until O moves to a position so that $hd_o > \max(hd'_o, hd_e)$. Since by definition of O there are no robots between Π_o and the vertical line through Ext, this movement is accomplished by O with no collisions. Moreover, since at time t_{ag} O is the first outermost robot, and during its movement O moves further away from Π_m, it clearly stays the first outermost robot. Until t_{out} all the other robots computed *null movements*, hence the lemma follows.

If $hd_o > hd_e \wedge hd_o = hd'_o$ at t_{ag}, then there are no robots between Π_o and the point p computed in Line 21 of the algorithm. The lemma follows by using an argument similar to the one adopted in the previous case. □

Lemma 4.5.4. *After time t_{out}, in a finite number of cycles Ext' is reached by a robot r, say at time t_{occ}. Until t_{occ} any collision is avoided, all the robots $r' \neq r$ compute* null movements, *and $\mathbb{G}_{t_{out}}$ is still an odd agreement configuration; M, O and O' are still the median, the first and the second outermost robot in $\mathbb{G}_{t_{out}}$, respectively; and Π_m, M, and Π'_o do not change position between time t_{out} and t_{occ}. Furthermore, at time t_{occ}, all the robots, with the possible exception of O, are in $\mathbb{SD}(t_{occ})$.*

Proof. If at time t_{out} a robot is already on *Ext'*, then the lemma clearly follows. Otherwise, the algorithm lets the robot r that at time t_{out} is on[5] Π'_o and closest to *Ext'* to move towards *Ext'*. Since r is the closest robot to *Ext'*, it follows that no robot can be between the position occupied by r at t_{out} and *Ext'*. Therefore, in a finite number of cycles r reaches *Ext'* moving on Π'_o, say at time t_{occ}. In the meanwhile, all the other robots with the possible exception of O (it can be still moving towards point p of Line 21, see previous lemma) can compute only *null movements*, hence the lemma follows. □

Lemma 4.5.5. *After time t_{occ}, from an odd agreement configuration, only an odd agreement configuration or a final configuration can be reached, and a final configuration will be reached in a finite number of moves, say at time t_f, while avoiding collisions. Furthermore, Π_m, M, and Π'_o do not change position between time t_{occ} and t_f, and $\mathbb{G}_t = \mathbb{G}_{t_f}$, for all $t \geq t_f$.*

Proof. From Lemmas 4.5.3 and 4.5.4, after an odd agreement configuration has been reached and until time t_{occ}, M, O and *O'* are always the median, the first and the second outermost robot, respectively. Therefore, the subsets of robots $\mathbb{K}$ and $\mathbb{K}'$ are still the same as at time t_{ag}.

According to the algorithm, after time t_{occ} the robots in the two sides defined by Π_m act concurrently aiming at reaching one of the points in *FP* in their side, by calling routine `Go_To_Points()` (Line 39). In particular, the robots in $\mathbb{K}$ aim at reaching the points in *FP* that are in *AsSide* or on Π_m below M, call this set of points $\mathbb{F}$; and the robots in $\mathbb{K}'$ aim at reaching the points in *FP* that are in *NotAsSide* or on Π_m above M, call this set of points $\mathbb{F}'$. By construction,

$$|\mathbb{K}| = |\mathbb{K}'| = |\mathbb{F}| = |\mathbb{F}'|, \tag{4.6}$$

and

$$\mathbb{F} \cap \mathbb{F}' = \emptyset, \ \mathbb{K} \cap \mathbb{K}' = \emptyset. \tag{4.7}$$

Note that every robot $r \neq O$ does not consider *Ext* as one position to be reached (Line 17): this position, in fact, is left to O. We distinguish two cases.

1. If *Ext* is the only point in *FP* left available at time t_{occ} (i.e., with no robot on it), then all the robots $r' \neq O$ are already on their final positions, and the only robot allowed to move is O. According to the algorithm, O moves towards *Ext*. Since by Lemma 4.5.3 no robot can be between the vertical line through O and the vertical line through *Ext*, any collision is avoided during this movement,

[5] By the way the pattern has been scaled, and by Lemma 4.5.3, at least *O'* is on Π'_o.

and it stays the first outermost robot during its move. Furthermore, since the scaling of the pattern is done according to the position of Π_m and of the topmost robot on Π'_o, and since these two robots are not allowed to move, the set *FP* does not change during the movement of *O*. Hence, from t_{occ} only odd agreement configurations are reached, and in a finite number of cycles, *O* reaches *Ext*. By Line 4 of the algorithm, the lemma follows.

2. If at time t_{occ} *Ext* is not the only point in *FP* with no robot on it, then each robot computes the subset of points in $FP \setminus \{Ext\}$ that are in the side where it lies, and that have no robots on it. Furthermore they compute the set of robots that are still not on one of the points in *FP* (they do not consider *O*). Let us consider what happens in *AsSide* (the argument is similar for the other side). Let us call *FreeP* the set of points $\mathbb{F} \setminus \{Ext\}$ such that no robot occupies a position in *FreeP*; and *FreeR* the set of robots in $\mathbb{K} \setminus \{O\}$ that are not on one of the points in *FreeP*. Moreover, let (r, p) be the pair that satisfies Equation 4.1 at time t_{occ}, with $r \in$*FreeR*, and $p \in$*FreeP*.

   By (4.6), $|FreeP| = |FreeR|$; furthermore all the robots not in $FreeR \cup \{O\}$ are already on a final position. According to the algorithm, after time t_{occ}, all the robots in *FreeR* execute `Go_To_Points(`$FreeR, FreeP, \Gamma_m, \Gamma_o$`)`, while all the others in $\mathbb{K}$ compute only *null movements*. Moreover, by Lemma 4.5.4, all the robots in *FreeR* are in $\mathbb{SD}(t_{occ})$. By (4.7), and since routine `Go_To_Points()` called by robots in $\mathbb{K}'$ guarantees that no robot in *NotAsSide* can ever enter in *AsSide* (i.e., they call `Go_To_Points(`$\cdot, \cdot, \Gamma_m, \Gamma'_o$`)`), by Theorems 4.3.1 and 4.3.2 applied on the set of robots $\mathbb{K} \setminus \{O\}$, *r* reaches *p* in a finite number of cycles, avoiding collisions, and never leaving the vertical stripe with borders Γ_m and Γ_o. Furthermore, until this happens, say at $t > t_{occ}$, all the other robots in *FreeR* do not move; hence, *M* and *O* are always the median and the first outermost robot, respectively. Furthermore, at *t* it still holds that $r' \in \mathbb{SD}(t)$, for all $r' \in$*FreeR*. Note that, by Lemma 4.5.4, after t_{occ} a robot is on *Ext'*, hence it never moves again (it is on one of the *Final_Positions*), Π'_o never changes position, and the scaling of the patterns is preserved.

   Thus, by iterating the above argument in both sides, in a finite number of cycles and avoiding collisions, every robot in $\mathbb{K} \cup \mathbb{K}' \setminus \{O\}$ will reach one of the points in $\mathbb{F} \cup \mathbb{F}' \setminus \{Ext\}$. Until this happens, *M*, *O* and Π'_o are always the median, the first outermost robot, and the second outermost vertical line, respectively. When *O* is the only robot not on one of the points in *FP*, previous case applies, and the lemma follows. □

From these lemmas, we conclude:

Theorem 4.5.1. *Algorithm 2 is a collision-free pattern formation algorithm, when n is odd and $\Upsilon'_o \not\equiv \Upsilon_m$ in $\mathbb{P}$.*

In the following sections we deal with Cases b. and c. The techniques to prove that also in these cases the problem is solvable are similar to the one used to solve the problem in Case a. Therefore, in the following we will give sketches of the algorithms, omitting the details of the proofs of correctness.

4.5.2 Case b.: $\Upsilon'_o \equiv \Upsilon_m \wedge \Upsilon'_o \not\equiv \Upsilon_o$

In this section we discuss the case when Υ'_o coincides with Υ_m in $\mathbb{P}$, but Υ'_o and Υ_o are distinct (refer to Figure 4.10.c and d). This implies that there are no vertices of $\mathbb{P}$ outside the area in $\mathbb{P}$ delimited by Υ_m and Υ_o. In this case, the algorithm is slightly different from Algorithm 2 (for a pictorial representation of the main steps of the algorithm, refer to Figure 4.15). In particular, Γ_m, *Outer_Robot*, *Outer_Robot'*, *Asym_Side*, Γ_o and Γ'_o are computed in the same way (Lines 1–7 of Algorithm 2). Then, Lines 8–10 are avoided: in fact, in this case we aim at reaching a scenario where Γ'_o coincides with Γ_m.

At this point, the orientation of the pattern is found, by executing `Orient_X()` (Line 11). That is, if in $\mathbb{P}$ we have Υ_m to the left of Υ_o (according to the local orientation of the x axis), then the x axis of *GCS* is oriented in such a way that Γ_m is to the left of Γ_o, and viceversa. In the following we assume that the first case applies (Υ_m to the left of Υ_o); hence, in *GCS*, the x axis is oriented from Γ_m towards Γ_o. The other case can be approached symmetrically.

Then the *Median_Robot* is computed (Line 12), by ordering the robots' positions left-right, top-down, according to the orientation of *GCS*. This robot is not allowed to move anymore (Lines 13). Then, the final positions are computed: the scaling cannot clearly be done with respect to the distance $\overline{\Upsilon_m \Upsilon'_o} = 0$, as done in Algorithm 2. In this case, the scaling is done according to the distance $\overline{\Upsilon_m \Upsilon_o}$, that is by construction not zero. In other words, p_m is mapped onto *Median_Robot*, and Υ_o onto Γ_o.

At this point, we distinguish two cases.

1. If $\Gamma'_o \equiv \Gamma_m$, then *Outer_Robot'* is on Γ_m. Let us call Λ the segment between *Median_Robot* and the topmost robot on Γ_m, and by $\mathcal{V}$ the vertical stripe with borders Γ_o and the portion of Γ_m below[6] *Median_Robot*. Note that, by the way the pattern has been scaled, by definition of p_m and since $\Upsilon'_o \equiv \Upsilon_m$, the number of robots on Λ equals the number of points in $\mathbb{P}$ on Υ_m above p_m.

[6] In the case when Γ_m is to the right of Γ_o, Λ is the segment between *Median_Robot* and the *bottommost* robot on Γ_m, and $\mathcal{V}$ the vertical stripe with borders Γ_o and the portion of Γ_m *above* *Median_Robot*.

Analogously, the number of robots in $\mathcal{V}$ equals the number of points in $\mathbb{P}$ that are in the area delimited by Υ_o and the part of Υ_m below p_m.

The robots in $\mathcal{V}$ aim at reaching the final positions in $\mathcal{V}$ by calling `Go_To_Points()` (similarly to what done in Algorithm 2, *Outer_Robot* is still the last robots to move and its final target is *External*). The robots on Λ aim at reaching the final positions on Λ as follows. All the movement happens vertically. Since there is agreement on direction and orientation of y, there is a total ordering among the robots and among the final positions on Λ. Hence, the $i-th$ robot in the ordering aim at reaching the $i-th$ final target on Λ. The movement happens so that, if a robot r sees a robot r' on its way, it stops before r', in order to avoid collisions.

2. Otherwise, $\Gamma'_o \not\equiv \Gamma_m$. In this case, if the horizontal distance between *Outer_Robot* and Γ_m equals the horizontal distance between *Outer_Robot'* and Γ_m, then *Outer_Robot* moves outwards (i.e., it move farther away from Γ_m). In this way, in subsequent computations, there is no danger to change the agreement on *Asym_Side* (similarly to what done in Algorithm 2 in Lines 18–23).

   Then, the robots in the vertical stripe with borders Γ'_o and Γ_m move towards Γ_m, and place themselves above[7] *Median_Robot*. In particular, the robots closest to Γ_m move first (any tie between two robot at the same horizontal distance from Γ_m is broken by moving the topmost robot). During this movements, the robots in *Asym_Side* do not move, hence the agreement on *Outer_Robot* is preserved. Once all these robots reached Γ_m, we have that $\Gamma'_o \equiv \Gamma_m$, and the previous case applies.

4.5.3 Case c.: $\Upsilon_o \equiv \Upsilon_m$

In this case, by Observation 4.5.1, $\mathbb{P}$ is a vertical line. Let d be the distance between the topmost and bottommost point in $\mathbb{P}$. Γ_m, *Outer_Robot*, *Outer_Robot'*, *Asym_Side*, Γ_o and Γ'_o are computed as in Lines 1–7 of Algorithm 2, with the exception that the test of Line 3 is eliminated. Moreover, also Lines 8–10 are avoided: in fact, in this case we aim at reaching a scenario where Γ'_o coincides with Γ_o. We distinguish three cases (for a pictorial representation of the main steps of the algorithm, refer to Figure 4.16).

1. If the robots are all on the same vertical line, then the pattern is scaled so that the distance between the topmost r and the bottommost robot r' on Γ_m

[7] In the case when Γ_m is to the right of Γ_o, these robots place themselves *below Median_Robot*.

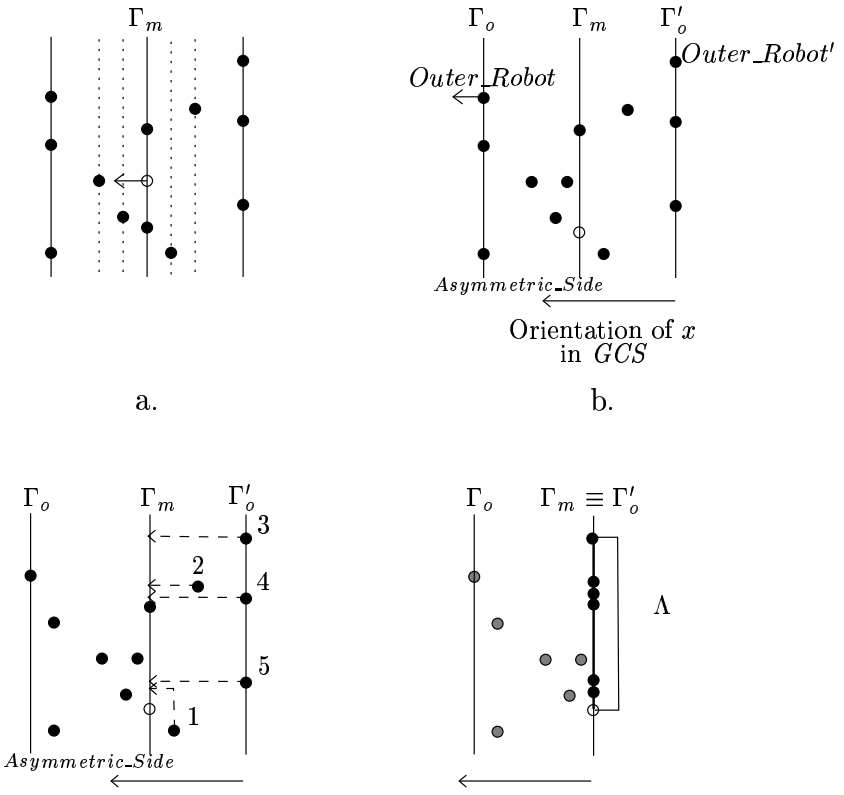


Figure 4.15: The algorithm for Case b., when $\Upsilon'_o \equiv \Upsilon_m \wedge \Upsilon'_o \equiv \Upsilon_o$. (a) The starting configuration is equidistant from Γ_m. Then, the robot that is median on Γ_m (the white circle) move towards its local right to break symmetry. (b) At this point an *Asym_Side* is identifiable, hence the two outermost robots. In this example, we are assuming that in *GCS* the x axis is oriented from Γ_m towards Γ_o. Since *Outer_Robot* and *Outer_Robot'* are at the same horizontal distance from Γ_m, according to the algorithm *Outer_Robot* move outwards. (c) All the robots in the region delimited by Γ_m and Γ'_o place themselves on Γ_m above *Median_Robot* (the white circle). The numbering on the robots denote the ordering they follow to move. That is, robot 1 moves first, then 2, and so on. (d) At this point, $\Gamma_m \equiv \Gamma'_o$. The black robots represent the robots on Λ, and the grey one the robots in $\mathcal{V}$.

equals d; hence, r and r' are already on their final positions, and they will never move again. At this point, since there is a total ordering among the robots (from the topmost to the bottommost) and among the final positions (from the topmost to the bottommost), the $i-th$ robot in the ordering aim at reaching the $i-th$ final positions. The movement happens so that, if a robot r sees a robot r' on its way, it stops before r', in order to avoid collisions.

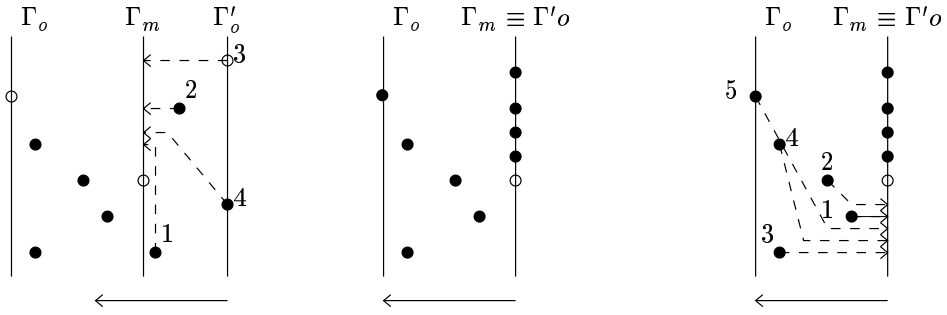
2. If $\Gamma'_o \equiv \Gamma_m \wedge \Gamma_m \not\equiv \Gamma_o$, then the x axis of *GCS* is oriented from[8] Γ_m towards Γ_o. At this point, a median robot r is found, by executing `Find_Median_Robot(`Γ_m`)`. All the robots in the region of the plane delimited by Γ_m and Γ_o aim at reaching Γ_m, and place themselves below r. In particular, the robots closest to Γ_m move first (any tie between two robot at the same horizontal distance from Γ_m is broken by moving the topmost robot). During this movements, the robots already on Γ_m do not move, hence the agreement on r is preserved. Once all these robots reached Γ_m, we have that $\Gamma_o \equiv \Gamma_m$, and the above case applies.

3. If $\Gamma'_o \not\equiv \Gamma_m$, then, by Observation 4.5.1, Γ_o, Γ'_o and Γ_m are three distinct vertical lines. If the horizontal distance between *Outer_Robot* and Γ_m equals the horizontal distance between *Outer_Robot'* and Γ_m, then *Outer_Robot* moves outwards (i.e., it move farther away from Γ_m). In this way, in subsequent computation, there is no danger to change the agreement on *Asym_Side*. Similarly to the previous case, the x axis of *GCS* is oriented from Γ'_o towards Γ_o, and a median robot r is found, by executing `Find_Median_Robot(`Γ_m`)`. Then, all the robots in the region of the plane delimited by Γ_m and Γ'_o aim at reaching Γ_m, and place themselves above r, similarly to what done in Case b.2. During these movements all the other robots (i.e., those on Γ_m and between Γ_m and Γ_o) do not move. When all these robots reach Γ_m, then $\Gamma'_o \equiv \Gamma_m$, and the previous case applies.

From Theorem 4.5.1, and from the above analysis of Cases a., b., and c., we can state that

Result 3. *With one axis direction and orientation agreement, an odd number of autonomous, anonymous, oblivious, mobile robots can form an arbitrary given pattern.*

[8]Since $\mathbb{P}$ is a vertical line, in contrast with Cases a. and b., in this case there is no need to orient the x axis of *GCS* according to the relative positions of Υ_m and Υ_o.



a. b. c.

Figure 4.16: The algorithm for Case c., when $\Upsilon_o \equiv \Upsilon_m$. In this case the orientation of the x axis is always from Γ'_o towards Γ_o. In (a) and (b), the robots in the region of the plane delimited by Γ_m and Γ'_o place themselves on Γ_m and above the median robot. The numbering has the same meaning as in Figure 4.15.c. (c) If $\Gamma_m \equiv \Gamma'_o$, according to the algorithm all the robots in the region of the plane delimited by Γ_m and Γ_o aim at reaching Γ_m, and place themselves below the median robot.

4.6 Partial Knowledge: Even Number of Robots

4.6.1 Characterization

We know from Section 4.4 that an arbitrary pattern can not be formed by an even number of robots (Corollary 4.4.1). In this section, we are interested in determining which class of patterns, if any, can be formed in this case. From now on, we will assume that the robots in the system have common knowledge on the direction and orientation of only the y axis[9], and that the number n of robots in the system is even.

We say that $\mathbb{P}$ is a *symmetric pattern* if it has at least one axis of symmetry Λ; that is, for each $p \in \mathbb{P}$ there exists exactly another point $p' \in \mathbb{P}$ such that p and p' are symmetric with respect to Λ (see Figure 4.17.b, c and d).

The proof of the unsolvability result of Theorem 4.4.1 is useful to better understand which kind of patterns can not be formed, hence which kind of pattern formation algorithm can not be designed. In fact, the ability to form a particular type of patterns would imply the ability to elect a robot in the system as the leader. More formally,

Theorem 4.6.1. *If an algorithm $\mathcal{A}$ lets the robots form*

[9]This implies that there is common knowledge also on the direction of the x axis, but not on its orientation.

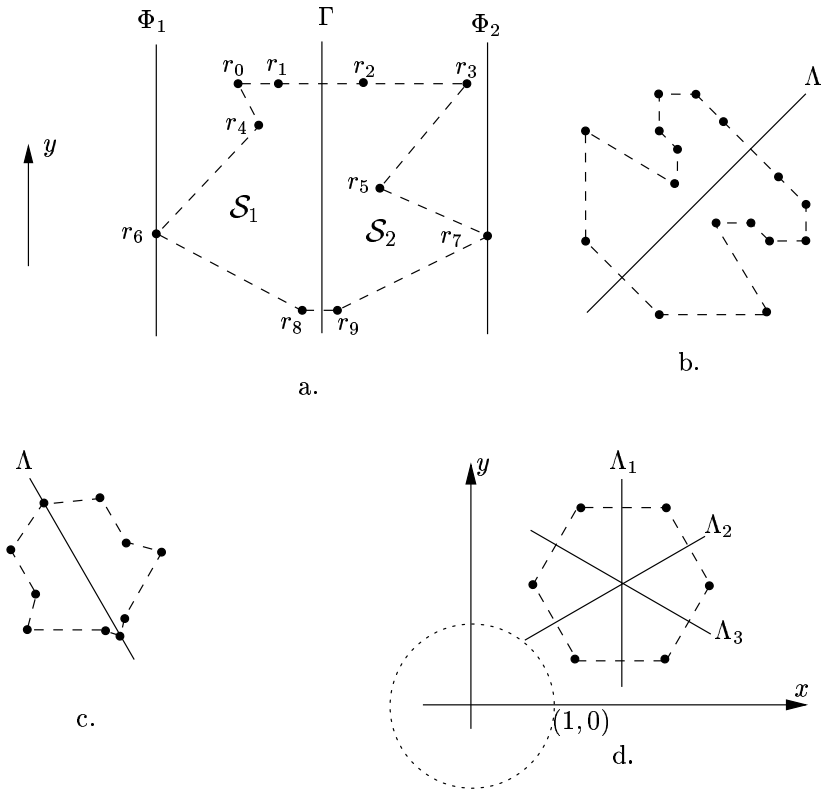


Figure 4.17: (a) An unachievable asymmetric pattern. In this example, the sorted sequence of pairs of robots from the proof of Theorem 4.6.1 is the following: (r_1, r_2), $(r_0, \emptyset)$, $(r_3, \emptyset)$, $(r_4, \emptyset)$, $(r_5, \emptyset)$, (r_6, r_7), (r_8, r_9). In this case r_0 would be elected as the leader. (b) An achievable pattern with one axis of symmetry not passing through any vertex. (c) An unachievable pattern. (d) An achievable pattern that has three axes of symmetry not passing through any vertex. Note that this pattern has also axes of symmetry passing through vertices. In this case, the routine `Choose(`$\mathbb{P}$`)` of Algorithm 3 would choose the axis Λ_2.

a. *an asymmetric pattern, or*

b. *a symmetric pattern that has all its axes of symmetry passing through some vertex,*

then $\mathcal{A}$ is a leader election algorithm.

Proof.

Part a. Let $\mathcal{A}$ be an algorithm that lets the robots form an asymmetric pattern $\mathbb{P}$ of n points. The task for the robots is to form $\mathbb{P}$ by executing $\mathcal{A}$. Let $\mathbb{G}$ be the final configuration after they execute the algorithm, starting from an arbitrary initial configuration. Moreover, let Φ_1 and Φ_2 be respectively the vertical axes passing through the outermost robots in $\mathbb{G}$, and let Γ be the vertical axis equidistant from Φ_1 and Φ_2 (e.g., see Figure 4.17.a). Γ splits the plane in two regions, $\mathcal{S}_1$ and $\mathcal{S}_2$. If some robots are on Γ, the highest on Γ can be elected as a leader, and the theorem follows. If no robot is on Γ, we can distinguish two cases:

1. $|\mathcal{S}_1| \neq |\mathcal{S}_2|$. In this case, the robots can agree on the most populated region as the positive side of x; hence, starting from any initial configuration, it is possible to elect a leader (e.g., the topmost rightmost one), and the theorem follows.
2. $|\mathcal{S}_1| = |\mathcal{S}_2|$. In this case, for each robot $r_i \in \mathcal{S}_1$, we build a pair (r_i, x), $x \in \mathcal{S}_2 \cup \emptyset$, as follows. Let $h(r)$ indicates the height of robot r. If there exists $r_j \in \mathcal{S}_2$ such that $h(r_i) = h(r_j)$ and $\overline{r_i\Gamma} = \overline{r_j\Gamma}$, then $x = r_j$; otherwise $x = \emptyset$. Analogously, we build pairs for each $r_j \in \mathcal{S}_2$. Given that (r_i, r_j) is defined if and only if (r_j, r_i) is defined, we can sort all the pairs in descending order, with respect to the height and the horizontal distance of the robots from Γ. Namely, (e.g., see in Figure 4.17.a):

$$\begin{array}{lll}
(r_i, \emptyset) > (r_j, \emptyset) & \Leftrightarrow & h(r_i) > h(r_j) \vee (h(r_i) = h(r_j) \wedge \overline{r_i\Gamma} < \overline{r_j\Gamma}) \\
(r_i, \emptyset) > (r_j, r_h) & \Leftrightarrow & h(r_i) > h(r_j) \vee (h(r_i) = h(r_j) \wedge \overline{r_i\Gamma} < \overline{r_j\Gamma}) \\
(r_i, r_j) > (r_h, \emptyset) & \Leftrightarrow & h(r_i) > h(r_h) \vee (h(r_i) = h(r_h) \wedge \overline{r_i\Gamma} < \overline{r_h\Gamma}) \\
(r_i, r_j) > (r_h, r_k) & \Leftrightarrow & h(r_i) > h(r_h) \vee (h(r_i) = h(r_h) \wedge \overline{r_i\Gamma} < \overline{r_h\Gamma})
\end{array}$$

We observe that the set of pairs obtained is independent from the orientation of the x axis in the local coordinate systems of the robots; moreover, since $\mathbb{G}$ is asymmetric w.r.t Γ by hypothesis, there must exist at least a pair with an $\emptyset$. It follows that we can elect as a leader the robot in the first pair that has $\emptyset$ as an element, and the theorem follows.

Part b. Let $\mathcal{A}$ be an algorithm that lets the robots form a symmetric pattern $\mathbb{P}$ that has all its axes of symmetry passing through some vertex in $\mathbb{P}$, starting from any arbitrary initial configuration. After the robots run $\mathcal{A}$, they are in a final configuration $\mathbb{G}$ whose positions correspond to the vertices of $\mathbb{P}$ (up to scaling and rotation); hence, $\mathbb{G}$ must be symmetric with all its axes of symmetry passing through some vertex (robot's position). We distinguish two cases.

1. $\mathbb{G}$ is not symmetric w.r.t. any line Γ' parallel to y. In this case, the same argument of Part a. can be used to conclude that a leader can be elected, and the theorem follows.
2. $\mathbb{G}$ is symmetric w.r.t. some Γ' parallel to y. Since by hypothesis Γ' must pass through a vertex, a leader can be elected (e.g., the topmost robot on Γ'), and the theorem follows. □

From Theorem 4.4.1 and Theorem 4.6.1, it follows that

Corollary 4.6.1. *There exists no pattern formation algorithm that lets the robots in the system form*

a. *an asymmetric pattern, or*

b. *a symmetric pattern that has all its axes of symmetry passing through some vertex.*

Let us call $\mathfrak{T}$ the class containing all the arbitrary patterns, and $\mathfrak{P} \subset \mathfrak{T}$ the class containing only patterns with at least one axis of symmetry not passing through any vertex (e.g., see Figures 4.17.b and 4.17.d); let us call *empty* such an axis. Corollary 4.6.1 states that if $\mathbb{P} \in \mathfrak{T}/\mathfrak{P}$, then $\mathbb{P}$ can not be in general formed; hence, according to Part b. of the previous corollary, the only patterns that might be formed are symmetric ones with at least one *empty axis*. In the following, we prove that all these patterns can actually be formed. In particular, we present an algorithm that lets the robots form exactly these kind of patterns, if local rotation of the pattern is allowed.

4.6.2 The Algorithm

In this section we present an algorithm that lets the robots form symmetric patterns with at least one *empty axis* (Algorithm 3). The idea behind the algorithm is as follows. First, the robots compute locally an *empty axis* of the input pattern $\mathbb{P}$, say Λ, and then rotate $\mathbb{P}$ so that Λ is parallel to the common understanding of the

orientation of y. Second, the robots elect the two topmost outermost[10] robots in the observed robots' positions, $Outer_1$ and $Outer_2$. If this is not possible (i.e., all the robots are on the same vertical axis), the second topmost robot on this axis moves to its right, so that $Outer_1$ and $Outer_2$ can be correctly computed. Then, $Outer_1$ and $Outer_2$ move until they are at the same height: these two robots will never move again. When this happens, the vertical axis Γ is computed: it is the median between the vertical axis passing through the positions of $Outer_1$ and $Outer_2$; Γ splits the plane in two halves, called *Left* and *Right*.

At this point, it is possible to compute the set of final positions of the robots: $\mathbb{P}$ is scaled with respect to the distance between $Outer_1$ and $Outer_2$, and is translated with respect to the positions of these two robots. We note that, by definition of $\mathbb{P}$, the number of final positions in *Left* and in *Right* is $n/2$. The robots' positions in each half can be sorted using the ordering defined in the proof of Theorem 4.6.1. Thus, the first $n/2$ robot in *Left* are directed towards final positions in *Left*, and the first $n/2$ robots in *Right* towards final positions in *Right*. If in one half there are more robots than final destination, the *extra* robots are directed towards Γ. Once there are no *extra* robots neither in *Left* nor in *Right*, the robots on Γ are directed, from the topmost to the bottommost, towards the final destinations that do not have any robots on them yet.

In Algorithm 3, routines `Choose()`, `Rotate()`, `Same_Vertical_Axis()`, `Pattern_Length()`, `Outermost()`, `Median_Axis()`, `Find_Final_Positions()`, `Sides()`, `My_Side()`, `Go_To_Points()`, and `Choose_On_`Γ_m`()` are called, whose behavior is described in the following.

The routine `Choose(`$\mathbb{P}$`)` locally chooses an *empty axis* of symmetry in the input pattern $\mathbb{P}$; since this is a local operation, and $\mathbb{P}$ is the same for all the robots, every robot can be made to choose the same axis of symmetry [11].

`Rotate(`$\mathbb{P}, S$`)` locally rotates $\mathbb{P}$ in such a way that the axis of symmetry Λ chosen with `Choose(`$\mathbb{P}$`)` becomes parallel to the y axis. The rotation is (locally) performed clockwise, and its result is stored in $\mathbb{P}_R$.

`Pattern_Length(`$\mathbb{P}_R$`)` returns the horizontal length of $\mathbb{P}_R$ according to the local unit distance, measured as the horizontal distance between the two outermost vertical axes tangent to $\mathbb{P}_R$.

As already stated previously, the algorithm locates the two outermost and topmost robots, so that the input pattern can be translated and scaled with respect

[10] That is, among the outermost robots, the two topmost ones.

[11] For instance, starting from the point $(1, 0)$ on the unit circle centered in the origin of the local coordinate system, they can choose the first *empty axis* that is hit moving counterclockwise (according to the local orientation of the x axis), after having translated all the *empty axes* in such a way that they pass through the origin. In the example depicted in Figure 4.17.d, the axis Λ_2 would be chosen.

Algorithm 3 Arbitrary Pattern Formation With Partial Agreement, n even

Input: An arbitrary pattern $\mathbb{P}$ described as a sequence of points $p_1, \ldots, p_n$, given in lexicographic order. $\mathbb{P}$ is symmetric and has at least one *empty axis*. The direction and orientation of the y axis is common knowledge. Me is the current position of the executing robot.

$\Lambda :=$ `Choose(`$\mathbb{P}$`)`;
$\mathbb{P}_R :=$ `Rotate(`$\mathbb{P}, \Lambda$`)`;
$P\_Length :=$ `Pattern_Length(`$\mathbb{P}_R$`)`;
$\Xi :=$ Vertical Axis With More Robots On It;
If ($|\Xi| = n$ **Or** $|\Xi| = n-1$) **Then** `Same_Vertical_Axis(`Ξ`)`.
$(Outer_1, Outer_2) :=$ `Outer_Most()`;
$\Gamma_1 :=$ Vertical Axis Through $Outer_1$;
$\Gamma_2 :=$ Vertical Axis Through $Outer_2$;
If $Outer_1.y \neq Outer_2.y$ **Then** `Fix_Outermosts(`$Outer_1$`,`$Outer_2$`)`.
If I Am $Outer_1$ **Or** $Outer_2$ **Then** `do_nothing()`;
$\Gamma_m :=$ `Median_Axis(`$Outer_1, Outer_2$`)`;
$Final\_Positions :=$ `Find_Final_Positions(`$\Gamma_m, \mathbb{P}_R, \Lambda,$
$P\_Length, Outer_1, Outer_2$`)`;
If I Am On One Of The $Final\_Positions$ **Then** `do_nothing()`;
$(Left, Right) :=$ `Sides(`Γ_m`)`;
If I Am In $Left$ **Or** In $Right$ **Then**
  $\mathcal{M}ySide :=$ `My_Side(`Γ_m`)`;
  $\Gamma :=$ Axis Among Γ_1 and Γ_2 in $\mathcal{M}ySide$;
  $Free\_Points :=$ {*Final_Positions In* $\mathcal{M}ySide$ *With No Robots On Them* };
  $Free\_Robots :=$ {*Robots' positions in* $\mathcal{M}ySide$ *not on Final_Positions*};
  If $Free\_Points \neq \emptyset$ **Then**
    `Go_To_Points(`$Free\_Robots, Free\_Points, \Gamma_m, \Gamma$`)`;
  Else %I am in $Free\_Robots$ but there are no $Free\_Points$ in $\mathcal{M}ySide$%
    `Choose_On_`Γ`(`$Free\_Robots, \Gamma_m$`)`;
If I Am On Γ_m **Then**
  If There Are Robots In $(Left \cup Right)$ Not On $Final\_Positions$ **Then**
    `do_nothing()`.
  $Robots\_On\_\Gamma_m :=$ {*Positions Of Robots On* Γ_m};
  $Available\_Points :=$ {*Points In Final_Positions With No Robot On Them*};
  `Go_To_Points(`$Robots\_On\_\Gamma_m, Available\_Points, \Gamma_1, \Gamma_2$`)`.

to them. Unfortunately, if all the robots are on the same vertical axis Ξ, these two robots can not be located. In Line 5, the algorithm handles this case. In particular, routine `Same_Vertical_Axis(`Ξ`)` lets one of the robots on Ξ move to its right, so that there are exactly two outermost topmost robots. The routine is the same described in Section 4.5.1.

`Outermost()` returns the current topmost outermost robots. In particular, let Φ_1 and Φ_2 be the two outermost vertical axes passing through some robot in a given configuration. This routine returns the topmost robot on Φ_1, $Outer_1$, and the topmost robot on Φ_2, $Outer_2$ (in the following, we will refer to these two robots as the two *topmost outermost robots*). Clearly, since there is no agreement on the orientation of the x axis, it is possible that the robots do not agree on which is $Outer_1$ and which is $Outer_2$.

Then, the algorithm calls the routine `Fix_Outermosts(`$Outer_1, Outer_2$`)`, that moves $Outer_1$ and $Outer_2$ until they reach the same height. Namely,

`Fix_Outermosts(`$Outer_1, Outer_2$`)`
  If $Outer_1.y > Outer_2.y$ **Then**
    If I Am Not $Outer_2$ **Then**
3:    `do_nothing();`
    Else
      $p := (Outer_2.x, Outer_1.y)$;
6:    `Move(`p`);`
  If $Outer_1.y < Outer_2.y$ **Then**
    If I Am Not $Outer_1$ **Then**
9:    `do_nothing();`
    Else
      $p := (Outer_1.x, Outer_2.y)$;
12:   `Move(`p`);`

The function `Median_Axis(`$Outer_1, Outer_2$`)` returns the vertical axis Γ_m, which is the median axis between the vertical axes passing through $Outer_1$ and $Outer_2$.

`Find_Final_Positions` $(\Gamma_m, \mathbb{P}_R, \Lambda, P\_Length, Outer_1, Outer2)$ performs the scaling and the translation of $\mathbb{P}_R$ with respect to the positions of $Outer_1$ and $Outer_2$. The scaling is computed as follows: Γ_m is viewed as the *empty axis* Λ computed in Line 1; furthermore, by definition of $\mathbb{P}$ and because of the way it has been rotated in Line 2, there are exactly two outermost topmost points in $\mathbb{P}_R$ that are symmetric with respect to Λ: these two points are viewed as $Outer_1$ and $Outer_2$ (which, after the execution of `Fix_Outermosts()`, are symmetric with respect to Γ_m). The common scaling of the input pattern is defined by identifying P_Length with the horizontal distance between $Outer_1$ and $Outer_2$ (that are at the same height and

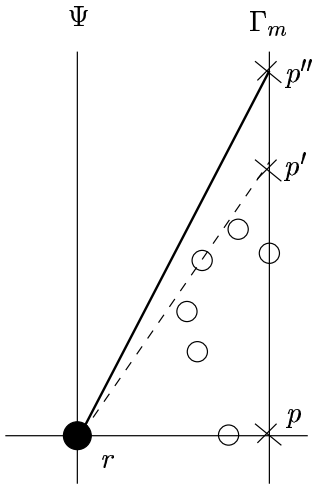


Figure 4.18: Routine `Choose_On_`Γ_m`()` determines the destination point for r on Γ_m. The white circles represent robots' positions. The thick line is the path followed by r to reach Γ_m.

already in their final positions).

The routine `Sides(`Γ_m`)` returns two sets, each containing the positions of robots currently lying *inside*[12] the two halves in which the plane is split by Γ_m: in particular, it returns respectively the robots' positions on the *Left* and on the *Right* of Γ_m, according to the local orientation of the calling robot's x axis. `My_Side(`Γ_m`)` returns the half of the plane where the calling robot currently lies.

Routine `Go_To_Points()` is as described in Figure 4.3. The only difference is in the way routine `Minimum()` breakes ties: if the calling robot is on Γ_m, any tie is broken by choosing the pair (r, p) such that r is the topmost robot on Γ_m; otherwise, the selected pair is the one such that r is the topmost closest robot to Γ_m (in the side where the calling robot is) and, in case of another tie, p is the topmost closest point to Γ_m.

If in $\mathcal{M}ySide$ there are more robots than final positions, the *extra* robots are directed towards Γ_m, in Line 24, by routine `Choose_On_`Γ_m`()`, that ensures that the movements towards Γ_m are done without collisions (refer to Figure 4.18). In particular,

`Choose_On_`Γ_m`(`$Free\_Robots, \Gamma_m$`)`

If I Am The Topmost And Closest To Γ_m in $Free\_Robots$ **Then**
$p :=$ Intersection Between Γ_m And Horizontal Line Passing Through Me;
If No Robot Is On The Line Passing Through Me And p **Then** `Move(`p`)`.

[12] i.e., the robots on Γ_m are not considered.

$\Psi :=$ Vertical Line Passing Through Me;
$\mathcal{V} :=$ Vertical Stripe With Borders Ψ and Γ_m;
$\mathcal{H} :=$ Half Plane Above Line Through r and p;
$\mathcal{R} := (\mathcal{V} \cap \mathcal{H}) \setminus \Psi$;
$Avoid := \{Robots'\ Positions\ In\ \mathcal{R}\}$;
$Intersections := \emptyset$;
For All $p' \in Avoid$ **Do**
$\Psi' :=$ Line Passing Through Me And p';
$Intersections := Intersections \cup \{\ Intersection\ Between\ \Gamma_m\ And\ \Psi'\}$;
End For
$p' :=$ Topmost Point In $Intersections$;
$p^* :=$ Point On Γ_m Above p' At Distance $\eta > 0$.
`Move(`p^*`).`
Else
`do_nothing().`

In other words, $\mathcal{R}$ is the region of the plane above $[rp]$, and delimited by Ψ and Γ_m; all the points on Ψ are not included in $\mathcal{R}$. `Choose_On_`Γ_m`()` allows the calling robot to choose a path the goes above all the robots that are inside $\mathcal{R}$, mantaining the invariant to remain the (closest to Γ_m) topmost robot in $Free\_Robots$. η is an arbitrary positive constant.

When all the robots in *Left* and *Right* are on $Final\_Positions$, all the robots on Γ_m, if any, are directed sequentially towards the available (i.e., with no robots on them) final positions. In particular, routine `Go_To_Points()` selects an unique robot from the set $Robots\_On\_\Gamma_m$, and lets it move towards one of the points in $Final\_Positions$ that are still available.

4.6.3 Correctness of Algorithm 3

To prove the correctness of Algorithm 3, we follow a strategy similar to the one used in the case with an odd number of robots. Let $\mathbb{G}_t$, $\Pi(\mathbb{G}_t)$ and $\Pi'(\mathbb{G}_t)$ be as defined in Section 4.5.1, and $\Pi_m(\mathbb{G}_t)$ be the vertical line that is median between $\Pi(\mathbb{G}_t)$ and $\Pi'(\mathbb{G}_t)$. Moreover, let $\Xi(\mathbb{G}_t)$ be the vertical line with the most number of robots on it in $(\mathbb{G}_t)$. We call the *outermost robots* in $\mathbb{G}_t$, the topmost robot on $\Pi(\mathbb{G}_t)$ and the topmost robot on $\Pi'(\mathbb{G}_t)$ (in contrast with the approach used in Section 4.5, in this case we do not distinguish between the first and the second outermost robot). Analogously, we call the *outermost vertical axes*, the vertical axes through the two outermost robots.

Lemma 4.6.1. *By executing Algorithm 3, if the robots at the beginning are on the*

same vertical line, i.e. $\Pi(\mathbb{G}_0) \equiv \Pi'(\mathbb{G}_0)$*, then in a finite number of cycles, say at time* t_{dis}*, the robots are in a configuration such that* $\Pi(\mathbb{G}_{t_{dis}}) \not\equiv \Pi'(\mathbb{G}_{t_{dis}})$*. Furthermore, until time* t_{dis} *any collision is avoided, and at* t_{dis} *all the robot are in* $\mathbb{SD}(t_{dis})$*.*

Proof. If $\Pi(\mathbb{G}_0) \equiv \Pi'(\mathbb{G}_0)$, then $|\Xi(\mathbb{G}_0)| = n$; let $\Xi = \Xi(\mathbb{G}_0)$. In this configuration the robots can only call routine `Same_Vertical_Axis(`Ξ`)` (Line 5). According to this routine, the second topmost robot r on $\Xi(\mathbb{G}_0)$ is the only robot allowed to move: it moves to a point p at horizontal distance d from Ξ, with d the distance between the topmost and the bottomost robot on Ξ. Let t_1 be the first time when r leaves Ξ. Since $n \geq 3$, $\Xi(\mathbb{G}_0) = \Xi(\mathbb{G}_{t_1}) = \Xi$. If r is observed when it is not on Ξ (i.e., when $|\Xi| = n - 1$), the second case of the routine is called: all the robots are forced to not move as long as r is not at horizontal distance d from Ξ, that is on p. Therefore, until this happens, all the other robots compute only *null movements*, and any collision is avoided. Let $t_{dis} \geq t_1$ be the first time when r is at horizontal distance d from Ξ (t_{dis} is finite, by Assumptions A1 and A2 of the model). Since at this time all the robots are in $\mathbb{SD}(t_{dis})$, the lemma follows. □

Let us denote by O and O' the two outermost robot's positions in $\mathbb{G}_{t_{dis}}$. Moreover, let $\Pi = \Pi(\mathbb{G}_{t_{dis}})$, $\Pi' = \Pi'(\mathbb{G}_{t_{dis}})$, and Π_m be the vertical axis that is median between Π and Π' (i.e., Π_m is the vertical axis that has the same horizontal distance from Π and Π'). In the following lemma, we show that in a finite number of cycles the two outermost robots place themselves at the same height.

Lemma 4.6.2. *In a finite number of cycles, say at time* $t_h \geq t_{dis}$*, the robots are in a configuration where* O *and* O' *are at the same height, by executing Algorithm 3. Furthermore, until this time any collision is avoided, and at* t_h *all the robot are in* $\mathbb{SD}(t_h)$*.*

Proof. We distinguish two cases.

1. If the robots at the beginning are not in a distinct configuration[13], then by previous lemma at time t_{dis} such a configuration is reached. Hence, $\Pi(\mathbb{G}_{t_{dis}}) \not\equiv \Pi'(\mathbb{G}_{t_{dis}})$.

   At this point, according to the algorithm, the two outermost robots in $\mathbb{G}(t_{dis})$ are localized, O and O' (called $Outer_1$ and $Outer_2$ in Line 6), and as long as they are not at the same height[14] all the other robots do not move (Lines 3 and 9 in routine `Fix_Outermosts()`). Routine `Fix_Outermosts()` lets the bottomost among O and O' to move vertically upwards. Since by definition no

[13] A *distinct* configuration has been defined in Section 4.5.1.

[14] We recall that we can talk about *same heights* because all the robots agree on the direction of y, hence they can commonly agree on this.

robot is above the two outermost robots on the vertical lines where they are, during this movement any collision is avoided. By Assumptions A1 and A2, in a finite number of cycles, say at time t_h, O and O' are at the same height. Furthermore, between time t_{dis} and t_h all the other robots can only compute *null movements*, and the lemma follows.

2. If the robots at the beginning are in a distinct configuration, then $t_{dis} = 0$, and the lemma follows as in the previous case. □

Let *FP* be the sets of point returned by routine `Find_Final_Positions(`Π_m,$\mathbb{P}_R$, Λ,*P_Length*,*O*,*O'*`)` at time t_h (i.e., when O is at the same height as O'), with Λ, $\mathbb{P}_R$, and *P_Length* as defined in Lines 1–3 of Algorithm 3.

Note 4.6.1. By the way the set *FP* is computed, at time t_h O and O' occupy two positions that are in *FP*. Furthermore, all the points in *FP* can neither be above O on Π, nor above O' on Π', nor outside the vertical stripe with borders Π and Π'. ⋆

In the following we will denote by *sides* the halves in which the plane is divided by Π_m. In particular, one side is the vertical stripe with borders Π and Π_m, say *Right* (Π_m does not belong to *Right*); and the other is the vertical stripe with borders Π' and Π_m, say *Left* (Π'_m does not belong to *Left*).

Note 4.6.2. Since $\mathbb{P}$ is symmetric w.r.t. Λ, by the way $\mathbb{P}$ has been rotated in Line 2, and by definition of set *FP*, no point in *FP* can be on Π_m. Furthermore, the number of points in *FP* that are in *Left* equals the number of points in *FP* that are in *Right*. ⋆

Let $FreeP(R,t)$ (resp. $FreeP(L,t)$) be the subset of points in *FP* that are in *Right* (resp. *Left*) and with no robots on them, at time t; and $FreeR(R,t)$ (resp. $FreeR(L,t)$) be the set of robots in *Right* (resp. *Left*) that are not on a point in $FreeP(R,t)$ (resp. $FreeP(L,t)$) at time t.

We shall call an *even agreement configuration* a distinct configuration of the robots in which (i) each robot is either on one of the points in *FP* or on Π_m, and (ii) there is at most one robot on each of the points in *FP*. Moreover, we define the *cardinality* of an even agreement configuration as the number of robots on Π_m.

Observation 4.6.1. Given a robot r, the only two routines in Algorithm 3 that compute a destination point for r are `Go_To_Points(`$Free\_Robots, Free\_Points, \cdot, \cdot$`)`, and `Choose_On_`$\Gamma_m$`(`$Free\_Robots, \Gamma_m$`)`. `Go_To_Points()` either computes a *null movement* for r, or let r move towards one of the points in *Free_Points*. `Choose_On_`Γ_m`()` either computes a *null movement* for r, or let it move on a point on Γ_m. ◇

In the following lemma, we show that Algorithm 3 lets the robots reach an even agreement configuration in a finite number of cycles, while avoiding collisions between robots.

Lemma 4.6.3. *If the robots at the beginning are not in a final configuration nor in an even agreement configuration, they will reach an even agreement configuration in a finite number of cycles, say at time t_{ag}, while avoiding collisions. Furthermore, at t_{ag} all the robots are in $\mathbb{SD}(t_{ag})$, O and O' are still the two outermost robots of $\mathbb{G}_{t_{ag}}$, and between t_h and t_{ag} O and O' never move.*

Proof. According to the previous lemma, O and O' are at the same height at time t_h. At this point, according to the algorithm, the robots aim at reaching the positions in *FP*. We note that, since $\mathbb{P}$ is symmetric with respect to Λ (computed in Line 1), and $\mathbb{P}_R$ is the result of the rotation of $\mathbb{P}$ in Line 2, also in $\mathbb{P}_R$ there are exactly two topmost outermost points at the same height.

The robots in the two *sides* act concurrently. As we will show, this is done without interference between the *Left* and *Right* side. First observe that (I) *a robot r that is in Left (resp. Right) does not go into Right (resp. Left) if there are robots in Right (Left) not on a point of FP*, that is if $|FreeR(R,t)| \neq 0$ (resp. $|FreeR(L,t)| \neq 0$). In fact, by Note 4.6.2 and Observation 4.6.1, the only way for $r \in Left$ (resp. $r \in Right$) to change side, is to execute routine `Choose_On_`Γ_m`()`, that allows r to reach and stop on a point on Π_m. Line 27, however, does not allow r (and in general any robot on Π_m) to leave Π_m and go into *Right* (resp. *Left*), as long as there are robots in *Right* (resp. *Left*) not on one of the points in *FP*. Moreover, by Line 24 and by Observation 4.6.1, we have that (II) *a robot $r \in Left$ (resp. $r \in Right$) is allowed to go on Π_m only if $FreeP(L,t) = \emptyset$ (resp. i.e., $FreeP(R,t) = \emptyset$).*

Let us consider the *Left* side at time t_h, and let $FreeR(L,t_h) \neq \emptyset$ and $FreeP(L,t_h) \neq \emptyset$ (the argument is symmetric for the *Right* side). By Lemma 4.6.2, at time t_h all the robots are in $\mathbb{SD}(t_h)$. According to the algorithm, all the robots in $FreeR(L,t_h)$ execute `Go_To_Points()`, while all the others that are in *Left* can compute only *null movements* (by definition, they are already on one of the points in *FP*). Let (r,p) be the pair that satisfies Equation (4.1) at time t_h, among all the robots in $FreeR(L,t_h)$ and all the points in $FreeP(L,t_h)$. By observation (I) above, no robot in *Right* or on Π_m can enter *Left*. Hence, according to Theorems 4.3.1 and 4.3.2, r will reach p in a finite number of cycles. By Note 4.6.1, during this movement O and O' do not change position, hence they are always the two outermost robots. Therefore, by iterating this argument, we can conclude that, (A) if the numbers of robots in *Left* at time t_h is smaller or equal than the number of points in *FP* that

are in *Left* at time t_h (i.e., $0 < |FreeR(L, t_h) \leq |FreeP(L, t_h)|$[15]), then in a finite number of cycles and avoiding collisions, $|FreeR(L, \cdot)| = 0$; let $t_1 \geq t_h$ be the first time such that $|FreeR(L, t_1)| = 0$. Furthermore, by Note 4.6.1, between time t_h and t_1, O and O' do not move, and they are the two outermost robots in $\mathbb{G}_{t_1}$. (B) if the numbers of robots in *Left* at time t_h is greater than the number of points in FP that are in *Left* at time t_h (i.e., $|FreeR(L, t_h)| > |FreeP(L, t_h)| > 0$), then in a finite number of cycles and avoiding collisions, $|FreeP(L, \cdot)| = 0$; let t_2 be the first time such that $|FreeP(L, t_2)| = 0$. Furthermore, by Note 4.6.1, between time t_h and t_2, O and O' do not move, and they are the two outermost robots in $\mathbb{G}_{t_2}$. We distinguish two cases.

1. Case (B) above applies: $|FreeR(L, t_h)| > |FreeP(L, t_h)| > 0$, and $|FreeP(L, t_2)| = 0$, with $t_2 \geq t_h$. Then, since $|FP| = n$ and by Note 4.6.2, the number of robots in *Right* at time t_h is smaller or equal than the number of points in FP in *Right* at time t_h; that is $0 < |FreeR(R, t_h) \leq |FreeP(R, t_h)|$. Thus, by Case (A) above applied to *Right*, there exists a finite time $t_1 \geq t_h$ such that $|FreeR(R, t_1)| = 0$.

   Let us consider what happens at time t_2 in *Left*. The *extra* robots in $FreeR(L, t_2)$ that are still in *Left* are sequentially directed towards Π_m (Line 24). In particular, the topmost robot in $FreeR(L, t_2)$ is allowed to move towards Π_m (ties are broken by choosing the closest robot to Π_m). Its destination point on Π_m is chosen by routine `Choose_On_`Γ_m`()`, using a strategy that avoids collisions (see Figure 4.18). In particular, if there is no robot between r and the intersection p between Π_m and the horizontal line passing through r, r moves towards p; otherwise, `Choose_On_`Γ_m`()` allows r to move only upwards. In both cases, during this movement, r remains the (closest to Π_m) topmost robot in $FreeR(L, t_2)$, hence it is the only one allowed to move until it reaches Π_m, avoiding collisions with robots in *Left*. Furthemore, the destination point on Π_m computed by r can not be destination points of any of the robots in *Right* (i.e., it can not collide on Π_m with a robot coming from *Right*). In fact, the only way for r to reach Π_m is to execute `Choose_On_`Γ_m`()` in Line 24. Since $0 < |FreeR(R, t_h) \leq |FreeP(R, t_h)|$, no robots in *Right* can move towards Π_m (i.e., the condition in Line 21 or in Line 26 is satisfied by a robot in *Right*), and r can not collide with robots coming from *Right*. Therefore, r reaches Π_m in a finite number of cycles, while avoiding collisions. By iterating this argument, all the *extra* robots in *Left* at time t_2 reach Π_m in a finite number of cycles while avoiding collisions, say at time t_3. Furthermore, by Note 4.6.1,

[15]This equation follows from the fact that no robots different from O and O' moved between time 0 and time t_h (by Lemmas 4.6.1 and 4.6.2), and since in $\mathbb{P}$ all the vertices are distinct.

during these movements O and O' do not move, hence they are always the two outermost robots. In conclusion, within a finite number of cycles, at time $t_{ag} = \max(t_1, t_3)$, either a robot is on one of the points in FP, or on Π_m, and the lemma follows.

2. Case (A) above applies: $0 < |FreeR(L, t_h) \leq |FreeP(L, t_h)|$, and $|FreeR(L, t_1)| = 0$. Then, since $|FP| = n$ and by Note 4.6.2, $|FreeR(R, t_h)| > |FreeP(R, t_h)| > 0$. Thus, by Case (B) above applied to *Right*, there exists a finite time $t_2 \geq t_h$ such that $|FreeP(R, t_2)| = 0$. The lemma follows by applying to *Right* the argument used in previous point 1. □

Moreover, we can prove the following

Lemma 4.6.4. *Given an even agreement configuration of cardinality greater or equal to one, within finite time, say at time $t_{ag'}$, another even agreement configuration of smaller cardinality will be reached, avoiding any collisions. Furthermore, in the new even agreement configuration O and O' are still the two outermost robots, and bewteen t_h and $t_{ag'}$ O and O' never move.*

Proof. By previous lemma, at time t_{ag} the robots reach their first agreement configuration. By definition, in any even agreement configuration, all the robots not on Π_m are on one of the points in FP, and all the robots not on one of the points in FP are on Π_m. Those on Π_m execute `Go_To_Points(`$FreeR, FreeP, \Pi, \Pi'$`)`, that takes in input the set $FreeR$ containing the positions of the robots on Π_m, and the set $FreeP$ of points in FP with no robots on them. All the other robots (not on Π_m) compute only *null movements* (they are on points in FP). Let (r, p) be the pair that satisfies Equation (4.1) at time t_{ag}, with respect to $FreeR$ and $FreeP$. By Note 4.6.2, no point in $FreeP$ is on Π_m, and r is the only robot allowed to move by `Go_To_Points()`; hence, as long as r stays on Π_m, every robot $r' \neq r$ computes *null movements* (either because it is on a point in FP; or because it is on Π_m, but it is not chosen by `Go_To_Points()` as the robot allowed to move). By A1, r will eventually leave Π_m; when this happens, say at time $t' \geq t_{ag}$, all $r' \neq r$ see one robot (i.e., r) in $Left \cup Right$, hence they can compute only *null movements* (by Line 14, if they are not on Π_m; by Line 26 if they are on Π_m and observe r when it is not on Π_m; or by Line 30, if they are on Π_m and observed r when it still was on Π_m). Therefore, by Theorems 4.3.1 and 4.3.2, r will reach p in a finite number of cycles, and avoiding collisions; hence a new even agreement configuration is reached, with one robot less on Π_m. Furthermore, O and O' never moves, and the lemma follows. □

Therefore, by Lemmas 4.6.2–4.6.4, we can state the following

Theorem 4.6.2. *Algorithm 3 is a collision-free pattern formation algorithm for* $\mathbb{P} \in \mathfrak{P}$.

Result 4. *An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of y axis, can form a pattern* $\mathbb{P}$ *if and only if* $\mathbb{P} \in \mathfrak{P}$.

4.6.4 Remarks on Rotation

In Section 4.6.1 we provided a characterization of the class of patterns which can be formed by an even number of anonymous robots, provided they have agreement on the direction and orientation of the y axis. This characterization assumes that the robots can locally *rotate* the input pattern. Should the robots be uncapable to perform this operation, the characterization is different; not surprisingly, the class of achievable patterns is smaller. Let $\mathfrak{P}' \subset \mathfrak{P}$ be the class of symmetric patterns with at least one *empty axis*, and with no *empty axis* parallel to y.

Theorem 4.6.3. *There exists no pattern formation algorithm that does not allow local rotation of the input pattern, and that lets the robots form a symmetric pattern* $\mathbb{P} \in \mathfrak{P}'$.

Proof. By contraddiction, let $\mathcal{A}$ be an algorithm that, starting from an arbitrary initial configuration, lets the robots form a pattern $\mathbb{P} \in \mathfrak{P}'$. Let $\mathbb{G}_f$ be the final configuration of the robots for $\mathbb{P}$ after they execute $\mathcal{A}$. Since no local rotation of the pattern is allowed, $\mathbb{G}_f$ is symmetric with no *empty axis* parallel to y. Let $\Pi = \Pi(\mathbb{G}_f)$, $\Pi' = \Pi'(\mathbb{G}_f)$, and $\Pi_m = \Pi_m(\mathbb{G}_f)$. If $\Pi_m \equiv \Pi \equiv \Pi'$, then all the robot are on Π_m, hence a leader can be elected (e.g., the topmost robot on Π_m), thus contraddicting Theorem 4.4.1. Otherwise, if $\mathbb{G}_f$ is symmetric with respect to Π_m, then there must be at least one robot on Π_m (by hypothesis, $\mathbb{G}_f$ has no *empty axis* parallel to y); hence, the topmost of these robots can be elected as leader, contraddicting Theorem 4.4.1. Therefore, $\mathbb{G}_f$ is not symmetric with respect to Π_m: also in this case a leader can be elected (e.g., following an approach similar to the one used in the proof of Theorem 4.6.1.a), thus contraddicting again Theorem 4.4.1. □

As a concluding remark, we note that skipping the `Rotate(`$\mathbb{P}$`)` at Line 2 in Algorithm 3, we have a pattern formation algorithm that does not make use of local rotation and allows the formation of a symmetric pattern that has at least one *empty axis* that is parallel to y. Hence, we can state the following

Result 5. *An even number of autonomous, anonymous, oblivious, mobile robots that agree on the direction and orientation of y axis, can form a pattern* $\mathbb{P}$ *if and only if* $\mathbb{P} \in \mathfrak{P}/\mathfrak{P}'$, *when no local rotation of* $\mathbb{P}$ *is allowed.*

4.7 Conclusions

We have shown that an arbitrary pattern cannot always be formed; it is interesting to understand in detail which patterns or classes of patterns can be formed under which conditions, because this indicates which types of agreement can be reached, and therefore which types of tasks can be performed.

In particular, we showed that when the robots have total agreement on the direction and orientation of both axes, any pattern can be formed. In contrast, when they do not have any kind of agreement on the orientation of the local coordinate system, there is no deterministic and oblivious algorithm that allows the robots to form arbitrary patterns.

Then we studied what kinds of patterns can be formed when the robots agree on the orientation and direction of only one axis, say y. We showed that, if the number of robots is odd, they can form any kind of patterns. Otherwise, the robots can form only patterns that are symmetric with respect to an axis not passing through any vertex of the pattern.

Chapter 5

Gathering with Limited Visibility

To conclude, all other living creatures live orderly and well, after their own kind; we see them flock and gather together, and ready to make head and stand against all others of a contrary kind: the lions as fell and savage as they be, fight not with one another; serpents sting not serpents, nor bite one another with their venomous teeth; nay the very monsters and huge fishes of the sea, war not amongst themselves in their own kind; but believe me, man at man's hand receiveth most harm and mischief.

Pliny The Elder (23 AD – 79 AD)

Abstract

In this chapter we are interested in *gathering* the robots when they have *limited visibility* of the environment, and no memory of the past (*oblivious*). In particular, they are required to meet in some not predetermined location on the plane in a finite number of cycles.

We show that, even in such a setting, it is possible for the robots to gather in the same location in finite time, if they have a *compass*. Our result implies that, with respect to gathering, orientation (i.e., agreement on a coordinate system) is at least as powerful as instantaneous actions.

5.1 Introduction

In this chapter we are interested in *gathering*: the basic task of having the robots meet in a single location (the choice of the location is not predetermined) starting from distinct positions in the plane (this defines a valid initial configuration for this problem). Since the robots are modeled as points in the plane, the task of robots gathering is also called the *point formation problem*.

Gathering (or point formation) has been investigated both experimentally and theoretically in the *unlimited visibility* setting, that is assuming that the robots are capable to sense ("see") the entire space (e.g., see [36, 52, 74, 78]). In general, and more realistically, robots can sense only a surrounding within a radius of bounded size. This setting, called the *limited visibility* case, is understandably more difficult, and only few algorithmic results are known [3, 77].

In the limited visibility setting, Ando *et al.* [3] presented a procedure which allows indistinguishable robots, placed on a plane without any common coordinate system, to converge towards a point. Their procedure does not require the robots to remember observations nor computations performed in the previous steps. Their result implies that gathering can be *asymptotically* achieved by robots with limited visibility which are indeed very simple: *anonymous*, *oblivious* and *disoriented*.

This powerful result is however based on a very strong "atemporal" assumption: the robots must be capable in each cycle to perform all the computing and moving *instantaneously*; that is, their procedure is designed to work in SYm. This assumption has many crucial consequences: if movement is instantaneous, a robot will *not* be seen by the others while moving; if computing is instantaneous, a robot always decides based on the correct current situation in its neighborhood. This assumption also dramatically limits the practical impact of the results, since instantaneous movement is not (yet ?) physically realizable. The immediate natural question is under what conditions point formation by anonymous oblivious robots with limited visibility is possible without this assumption, i.e., in the most general (and realistic) *asynchronous* setting, when there is no common notion of time, and both computations and movements require a (finite but otherwise) unpredictable amount of time. The quest is for a condition which can be effectively realized, possibly with little effort and cost. Notice that, if movement is not instantaneous, a robot can be seen by the others while moving, and that position mistaken for a destination point; if computing is not instantaneous, a robot might make a decision based on a view of its neighborhood that is totally outdated.

In this chapter, we provide an effective answer. In fact, we prove that the availability of *orientation*[1] enables anonymous oblivious robots with limited visibility to gather within a *finite* number of *moves* even if they are fully asynchronous. In fact, we present an algorithm for solving the point formation problem in the *asynchronous* setting by *anonymous*, *oblivious* robots with *limited visibility* but with *orientation*. We then prove its correctness showing that the robots will gather in a point, and will do so within a finite amount of time. This result holds not only allowing each activity and inactivity of the robots to be totally unpredictable (but finite) in duration, but

[1] i.e., total agreement on local coordinate systems.

also making their movement towards a destination unpredictable in length (but not infinitesimally small).

This shows that gathering can be performed in a finite number of moves by simpler robots with fewer restrictions than known before, provided they have a common orientation.

From a practical point of view, this result has immediate consequences. In fact, it solves the problem without requiring the robots to have physically unrealizable motorial and computing capabilities ("instantaneous actions"), and using instead a property ("orientation") which is both simple and inexpensive to provide (e.g., by a *compass*).

From a theoretical point of view, it shows that computationally, with respect to the gathering problem, *orientation is at least as powerful as instantaneous action.*

5.1.1 Visibility and Geometric Properties

As already stated, we study this problem under the assumption that the robots can sense only a portion of the plane, namely up to at most distance V (all the robots have common knowledge on V); and that they have total agreement on the local coordinate systems.

The result of an observation by a robot (in the *Look* state) will be the positions of the robots in its circle of visibility at that time. More precisely, the *circle of visibility* $\mathcal{C}_i(t)$ of a robot r_i at time t is the circle of radius V centered in r_i, if $r_i \in \mathbb{L}(t)$; otherwise, $\mathcal{C}_i(t) = \mathcal{C}_i(t')$, where $t' = \max\{\bar{t} \leq t | r_i \in \mathbb{L}(\bar{t})\}$. In other words, if a robot is observing, its circle of visibility is the circle of radius V centered in itself; otherwise, it is the circle of visibility in its most recent *Look*. Where no ambiguity arises, the parameter t in $\mathcal{C}_i(t)$ will be omitted.

The following are elementary useful geometric properties.

Lemma 5.1.1. *Every internal chord of a triangle has length less or equal to the longest side of the triangle.*

Lemma 5.1.2. *Let $\mathcal{Q}$ be a convex quadrilateral. If all the sides and the two internal diagonals have length less or equal to V then every internal chord of $\mathcal{Q}$ has length less or equal to V.*

Another useful property is shown in the following (see Figure 5.1.b)

Lemma 5.1.3. *Let $[OB]$ be the radius of a circle centered in O and let $B\widehat{O}D = \beta$, with $0 \leq \beta \leq 90°$, for D on the circle. Then $dist(p, C) \leq dist(B, C)$, $\forall p \in arc(\beta)$ and $\forall C \in [OD]$.*

Proof. Let $p \in arc(\beta)$ and β_p the angle $p\widehat{O}C$. By the law of cosines we have:

$$dist(p,C)^2 = dist(O,C)^2 + dist(O,B)^2 - 2dist(O,C) \cdot dist(O,B)\cos\beta_p.$$

Since $0 \leq \beta_p \leq \beta \leq 90°$, we have that $\cos\beta_p \geq \cos\beta$. Thus,

$$\begin{aligned} dist(p,C)^2 &= dist(O,C)^2 + dist(O,B)^2 - 2dist(O,C) \cdot dist(O,B)\cos\beta_p \\ &\leq dist(O,C)^2 + dist(O,B)^2 - 2dist(O,C) \cdot dist(O,B)\cos\beta \\ &= dist(B,C)^2, \end{aligned}$$

and the lemma follows. □

5.2 The Algorithm

Let *Left* and *Right* be the leftmost and rightmost vertical axis, respectively, where some robot initially lie, and let us call *Universe* ($\mathfrak{U}$) the portion of the plane delimited by *Left* and *Right*[2].

The idea of the algorithm (Algorithm 4) is to make the robots move towards *Right*, in such a way that, after a finite number of steps, they will reach it and gather at the bottommost position occupied by a robot at that time.

A robot r will be allowed to move only if it does not see any robot neither to its left nor above on the vertical axis where it lies. Several situations can arise depending on the positions of the robots in its area of visibility; for each, the robot will follow different rules:

- If r sees robots to its left or above on its vertical axis, it does not move.
- If r sees robots only below on its vertical axis, it moves down to the nearest robot.
- If r sees robots only to its right, it moves horizontally to the vertical axis of the nearest robot.
- If r sees robots both below on its axis and on its right, it computes a destination point and performs a diagonal move to the right and down, as explained later.

Recall that all of the above movements may stop before the destination is reached.

Let $[AA']$ be the vertical diameter of $\mathcal{C}_i$ with A' the top and A the bottom end point; let $\mathcal{R}_i$ and $\mathcal{L}_i$ denote the topologically open regions to the right and to the left of r_i, respectively (see Figure 5.1.a). Let $S_p = [r_iA']$ and $S_o = [r_iA]$. Then, the algorithm for gathering in a point described for robot r_i is described in Algorithm 4.

[2]Recall that the robots have total agreement on the local coordinate systems.

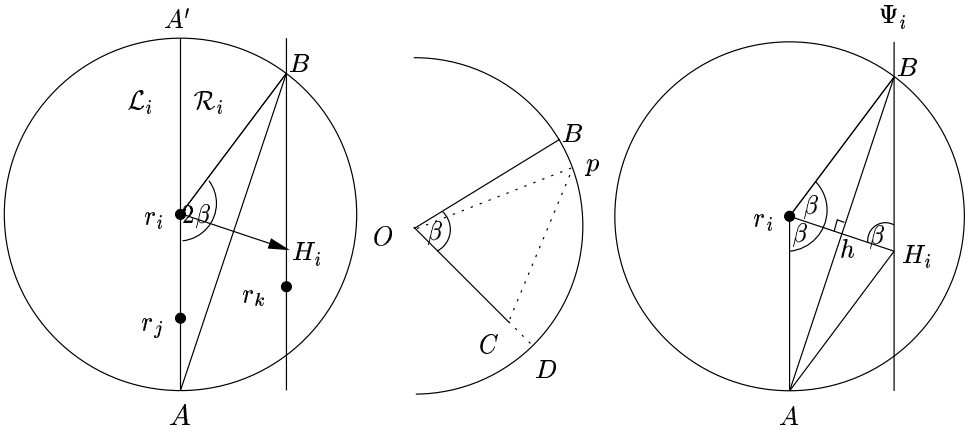


Figure 5.1: (a) The notation used in the gathering algorithm; (b) Lemma 5.1.3; (c) Routine `Diagonal_Movement(`Ψ_i`)` and Lemma 5.3.2.

Algorithm 4 Gathering With Limited Visibility

Extreme := $(|\mathcal{L}_i| = 0 \wedge |S_p| = 0)$;
If $\neg$*Extreme* **Then** `do_nothing()`.
Else
Case $(|\mathcal{R}_i|, |S_o|)$
- $(0, 0)$:
  `do_nothing()`;
- $(0, \neq 0)$: %Vertical Move%
  r_j := Nearest Visible Robot On S_o;
  p := Position Of r_j;
  `Move(`p`)`.
- $(\neq 0, 0)$: %Horizontal Move%
  Ψ_i := `Nearest()`;
  H_i := `H_Destination(`Ψ_i`)`;
  `Move(`H_i`)`.
- $(\neq 0, \neq 0)$: %Diagonal Move%
  Ψ_i := `Nearest()`;
  `Diagonal_Movement(`Ψ_i`)`.

In the algorithm, the functions `Nearest()`, `H_Destination(`Ψ_i`)`, and `Move(p)` are as follows.

`Nearest()` returns the vertical axis Ψ_i through the position of the robot in $\mathcal{R}_i$ with smallest horizontal distance from the line passing through A and A'.

`H_Destination(`Ψ_i`)` returns the intersection between Ψ_i and the horizontal line passing through r_i.

`Move(`p`)` terminates the local computation of the calling robot, and starts the *Move* towards p.

In the last case of the Algorithm 4, r_i sees some robots below it and some to its right; therefore, to avoid losing some robots, it moves diagonally, according to the following routine (see Figure 5.1.c).

`Diagonal_Movement(`Ψ_i`)`

$B :=$ Upper Intersection Between $\mathcal{C}_i$ And Ψ_i;
2: $A :=$ Point On S_o At Distance V From r_i;
$2\beta = A\widehat{r_i}B$;
4: **If** $\beta < 60°$ **Then** $B :=$ `Rotate(`r_i, B`)`;
$H_i :=$ `D_Destination(`V, Ψ_i, A, B`)`;
6: `Move(`H_i`)`.

where `Rotate(`r_i, B`)` and `D_Destination(`V, Ψ_i, A, B`)` are as follows.

`Rotate(`r_i, B`)` rotates the segment $[r_iB]$ in such a way that $\beta = 60°$ and returns the new position of B.

`D_Destination(`V, Ψ_i, A, B`)`, computes the destination of r_i in the following way: the *direction of its movement* is given by the perpendicular to the segment $[AB]$; the destination point is the intersection H_i between the direction of movement of r_i and Ψ_i.

5.3 Correctness

In this section, we prove the correctness of the algorithm by first showing that the robots which are mutually visible at any point of the computation, will stay mutually visible until the end of the computation, and then proving that at the end of the computation, all robots will gather in one point.

Since by hypothesis there is total agreement on the local coordinate systems, all the robots have a common understanding of which robots are to the right of a given robot, or which are above it; hence, for ease of expression, in the following we use "to the right", "above", etc. to denote the projection onto a single coordinate axis. Furthermore, we will call the vertical axis passing through the position of a robot

r, the vertical axis *of* r.

5.3.1 Basic Properties

We first introduce some lemmas. Let us consider a robot r_i executing the algorithm. Let β be the smallest of the two angles between the vertical axis of r_i and the direction of its movement (e.g., $A\widehat{r_i}H_i$ in Figure 5.1.c).

Lemma 5.3.1. *The length of line segment* $[r_iH_i]$ *is always smaller or equal to* V.

Proof. It follows from the fact that $dist(r_i, H_i) = 2V cos\beta$ and that, by construction, $\beta \geq 60°$. □

Lemma 5.3.2. $dist(B, H_i) = dist(A, H_i) = V$, *and,* $\forall\ p \in [r_iA]$, $dist(p, H_i) \leq V$.

Proof. By construction, we have that $dist(r_i, B) = V$ (Line 2 of routine `Diagonal_Movement(·)`). Since the triangle $\triangle(A, r_i, B)$ is isosceles and $[r_ih]$ is its height, then $A\widehat{r_i}h = \beta$. As a consequence, $r_i\widehat{H_i}B = A\widehat{r_i}h = \beta$ and, thus, also the triangle $\triangle(r_i, B, H_i)$ is isosceles and $dist(B, H_i) = dist(r_i, B) = V$.

Moreover, since $\triangle(A, r_i, B)$ is isosceles and $[r_ih]$ is perpendicular to $[AB]$, it follows that $dist(A, h) = dist(h, B)$; furthermore, $h\widehat{H_i}B = \beta$ and $dist(B, H_i) = V$; hence we can derive:

$$dist(A, H_i) = \sqrt{dist(h, H_i)^2 + dist(A, h)^2} =$$

$$\sqrt{(dist(B, H_i) \cdot \cos\beta)^2 + dist(h, B^2)} = \sqrt{V^2 \cdot \cos^2\beta + V^2 \cdot \sin^2\beta} = V.$$

The second part of the statement follows by Lemmas 5.1.1 and 5.3.1 on $\triangle(r_i, A, H_i)$. □

Thus, points A, r_i, B, and H_i form a parallelogram, that will be denoted by $\diamond(A, r_i, B, H_i)$.

5.3.2 Preserved Visibility

We now introduce the definition of *visibility graph*. The *visibility graph* $G = (N, E)$ of the robots is a graph whose node set N is the set of the input robots and, $\forall r_i, r_j \in N$, $(r_i, r_j) \in E$ iff r_i and r_j are initially at distance smaller than the visibility radius V.

Clearly, the visibility graph must be connected for the gathering problem to be solvable. In fact, we can state that

Theorem 5.3.1. *Necessary conditions for the problem to be solvable is that the visibility graph* G *is connected.*

Proof. Let us assume, by contradiction, that there exists a deterministic algorithm $\mathcal{A}$ that solve the problem even if G is disconnected. Let us assume that: there are two robots in input, say r and r'; $\mathcal{C}_r \cap \mathcal{C}_{r'} = \emptyset$; $\mathcal{A}$ is executed according to a synchronous activation schedule; and that the algorithm $\mathcal{A}$ lets them gather in $\mathfrak{p}$. Since the algorithm is deterministic, both r and r' see the world in the same way (namely no other robot is in their circles of visibility), and there is total agreement on the local coordinate systems, they will always move in the same direction and the distance between them will never decrease. Therefore they can not gather in $\mathfrak{p}$, having a contradiction. □

Thus, in the following we will always assume that G is connected.

We will now prove that the visibility relationship described by G is preserved during the entire execution of the algorithm. We do so by introducing the notion of *mutual visibility*, by showing that the robots which are connected by an edge in the visibility graph (i.e., those which are initially within distance V) will eventually become mutually visible, and that two robots that are mutually visible at some point in the algorithm will stay mutually visible until the end of the computation.

Informally speaking, we say that two robots are mutually visible if each robot includes the other one in its computation. Formally, two robots r_1 and r_2 are *mutually visible* at time t iff

- $r_1 \in (\mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{L}_S(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)) \wedge r_2 \in \mathcal{C}_1(t) \wedge r_2 \in (\mathbb{W}(t) \cup \mathbb{L}_S(t))$, or

- $r_2 \in (\mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{L}_S(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)) \wedge r_1 \in \mathcal{C}_2(t) \wedge r_1 \in (\mathbb{W}(t) \cup \mathbb{L}_S(t))$.

Since all the robots at the beginning are waiting, from the above definition we have that the robots that at the beginning are within distance V will become mutually visible in finite time. That is,

Lemma 5.3.3. *Let r_i and r_j be two robots that at the beginning are within distance V. Robots r_i and r_j will become mutually visible in a finite number of cycles.*

Proof. Let r_i enter its first look phase before r_j, at time t; that is $r_i \in \mathbb{L}_S(t)$ and $r_j \in \mathbb{W}(t)$. Since they are within distance V, $r_j \in \mathcal{C}_i(t)$, and by definition of mutual visibility the lemma follows. Similarly, the lemma holds if r_j enters its look phase before r_i, or if they enter it at the same time. □

We now introduce a lemma which will be useful to prove that mutually visible robots will stay so until the end of the algorithm. Let r_i be a robot on a vertical axis Γ. Let Γ' and Γ'' be two distinct vertical axes to the right of Γ, such that Γ' and Γ'' both intersect $\mathcal{C}_i$. Then we have (refer to Figure 5.2.a):

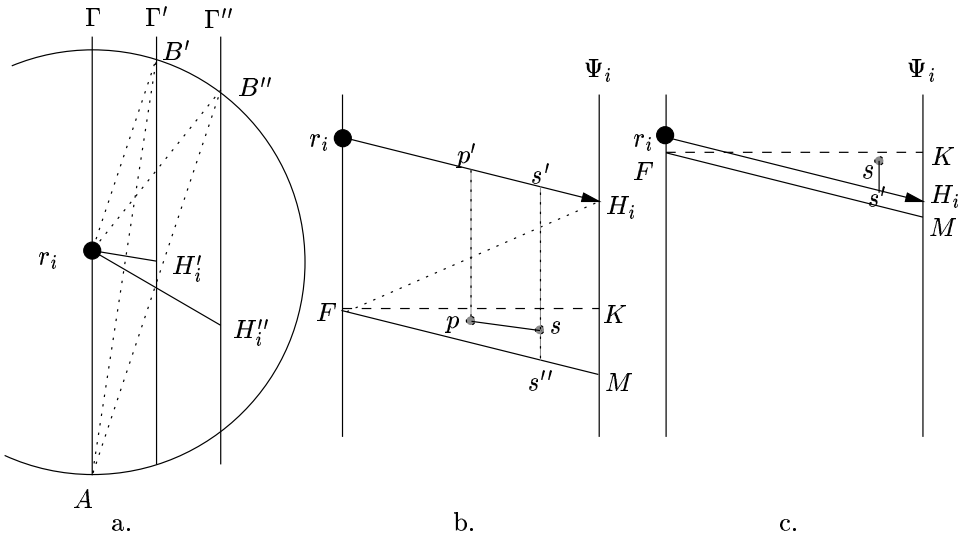


Figure 5.2: (a) Lemma 5.3.4; (b) and (c) Lemma 5.3.5.

Lemma 5.3.4. *Let $\beta_{\Gamma'}$ and $\beta_{\Gamma''}$ be the angles computed by the routines* `Diagonal_Movement(Γ′)` *and* `Diagonal_ Movement(Γ″)`*, respectively. Then,* $\overline{\Gamma\Gamma'} < \overline{\Gamma\Gamma''} \Leftrightarrow \beta_{\Gamma'} > \beta_{\Gamma''}$.

Proof. Let A be a point on Γ below r_i at distance V. We have that $\overline{\Gamma\Gamma'} < \overline{\Gamma\Gamma''}$ iff the upper intersection B' between $\mathcal{C}_i$ and Γ' is higher than the upper intersection B'' between $\mathcal{C}_i$ and Γ''; hence, $\overline{\Gamma\Gamma'} < \overline{\Gamma\Gamma''}$ iff B'' is to the right of B'. Therefore, since $2\beta_{\Gamma'} = A\widehat{r_i}B'$ and $2\beta_{\Gamma''} = A\widehat{r_i}B''$, the lemma follows. □

Let r_i be a robot, $F \neq r_i$ be a point at distance at most V from r_i on its vertical axis and below r_i, H_i the destination point of the planned move of r_i (according to the Diagonal Case of Algorithm 4), and Ψ_i the vertical axis where H_i is (the situation is depicted in Figure 5.2.b). Moreover, let K be the horizontal projection of F on Ψ_i, and M be the point on Ψ_i below H_i at distance $dist(r_i, F)$ from H_i.

Lemma 5.3.5. *Let $[ps]$ be a segment in $\triangle(F, M, K)$, with s to the right of p, and s' be the projection of s over $[r_iH_i]$. Then we have*

$$\forall\ l\ \in [ps],\ \forall\ l' \in [s'H_i], \qquad dist(l, l') \leq V.$$

Proof. We distinguish three cases.

Case 1: both p and s are below their vertical projections on $[r_iH_i]$ (see Figure 5.2.b).

By Lemma 5.3.1, we have

$$dist(s', H_i) \leq dist(r_i, H_i) \leq V. \tag{5.1}$$

Moreover, let s'' be the projection of s on $[FM]$. Since $dist(F, H_i) \leq V$ (by Lemma 5.3.2), $dist(H_i, M) = dist(r_i, F) \leq V$, and $dist(F, M) = dist(r_i, H_i) \leq V$ (by Lemma 5.3.1), it follows by Lemma 5.1.1 on $\triangle(F, H_i, M)$ that

$$dist(s'', H_i) \leq V, \tag{5.2}$$

$$dist(p, H_i) \leq V, \tag{5.3}$$

and that $dist(F, K) \leq V$. By applying again Lemma 5.1.1 on $\triangle(F, K, M)$, we have that

$$dist(p, s'') \leq V. \tag{5.4}$$

Moreover, we get:

$$dist(s', s'') = dist(r_i, F) \leq V. \tag{5.5}$$

Let p' be the projection of p on $[r_i H_i]$. Since $dist(p', p) \leq dist(r_i, F) \leq V$, by (5.3) and Lemma 5.1.1 on $\triangle(p', p, H_i)$, we have

$$dist(p, s') \leq V. \tag{5.6}$$

By (5.1)–(5.6), and Lemma 5.1.2 on $\diamond(s', p, s, H_i)$, the lemma follows.

Case 2: s is above s' (see Figure 5.2.c).

Since $[r_i H_i]$ is parallel to and above $[FM]$, and $r_i \neq F$, we clearly have that both s' and H_i are inside $\triangle(F, K, M)$. Since by construction also p is inside this triangle, the lemma follows by Lemma 5.1.1.

Case 3: p is above the vertical projection p' of p on $[r_i H_i]$.

Since p' must be inside $\triangle(F, K, M)$, we have that also s' and H_i must be inside it, and the lemma follows by Lemma 5.1.1 on $\triangle(F, K, M)$. □

We are now ready to show that, as soon as two robots become mutually visible, they will stay mutually visible. We first prove that this property holds when two mutually visible robots lie on the same vertical axis; and then we prove that it holds for two robots lying on different vertical axes.

In the next lemma we will refer to the notation introduced in Lemma 5.3.4 and 5.3.5.

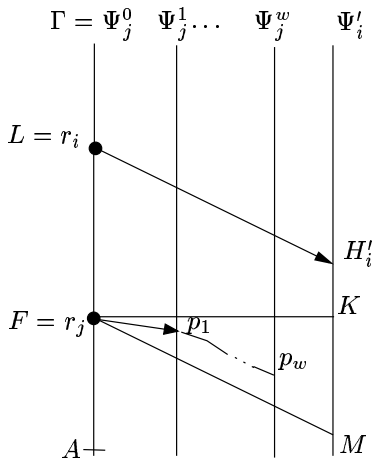


Figure 5.3: Case ii of Lemma 5.3.6.

Lemma 5.3.6. *Let r_i and r_j be two robots which are mutually visible at time t. Moreover, let them lie, at time t, on the same vertical axis Γ, with r_j being below r_i. There is a time $t' > t$ when r_i and r_j are still mutually visible. Moreover, between t and t', $dist(r_i, r_j) \leq V$.*

Proof. Let us first consider the case when $\mathcal{R}_i$ is empty. Since r_i and r_j are mutually visible at t and are on the same axis Γ, r_j can not move, and r_i can only move vertically towards r_j. More precisely, let $t'' = \min\{t^* \geq t \mid r_i \in \mathbb{L}_S(t^*)\}$. Clearly, according to the algorithm and since they are mutually visible, r_j does not move between time t and t''. Furthermore, if $\mathcal{R}_i$ is still empty at time t'', r_i can only perform a Vertical Move towards r_j (shortening their distance). According to the algorithm, r_i can not pass r_j, and r_j can not move as long as r_i is above it on its vertical axis. Let $t' > t''$ be the first time r_i stops in its movement towards r_j. From the considerations above, $r_j \in \mathbb{W}(\cdot) \cup \mathbb{L}_\emptyset(\cdot) \cup \mathbb{C}_\emptyset(\cdot) \cup \mathbb{M}_\emptyset(\cdot)$ between time t'' and t'; hence, at time t', the mutual visibility definition holds (r_i stops at time t, that is it finishes its *Move*; hence, $r_i \in \mathbb{W}(t) \cup \mathbb{L}_S(t)$), and the lemma follows.

Let us now consider the more interesting case when $\mathcal{R}_i$ is not empty. Let L be the position occupied by r_i on Γ, H_i' be the point where r_i first stops in its (diagonal) movement towards the computed destination, and Ψ_i' be the vertical axis where H_i' resides. In the following we shall consider the two possible situations:

Case 1: r_j does not look until r_i reaches H_i'. Since r_i and r_j are mutually visible at time t, we have that $r_j \in \mathbb{W}(\cdot) \cup \mathbb{C}_\emptyset(\cdot) \cup \mathbb{M}_\emptyset(\cdot)$ while r_i is moving towards H_i'. Since $dist(A, H_i) \leq V$ (Lemma 5.3.2) and $dist(r_i, H_i) \leq V$ (Lemma 5.3.1), from Lemma 5.1.1 on $\triangle(r_i, A, H_i)$, it follows that the distance between r_i and

r_j is always at most V while r_i is moving towards H_i' (recall that H_i is the destination point computed by r_i). Therefore, the first time r_i stops along its path towards H_i (i.e., on H_i'), the mutual visibility definition applies and the lemma follows.

Case 2: r_j looks while r_i is moving towards H_i'. Since r_i and r_j are mutually visible at time t, $r_j \in \mathbb{W}(t) \cup \mathbb{L}_\emptyset(t) \cup \mathbb{L}_S(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$. Moreover, since r_i and r_j are both on Γ, with r_i above r_j, as long as r_i does not move, r_j does not move; that is $r_j \in Swaitt \cup \mathbb{L}_\emptyset(t) \cup \mathbb{C}_\emptyset(t) \cup \mathbb{M}_\emptyset(t)$. If, after r_i starts moving, r_j decides not to move (because it sees some other robot above), and never moves until r_i reaches H_i', the proof is similar to the one of Case 1. Let r_j decide to move after r_i starts moving. Since r_i is to the right of r_j as r_j looks, r_j can not perform a Vertical Move. Therefore, it will perform either an Horizontal Move (if it does not see any robot below on its vertical axis), or a Diagonal Move (otherwise). The following applies to both situations.

While r_i is moving, r_j might execute several cycles, resulting in several observations (and moves). Let us call Ψ_j^w the w^{th} axis, counting from Γ, where r_j stops and looks while r_i is still moving towards H_i', and let p_w be the point on Ψ_j^w from where r_j observes. Let $\Psi_j^0 = \Gamma$, and $F = p_0$ coincide with the position of r_j on Γ (refer to Figure 5.3). In the following we will prove by induction that

a. Ψ_j^w is to the left of Ψ_j^{w+1}, and Ψ_j^{w+1} is to the left of robot r_i at the time at which r_j decides to move to a point on Ψ_j^{w+1};

b. The destination point d_{w+1} that r_j computes when it is on Ψ_j^w is inside $\triangle(F, K, M)$;

c. $dist(p_{w+1}, r_i) \leq V$.

Basis Let d_1 be the first destination point r_j computes. Since r_j can only perform a Horizontal or Diagonal Move, d_1 must be to the right of Ψ_j^0, and, as a consequence, Ψ_j^0 is to the left of Ψ_j^1; moreover, according to the algorithm, d_1 can not be to the right of r_i at that time; hence Ψ_j^1 is to the left of r_i, and claim **a.** follows. Furthermore, $\overline{\Gamma\Psi_j^1} < \overline{\Gamma\Psi_i'}$ and, by Lemma 5.3.4, d_1 must lie above M; hence, p_1 (that is on $[Fd_1]$) and d_1 must be within $\triangle(F, K, M)$, and claim **b.** follows. Finally, since d_1 is to the left of r_i at the time it is computed, by applying Lemma 5.3.5 on segment $[Fd_1]$, we have that $dist(p_1, r_i) \leq V$, and claim **c.** follows. Hence the basis of the induction holds.

Inductive Step Let us assume that all the statements are true for vertical axes $1, \ldots, w-1$. Since by inductive hypothesis Ψ_j^w is to the left of r_i at the time when r_j observes r_i from Ψ_j^w, r_j can only perform a Horizontal or a Diagonal Movement; therefore d_{w+1} must be to the right of Ψ_j^w and can not be to the right of r_i (because of how `Diagonal_Movement(·)` and `H_Destination(·)` are defined). As a consequence, Ψ_j^w is to the left of Ψ_j^{w+1}, Ψ_j^{w+1} is to the left of r_i, and claim **a.** follows.

Moreover, $\overline{\Psi_j^w \Psi_i'} < \overline{\Gamma \Psi_i'}$ and, by Lemma 5.3.4 the angle β computed by r_j from p_w must be greater than the angle computed by r_i from L; hence, since by construction segment $[FM]$ is parallel to $[LH_i']$, $[p_w d_{w+1}]$ is above $[FM]$, but cannot be above $[FK]$ (because the algorithm does not allow "up" movements). Therefore claim **b.** follows.

Furthermore, since d_{w+1} is to the left of r_i, claim **b.** holds, and Ψ_j^{w+1} can not be to the right of d_{w+1}, by applying Lemma 5.3.5 on segment $[p_w d_{w+1}]$, we have that $dist(p_{w+1}, r_i) \leq V$, and claim **c.** follows, and the induction is proved.

Now we know that all the stops that r_j performs while r_i is moving towards H_i' are inside $\triangle(F, K, M)$, and hence, by Lemma 5.3.5 and by above point **c.**, within distance V from r_i. Thus, when r_i reaches H_i' (i.e., it stops on H_i'), r_j is on its left; at this time, according to Algorithm 4 it follows that, as long as r_j is to the left of r_i, r_i can be only in $\mathbb{W}(\cdot) \cup \mathbb{L}_{\emptyset}(\cdot) \cup \mathbb{C}_{\emptyset}(\cdot) \cup \mathbb{M}_{\emptyset}(\cdot)$. Therefore, the first time that r_j stops after r_i reaches H_i', say at time $t' > t$, r_i and r_j will be mutually visible. Moreover, between t and t', by Lemma 5.3.5 and by above point **c.**, $dist(r_i, r_j) \leq V$, and the lemma follows. □

In the following lemma, we show that if a robot is mutually visible with some robots on its right, it will never lose them during the computations. Let B and C be respectively the upper and lower intersection between Ψ_i and $\mathcal{C}_i$, and z_i be the intersection between $\mathcal{C}_i$ and the line passing through r_i and H_i (refer to Figure 5.4).

Lemma 5.3.7. *Let $\mathbb{H}$ be a set of robots which are mutually visible with r_i at time t and that are located to the right of Ψ_i. There exists a time $t' > t$ after which r_i will still be mutually visible with the robots in $\mathbb{H}$. Moreover, between t and t', $\forall\ r^* \in \mathbb{H}$, $dist(r_i, r^*) \leq V$.*

Proof. According to the algorithm and because of the hypothesis on mutual visibility, as long as r_i is on the left of the robots in $\mathbb{H}$ after t, these robots cannot perform any movement, that is $\forall r^* \in \mathbb{H}$, $r^* \in \mathbb{W}(\cdot) \cup \mathbb{L}_{\emptyset}(\cdot) \cup \mathbb{C}_{\emptyset}(\cdot) \cup \mathbb{M}_{\emptyset}(\cdot)$. Let $t^* \geq t$ be the first time when r_i enters a *Look* state, and let H_i be the destination

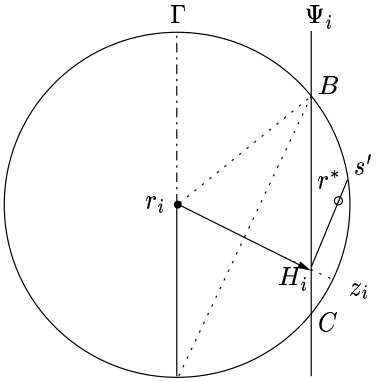


Figure 5.4: Lemma 5.3.7.

point it computes. Clearly, according to the algorithm, H_i can not be to the right of any robot in $\mathbb{H}$.

Let us first prove that $\forall\ r^* \in \mathbb{H}$ and $\forall l \in [r_i H_i]$, $dist(l, r^*) \leq V$. From Lemmas 5.1.3 and 5.3.2, it follows that:

$$\forall p \in arc(Bz_i), \quad dist(p, H_i) \leq dist(BH_i) = V. \tag{5.7}$$

Moreover, $dist(H_i, C) = dist(B, C) - dist(B, H_i) \leq 2V - V = V$, and from Lemma 5.1.3 we have:

$$\forall p \in arc(z_i C), \quad dist(p, H_i) \leq dist(H_i, C) \leq V. \tag{5.8}$$

Combining (5.7) and (5.8), we obtain:

$$\forall p \in arc(BC), \quad dist(p, H_i) \leq V. \tag{5.9}$$

Let us now consider a robot $r^* \in \mathbb{H}$. First observe that, by definition of Ψ_i, $r^* \in segm(BC)$ (that is in the area of $\mathcal{C}_i$ to the right of Ψ_i and in $\mathcal{C}_i$). Let s' be the intersection between $arc(BC)$ and the line passing through H_i and r^*. We have that $dist(H_i, r^*) \leq dist(H_i, s') \leq V$ (by (5.9)), $dist(r_i, r^*) \leq V$, and $dist(r_i, H_i) \leq V$. Therefore, applying Lemma 5.1.1 to $\triangle(r_i, r^*, H_i)$ we have that, $\forall l \in [r_i H_i]$, $dist(l, r^*) \leq V$. In conclusion, the first time r_i stops on $[r_i H_i]$ (it can happen on Ψ_i or before), say at time $t' > t^*$, all the robots in $\mathbb{H}$ are to its right; all these robots can only be in $\mathbb{W}(t') \cup \mathbb{L}_{\emptyset}(t') \cup \mathbb{L}_S(t') \cup \mathbb{C}_{\emptyset}(t') \cup \mathbb{M}_{\emptyset}(t')$, and the lemma follows. □

By Lemmas 5.3.3, 5.3.6 and 5.3.7 we can conclude that:

Theorem 5.3.2. *The visibility relationship described by the (initial) visibility graph G is preserved during the execution of Algorithm 4.*

5.3.3 Finiteness

In this Section we prove that, after a finite number of steps, the robots will gather in a point. Recall that we assume the visibility graph of the robots to be connected initially and therefore, by Theorem 5.3.2, at all times.

Lemma 5.3.8. *Let Λ be a vertical axis with several robots on it, and with no robots to its left. Let r be the topmost robot on Λ that has a robot to the right of Λ in its circle of visibility, and let $\mathbb{S}$ be the set of robots on Λ above r. In a finite number of cycles, either all robots in $\mathbb{S}$ will reach r, or one of the robots in $\mathbb{S} \cup \{r\}$ will leave Λ.*

Proof. If $\mathbb{S} = \emptyset$, according to the algorithm, the lemma trivially follows. Let $\mathbb{S} \neq \emptyset$. Assume, by contradiction, that no robot in $\mathbb{S} \cup \{r\}$ leaves Λ, and that there is a nonempty set $\mathbb{H} \subseteq \mathbb{S}$ of robots that do not reach r in a finite number of cycles. Let $r' \in \mathbb{H}$; since r' does not reach r, there exists at least a point on Λ, above r, which r' does not pass in a finite number of cycles. Let $p(r')$ be the topmost such point, let p be the topmost of $p(r')$ for all $r' \in \mathbb{H}$, and let $\mathbb{H}' \subseteq \mathbb{H}$ be the set of robots that never pass p. Within finite time, all robots in $\mathbb{S} \setminus \mathbb{H}'$ will be below p; consider a time t when only robots in $\mathbb{H}'$ are above p. Since the cycle time of any robot is finite (Assumption A1 of the model), and by Theorem 5.3.2 the connectivity of the visibility graph is always preserved, the topmost robot in $\mathbb{H}'$ must perform a movement in finite time; its move must be vertical because the robot is not allowed to leave Λ by hypothesis. It follows that all robots in $\mathbb{H}'$ will get closer and closer to p. After a finite number of cycles, all the robots in $\mathbb{H}'$ will be at distance smaller than δ from[3] p; let $t' > t$ be a time when this happens. Consider now a point p' above p and below the bottommost (at time t') robot in $\mathbb{H}'$. We know that all robots in $\mathbb{H}'$ must pass p' in a finite number of moves. Since there are no robots between p' and p, the first robot r' which passes p' must have as its destination a point below p. Since the distance between r' and p is smaller than δ, r' will pass p (Assumption A2 of the model) leading to a contradiction. Thus, in a finite number of cycles, either all the robots in $\mathbb{H}$ reach r, or at least one of them leaves Λ. □

The next Lemma shows that all robots in the system go onto the *Right* axis of the universe $\mathfrak{U}$.

Lemma 5.3.9. *If Left $\not\equiv$ Right, then for any given vertical axis to the left of Right, all the robots that are on its left at the beginning of the algorithm, will pass it in a finite number of cycles.*

[3] δ has been introduced in Section 3.1.

Proof. Assume, by contradiction, that there exists at least one vertical axis in $\mathfrak{U}$ before *Right* such that a subset of the robots that are to the left of it at the beginning of the algorithm will not pass it in a finite number of cycles. Let Ψ be the leftmost of these axes. Let $\mathbb{H}$ be the set of robots which do not pass Ψ in a finite number of moves. Let $\mathbb{H}_a \subseteq \mathbb{H}$ contain the robots that reach (but do not pass) Ψ in a finite number of cycles, and $\mathbb{H}_b \subseteq \mathbb{H}$ contain the robots that will not even reach Ψ in a finite number of cycles ($\mathbb{H} = \mathbb{H}_a \cup \mathbb{H}_b$). Consider Ψ at a time t when only the robots in $\mathbb{H}$ have not passed Ψ.

First of all, we note that Ψ cannot be the first vertical axis of the system (*Left*). In fact, if Ψ is *Left*, let r be the topmost robot of $\mathbb{H}$ on this first axis that sees some robots to its right (since $Left \not\equiv Right$, and by Theorem 5.3.2 the connectivity of G is preserved, such a robot must exist); by Lemma 5.3.8 either one of the robots above r will leave Ψ, or r will be reached by all the robots above it, and then it will be allowed to move to its right passing Ψ. Hence $\Psi \neq Left$.

By hypothesis, Ψ is the first axis in $\mathfrak{U}$ which is not passed by robots in $\mathbb{H}$, hence the robots in $\mathbb{R}_b$ must pass, in a finite number of cycles, any vertical axis in $\mathfrak{U}$ to the left of Ψ, hence getting infinitely closer to Ψ. We now consider two cases:

Case 1: $\mathbb{H}_b = \emptyset$.

In this case, after a finite number of cycles, all robots in $\mathbb{H}$ reach Ψ. Since the connectivity of G is preserved (by Theorem 5.3.2), and $\Psi \neq Right$, at least one robot r in $\mathbb{H}$ must see some robot to the right of Ψ. Let r be the topmost such robot.

a. If r cannot see any robot above it on Ψ, it can move either horizontally or diagonally, thus passing Ψ and leading to a contradiction.

b. Otherwise, by Lemma 5.3.8, we have that, in a finite number of cycles, either one of the robots above r on Ψ will leave it; or r will be reached by all the robots above it on Ψ, allowing it to move either horizontally or diagonally. In both cases, we have a contradiction.

Case 2: $\mathbb{H}_b \neq \emptyset$.

In this case, the robots in $\mathbb{H}_a$ reach Ψ in a finite number of cycles, while the robots in $\mathbb{H}_b$ just converge to it. Fix an arbitrary vertical axis Ψ' to the left of Ψ, at horizontal distance smaller than $\delta' < \frac{\sqrt{3}}{2}\delta$ from Ψ. In a finite number of cycles, the robots in $\mathbb{H}_b$ will pass Ψ'; they must also stop at least once to the right of Ψ' (otherwise they would pass Ψ). Let us now consider the computation after the robots in $\mathbb{H}_a$ have reached Ψ, and the robots in $\mathbb{H}_b$ have passed Ψ' stopping at least once to the right of it; let t be the first time when this happens. Consider another arbitrary axis Ψ'' at time t, to the right of the

rightmost robot in $\mathbb{H}_b$ and to the left of Ψ, at horizontal distance smaller than $\delta' < \frac{\sqrt{3}}{2}\delta$ from it. By definition, the robots in $\mathbb{R}_b$ converge to Ψ, and hence they will pass Ψ'' in a finite number of moves. Since, by construction, there are no robots between Ψ'' and Ψ, the first robot $r \in \mathbb{H}_b$ that passes Ψ'' must have as its destination either a point to the right of Ψ or on Ψ. According to the algorithm, r will move on a straight line towards the right either horizontally or diagonally at an angle β, $60° \leq \beta < 90°$. Since $\delta' < \frac{\sqrt{3}}{2}\delta$ then, by Assumption A2, r will reach Ψ in one move regardless of β, creating a contradiction. □

We now prove that all robots in the system actually reach the *Right* axis of $\mathfrak{U}$.

Lemma 5.3.10. *After a finite number of cycles, all the robots in the system reach Right.*

Proof. If *Left* ≡ *Right*, then the lemma trivially follows. Otherwise, by contradiction, assume that there is a subset $\mathbb{H}$ of robots that will converge to *Right* never reaching it. Thus, in a finite number of cycles, at time t, the only robots which do not reach *Right* are the robots in $\mathbb{H}$. Consider a vertical axis Ψ before *Right* at distance smaller or equal to $\delta' < \frac{\sqrt{3}}{2}\delta$ from it. Since the robots in $\mathbb{H}$ converge to *Right*, they will pass Ψ in a finite number of cycles. Let us call $r \in \mathbb{H}$ the first robot that does it. Applying a similar argument to the one of the previous lemma, we have that r must reach *Right*, leading to a contradiction. □

The following lemma shows that if all robots lie on the same vertical axis, they will reach the bottommost robot in a finite number of cycles.

Lemma 5.3.11. *If all the robots of the system lie on the same vertical axis Λ, then in a finite number of cycles all the robots will reach the bottommost robot on Λ.*

Proof. Assume by contradiction that there is a nonempty set $\mathbb{H}$ of robots which will not reach the bottommost robot r_b on Λ. Let $r \in \mathbb{H}$; let $p(r)$ be the topmost point (below r) that is not passed by r; let p be the topmost among all $p(r)$, for all $r \in \mathbb{H}$, and let $\mathbb{H}' \subseteq \mathbb{H}$ be the set of robots that never pass p. Within finite time, all robots not in $\mathbb{H}'$ will be below p; consider a time t when only robots in $\mathbb{H}'$ are above p. Since the cycle time of any robot is finite (Assumption A1), and by Theorem 5.3.2 the connectivity of the visibility graph is always preserved, the topmost robot in $\mathbb{H}'$ must perform a vertical movement in finite time. It follows that all robots in $\mathbb{H}'$ will get closer and closer to p. After a finite number of cycles, all the robots in $\mathbb{H}'$ will be at distance smaller than δ from p. Now, the same argument of Lemma 5.3.8 can be applied to derive the contradiction. □

By Lemmas 5.3.9–5.3.11 we can finally conclude that:

Theorem 5.3.3. *In a finite number of cycles, all the robots in the system gather in a point on Right, by executing Algorithm 4.*

As a final remark, we note that Algorithm 4 gather the robots even if they do not occupy distinct positions on the plane at the beginning.

Result 6. *With total agreement on the local coordinate systems, there exists a deterministic oblivious algorithm that gather the robots in a point in a finite number of cycles.*

5.4 Conclusions

In this chapter we have analyzed the gathering problem, for groups of autonomous mobile robots, and presented a gathering algorithm which solves the problem within *finite time*. Our solution operates in *fully asynchronous* settings and can be achieved by robots which are anonymous, oblivious and with limited visibility, provided they have orientation.

The results presented here are a further step towards a better understanding of the minimal robot capabilities enabling the collective resolution of a given global task. In particular, they show how one assumption ("orientation") has at least the same computational power for the gathering problem as another property ("instantaneous action"). This fact raises many intriguing questions, including: Are there other (simpler) properties comparable to these two? How do we compare different properties?

Chapter 6

Gathering with Unlimited Visibility[1]

A conference is a gathering of important people who singly can do nothing, but together can decide that nothing can be done.

Fred Allen (1894 – 1956)

Abstract

In this chapter we will study again the gathering problem. In contrast with the previous chapter, we will drop the condition that the robots can sense only a portion of the plane, and see how the capability to sense all the plane can influence the solvability of this problem.

6.1 Introduction

In this chapter we analyze again the gathering problem (given n robots $r_1, \ldots, r_n$ on distinct positions in the plane, they are required to gather on a point in a finite number of cycles), but under different conditions. In particular, in this case the robots can sense all the plane: they have unlimited visibility. In contrast with the result obtained in the previous chapter, however, the robots do not have any kind of agreement on the local coordinate systems. That is, we are trading the *compass* used to solve the problem in Chapter 5 with more power on the sensors. We want to understand how this change of *power* on the capabilities of the robots can influence the solvability of this problem.

[1]The ideas and results presented in this chapter have been partially developed in collaboration with Mark Cieliebak, ETH Zürich.

6.1.1 The Weber Point

An easy solution to this problem is given by the *Weber Point*.

Definition 6.1.1. The distance between a point p and a set $\mathbb{B} = \{p_1, \ldots, p_n\}$ of distinct points on the plane is defined as

$$D(p, \mathbb{B}) = \Sigma_{q \in \mathbb{B}} dist(p, q).$$

The Weber Point WP for $\mathbb{B}$ is the point on the plane that minimizes its distance from $\mathbb{B}$, i.e.

$$D(WP, \mathbb{B}) = \min_{p \in \mathbb{R}^2} D(p, \mathbb{B}).$$

◁▷

It has been proven in [81] that, if the n points are not collinear, their Weber point always exists and is unique. If they are collinear, then it is unique only for an odd number of robots (it is the median of the n points). An interesting property of the WP is that

Lemma 6.1.1 ([81]). *If WP is the Weber point of $p_1, \ldots, p_n$ non collinear and distinct points on the plane, then WP is the Weber point of $q_1, \ldots, q_n$, with $q_i \in [p_i, WP]$.*

The above lemma gives us an easy algorithm to solve the gathering problem: all the robots have simply to move towards their Weber point. Unfortunately, it has been proven in [24] that in general the Weber point is not computable. Therefore, we need a different strategy. In the remaining of the chapter, we will present four algorithms that solve the problem with 2, 3, 4, and more than 4 robots, respectively.

6.2 Geometric Definitions and Properties

In this section we introduce some geometric definitions that will be useful in the following. First, we introduce a definition of angle more general than the one introduced in Section 3.1.2.

Definition 6.2.1 (Angle $c\lhd^{q}_{q'}$). Given c, q and q' three distinct points in the plane, and an orientation (clockwise or counterclockwise), let $\mathcal{S}(q, c, q')$ be the region of the plane obtained by a rotation centered in c of the half-line $[cq)$ onto $[cq')$, according to the given orientation. We define $c\lhd^{q}_{q'}$ as the angle defined by $\mathcal{S}(q, c, q')$ (see Figure 6.1.a and b); namely

$$c\lhd^{q}_{q'} = \begin{cases} q\widehat{c}q' & \text{if } \mathcal{S}(q, c, q') \cap q\widehat{c}q' = \mathcal{S}(q, c, q') \\ q\check{c}q' & \text{if } \mathcal{S}(q, c, q') \cap q\widehat{c}q' = [cq) \cup [cq'). \end{cases}$$

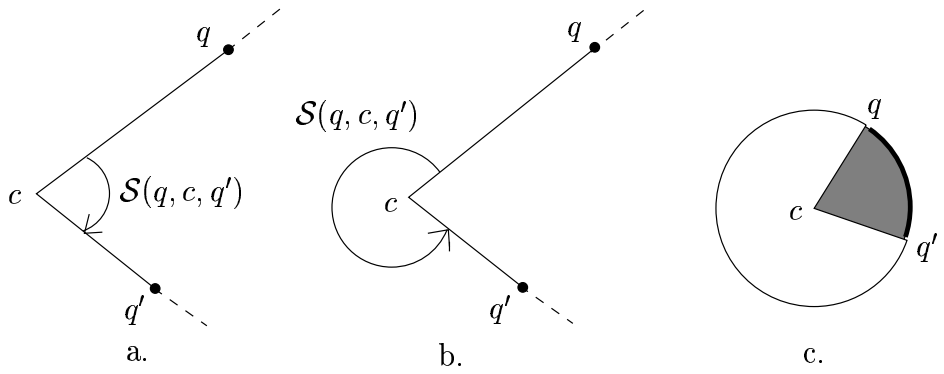


Figure 6.1: (a) Definition of ${}_c\lhd^q_{q'}$ with a clockwise orientation, and (b) with a counterclockwise orientation. (c) Definition of $arc({}_c\lhd^q_{q'})$ (the thick line) and $sector({}_c\lhd^q_{q'})$ (the grey area), with the circle oriented clockwise.

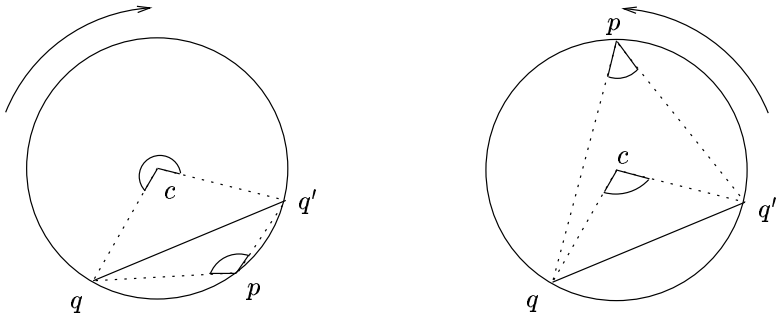


Figure 6.2: Corollary 6.2.1. To the left it is represented the scenario where the circle is oriented clockwise, while to the right the orientation is counterclockwise.

In the following, ${}_c\lhd^q_{q'}$ will denote also the width of the angle. ◁▷

A well known relationship between angles at the centre and angles at the circumference of a given circle is described by the following

Theorem 6.2.1. *The angle at the centre of a circle is double the angle at the circumference standing on the same arc.*

Hence (refer to Figure 6.2),

Corollary 6.2.1. *Let $\mathcal{C}$ be a circle with center c, and q and q' two non opposite points on its circumference. Then*

$$q\widehat{p}q' = \frac{1}{2}{}_c\lhd^q_{q'},$$

with $p \in arc({}_c\lhd^{q'}_q)$.

For ease of notation, in the following we will denote $arc({}_c\lhd^q_{q'})$ also by $arc(q, q')$, when c is understood (a similar notation applies to $sector({}_c\lhd^q_{q'})$). Next, we define the smallest circle enclosing n points.

Definition 6.2.2 (SEC($p_1, \ldots, p_n$)). Given n points $p_1, \ldots, p_n$ on the plane, their *Smallest Enclosing Circle* $SEC(p_1, \ldots, p_n)$, shortly SEC if no ambiguity arises, is the smallest circle such that all the points are on its circumference or inside it. We say that $p_1, \ldots, p_n$ are the points that *define* SEC. ◁▷

Observation 6.2.1. A smallest enclosing circle passes either through two points that are on the same diameter (opposite points), or through at least three points. According to [82], the SEC of n distinct points is unique and can be computed in at most $O(n \log n)$ time. Moreover, it does not change by eliminating all the points defining SEC that are inside it. ◇

The center of the SEC will be denoted by c. Let p_i be one of the points that define $SEC(p_1, \ldots, p_n)$, and q be the intersection between the circumference of SEC and $[cp_i)$. We denote by $\Psi(p_i)$ the segment $[cq]$.

The following lemma determines which points can be eliminated from the set of points that defines SEC, without causing changes in the SEC.

Lemma 6.2.1. *Let SEC be the smallest circle enclosing n points on the plane, and let c and $Rad(SEC)$ its center and radius, respectively. Moreover, let $l_1, \ldots, l_k$, $4 \leq k \leq n$ be the subset of the points that define SEC that are on its circumference (let us assume w.l.o.g they are in clockwise order).*

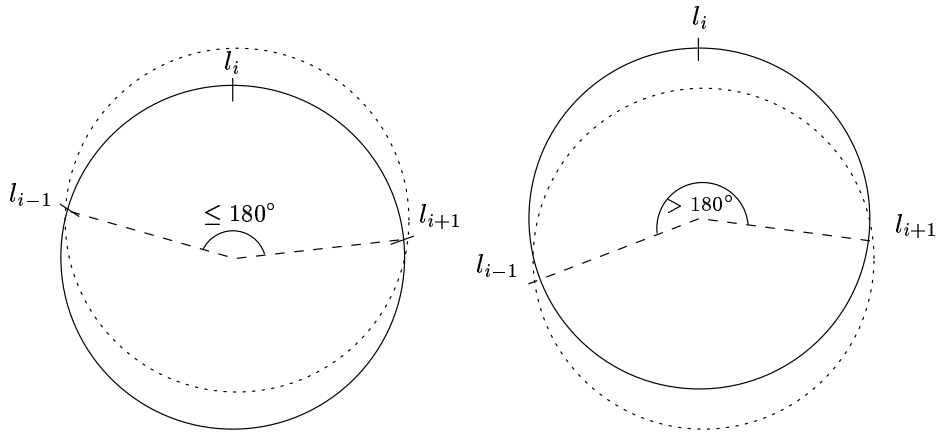


Figure 6.3: Lemma 6.2.1. The dashed circle represents SEC'; here, $Rad(SEC) = Rad(SEC')$, and the circles have a clockwise orientation. (a) SEC' includes l_{i-1} and l_{i+1}; therefore, since ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} \leq 180°$, it must contain l_i too. (b) If ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} > 180°$, it is possible that SEC' does not contain any more l_i.

1. *There exists a point l_i, $1 \leq i \leq k$, such that ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} \leq 180°$ (index operations are modulo k). Furthermore,*

2. *SEC does not change by eliminating l_i from the set of points that define it.*

Proof. Let us assume that there exists no l_i such that ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} \leq 180°$. Then, ${}_c\sphericalangle_{l_3}^{l_1} + {}_c\sphericalangle_{l_1}^{l_3} > 360°$ (recall that $n \geq 4$), having a contradiction since $\Sigma_{i=1}^{n} {}_c\sphericalangle_{l_{i+1}}^{l_i} = 360°$; hence statement 1. follows.

In order to prove the second part of the lemma, let l_i be a point such that ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} \leq 180°$. If ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} = 180°$, the circle does not change by removing l_i, since l_{i-1} and l_{i+1} are opposite, and the smallest enclosing circle must pass through them (Observation 6.2.1). Therefore, let us assume ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} < 180°$. By contradiction, let us suppose that the circle changes when eliminating l_i, and let SEC' be the new smallest circle enclosing $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$. Since by definition SEC encloses $l_1, \ldots, l_{i-1}, l_{i+1}, \ldots, l_n$, we have that $Rad(SEC') \leq Rad(SEC)$. SEC and SEC' intersect in at most two points, and l_{i-1} and l_{i+1} must be either inside SEC' or on it. Furthermore, since ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}} < 180°$, it follows that $arc(l_{i-1}, l_{i+1})$ is completely inside SEC' (see Figure 6.3.a). Therefore, if we put again l_i in its original position, SEC' can not change (since l_i is now inside SEC'). Hence, we would have two different smallest circles enclosing n points, thus contradicting the uniqueness of SEC. □

According to the notation introduced in the previous lemma, we will call ${}_c\sphericalangle_{l_{i+1}}^{l_{i-1}}$ the angle *created* by l_i.

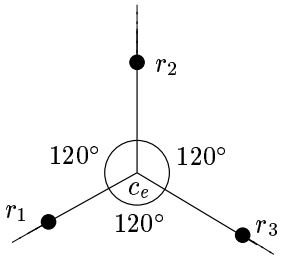


Figure 6.4: Center of equiangularity of r_1 and r_2 and r_3.

6.3 Case a.: $n = 2$

The gathering for two robots seems an easy problem. Suzuki *et al.* proved in [78], however, that this problem is unsolvable in SYm when the robots can cross each other (i.e., they have no dimension); that is, if the robots do not stop when they are moving on the same line in opposite direction and they meet: we will say that the robots do not *bump into each other*. Hence, by Corollary 3.3.1, this problem has no solution in CORDA when the robots do not stop when they meet.

It can be easily solved, however, when the robots can bump into each other. In fact, in this case, it is sufficient that each one of the two robots simply start moving towards the other robot. Following this simple rule, in a finite number of cycles the two robots will bump into each other, hence gather in a point. Therefore, we can state the following

Result 7. *There exists a deterministic and oblivious algorithm that solves the gathering problem for two robots if and only if the robots can bump into each other.*

6.4 Case b.: $n = 3$

In the case with 3 robots, the algorithm we propose takes advantage from the property of the center of equiangularity of three distinct points in the plane (see Figure 6.4).

Definition 6.4.1. A point c_e is the *center of equiangularity* of three distinct points in the plane r_1, r_2 and r_3, if $r_1\widehat{c_e}r_2 = r_1\widehat{c_e}r_3 = r_3\widehat{c_e}r_2 = 120°$. ◁▷

It is easy to see that

Theorem 6.4.1. *If the center of equiangularity of three distinct points in the plane exists, then it is unique and is inside the triangle having as vertices the three given points.*

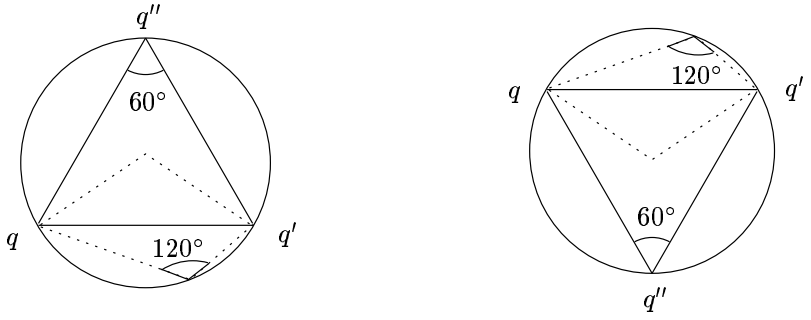


Figure 6.5: The two *120-Circles* defined by q and q'.

Furthermore, we observe that (refer to the example depicted in Figure 6.4)

Observation 6.4.1. If c_e is the center of equiangularity of three distinct points in the plane r_1, r_2 and r_3, then c_e is the center of equiangularity with respect to p_1, p_2 and p_3, with $p_1 \in [c_e r_1)$, $p_2 \in [c_e r_2)$, and $p_1 \in [c_e r_3)$. ◇

Hence, the idea of the algorithm is to let the three robots move towards their center of equiangularity. In order to do this, we need a way to locate such a point. Given two distinct points q and q', we call the *120-Circle* defined by q and q' the circle passing through q, q' and q'', with q'' such that $\triangle(q, q', q'')$ is equilater. This triangle will be referred to as the triangle *associated to the 120-Circle*. We note that there are two different *120-Circles* defined by q and q', depending in which halfplane q'' lies (see Figure 6.5). Furthermore, by Corollary 6.2.1 it follows that

Corollary 6.4.1. *Given a* 120-Circle *with center c defined by two distinct points q and q' in the plane, $q\widehat{p}q' = 120°$, for all $p \in arc(q\widehat{c}q')$.*

Let p_1, p_2 and p_3 be three distinct points in the plane. Moreover, let $\mathcal{C}_{ij}$, with $1 \leq i \neq j \leq 3$, the *120-Circle* defined by p_i and p_j such that the triangle associated to $\mathcal{C}_{ij}$ does not intersect $\triangle(p_1, p_2, p_3)$, and c_{ij} its center. Furthermore, let w_i the intersection between $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$, with $1 \leq i \neq j \neq k \leq 3$ and $i \neq k$, such that $w_i \notin \{p_1, p_2, p_3\}$. By construction, it follows that

Observation 6.4.2. If $w_i \in arc(p_i\widehat{c_{ij}}p_j)$ and $w_i \in arc(p_i\widehat{c_{ik}}p_k)$, then $w_i \tilde{\in} \triangle(p_1, p_2, p_3)$. ◇

For instance, in the example depicted in Figure 6.6, $w_1 = c_e$, $w_1 \in arc(p_1\widehat{c_{12}}p_2)$, $w_1 \in arc(p_1\widehat{c_{13}}p_3)$ and w_1 is strictly inside $\triangle(p_1, p_2, p_3)$. Furthermore,

Observation 6.4.3. If $p_i \tilde{\in} segm(p_j\widehat{c_{jk}}p_k)$, then $p_j \notin segm(p_i\widehat{c_{ik}}p_k)$, and $p_k \notin segm(p_i\widehat{c_{ij}}p_j)$. ◇

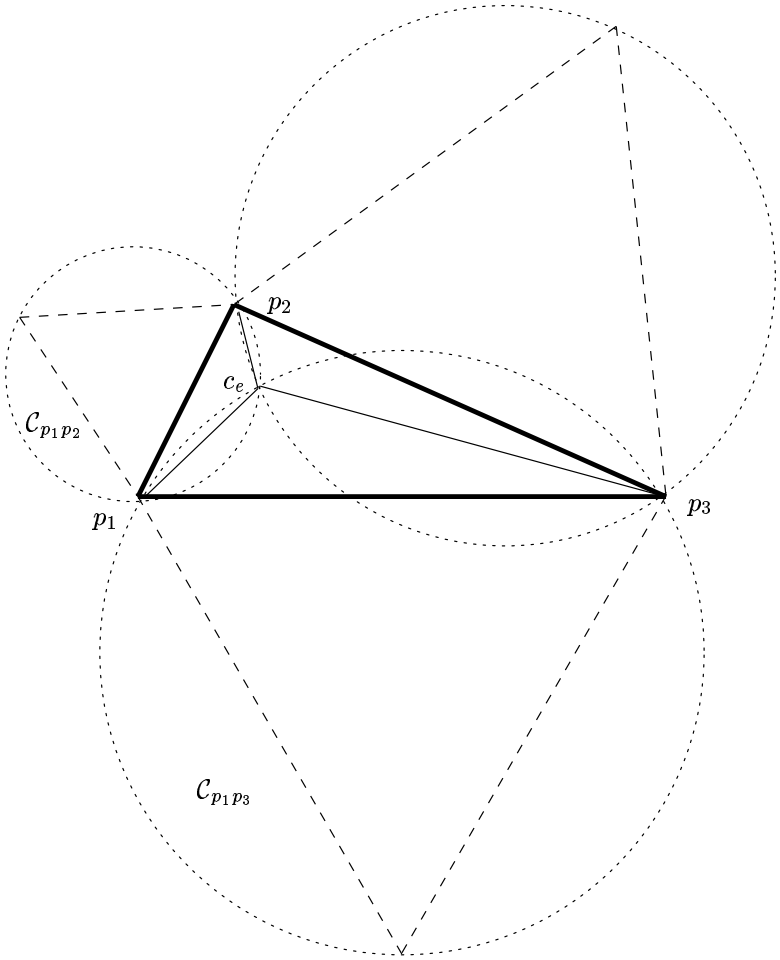


Figure 6.6: Lemma 6.4.1.

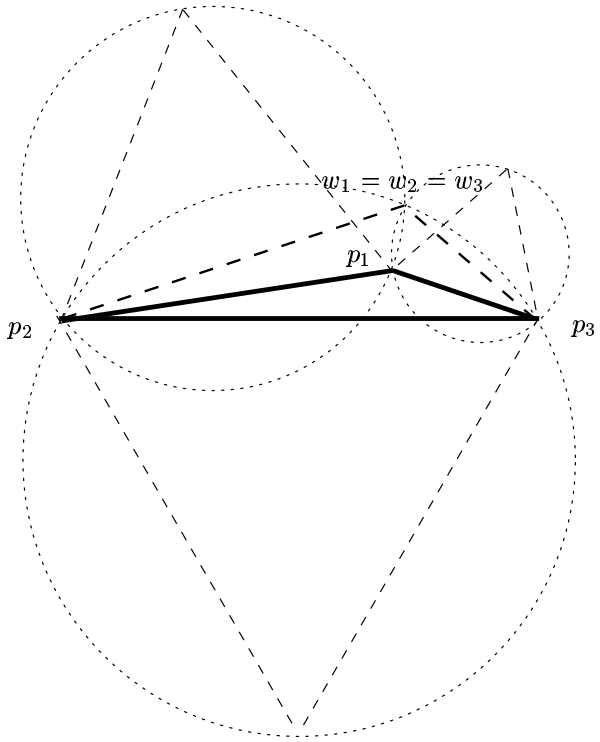


Figure 6.7: Lemma 6.4.2.

For instance, in the example depicted in Figure 6.7, $p_1 \stackrel{\sim}{\in} segm(p_2\widehat{c_{23}}p_3)$, while $p_2 \notin segm(p_1\widehat{c_{13}}p_3)$, and $p_3 \notin segm(p_1\widehat{c_{12}}p_2)$.

Now, we are ready to show the relationship between *120-Circles* and center of equiangularity.

Lemma 6.4.1. *If $w_i \neq \emptyset$ and $w_i \stackrel{\sim}{\in} \triangle(p_1, p_2, p_3)$, then w_i is the center of equiangularity of the three given points. Furthermore, $c_e = w_i = w_j = w_k$.*

Proof. By the way $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$ have been chosen (their associated triangles do not intersect $\triangle(p_1, p_2, p_3)$), and by Corollary 6.4.1, it follows that $p_i\widehat{w_i}p_j = p_i\widehat{w_i}p_k = 120°$ (refer to Figure 6.6, with $i = 1$ and $j = 2$). Therefore, $p_j\widehat{w_i}p_k = 120°$, and, by Theorem 6.4.1, the lemma follows. □

Hence, if at the beginning the positions of the robots satisfy the conditions of the previous lemma, the robots have simply to move towards w_i. Unfortunately, three points not always have a center of equiangularity. This happens when $w_i \notin \triangle(p_1, p_2, p_3)$. In the following, we describe a property of this scenario, that will help us in solving the problem in this case.

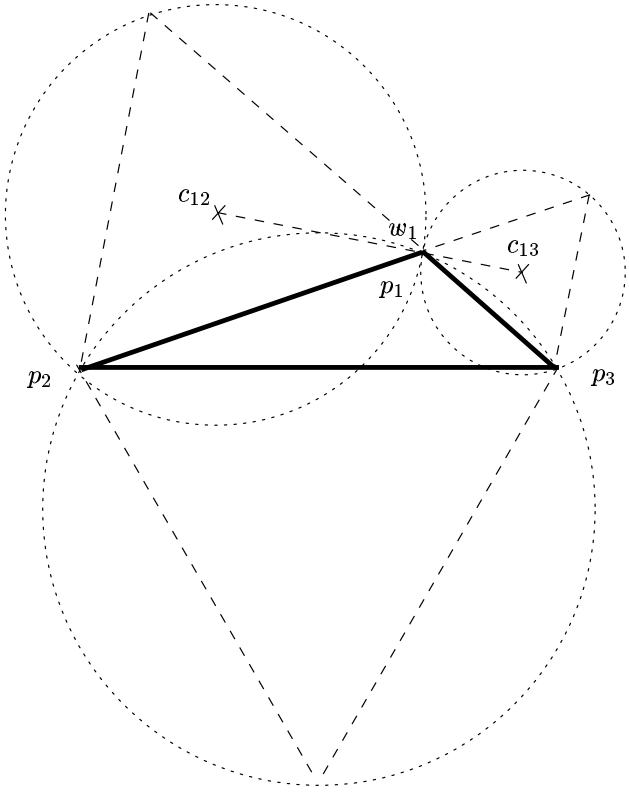


Figure 6.8: Lemma 6.4.3.

Lemma 6.4.2. *If $w_i \neq \emptyset$ and $w_i \notin \triangle(p_1, p_2, p_3)$, then there exists i' such that $p_{i'} \tilde{\in} segm(p_{j'}\widehat{c_{j'k'}}p_{k'})$, with $1 \leq i' \neq j' \neq k' \leq 3$ and $i' \neq k'$.*

Proof. By Observation 6.4.2, we distinguish two cases:

1. $w_i \in arc(p_i\widehat{c_{ij}}p_j)$ and $w_i \notin arc(p_i\widehat{c_{ik}}p_k)$ (refer to Figure 6.7, with $i = 2$ and $j = 3$). By Corollary 6.4.1, $p_i\widehat{w}_ip_j = 120°$, and by Corollary 6.2.1 $p_i\widehat{w}_ip_k = 60°$. Therefore, since the triangles associated to $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$ do not intersect $\triangle(p_1, p_2, p_3)$ and $w_i \notin \triangle(p_1, p_2, p_3)$, p_k is strictly inside $\triangle(w_i, p_i, p_j)$; hence, strictly inside $segm(p_i\widehat{c_{ij}}p_j)$, and the lemma follows.

2. $w_i \notin arc(p_i\widehat{c_{ij}}p_j)$ and $w_i \notin arc(p_i\widehat{c_{ik}}p_k)$ (refer to Figure 6.7, with $i = 1$ and $j = 2$). By Corollary 6.2.1, $p_i\widehat{w}_ip_k = p_i\widehat{w}_ip_j = 60°$. Therefore, $p_j\widehat{w}_ip_k = 120°$ and, since the triangles associated to $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$ do not intersect $\triangle(p_1, p_2, p_3)$, $p_i \tilde{\in} \triangle(w_i, p_j, p_k)$; hence, strictly inside $segm(p_j\widehat{c_{jk}}p_k)$, and the lemma follows.

□

The last case we have to analyze, is when $w_i = \emptyset$.

Lemma 6.4.3. *If $w_i = \emptyset$, then $p_j\widehat{p_i}p_k = 120°$.*

Proof. By definition, w_i is the intersection between $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$, such that $w_i \notin \{p_1, p_2, p_3\}$. Hence, if $w_i = \emptyset$ then $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$ intersect only in p_i; that is, $\mathcal{C}_{ij}$ and $\mathcal{C}_{ik}$ are tangent in p_i (see Figure 6.8, with $i = 1$). Moreover, $[p_i c_{ij})$ is the bisector of an angle of 60° (namely, one of the angles of the triangle associated to $\mathcal{C}_{ij}$); hence, $p_j\widehat{p_i}c_{ij} = 30°$. Similarly, $p_k\widehat{p_i}c_{ik} = 30°$. Therefore, $p_j\widehat{p_i}p_k = 180° - p_j\widehat{p_i}c_{ij} - p_k\widehat{p_i}c_{ik} = 180° - 60° = 120°$, and the lemma follows. □

Theorem 6.4.2. *Given three distinct points in the plane p_1, p_2, and p_3, their center of equiangularity exists if and only if $p_2\widehat{p_1}p_3 < 120°$, $p_1\widehat{p_3}p_2 < 120°$, and $p_3\widehat{p_2}p_1 < 120°$.*

Proof. We prove separately the two parts.

- ⇒ Let c_e the center of equiangularity of p_1, p_2 and p_3. By Theorem 6.4.1 c_e is inside $\triangle(p_1, p_2, p_3)$ (refer to Figure 6.6). By definition of c_e, $p_1\widehat{c_e}p_2 = p_1\widehat{c_e}p_3 = p_2\widehat{c_e}p_3 = 120°$; hence, $c_e\widehat{p_1}p_2 < 60°$ and $c_e\widehat{p_1}p_3 < 60°$. Therefore, $c_e\widehat{p_1}p_2 + c_e\widehat{p_1}p_3 = p_2\widehat{p_1}p_3 < 120°$. Similarly, it can be proven that $p_1\widehat{p_3}p_2 < 120°$, and $p_3\widehat{p_2}p_1 < 120°$.
- ⇐ Let us assume that $\triangle(p_1, p_2, p_3)$ has no angle bigger than or equal to 120°. We distinguish three cases.
  1. If $w_i \neq \emptyset$ and $w_i \tilde{\in} \triangle(p_1, p_2, p_3)$, then by Lemma 6.4.1 w_i is the center of equiangularity, and the theorem follows.
  2. If $w_i \neq \emptyset$ and $w_i \notin \triangle(p_1, p_2, p_3)$, then by Lemma 6.4.2 there exists i' such that $p_{i'} \tilde{\in} segm(p_{j'}\widehat{c_{j'k'}}p_{k'})$, with $1 \leq i' \neq j' \neq k' \leq 3$ and $i' \neq k'$. Without loss of generality, let us assume $i' = 1$. Then, $p_1 \tilde{\in} segm(p_2\widehat{c_{23}}p_3)$ (refer to Figure 6.7). By Corollary 6.4.1, $p_2\widehat{p_1}p_2 > 120°$, having a contradiction.
  3. If $w_i = \emptyset$, then by Lemma 6.4.3 $p_j\widehat{p_i}p_k = 120°$, having a contradiction.

  □

6.4.1 The Algorithm

Now, based on the geometrical properties showed, we are ready to describe the complete algorithm that solves the gathering problem for three robots that are able to distinguish multiplicity (Algorithm 5). Recall that by Theorem 3.4.2, such a capability is necessary in order for the robots to correctly solve the problem.

In order to prove the correctness of Algorithm 5, we note that, given 3 arbitrary distinct points in the plane, the following are the only possible configurations:

Algorithm 5 Gathering With Unlimited Visibility, $n = 3$

If There Is One Point q With Multiplicity > 1 **Then** `Move(`q`)`.
If All The Robots Are On The Same Line **Then**
  $q :=$ Position Occupied By The Median Robot On The Line;
  If I Am Not The Median Robot **Then** `Move(`q`)`.
5: **Else** `do_nothing()`.
If $\exists r_i | r_j \widehat{r_i} r_k \geq 120°$, $1 \leq i \neq j \neq k \leq 3$ And $i \neq k$ **Then**
  If I Am r_i **Then** `do_nothing()`.
  Else
    $q :=$ Position Occupied By r_i;
10: `Move(`q`)`.
$c_e = w_1 = w_2 = w_3 :=$ Center Of Equiangularity Of r_1, r_2, r_3;
`Move(`c_e`)`.

(I) they can either be all on the same line (Figure 6.9.a); or

(II) there is no angle in $\triangle(p_1, p_2, p_3)$ greater than or equal to 120°; or

(III) there exists r_i such that $r_j \widehat{r_i} r_k \geq 120°$.

The following lemmas deal with scenarios (I)–(III).

Lemma 6.4.4. *According to the algorithm, if the robots are at the beginning on the same line, they will gather on the position occupied by the median robot on the line in a finite number of cycles, by executing Algorithm 5.*

Proof. The median robot, say r, is never allowed to move, while the other two move towards the position occupied at the beginning by r, say q. During their movements, the three robots are always on the same line. Hence, in a finite number of cycles, either both robots reach simultaneously q, or there exists only one point in the plane with multiplicity greater than one, namely q. In the first case, the lemma follows. In the second case, the robot not on q is the only robot allowed to move, and it moves towards q. Hence, in a finite number of cycles, all the robots are on q and the lemma follows. □

Let q_1, q_2 and q_3 be the positions of r_1, r_2 and r_3 at the beginning, respectively.

Lemma 6.4.5. *If at the beginning there exists no r_i such that $r_j \widehat{r_i} r_k \geq 120°$, with $1 \leq i \neq j \neq k \leq 3$ and $i \neq k$, then the robots gather in a finite number of cycles in the center of equiangularity of q_1, q_2 and q_3, by executing Algorithm 5.*

Proof. By Theorem 6.4.2 and Lemma 6.4.1, w_1 is the center of equiangularity of q_1, q_2 and q_3. According to the algorithm, the three robots are allowed to move

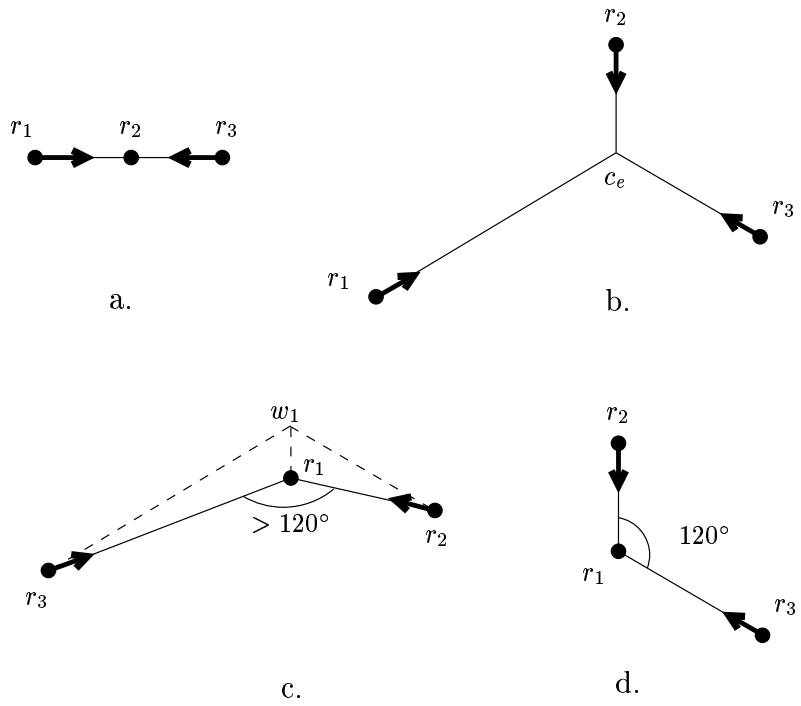


Figure 6.9: The gathering algorithm with three robots. In (a) is depicted the case when the robots are on the same line: the external robots move towards the median robot on the line. In (b) the case when there exists a center of equiangularity c_e: all the robots move towards it (scenario II). In (c), $r_3\widehat{r_1}r_2 > 120°$, while in (d) $r_3\widehat{r_1}r_2 = 120°$ (scenario III).

towards w_1. By Observation 6.4.1, during their movements and as long as no robot reaches w_1, w_1 is always the center of equiangularity of the positions occupied by the three robots. In a finite number of movements, at least one robot reaches w_1. We distinguish two cases.

1. If more than one robot reaches w_1 simultaneously, then w_1 is the only point on the plane with multiplicity greater than one; hence, according to the algorithm, the robot not on it (if any) will keep computing w_1 as its destination point, and the lemma follows.

2. If only one robot reaches first w_1, say r_1 at time t, then at t we have that $r_2\widehat{r_1}r_3 = 120°$. According to the algorithm, r_1 is not allowed to move any more, while the other two robots keep computing w_1 (the position occupied by r_1 after t) as destination point. Hence, in a finite number of cycles, at least one other robot will reach w_1, and the lemma follows as in the previous case (the proof is similar if r_2 or r_3 reach first w_1).

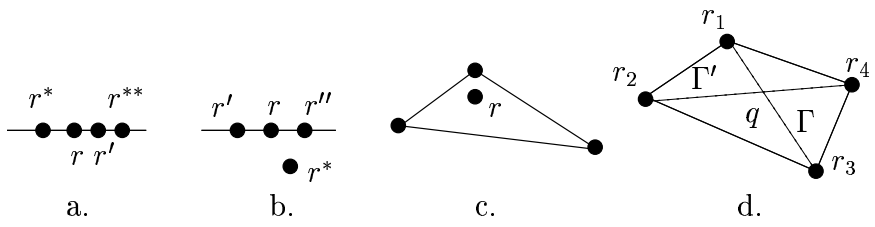


Figure 6.10: The four possible configurations with four points in the plane

□

Lemma 6.4.6. *If at the beginning there exists r_i such that $r_j\widehat{r_i}r_k \geq 120°$, with $1 \leq i \neq j \neq k \leq 3$ and $i \neq k$, then the robots gather in q_i in a finite number of cycles by executing Algorithm 5.*

Proof. According to the algorithm, r_j and r_k move towards the position occupied by r_i, and r_i never moves. While r_j and r_k approach r_i, the angle $r_j\widehat{r_i}r_k$ is always $\geq 120°$. As soon as one of the two robots reaches q_i (by Assumptions A1 and A2, in a finite number of cycles), there is only one point in the plane with multiplicity greater than one (q_i). Therefore, in a finite number of cycles, the robots gather in q_i. □

Finally, by Lemmas 6.4.4–6.4.6, we have

Theorem 6.4.3. *Algorithm 5 is a deterministic and oblivious algorithm that lets three robots that can detect multiplicity and that at the beginning occupy distinct positions on the plane, gather in a point in a finite number of cycles.*

Result 8. *Three robots that can detect multiplicity can always gather in a point in a finite number of cycles.*

6.5 Case c.: $n = 4$

In this section we give an algorithm that lets 4 robots r_1, r_2, r_3, and r_4 gather in a point. Given 4 arbitrary distinct points in the plane, the following are the only possible configurations:

(I) they can either be all on the same line (see Figure 6.10.a); or

(II) three of them can be on the same line (see Figure 6.10.b) ; or

(III) one of them can be strictly inside their convex hull (see Figure 6.10.c); or

(IV) they are the vertices of a convex quadrilater (i.e., none of them is inside their convex hull, see Figure 6.10.d).

Similarly to the case $n = 3$, also in this case the algorithm takes distinct actions according to the configuration of the robots at the beginning (Algorithm 6). In particular, if the robots are all on the same line, the two middle robots meet in the center of SEC. If three of them are on the same line, all the robots meet on the position occupied by the central robot on that line. If at the beginning the robots are in scenario (III), then all the robots meet in the position occupied by the robots inside the convex hull. Finally, if they form a convex quadrilater, they meet in the intersections of the two diagonals of the quadrilater.

Algorithm 6 Gathering With Unlimited Visibility, $n = 4$

If There Is One Point q With Multiplicity > 1 **Then** `Move(`q`)`.
$SEC := SEC(r_1, r_2, r_3, r_4)$;
$c :=$ Center Of SEC;
If All The Robots Are On The Same Line **Then**
    $(r, r') :=$ The Two Robots Inside SEC;
    If I Am r or r' **Then** `Move(`c`)`.
    Else `do_nothing()`.
If Three Robots Are On The Same Line **Then**
    $(r, r', r'') :=$ Robots On The Same Line, With r' Between r and r'';
    If I Am r' **Then** `do_nothing()`.
    Else
        $q :=$ Position Occupied By r';
        `Move(`q`)`.
$CH :=$ `Convex_Hull(`r_1, r_2, r_3, r_4`)`;
If One Robot Is Strictly Inside CH **Then**
    $r :=$ Robot Inside CH;
    If I Am r **Then** `do_nothing()`.
    Else
        $q :=$ Position Occupied By r;
        `Move(`q`)`.
If No Robot Is Inside CH **Then**
    $q :=$ Intersection Of The Two Diagonals Of CH;
    `Move(`q`)`.

In Algorithm 6, routines `Move(`l`)` and `Convex_Hull()` are called. The first one moves the calling robots r towards the point l in the plane as follows: if r is already on l, it does not move (in this case, is as r executes `do_nothing()`). If no robot is on the line $[rl]$, then r moves towards l. Otherwise, let r' be the robot on $[rl]$ closest to

r; then, r can reach at most a point at distance $d > 0$ from r'. In this way, collisions are avoided.

Routine `Convex_Hull`(r_1, r_2, r_3, r_4) returns the convex hull of the points occupied by r_1, r_2, r_3, and r_4.

In the following we will prove that Algorithm 6 is correct and lets the robots gather in a point in a finite number of cycles by distinguishing the four possible configurations the robots can be in at the beginning. In the following, SEC denotes the smallest circle enclosing the robots at the beginning.

Lemma 6.5.1. *If the robots are at the beginning in configuration (I), they will gather in the center of SEC in a finite number of moves.*

Proof. By definition of SEC, two robots are on the SEC, say r^* and r^{**}, and the other two are inside it, say r and r'; furthermore, r^* and r^{**} are opposite (refer to Figure 6.10.a). The algorithm allows only r and r' to move towards the center c of the SEC, while the robots on the SEC can only compute *null movements*. While r and r' move, r^* and r^{**} do not move; therefore SEC, hence c, can not change. By the way routine `Move()` has been defined, it follows that in a finite number of cycles, r and r' meet in c (if one of them reaches first c, it will keep computing c as its destination point).

At this point, there is only one position in the plane with multiplicity greater than one; namely the position occupied by both r and r', say q. The algorithm allows r^* and r^{**} move towards q, and the lemma follows. □

The following lemma handles the case when the robots start from configuration (II). Let r, r' and r'' be the three points that at the beginning lie on the same line Γ, with r' between r and r''. Moreover, let r^* be the fourth robot (refer to Figure 6.10.b).

Lemma 6.5.2. *If the robots are at the beginning in configuration (II), they will gather on the position occupied by r', say q, in a finite number of moves.*

Proof. The algorithm never allows r' to move; hence r' is already at the beginning in the meeting point. Furthermore, all the other robots compute as destination point q, and move towards it. While r and r'' move towards r', they stay on Γ, while r^* is not. Therefore, as long as no robots reaches r', the robots are in configuration (II), r' computes only *null movements*, and the others compute q as destination point.

In a finite number of cycles, at least one robot among r, r'' and r^*, reaches r'. At this point, there is only one position in the plane with multiplicity greater than one, namely q. The destination point that the robots not on q compute is still q; hence, in a finite number of cycles, all the robots meet in q and the lemma follows. □

The following lemma handles scenario (III). Let r be the only robot inside the convex hull of four robots' positions (refer to Figure 6.10.c).

Lemma 6.5.3. *If the robots are at the beginning in configuration (III), they will gather on the position occupied by r, say q, in a finite number of moves.*

Proof. The algorithm never allows r to move; hence r is already at the beginning in the meeting point. Furthermore, all the other robots compute as destination point q, and move towards it. While they move towards q, r stays always strictly inside the convex hull of the robots' positions. Therefore, as long as no robots reaches r, the robots are in configuration (III), r compute only *null movements*, and the others compute q as destination point.

In a finite number of cycles, at least one robot reaches r'. At this point, there is only one position in the plane with multiplicity greater than one, namely q. The destination point that the robots not on q compute is still q; hence, in a finite number of cycles, all the robots meet in q and the lemma follows. □

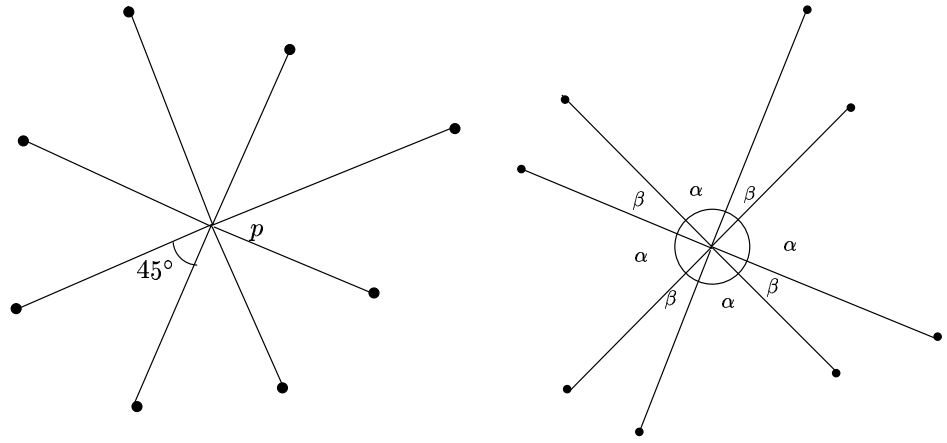
Let the four robots form at the beginning a convex quadrilater, and let us assume, without loss of generality, that r_1 and r_3 are the end points of one diagonal of the quadrilater, and r_2 and r_4 the end points of the other diagonal. Moreover, let Γ be the line passing through r_1 and r_3, Γ' be the line passing through r_2 and r_4, and q be the intersection of the two diagonals (refer to Figure 6.10.d).

Lemma 6.5.4. *If the robots are at the beginning in configuration (IV), they will gather on q in a finite number of moves.*

Proof. According to the algorithm, all the robots compute q as destination point, and start moving towards it. During their movements, they never leave Γ and Γ'; hence, as long as no robot reaches q, they always form a convex quadrilater whose diagonals intersect in q.

In a finite number of cycles, at least one robot reaches q. If more than one robot reaches q simultaneously, then there is only one position in the plane with multiplicity greater than one, namely q. The destination point that the robots not on q compute is still q; hence, in a finite number of cycles, all the robots meet in q and the lemma follows.

If only one robot reaches first q, say r, then the robots are in configuration (II). At this point, the algorithm does not allow r to move again, while all the other robots still compute q as destination point. Hence, in a finite number of steps at least another robot reaches q. When this happens, there is only one position in the plane with multiplicity greater than one, namely q. The destination point that the robots not on q compute is still q; hence, in a finite number of cycles, all the robots meet in q and the lemma follows. □



a. b.

Figure 6.11: (a) An equiangular configuration with 8 points. (b) A biangular configuration with 8 points.

Finally, by Lemmas 6.5.1–6.5.4, we can state the following

Theorem 6.5.1. *Algorithm 6 is a deterministic and oblivious algorithm that lets four robots that can detect multiplicity and that at the beginning occupy distinct positions on the plane, gather in a point in a finite number of cycles.*

Result 9. *Four robots that can detect multiplicity can always gather in a point in a finite number of cycles.*

6.6 Case d.: $n \geq 5$

In this section, we will describe an algorithm that solves the problem when there are more than 4 robots in the system. Unfortunately, our strategy works *almost* always. In particular, it does not work if the robots at the beginning are in an *equiangular* or *biangular* configuration. Informally speaking, n distinct points on the plane are in a *equiangular* configuration if there exists a point p (the *center*) and an ordering of the n points such that each adjacent two points form exactly an angle of $\frac{360°}{n}$ with p. They are in a *biangular* configuration, if there exist a point p (the *center*), an ordering of the points, and two angles $\alpha, \beta > 0$ such that each two adjacent points form α or β with p, and the angles alternate (see Figure 6.11). We need to distinguish these two cases for the following reason. If the robots are at the beginning the vertices of a regular n-gone, there is no way to elect a robot to move: all the robots will take the same decision, hence move. In this two cases, it can be proven that the centers do not change while the robots approach them. Hence, it

is necessary to have a way to realize when, given n points arbitrarily placed in the plane, they are in an equiangular or biangular configuration, so that the robots can move towards the centers, and gather there. Indeed, we have a strategy that allows us to understand whether the centers exist or not (it is an extension of the strategy used in Section 6.4). Until now, however, we were not able to merge these cases with the general case when there exists no center of equiangularity or biangularity for the robots at the beginning. In fact, it can happen that at the beginning the robots are neither in an equiangular nor in a biangular configuration. Hence, they start executing the general algorithm. Unfortunately, it can happen that, while they move, they end up in one of these two special configurations; since the robots act completely asynchronously, some of them could realize before others this change of scenario, and apply the algorithm that deal with the two special cases. In other words, not all the robots would agree on the configuration they are in, hence they would not take *coherent* actions, and the algorithm never converge.

Hence, in this section we describe the algorithm that solves the problem only when the robots are not in an equiangular or biangular configuration at the beginning.

6.6.1 Basic Definitions on Strings

In this section, we introduce some basic notations on strings.

Definition 6.6.1 (String). A string s is a tuple $< s_0, \ldots, s_{n-1} >$, where s_i are called the *characters* of s; the $i-th$ character in s will be denoted also by $s[i]$. ◁▷

For our purposes, we assume that the characters of a string are real numbers. Given a string s, its length is denoted by $|s|$, and $s[i..j] =< s[i], \ldots, s[j] >$, with $i \leq j$. Given two strings s and s', we denote the concatenation of s and s' by $s \circ s' =< s_0, \ldots, s_{n-1}, {s'}_0, \ldots, {s'}_{n-1} >$.

Given two strings s and s', with $|s| = |s'|$, we say they are *lexicographically equal*, denoted by $s = s'$, if $s[i] = s[i']$, for all $0 \leq i \leq n-1$. We say s is *lexicographically smaller* than s', denoted by $s < s'$, if there exists an index j, $0 \leq j \leq n-1$, such that $s[i] = s'[i]$, for all $i < j$, and $s[j] < s'[j]$. Finally, the *minimum* between them is defined as follows:

$$\min(s, s') = \begin{cases} s & \text{if } s < s' \text{ or } s = s' \\ s' & \text{if } s' < s. \end{cases}$$

Definition 6.6.2 (Period). We say that a string p is a period of a string s (or that s is periodic of period p), with $|s| \geq |p|$, if $|s| \bmod |p| = 0$ and there exists an index $0 \leq f \leq |p| - 1$ such that $s[(f+i) \bmod |s|] = p[i \bmod |p|]$, for all $0 \leq i \leq |s| - 1$. We say that s is *perfectly* periodic of period p, if $f = 0$.

The substrings $s[(f + i|p|) \bmod |s|..(f + i|p| + |p| - 1) \bmod |s|]$, for all $0 \leq i \leq |s|/|p| - 1$ are called the *occurrences* of p in s. ◁▷

For instance, $s =< a, b, c, d, a, b, c, d, a, b, c, d >$ is perfectly periodic of period $p =< a, b, c, d >$; and $s' =< c, d, a, b, c, d, a, b, c, d, a, b >$ is periodic of period p, with $f = 2$. From the definition of periodic string, it follows that

Lemma 6.6.1. *Given a string $s[0..k-1]$ and a positive integer $x \leq k$ such that $s[(f + i) \bmod k] = s[(f + i + x) \bmod k]$, for some $0 \leq f \leq x - 1$ and for all $0 \leq i \leq k - 1$, then s is periodic of period $p = s[f..(f + x - 1) \bmod k]$. Furthermore, there are k/x occurrences of p in s.*

In s' of the above example, $f = 2$ and $x = 4$.

6.6.2 The String of Angles SA and the Set S

Now, we are ready to combine the definition of angles and strings, in order to obtain the definition of a *string of angles*.

Definition 6.6.3 (String of Angles SA). Given n points $l_1, \ldots, l_n$ on the plane arbitrarily ordered, another point c distinct from the others, and an orientation (clockwise or counterclockwise), we define the *string of angles* of the points $l_1, \ldots, l_n$ with respect to c, as the string $SA(l_1, \ldots, l_n, c) = < \alpha_0, \ldots, \alpha_{n-1} >$ (shortly SA, if no ambiguity arises), where $\alpha_{i-1} = {}_c\sphericalangle^{l_i}_{(l_{i+1} \bmod (n+1))+1}$, $1 \leq i \leq n$. Moreover, we denote by rSA the reverse of SA, and by $SA[i]$ ($rSA[i]$) the $i-th$ character in SA (rSA). ◁▷

An example of string of angles with respect to the center of $SEC(r_1, \ldots, r_n)$ is depicted in Figure 6.13.

Definition 6.6.4 (Valid String of Angles). A String of Angles is valid if all its characters are not zero. ◁▷

The basic idea of the algorithm is that in every cycle each robot computes the smallest circle enclosing the robots' positions, and its center c; it then gives an (arbitrary) orientation to the circle, hence it sorts all the other robots' positions according to this orientation of the circle and to their distances from c; let $r_1, \ldots, r_n$ be the sorted sequence of robots. At this point, it computes the string of angles $SA(r_1, \ldots, r_n, c)$ with respect to c. Finally, it decides its next destination point according to the form of SA.

In the following we will assume that the robots occupy distinct positions in the plane, that SEC is always computed with respect to the robots' positions retrieved in

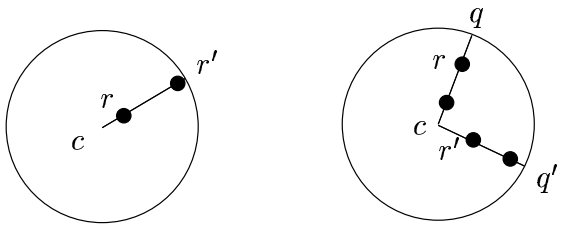


Figure 6.12: Routine `Succ()` in `Compute_SA()`. The circle is oriented clockwise.

the last *Look* state, and SA computed with respect to the center of the smallest circle enclosing all the robots and with respect to a particular ordering of the positions of the robots. More formally, SA is computed according to the following routine

`Compute_SA(`SEC`)`
  $SA := \emptyset$;
  $r :=$ `Choose_Robot()`;
  $Orientation :=$ `Orient(`SEC, r`)`;
  $i = 0$;
  While $i \leq n$ **Do**
    $r' :=$ `Succ(`$r, Orientation$`)`;
    $SA[i] := {}_c\!\sphericalangle^r_{r'}$;
    $r := r'$;
    $i := i + 1$;
  End While
  Return SA.

First, a robot is arbitrarily chosen by calling `Choose_Robot()`, and an arbitrary orientation of SEC is chosen by `Orient(`SEC, r`)`. For instance, in the example depicted in Figure 6.13, `Choose_Robot()` returned r_1, and the chosen $Orientation$ is clockwise. Then, the successor $r' \neq r$ of r returned by `Succ(`$r, Orientation$`)` is

- either the robot r' on $\Psi(r)$ such that $dist(c, r') > dist(c, r)$ (if such an r' exists, see Figure 6.12.a); or

- the robot r' such that, according to $Orientation$, there is no robot inside $sector(q, q')$, and there is no robot between c and r' on $[cr']$, where q is the intersection between SEC and $\Psi(r)$, and q' is the intersection between SEC and $\Psi(r')$ (see Figure 6.12.b).

It is easy to show the following

Lemma 6.6.2. *If SA is computed according to routine* `Compute_SA(`SEC`)` *and two robots are on the same radius of SEC, then SA is not valid.*

Proof. Let us assume that r and r' are such that $r' \in \Psi(r)$ and that, without loss of generality, r' is after r in the ordering established by `Compute_SA(`SEC`)`. If $r' =$`Succ(`$r, Orientation$`)`, then ${}_c\triangleleft^r_{r'} = 0$, and the lemma follows. Otherwise, there exists at least one robot between r and r' on $[rr']$. Let r'' be the closest robot to r, with $r'' \in [rr']$. Then, $r'' =$`Succ(`$r, Orientation$`)`; hence ${}_c\triangleleft^r_{r''} = 0$, and the lemma follows. □

Given SA, we will call the strings $SA[f], SA[f+1], \ldots, SA[n-1], SA[0], \ldots, SA[f-1]$, for all $0 \leq f \leq n-1$, the strings *within* SA, denoted by $\mathcal{W}SA(f)$; we will say that $\mathcal{W}SA(f)$ *starts* in SA from f. Similarly, we can define the strings within rSA, denoted by $\mathcal{W}rSA(f)$. Furthermore, we can define

Definition 6.6.5 (lmin(SA), lmin(rSA)). The *lexicographically minimum string within SA*, shortly $lmin(SA)$, is the string s within SA such that $s = \min(s, \mathcal{W}SA(f))$, for all $0 \leq f \leq n-1$ (an analogous definition holds for rSA). ◁▷

For instance, if $SA =< \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma, \alpha >$ is the string of angles depicted in Figure 6.13, then $lmin(SA)$ is $s =< \alpha, \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma >$, given that $\alpha < \beta < \gamma$ (and $lmin(rSA)$ is $s' =< \alpha, \alpha, \gamma, \beta, \alpha, \alpha, \gamma, \beta >$).

According to `Compute_SA(`SEC`)`, each index $0 \leq i \leq n-1$ can be *associated* to one robot r and to one point on the SEC. In fact, if $SA[i] = \alpha = {}_c\triangleleft^r_{r'}$, i can be associated to robot r. We denote the robot associated to the position i in[2] SA (resp. rSA) by $\mathfrak{r}(i)$ (when it is clear wheter i refers to a position in SA or in rSA, we will refer to i simply as the position associated to $\mathfrak{r}(i)$)[3]. Viceversa, it follows from the above definition that each robot can be associated to one position in SA (resp. rSA): namely, r is associated to the position i in SA (resp. rSA) such that $r = \mathfrak{r}(i)$ (we are assuming that the robots' positions are distinct). We say that a robot r *is in* s, with $s = SA[i..j]$ (resp. $s = rSA[i..j]$), if there exists a position $i \leq k \leq j$ such that r is associated to k in SA (resp. rSA).

Furthermore, we denote by $\mathfrak{l}(i)$ the intersection between SEC and $\Psi(\mathfrak{r}(i))$. We will call $\mathfrak{l}(i)$ *the point on the SEC associated to i*. Note that $\mathfrak{l}(i)$ is the point from where $arc(\alpha)$ starts, according to the orientation of SA (resp. rSA).

[2] i.e., to the $i-th$ position in SA.

[3] Note that, if i is a position in SA, then $\mathfrak{r}(i)$ is computed according to SA and its orientation; similarly, if i is a position in rSA, then $\mathfrak{r}(i)$ is computed according to rSA and its orientation.

Note 6.6.1. Given an index w in SA, by definition of rSA it follows that $rSA[|SA| - w - 1] = SA[w]$. That is,

$$rSA[i] = SA[n-1-i], \quad \forall\ 0 \leq i \leq n-1. \tag{6.1}$$

We call $w' = |SA| - w - 1$ the index (or position) in rSA corresponding to w. Furthermore, $\mathfrak{r}(w) = \mathfrak{r}(w'+1)$. ⋆

For instance, in Figure 6.13 the second position in SA, $SA[1] = \beta = {}_c\!\lhd^{r_2}_{r_3}$, is associated to r_2 ($\mathfrak{r}(1) = r_2$) and to l_2 ($\mathfrak{l}(1) = l_2$). Furthermore, the index in rSA associated to $w = 1$ is $w' = 6$ ($SA[1] = rSA[6]$), and r_2 is associated to ${}_c\!\lhd^{r_2}_{r_3}$ in SA and to ${}_c\!\lhd^{r_2}_{r_1}$ in rSA ($\mathfrak{r}(w) = \mathfrak{r}(1) = r_2 = \mathfrak{r}(w'+1) = \mathfrak{r}(7)$.

Note 6.6.2. Note that, by definition of `Compute_SA(`SEC`)`, there cannot exist two different positions in SA associated to the same robot r, while it is possible that two different positions are associated to the same point on the SEC (this happens if SA is not valid). ⋆

Let $Lex\_Min\_String = \min(lmin(SA), lmin(rSA))$. It is possible that there are several positions in SA (resp. rSA) from where Lex_Min_String starts. In the example above, $Lex\_Min\_String = lmin(SA)$, and its starting positions within SA are the third and the seventh.

Definition 6.6.6 (pos(SA), pos(rSA)). $pos(SA)$ (resp. $pos(rSA)$) is a set of tuples $h^j_{SA} =< On\_SA, \texttt{SA} >$ (resp. $h^j_{rSA} =< On\_rSA, \texttt{rSA} >$), $j \geq 1$, where On_SA (resp. On_rSA) represents a position in SA (resp. rSA) from where Lex_Min_String starts; that is, $Lex\_Min\_String = \mathcal{W}SA(On\_SA)$ (resp. $Lex\_Min\_String = \mathcal{W}rSA(On\_rSA)$). `SA` (resp. `rSA`) stores the information on the orientation of SA (resp. rSA). It is forbidden in $pos(SA)$ (resp. $pos(rSA)$) to have two distinct tuples with the same value of On_SA (resp. On_rSA). ◁▷

We denote by $h^j_{SA}.On\_SA$ (resp. $h^j_{rSA}.On\_rSA$) the first field of h^j_{SA} (resp. h^j_{rSA}). Furthermore, we assume there are no two different tuples in $pos(SA)$ (resp. $pos(rSA)$) with the same first field; that is $h^j_{SA}.On\_SA \neq h^i_{SA}.On\_SA$, for all $1 \leq i \neq j \leq |pos(SA)|$ (the same clearly holds for $pos(rSA)$).

Note that, if $Lex\_Min\_String = lmin(SA) \wedge Lex\_Min\_String \neq lmin(rSA)$, then $pos(rSA) = \emptyset$, and viceversa. For instance, in the example started above and referring to Figure 6.13, we have that $pos(SA) = \{< 3, \texttt{SA} >, < 7, \texttt{SA} >\}$, while $pos(rSA) = \emptyset$.

Now we are ready to define when a string of angles is equiangular or biangular.

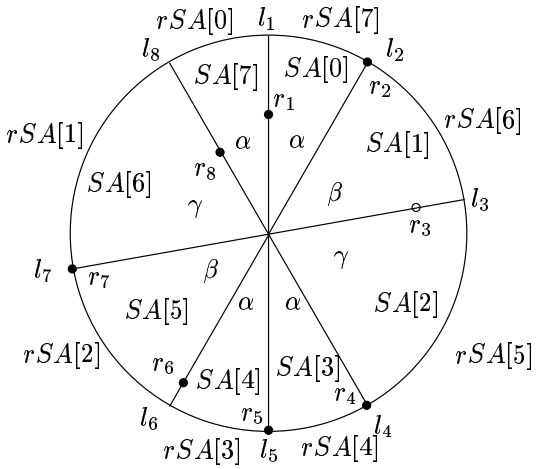


Figure 6.13: Assuming $SEC(r_1, \ldots, r_n)$ oriented clockwise, and that r_1 is the robot chosen in `Compute_SA(`SEC`)`, we have that $SA =< \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma, \alpha >$; $lmin(SA) =< \alpha, \alpha, \beta, \gamma, \alpha, \alpha, \beta, \gamma >$; $lmin(rSA) =< \alpha, \alpha, \gamma, \beta, \alpha, \alpha, \gamma, \beta >$. Hence, $Lex\_Min\_String = lmin(SA)$, $pos(SA) = \{< 3, \texttt{SA} >, < 7, \texttt{SA} >\}$, and $S = pos(SA)$. In this case SA is periodic of a period that starts in l_4 and ends in l_8, that is $p = \alpha, \alpha, \beta, \gamma$ (Lemma 6.7.1).

Definition 6.6.7. A string of angles $SA =< \alpha_0, \ldots, \alpha_{n-1} >$ is *equiangular* if $\alpha_i = 360°/n$, for all $0 \leq i \leq n-1$.

SA is *biangular* if n is even, $\alpha_0 \neq \alpha_1$, $\alpha_i = \alpha_{i+2}$, for all $0 \leq i \leq n-3$, and $\alpha_0 + \alpha_1 = 2 \cdot 360°/n$. ◁▷

Finally, we define

Definition 6.6.8 (The Set S). Given SA, rSA, Lex_Min_String, $pos(SA)$ and $pos(rSA)$, $S = pos(SA) \cup pos(rSA)$. Therefore, S is a set of 2-tuples $t_i =< pos, orient >$, $i \geq 1$, where pos represents a position in either SA or rSA, and $orient$ is either `SA` or `rSA`. ◁▷

In the following we will denote by $t_i.pos$ and $t_i.orient$ the two fields in a tuple $t_i \in S$. In our example, $S = pos(SA) = \{< 3, \texttt{SA} >, < 7, \texttt{SA} >\}$. Given a tuple $t \in S$, $\mathfrak{r}(t.pos)$ is called the robots associated to t. In the following, when no ambiguity arises, we will omit `SA` (resp. `rSA`) from the tuples in $pos(SA)$ (resp. $pos(rSA)$). Moreover, we will denote by $s(h^j_{SA})$ the string within SA that starts from $h^j_{SA}.On\_SA$, by $s(h^j_{rSA}) = \mathcal{W}rSA[h^j_{rSA}.On\_rSA]$, and by

$$s(t_i.pos) = \begin{cases} s(h^j_{SA}) & \text{if } t_i.orient = \texttt{SA} \\ s(h^j_{rSA}) & \text{if } t_i.orient = \texttt{rSA}, \end{cases}$$

with $t_i \in S$.

Note 6.6.3. If $pos(SA) \neq \emptyset$, by definition of S, Lex_Min_String starts in SA from all $t_i.pos$ such that $t_i.orient = \texttt{SA}$, with $t_i \in S$; hence from all the $h^j_{SA}.On\_SA$, with $h^j \in pos(SA)$. Therefore $s(h^j_{SA}) = s(h^{j+1}_{SA})$, for all $1 \leq j \leq |pos(SA)| - 1$. If $pos(rSA) \neq \emptyset$, this holds symmetrically also for rSA and $pos(rSA)$. $\star$

6.7 Properties of SA

In this section, we will show some properties of the string of angles SA as computed by routine `Compute_SA(`SEC`)`, that will be useful in the following.

Lemma 6.7.1. *Let SA be valid and not equiangular. The following statements are true:*

1. *Let $|pos(SA)| \geq 1$. If $|pos(SA)| = 1$, then let $p = s(h^1_{SA})$; otherwise, let $p = SA[h^1_{SA}.On\_SA..h^2_{SA}.On\_SA - 1]$, and $|p|$ its length. SA is periodic of period p, Lex_Min_String is perfectly periodic of period p, and there are $|pos(SA)| = |SA|/|p|$ occurrences of p in SA. We call p the* period induced by $pos(SA)$ *on SA. Moreover $|pos(SA)| \leq n/3$. The same holds for rSA.*

2. *If $pos(SA) \neq \emptyset \wedge pos(rSA) \neq \emptyset$, then $|pos(SA)| = |pos(rSA)|$. Moreover, the period induced by $pos(SA)$ on SA and by $pos(rSA)$ on rSA are the same.*

3. *If $pos(SA) \neq \emptyset \wedge pos(rSA) \neq \emptyset$, there exists no i and j, $1 \leq i, j \leq |pos(SA)|$ such that $h^j_{rSA}.On\_rSA = |SA| - h^i_{SA}.On\_SA$.*

Proof.

1. The first part of the statement follows from Note 6.6.3 and from Lemma 6.6.1. Moreover, since we assume the robots are not in equiangular/biangular positions at the beginning, p must be greater than 3, hence $|pos(SA)| \leq n/3$.

2. If $pos(SA) \neq \emptyset \wedge pos(rSA) \neq \emptyset$, by previous point both SA and rSA are periodic, say of period p and p', respectively. Moreover, also Lex_Min_String is periodic of period p and p', and let us assume by contradiction that $|p| < |p'|$. By definition, in $pos(rSA)$, there are all the positions in rSA from where Lex_Min_String starts; therefore, starting from $rSA[h^1_{rSA}.On\_rSA]$, Lex_Min_String starts every $|p'|$ characters. But since Lex_Min_String is also periodic of period p, we have that Lex_Min_String starts in rSA every $p < p'$ characters, hence $pos(rSA)$ does not store all the starting positions of Lex_Min_String in rSA, having a contradiction. A similar argument applies

if $|p| > |p'|$. Therefore $|p| = |p'|$, hence $|pos(SA)| = |pos(rSA)|$. Finally, since p and p' are both periods of Lex_Min_String, and by definition the first $|p|$ characters of Lex_Min_String must be those of both p and p', we can conclude that p and p' must be the same.

3. By contradiction, let us assume that such i and j exist (for instance, referring to Figure 6.13, it is not possible that $h^1_{SA}.On\_SA =< 6, \texttt{SA} >$, and $h^1_{rSA}.On\_rSA =< 2, \texttt{rSA} >$). Without loss of generality, let us assume that SA is oriented clockwise, and rSA counterclockwise. By definition, $s(h^i_{SA}) = s(h^j_{rSA}) = Lex\_Min\_String$, hence their first character must be the same. Moreover it can not be 0, since SA is valid. Let α be this character, and $s' = s(h^i_{SA}) \setminus \alpha$ the string $s(h^i_{SA})$ without its first character. Let $x = h^j_{rSA}.On\_rSA$ and $y = h^i_{SA}.On\_SA$; by Note 6.6.1, $rSA[x] = SA[n-1-x] = SA[n-1-|SA|+y] = SA[y-1]$. Therefore, $s(x) = s(h^j_{rSA})$ is the string $< SA[y-1] = \alpha, SA[y-2], \ldots, SA[y+1], SA[y] >$, while $s(y) = s(h^i_{SA})$ is the string $< SA[y] = \alpha, SA[y+1], \ldots, SA[y-2], SA[y-1] >$. Hence, since SA is not equiangular, the string $< SA[y-1], SA[y], SA[y+1], \ldots >=< \alpha, \alpha \circ s' >$ is lexicographically smaller than $s(h^i_{SA})$, thus contradicting the hypothesis that the strings starting from the positions in the tuples of $pos(SA)$ are the lexicographically smallest. □

Following the result shown in Point 2. of previous lemma, from now on we will refer to p also as the *period induced by* S, meaning that either $pos(SA) \neq \emptyset$ and p is the period induced by $pos(SA)$ on SA, or that $pos(rSA) \neq \emptyset$ and p is the period induced by $pos(rSA)$ on rSA.

6.8 Properties of S

Similarly to what done in the previous section, we want to highlight some properties of the set S. First, we analyze the case when $|S| = 2$.

Lemma 6.8.1. *Let SA be valid and $|S| = 2$, with $w = t_1.pos$ and $w' = t_2.pos$, all the robots be on the smallest circle enclosing them, not in equiangular/biangular configuration. Moreover, let $r = \mathfrak{r}(w)$ and $r' = \mathfrak{r}(w')$ be adjacent on the SEC, $\alpha = l\widehat{c}l'$, and $\tilde{l}$ and l^* be the opposite points to $l = \mathfrak{l}(w)$ and $l' = \mathfrak{l}(w')$, respectively. Then,*

1. *$t_1.orient \neq t_2.orient$;*

2. *$SA[w+j] = rSA[w'+j]$, for all $0 \leq j \leq |SA| - 1$;*

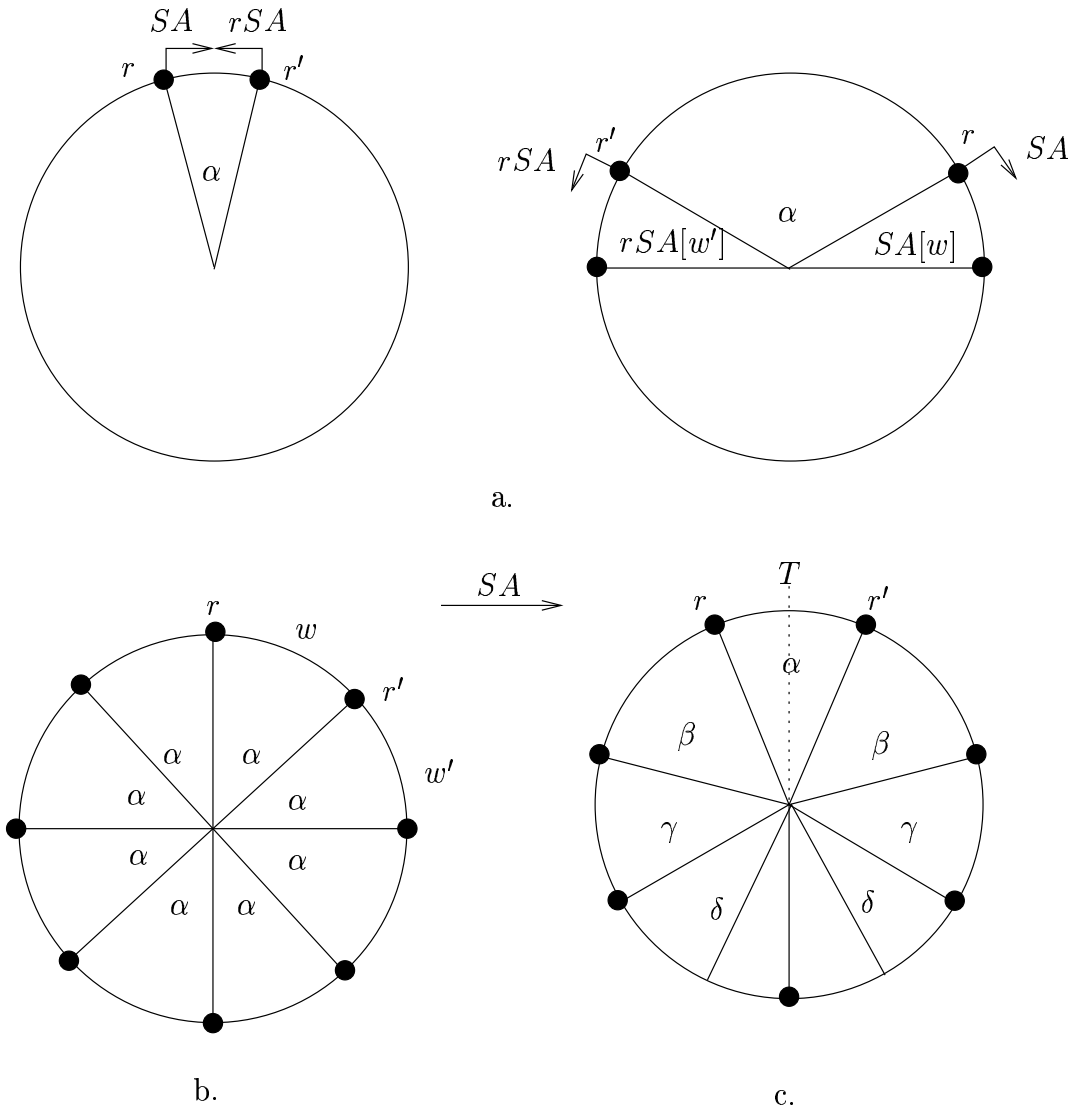


Figure 6.14: (a) The two possible ways in which r and r' can be adjacent on the SEC in Lemma 6.8.1. In the left scenario, $\alpha = SA[w] = rSA[w']$ and $r' = \mathfrak{r}(w+1)$; in the second scenario, $\alpha = SA[w-1] = rSA[w'-1]$ and $r' = \mathfrak{r}(w-1)$. (b) Case 1. of Lemma 6.8.1. In this case $t_1.orient = t_2.orient = \texttt{SA}$. (c) Cases 2. and 3. of Lemma 6.8.1. $t_1.orient \neq t_2.orient$, and $\alpha = SA[w] = rSA[w']$.

3. *Let T be the diameter of SEC that cuts in half α. Given a position $w+i$ in SA, $\wp(w+i)$ is symmetric to $\wp(w'-i)$ with respect to T, for all $0 \leq i \leq |SA|-1$*

Proof. By hypothesis, r and r' are adjacent on the SEC. Two cases can apply, depending on whether r' is after r in the ordering given to the robots by `Compute_SA(`SEC`)`, or viceversa; in other words, whether $r' = \mathfrak{r}(w+1)$ or $r' = \mathfrak{r}(w-1)$ (see Figure 6.14.a). In the following, we will prove the lemma assuming that r' is after r; the other case can be proven similarly.

1. Let us assume by contradiction that $t_1.orient = t_2.orient =$ `SA` (refer to Figure 6.14.b). Since r' is a successor of r, by definition of SA and by Note 6.6.2, $w' = w+1$. Let $s(w) =< SA[w] = \alpha, SA[w+1], \ldots, SA[w-1] >$ and $s(w') =< SA[w'], \ldots, SA[w'-1] >$ be the two lexicographically minimum strings starting respectively from w and w'. Since by definition of S, $s(w') = s(w)$, we have that $s(w')[i] = s(w)[i]$, for all $0 \leq i \leq |SA|-1$; that is $SA[w'+j] = SA[w+j]$, for all $0 \leq j \leq |SA|-1$. Therefore, since $w' = w+1$, $SA[w'+j] = SA[w+1+j] = SA[w+j]$, for all j; hence $s(w) =< \alpha, \alpha, \ldots, \alpha >$, and SA would be equiangular, having a contradiction. A similar argument holds if $t_1.orient = t_2.orient =$ `rSA`.

2. Since by previous Point 1. $t_1.orient \neq t_2.orient$, and r' is after r according to the ordering given by `Compute_SA(`SEC`)` (hence, to the orientation of SA), it follows that r is after r' according to the orientation of rSA; that is, by Note 6.6.2, $SA[w] = rSA[w']$ (see Figure 6.14.a). Hence, since $s(w) =< SA[w] = \alpha, SA[w+1], \ldots, SA[w-1] >$ and $s(w') =< rSA[w'] = \alpha, rSA[w'+1], \ldots, rSA[w'-1] >$ are both lexicographically minimum strings, by definition of S we have that $s(w) = s(w')$. Therefore, $SA[w+j] = rSA[w'+j]$, for all $0 \leq j \leq |SA|-1$.

3. Since T is the bisector of α, and by previous Point 2. $\alpha = SA[w] = rSA[w']$, it follows by Note 6.6.2 that $\alpha = {}_c\!\sphericalangle^{\wp(w)}_{\wp(w')}$ (according to the orientation of SA), hence $\wp(w)$ and $\wp(w')$ are symmetric with respect to T. Let us assume by induction that the statement is true for all $0 \leq i \leq j-1$. Let $w+j$ be the next position in SA, and $\wp(w+j)$ its asociated point on the SEC. By previous Point 2., $SA[w+i] = rSA[w'+i]$, $0 \leq i \leq j-1$; moreover, by inductive hypothesis, $\wp(w+i)$ and $\wp(w'+i)$ are symmetric with respect to T. Therefore, by Note 6.6.2 and since $SA[w+j] = rSA[w'+j]$, we have that T must cut the angle[4] ${}_c\!\sphericalangle^{\wp(w+j)}_{\wp(w'+j)}$ in half; hence $\wp(w+j)$ and $\wp(w'+j)$ are symmetric with respect to T too, and the lemma follows. □

[4]This is true regardless the orientation used to compute the angle.

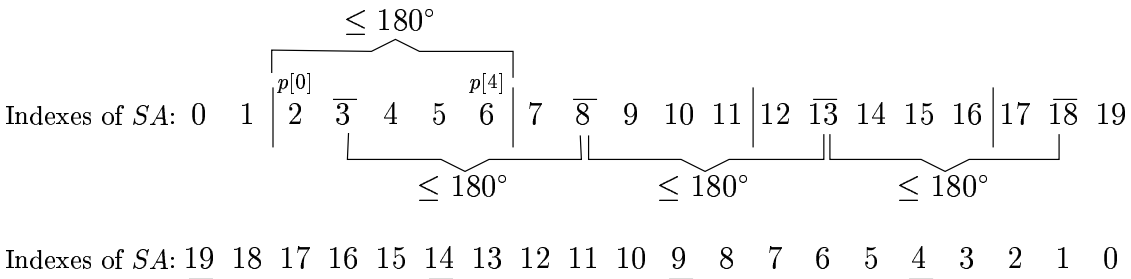


Figure 6.15: Example for Lemma 6.8.2. In this case, $SA : < \alpha_0, \ldots, \alpha_{19} >$, $rSA : < \beta_0, \ldots, \beta_{19} >$, $\beta_i = \alpha_{|SA|-i-1} = \alpha_{19-i}$, $p = < \alpha_2, \ldots, \alpha_6 >$, $\tilde{x} = 3$, $\tilde{y} = 4$, $M(\tilde{x}) = \{\mathfrak{r}(k) | k = 3, 8, 13, 18\}$, $M'(\tilde{y}) = \{\mathfrak{r}(k) | k = 4, 9, 14, 19\}$. The overlined indexes represent robots in $M(\tilde{x})$, while the underlined ones robots in $M'(\tilde{y})$. Note that, at most two robots in each occurrence of p in SA are also in $M(\tilde{x}) \cup M'(\tilde{y})$. Furthermore, $_c\sphericalangle^{\mathfrak{l}(w)}_{\mathfrak{l}(w+|p|)} \leq 180°$, for all $0 \leq w \leq |pos(SA)| - 1$; hence, by removing all the robots in $M(\tilde{x}) \cup M'(\tilde{y})$ no angle bigger than 180° is created.

The following lemma locates a set of robots that can be removed from the circumference of SEC without having SEC changed, when $|S| > 2$.

Lemma 6.8.2. *Let SA be valid and $|S| > 2$, and let all the $n \geq 4$ robots be on the smallest circle enclosing them, not in an equiangular/biangular configuration. Let p be the period induced by S. Moreover, let $M(\tilde{x})$ and $M'(\tilde{y})$, $0 \leq \tilde{x}, \tilde{y} \leq |p| - 1$, be defined as follows:*

$$M(\tilde{x}) = \{\mathfrak{r}(k) | k = \tilde{x} + j|p|, \forall 0 \leq j \leq \frac{|SA|}{|p|} - 1\},$$

if $pos(SA) \neq \emptyset$, otherwise $M(\tilde{x}) = \emptyset$; and

$$M'(\tilde{y}) = \{\mathfrak{r}(k) | k = \tilde{y} + j|p|, \forall 0 \leq j \leq \frac{|rSA|}{|p|} - 1\},$$

if $pos(rSA) \neq \emptyset$, otherwise[5] $M'(\tilde{y}) = \emptyset$. SEC does not change if the robots in $M(\tilde{x}) \cup M'(\tilde{y})$ are removed.

Proof. $M(\tilde{x})$ and $M'(\tilde{y})$ are constituted by the robots associated respectively to the $\tilde{x} - th$ position in each occurrence of p in SA and to the $\tilde{y} - th$ position in each occurrence of p in rSA. Two cases are analyzed.

1. $pos(rSA) \neq \emptyset \wedge pos(SA) \neq \emptyset$. By point 2 of Lemma 6.7.1, $|pos(rSA)| = |pos(SA)| = |S|/2$ (hence $|S| \geq 4$), and SA and rSA are both periodic of

[5]All the operations are modulo $|SA|$ (resp. $|rSA|$). Moreover, note that $M(\tilde{x})$ is computed according to positions in SA, while $M'(\tilde{y})$ according to positions in rSA.

period p; furthermore, each period starts in SA and rSA from one of the positions stored in S. By hypothesis, the robots are not in equiangular or biangular configuration, hence $|p| \geq 3$ and there are at least three robots in each occurrence of p. Let $0 \leq \tilde{x}, \tilde{y} \leq |p| - 1$, $w = \tilde{x} + j|p|$, with $0 \leq j \leq |SA|/|p| - 1$, and $w' = \tilde{y} + j'|p|$, with $0 \leq j' \leq |rSA|/|p| - 1$. Since there are $|pos(SA)| = |SA|/|p| \geq 2$ occurrences of p in SA, we have that (recall that the characters in p are angles):

$$\Sigma_{i=0}^{|p|-1} p[i] = 360°/|pos(SA)| \leq 180°, \tag{6.2}$$

hence ${}_c\!\sphericalangle_{l'}^{l} = \Sigma_{i=0}^{|p|-1} p[i] \leq 180°$, with $l = \mathfrak{l}(w)$ and $l' = \mathfrak{l}(w + |p|)$ (refer to the example depicted in Figure 6.15).

Let o be an occurrence of p in SA (clearly, it is also an occurrence of p in rSA). If $\mathfrak{r}(w)$ is in o, then $\mathfrak{r}(w + |p|)$ is not in o; the same holds also for $\mathfrak{r}(w')$. Therefore, by definition of M and M', at most two distinct robots in o are also in $M(\tilde{x}) \cup M'(\tilde{y})$, and there are no more than two distinct robots in $M(\tilde{x}) \cup M'(\tilde{y})$ that are in o. Hence, since there are at least three robots in every occurrence o of p, if the robots in $M(\tilde{x}) \cup M'(\tilde{y})$ are eliminated from the SEC, by (6.2) the angle between two any adjacent robots left on the circle is smaller or equal than 180°. The lemma follows by Lemma 6.2.1.

2. If $pos(rSA) = \emptyset$, the proof follows similarly to the previous case. Symmetrically, the lemma holds if $pos(SA) = \emptyset$. □

6.9 The Algorithm

Based on the definitions introduced in the previous sections, we are now ready to present an algorithm (Algorithm 7) that allows $n \geq 5$ robots to gather in a point in a finite number of cycles, provided they are not in an equiangular or biangular configuration at the beginning. In particular, let $start_i$ be the position of r_i in the initial configuration, with $start_i \neq start_j$ for all $1 \leq i \neq j \leq n$, and c the center of $SEC(start_1, \ldots, start_n)$.

Definition 6.9.1. The robots are in a *equiangular configuration* at the beginning if $SA(start_1, \ldots, start_n, c)$ is equiangular.

They are in a *biangular configuration* at the beginning if $SA(start_1, \ldots, start_n, c)$ is biangular. ◁▷

Algorithm 7 does not require to the robots any particular kind of agreement on the orientation of the local coordinate systems; in other words, it works even if the robots have no agreement on the orientation of the local coordinate system. The only ability the robots must have is to detect multiplicity. In fact, as stated in Theorem 3.4.2, such a capability is necessary in order to solve the gathering problem.

The idea behind the algorithm is described in the following. First, the algorithm checks if there is a point q in the plane with multiplicity greater than one. In this case, all the robots move towards q avoiding collisions. In fact, while they move, the robots have to pay attention to not occupy simultaneously another point $q' \neq q$ on the plane. In other words, q has to be the only point on the plane with multiplicity greater than one. This is accomplished by routine `Move(`q`)`, that moves the calling robot r towards q on a straight path and avoiding collisions. In particular, if there is a robot r' on the segment connecting the current position of r and q (in other words, r' is on r's way), the destination point of r is a point on the segment $[rr']$ at distance $d > 0$ from r' (i.e., r stops before colliding with r').

Then, the smallest circle SEC enclosing all the robots' positions, and its center c, are computed. At this point, the algorithm checks whether there is only one robot r inside SEC and on c. In this case, r chooses arbitrarily one of the robots on the circumference of SEC, and moves towards it. If r is on c, but it is not the only robot inside SEC, then all the robots inside SEC and closest to c move towards c.

In the case when no robot occupies the center of SEC, the string of angles SA is computed, according to routine `Compute_SA(`SEC`)` described in Section 6.6.2; its reverse rSA is computed too.

If SA is valid, the set S as defined in Definition 6.6.8 is first computed, and then sorted by routine `Sort(`S`)`. In particular, this routine sorts the tuples in S in ascending order, according to the following rule:

$$t_i < t_i \;\Leftrightarrow\; (t_i.orient = t_j.orient \wedge t_i.pos < t_j.pos) \vee (t_i.orient = \texttt{SA} \wedge t_j.orient = \texttt{rSA}).$$

Then, depending on the cardinality of set S, the algorithm analyzes separately three cases. In particular, if in S there is only one tuple t_1 (i.e., $|S| = 1$), we have

`Case(`$|S| = 1$`)`
  If All The Robots Are On The SEC **Then**
    $r :=$ `Elected();`
    If I Am r **Then** `Move(`c`).`
    Else `do_nothing().`
  If Only One Robot r Is Inside The SEC **Then**
    If I Am r **Then** `Move(`c`).`

Algorithm 7 Gathering With Unlimited Visibility, $n \geq 5$

```
    If There Is One Point q With Multiplicity > 1 Then Move(q).
    (SEC, c) := Compute SEC Of Robots' Positions, And Its Center;
    If One Robot r Is On c Then
        If Nobody Else Is Inside SEC Then
 5:         If I Am r Then
                q := Position Of One Of The Robots On The Rim Of SEC;
                Move(q).
            Else do_nothing().
        Else %Some Other Robot Is Inside SEC Besides r%
10:         If I Am Inside SEC Then
                Moving := {Robots Inside SEC And Closest To c};
                If I Am In Moving Then Move(c).
                Else do_nothing().
            Else do_nothing().
15: Else %No Robot Is On c%
        SA := Compute_SA(SEC);
        rSA := Reverse Of SA;
        If SA Is Valid Then
            S := Compute The Set S As Defined In Definition 6.6.8;
20:         Sort(S);
            Case |S|
                • = 1
                    Case(|S| = 1).
                • = 2
25:                 Case(|S| = 2).
                • > 2
                    Case(|S| > 2).
        Else %SA Is Not Valid%
            If Only One Robot r Is Inside SEC Then
30:             If I Am r Then
                    w := Index In SA Such That r = 𝔯(w);
                    q := 𝔩(w);
                    Move(q).
                Else do_nothing().
35:         Else %More Than One Robot Is Inside SEC%
                If I Am Inside SEC Then Move(c).
                Else do_nothing().
```

Else `do_nothing()`.

If More Than One Robot Is Inside SEC **Then**

If I Am One Of The Robots Closest To c **Then** `Move(`c`)`.

Else `do_nothing()`.

Routine `Elected()` elects an unique robot r as follows. If $t_1.orient = \texttt{SA}$, it looks for a position $j = t_1.pos + i$ in SA, $0 \leq i \leq |SA| - 1$, such that $c\sphericalangle^{\mathfrak{l}(j-1)}_{\mathfrak{l}(j+1)} \leq 180°$; then $r = \mathfrak{r}(j)$. If $t_1.orient = \texttt{rSA}$, it looks for j in rSA.

If $|S| = 2$, the following actions are taken.

`Case(`$|S| = 2$`)`

$(w, w') := (t_1.pos, t_2.pos)$;

$(r, r') := (\mathfrak{r}(w), \mathfrak{r}(w'))$;

$(l, l') := (\mathfrak{l}(w), \mathfrak{l}(w'))$;

$(\tilde{l}, l^*) :=$ Points On The SEC Opposite To l And l', Respectively;

$\alpha := l\hat{c}l'$;

$Moving\_Robots := \emptyset$;

$Selected := \{\mathfrak{r}(w+1), \mathfrak{r}(w-1), \mathfrak{r}(w'+1), \mathfrak{r}(w'-1)\} \setminus \{r, r'\}$;

If $\alpha = 180°$ **Or** ($\alpha < 180°$ **And** r, r' Are Adjacent On SEC) **Then**

$Moving\_Robots := Selected$;

If $\alpha < 180°$ **And** r, r' Are On SEC But Not Adjacent **Then**

$Moving\_Robots :=$ { Robots In $Selected$ That Lie On $arc(\alpha)$}; %(at most 2)%

If All The Robots Are On SEC **Then**

If I Am In $Moving\_Robots$ **Then** `Move_Carefully(`$c, Moving\_Robots$`)`.

Else `do_nothing()`.

If Only One Robot $\overline{r}$ Is Inside SEC **Then**

If $\overline{r} \in Moving\_Robots$ **Then**

If I Am In $Moving\_Robots$ **Then** `Move_Carefully(`$c, Moving\_Robots$`)`.

Else `do_nothing()`.

Else $\%_{\overline{r} \notin Moving\_Robots}\%$

If I Am $\overline{r}$ **Then** `Move(`c`)`.

Else `do_nothing()`.

If More Than One Robot Is Inside SEC **Then**

If All The Robots Inside SEC Are In $Moving\_Robots$ **Then**

If I Am In $Moving\_Robots$ **Then** `Move_Carefully(`$c, Moving\_Robots$`)`.

Else `do_nothing()`.

Else

If I Am Inside SEC **Then** `Move(`c`)`.

Else `do_nothing()`.

Finally, `Case(`$|S| > 2$`)` is defined as follows.

`Case(`$|S| > 2$`)`
If All The Robots Are On SEC **Then**
If $pos(SA) \neq \emptyset$ **Then**
$Moving\_Robots := \{\mathfrak{r}(t_i.pos), \forall t_i \in S | t_i.orient = \texttt{SA}\}$;
Else
$Moving\_Robots := \{\mathfrak{r}(t_i.pos), \forall t_i \in S | t_i.orient = \texttt{rSA}\}$.
If I Am In $Moving\_Robots$ **Then** `Move_Carefully(`$c, Moving\_Robots$`)`.
Else `do_nothing()`.
If There Is Exactly One Robot r Inside SEC **Then**
If $pos(SA) \neq \emptyset$ **Then**
$y :=$ First Index In SA Such That $\mathfrak{r}(y) = r$;
Else
$y :=$ First Index In rSA Such That $\mathfrak{r}(y) = r$;
$p :=$ Period Induced By S;
$Moving\_Robots' := \{\mathfrak{r}(k) | k = (y \bmod |p| + j|p|), \forall 0 \leq j \leq \frac{|SA|}{|p|} - 1\}$.
If I Am In $Moving\_Robots'$ **Then** `Move_Carefully(`$c, Moving\_Robots'$`)`.
Else `do_nothing()`.
If There Is More Than One Robot Inside SEC **Then**
If I Am Inside SEC **Then** `Move(`c`)`.
Else `do_nothing()`.

Note that, in the first case of `Case(`$|S| > 2$`)` (i.e., all the robots are on SEC), in $Moving\_Robots$ there are the robots associated to the positions in SA (resp. rSA) from where Lex_Min_String start. Routine `Move_Carefully(`$c, Moving$`)` moves the calling robot $r \in Moving$ towards c, with the restriction that r is not allowed to reach c unless all the other robots in the set $Moving$ are inside SEC.

If SA is not valid and there is only one robot r inside SEC, it moves towards the rim of SEC staying on the radius where it currently lies. If r is not the only robot inside SEC, then all the robots inside SEC move towards c avoiding collisions.

6.10 Correctness of Algorithm 7

In this section, we will show that Algorithm 7 correctly solves the gathering problem for $n \geq 5$ robots with unlimited visibility that can distinguish multiplicity.

At the beginning the algorithm analyzes if SA is valid and, if this is the case, acts according to the number of tuples in S. In the following we will analyze all

the cases that can arise, showing that, independently from the initial configuration (given that it is neither equiangular nor biangular), the robots always gather in one point in a finite number of moves. In particular, if SA is valid at the beginning, the following properties are true:

(P1) SEC never changes until two robots meet in a point.

(P2) In a finite number of steps SA becomes not valid. More specifically, either two robots meet, or one robot reaches c. Moreover SA does not change until SA becomes not valid.

(P3) When SA becomes not valid, the robots gather in a point in a finite number of cycles.

To prove the three properties stated, we will analyze the different situations that can arise at the beginning, showing that the type of initial configuration can change at most once (from valid SA to non valid SA), and that SEC is never affected until two robots meet in a point.

Lemma 6.10.1. *Let SA be not equiangular/biangular and valid at the beginning, and $|S| = 1$. Properties P1–P3 are true.*

Proof. We distinguish the possible cases, according to the actions executed by the algorithm.

a. *At the beginning, every robot is on the SEC.* The algorithm elects a robot r, according to the procedure described in `Case(`$|S| = 1$`)`; by Lemma 6.2.1, such a robot always exists; note that, since $|S| = 1$, routine `Elected()` returns an unique robot r, independently from the orientation given to SEC. According to the algorithm, r is the only robot allowed to move, and it moves towards c, while all the others can compute only *null movements*.

Since r has been chosen such that its position on the SEC does not create an angle $> 180°$, when it leaves SEC to enter inside it, according to Lemma 6.2.1, the circle does not change; moreover, since r moves radially, SA is still valid, and $|S| = 1$. After r is inside SEC, according to `Case(`$|S| = 1$`)`, r is still the only robot allowed to move. Moreover, SEC, SA and $|S|$ are invariant until, in a finite number of moves, r reaches c; when this happens, SA becomes not valid, and property (P2) follows. At this point, there is a robot on c, while all the others are on SEC, having computed only *null movements* until now. By Line 5, r is once again the only robot allowed to move, and it starts moving radially towards one of the robots on the circle. During this movement, SA

stays not valid (two robots are on the same radius, see Lemma 6.6.2), and SEC cannot change since all the other robots are on the SEC computing *null movements* (by Line 34). Therefore, in a finite number of steps, two robots meet in a point q, and property (P1) follows. Note that, until now, all the robots but r computed only *null movements*. Therefore, now they realize that there is only one point in the plane with two robots on it, and, according to Line 1 of the algorithm, they move towards it following a straight path. Since these movements are done avoiding that two robots collide (see description of procedure `Move()`), the invariant that there is only one point with multiplicity greater than 1 holds. In a finite number of steps, all the robots gather in q, and property (P3) follows.

b. *At the beginning, exactly one robot r is inside the SEC.* According to `Case(`$|S| = 1$`)`, r is the only robots allowed to move towards c, and the argument follows as in the previous case.

c. *At the beginning, more than one robot is inside the SEC.* According to `Case(`$|S| = 1$`)` of the algorithm, all the robots inside SEC and closest to c are allowed to move towards c, while all the others (either on the SEC or inside it, but not closest to c) can only compute *null movements*. Since only the closest to c are allowed to move, any collision is avoided while they approach c. Furthermore, since SA is valid at the beginning and all the robots move radially towards c, SA can not change during these movements, hence it stays valid, with $|S| = 1$, until one (or more) of them reaches c, in a finite number of moves, and property (P2) follows. If only one robot reaches the center first, Line 15 forces all the robots on the circle to compute still *null movements*, and Line 11 all the other inside SEC and closest to c (at least one) to move towards c. Therefore, in a finite number of movements, two robots will meet in c. Property (P1) clearly holds, since no one of the robots on the circle ever move until two robots meet in the center of the circle (Observation 6.2.1). At this point, all the robots on SEC will start moving radially towards c (Line 1), therefore, in a finite number of steps, all the robots gather in c, and property (P3) follows. A similar argument applies if more than one robot reaches c simultaneously. □

Lemma 6.10.2. *Let SA be not equiangular/biangular and valid at the beginning, and $|S| = 2$. Properties P1–P3 are true.*

Proof. Following the notation introduced in Lemma 6.8.1, let $w = t_1.pos$ and $w' = t_2.pos$, $r = \mathfrak{r}(w)$ and $r' = \mathfrak{r}(w')$ be the robots associated to w and w', and $\tilde{l}$ and l^* be respectively the opposite points to $l = \mathfrak{l}(w)$ and $l' = \mathfrak{l}(w')$.

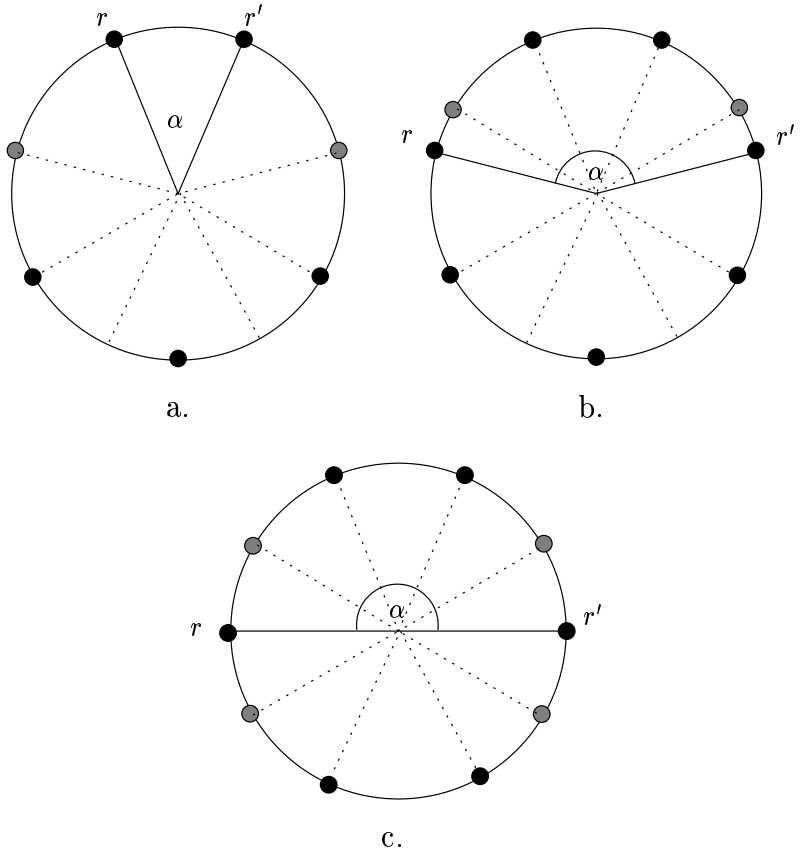


Figure 6.16: Lemma 6.10.2. The grey circles represent robots in *Moving_Robots* when (a) $\alpha < 180°$, and r and r' are adjacent on SEC; (b) $\alpha < 180°$, and r and r' are on SEC, but not adjacent; (c) $\alpha = 180°$.

The first thing the algorithm does is to compute *Moving_Robots*, according to $\alpha = l\widehat{c}l'$. Either $\alpha = 180°$ or smaller, *Moving_Robots* is a set of robots on the SEC that does not include neither r nor r'.

a. *At the beginning, every robot is on the SEC.* The possible cases are analyzed:

(1) $\alpha < 180$ and r and r' are adjacent on the SEC. In this case the robots in MR = *Moving_Robots* are allowed to move. In this set there are the robots that lie on the radiuses closest to the ones where r and r' lie (refer to Figure 6.16.a). Since r and r' are adjacent on SEC and they are not in *Moving_Robots*, and SA is valid, we have $|Moving\_Robots| = 2$. According to `Case(`$|S| = 2$`)`, the two robots in *Moving_Robots* start moving towards c. Since $n \geq 5$ and by hypothesis all the robots are on the circumference of SEC, there is at least one more robot on SEC. Therefore, by Lemma 6.8.1, when these two robots leave SEC, no angle bigger than 180° can be created, and SEC does not change. Since they move radially, SA does not change when one of them (or both) move inside the circle; hence, *Moving_Robots* does not change, and `Case(`$|S| = 2$`)` of the algorithm allows only these robots to keep moving towards c, with the restriction that they are allowed to reach c only if all the *Moving_Robots* are inside SEC (see description of `Move_Carefully()`). All the other robots stay on the SEC computing *null movements*. Hence, in a finite number of movements, at least one of the robots in MR reaches c, and property (P2) follows.

If the robots in MR reach simultaneously c, then the lemma follows by Line 1 of the algorithm (similarly to Case c. of the previous lemma). Otherwise, one robot is on c and another one is inside SEC (because of definition of `Move_Carefully()`); therefore, by Line 10, the robot inside SEC (that is in MR) moves towards c. Note that, only the robots in MR are allowed to move, while all the others stay on the SEC computing *null movements* until the two robots in MR meet in c; moreover, SA stays not valid (one robots is on c), and in a finite number of movements, two robots will meet in c, hence property (P1) follows. Property (P3) follows by Line 1 of the algorithm (as in Case c. of Lemma 6.10.1).

(2) $\alpha < 180$ and r and r' are not adjacent on the SEC (refer to Figure 6.16.b). In this case, in *Moving_Robots* there are the robots on the $arc(\alpha)$ that lie on the radiuses closest to the radiuses where r and r' are (at most two). Since $\alpha < 180$, when these robots start moving, they cannot create any angle $> 180°$, therefore by Lemma 6.2.1, SEC cannot change. The proof follows similarly to the previous case.

(3) $\alpha = 180°$ (refer to Figure 6.16.c). In this case, the algorithm allows only the robots in *Moving_Robots* (at most 4) to move towards c, while all the others can only compute *null movements*. Since r and r' are opposite points on the SEC, and neither r nor r' are in *Moving_Robots*, by Lemma 6.2.1, SEC does not change when the robots in *Moving_Robots* enter inside it. The proof follows as in Case (1).

b. *At the beginning, exactly one robot is inside the SEC.* The algorithm checks in `Case(`$|S| = 2$`)` if this robot, say $\overline{r}$, belongs to *Moving_Robots*. In this case, it keeps moving towards c, but it is not allowed to reach it unless all the others in *Moving_Robots* are inside SEC, and the argument follows similarly to the previous Case a.(2) and Case a.(3).

Otherwise, ($\overline{r}$ does not belong to *Moving_Robots*) it is allowed to go towards c and to reach it. Note that all the other robots stay on the SEC, computing *null movements*; in fact, in this case the set *Moving_Robots* is ignored. Since $\overline{r}$ moves radially, SA does not change until it reaches c, and property (P2) follows. At this point there is a robot on c, and all the others are on SEC, having computed only *null movements* until now. Properties (P1) and (P3) follow as in Case a. of Lemma 6.10.1.

c. *At the beginning, more than one robot is inside the SEC.* In this case, if all the robots inside SEC belong to *Moving_Robots*, these are once again the only robots allowed to move towards c (`Case(`$|S| = 2$`)`). In this case, they *move carefully* towards c, in order to allow all the other *Moving_Robots* that could still be on the rim of the circle to come inside SEC, before one of them reaches c. All the other robots (those not in *Moving_Robots*) stay on the SEC computing *null movements*. Therefore, in a finite number of movements, at least one of the robots in *Moving_Robots* reaches c, and property (P2) follows. The argument follows as in previous Case a.(1).

Otherwise (not all the robots inside SEC are in *Moving_Robots*), the robots inside SEC start moving towards c, while all the robots on the circle can compute only *null movements* (in this case *Moving_Robots* is ignored). Since all the movements are radial, SA does not change until one of them reaches c, and property (P2) follows. Now there is one robot on c, and at least another robot inside SEC. Properties (P1) and (P3) follow as in Case c. of Lemma 6.10.1. □

Lemma 6.10.3. *Let SA be not equiangular/biangular and valid at the beginning, and $|S| > 2$. Properties P1–P3 are true. Moreover, the following invariant holds:*

(I) The only destination point a robot can compute is the center c of SEC.

Proof. Analogously to the previous cases, we analyze what happens in the possible initial configurations.

a. *At the beginning, every robot is on the SEC.* By Lemma 6.7.1, SA and its reverse are periodic. Let p be the period induced by S. Let $Moving_b$ be the set of robots that are in $Moving\_Robots$ at the beginning, where $Moving\_Robots$ is as defined in `Case(`$|S| > 2$`)` of the algorithm. It follows by Lemma 6.7.1 that $Moving_b \subseteq M(h^1_{SA}.On\_SA) \cup M'(h^1_{rSA}.On\_rSA)$, where $M()$ and $M'()$ are as defined in Lemma 6.8.2. Furthermore, according to the algorithm, the robots in $Moving_b$ are the only robots allowed to move. If all these robots leave the rim of SEC, from Lemma 6.8.2, and since these robots move radially towards c, SEC and SA do not change. Moreover, all the other robots stay on the SEC computing *null movements*.

When one of the $Moving_b$ enters inside SEC, $Moving\_Robots'$ is computed. It is easy to see that the only robots that can be in this set are exactly those in $Moving_b$. Hence, once again these robots keep *moving carefully* towards c, while the others can only compute *null movements*.

When more than one of the $Moving_b$ enter inside SEC, all the robots inside SEC are still the only robots allowed to move (note that these robots are in $Moving_b$ too). Therefore, in a finite number of movements, one (or more simultaneously) of the $Moving_b$ reaches c, and property (P2) follows.

If two robots reach c simultaneously, property (P1) follows. Otherwise, there is one robot on c, at least another one is inside SEC, some of the $Moving_b$ can be still on the SEC, and all the others on SEC can only compute *null movements*. Therefore, according to Line 10 of the algorithm, the robots inside SEC move towards c, while the robots in $Moving_b$ still on the SEC can decide to enter inside SEC, and all the others keep computing *null movements*. SA stays not valid (one robot is on c), and by Lemma 6.8.2 SEC can not change. Moreover, in a finite number of movements, two robots will meet in c, and property (P1) follows.

Now, all the robots not in $Moving_b$ (that until now computed *null movements*), start moving radially towards c (Line 1 of the algorithm), and in a finite number of movements they will gather in c. Also the robots in $Moving_b$ that at this time are inside SEC keep computing c as destination point; hence they gather in c in a finite number of movements too. Finally, all the robots have always computed as destination point c (at the beginning because it was the center of SEC, and now because it is the only point in the plane with two

robots on it). Therefore, invariant (I) holds, and all the robots gather in c in a finite number of movements, and property (P3) follows.

b. *At the beginning, exactly one robot is inside the SEC.* If one robot r is inside SEC, the algorithm allows to move only the robots[6] in $Moving\_Robot'$ as defined in `Case(`$|S| > 2$`)`, with $y = t.pos$ and t the first tuple in S such that $\mathfrak{r}(t.pos) = r$. All the robots in $Moving\_Robots'$ move carefully towards c, while the others compute *null movements*. Since $Moving\_Robot' \subseteq M(t.pos \bmod |p|) \cup M'(t.pos \bmod |p|)$, with p the period induced by S, by Lemma 6.8.2 SEC does not change when these robots leave the rim of SEC. The argument follows as in the previous case.

c. *At the beginning, more than one robot is inside the SEC.* All these robots are elected as $Moving\_Robots$, and are allowed to move towards c, while the others (on SEC) can only compute *null movements*. Since all the movements are radial, SA does not change until one of them reaches c in a finite number of movements, and property (P2) follows. When this happens, by Line 10, the only robots allowed to move are once again the ones inside, while those on SEC keep computing *null movements*. Therefore, SA stays not valid, SEC can not change, and in a finite number of movements two robots meet in c, and property (P1) follows. Property (P3) and invariant (I) follow as in previous Case a. □

Hence, by Lemmas 6.10.1–6.10.3, we can state the following

Theorem 6.10.1. *Let SA be not equiangular/biangular and valid at the beginning. The following properties are true:*

(P1) SEC never changes until two robots meet in a point.

(P2) In a finite number of steps SA becomes not valid. More specifically, either two robots meet, or one robot reaches c. Moreover the kind of configuration stays the same until SA becomes not valid.

(P3) When SA becomes not valid, the robots gather in a point in a finite number of steps.

Finally, the following lemma states that, if SA is not valid, the robots gather in a point in a finite number of cycles.

[6] If this case applies after the previous one, $Moving\_Robot$ computed now coincides with the set $Moving_b$.

Theorem 6.10.2. *Let SA be not valid at the beginning. In a finite number of steps the robots gather in a point.*

Proof. By definition of not valid, there must be at least one robot inside SEC. Two cases can happen.

1. If one robot r is inside the circle, and it is also on c, the algorithm elects r as the only robot allowed to move (Line 5). It starts moving radially towards a robot r' on the SEC, while all the others can only compute *null movements*. Note that during this movement, SA stays not valid and the SEC can not change. In a finite number of movements, r and r' will be on the same point q. The theorem follows as in Case a. of Lemma 6.10.1.

   If r is not on the center, let w be the position in SA such that $r = \mathfrak{r}(w)$. Since all the other robots are on the SEC, and by definition of not valid string of angles, there is a robot r' on $\mathfrak{l}(w)$. According to Lines 31–34 of the algorithm, r starts moving towards r', while all the others can only compute *null movements*. The theorem follows as in the previous case.

2. If more than one robot is inside SEC, the algorithm let the robots closest to the center of SEC to compute as destination point c (Line 10 if one robot is on c, and Line 36 otherwise), while all the others can compute only *null movements*. Note that, since these robots perform only radial movements, SA stays not valid. Moreover, since only robots inside SEC are allowed to move, SEC can not change. Furthermore, in a finite number of movements, one robot reaches c. We distinguish two cases.

   (a) If one robot was already on the center of SEC (at the beginning), c is the only point on the plane with two (or more) robots on it, therefore every robot will compute again c as destination point (Line 1 of the algorithm). Since c is the only destination point a robot can compute (at the beginning because it is the centre of SEC, and then because it is the only point in the plane with multiplicity greater than one), all the robots will gather there in a finite number of movements.

   (b) Otherwise (no robot was on c at the beginning), since SA is still not valid, c is again the only destination point that can be computed (Line 10) and SEC does not change. Eventually, two robots will meet in c, and the proof follows as in the previous case. □

From Theorems 6.10.1 and 6.10.2, we can state that

Result 10. *There exists a deterministic and oblivious algorithm that, in a finite number of cycles, gather in a point a set of $n \geq 5$ robots with unlimited visibility that can detect multiplicity, when they are not in equiangular or biangular configuration at the beginning.*

6.11 Conclusions

In this chapter we studied the gathering problem for a set of robots with unlimited visibility and with no agreement on their local coordinate systems. We showed that 2 robots can deterministically gather in a point only if they can bump into each other. If $3 \leq n \leq 4$, we presented algorithms that allow them always meet in a point in a finite number of cycles.

In contrast, if $n \geq 5$, we designed an algorithm that works only when the robots are not at the beginning in equiangular or biangular configuration. Therefore, a main open problem is to understand what can be done in these cases.

Another interesting problem is to see if there exists an *easier* algorithm that works in all cases, and unifies the algorithms presented in Cases b–d.

Chapter 7

Flocking[1]

[...] and they will hear My voice. So they will become one flock with one shepherd.

John 10:16

Abstract

In this chapter, the model is slightly changed in order to study an interesting problem: the *flocking*. In this case, one robot, called the *leader* is different from all the others, called the *followers*, and it can be recognized by the other $n-1$ robots in their observations. The leader moves following an arbitrary path. The task for the followers is to follow the leader while moving in flock; that is, they have to follow the leader wherever it goes while keeping a formation (described as a set of points). We present an algorithm that solves the problem for robots with no agreement on the local coordinate systems that have to keep symmetric formations, and we test our solution by computer simulation.

7.1 Introduction

In this chapter we study the *flocking problem*: a set of mobile units are required to follow a leader unit while keeping a predetermined formation (i.e., they are required to move in flock, like a group of soldiers). Moreover, the units in the flock do not know beforehand the path the leader will take: their task is just to follow him wherever he goes, and to keep the formation while moving. This is a problem often studied in robotics, with several applications (e.g. military [9], or in factories, where robots can be asked to move heavy loads). The approach usually adopted to study this and similar problems, is to design solutions based on heuristics and tailored on

[1]The results presented in this chapter appear also in [46].

the capabilities of the robots employed, and then test them by computer simulations, or on real robots.

For instance, in [58], experiments are conducted on a team of simple mobile units in order to produce *complex* behaviors, by compounding *basic* ones (such as safe-wondering, i.e. the ability to avoid collisions while moving; dispersion, i.e. the ability of the robots to spread out over an area; aggregation, i.e. the ability to gather; and homing, i.e. the ability to reach a predetermined destination). In particular, the author points out that flocking can be obtained combining safe-wandering, aggregation, dispersion, and homing. Hence, in her experiments, all the units have a common destination to reach.

T. Balch and C. Arkin studied formation and navigation problems in multi-robot teams. In particular in [5] the problem of specifying the behavior for the navigation of a mobile unit is analyzed, and results of both computer simulation and real experimentation are reported. In [9] the approach is extended to multi-robot teams that navigate the environment maintaining particular formations: in particular the case of a line, column, diamond and wedge are examined. In their study, the authors assumed that the path along which the group of robots has to move is known in advance to every unit. This same assumption is made in [22], where the robots are asked to move in a matrix shape along a path represented by a straight line followed by a right turn and then a straight line again. In contrast, in this paper we do not assume any knowledge by the followers of the path that the leader will follow. The followers have only a common description of the formation they have to keep while moving.

A similar problem is studied in [79], where the author derives equations describing navigational strategies for robots moving in formation, and following the movement described by one (ore more) leader. In the studied framework, the robots have identities, hence their positions in the formation are fixed. Moreover, in order for the i-th robot to compute its position at time t, it has to know the position of either the $(i-1)$-th robot or the leader at time t. Hence, some degree of synchrony has to be introduced in order to implement these strategies.

In this chapter, we analyze the flocking problem by using very simple units and by dropping the assumption that all the robots in the flock know the path, or that they can derive it (e.g. by observing the orientation of the leader's prow, or by deriving it by observing the leader in different positions). We describe an algorithm that allows the followers to keep a formation given to them as input, while following a path determined at run-time by the leader, that acts completely independently and does not behave according to the followers' algorithm. Moreover, we present results of computer simulations that show the effectiveness of the proposed solution.

7.2 Changes to the Basic Model

The model presented in Chapter 3 has been adapted to the particular problem under study. First, there are two kinds of robots in the environment: the *leader*, denoted by L, and the *followers*, denoted by $f_1, \ldots, f_n$. The leader acts independently from the others, and we can assume that it is driven by a human pilot. In the following we will discuss only about the followers. Each follower can observe all the other robots in the system: in particular, it can distinguish the leader from all the other followers.

All the followers have as input, however, the same pattern $\mathbb{F}$ representing the flock to be kept. $\mathbb{F}$ is described as a set of coordinates in the plane, relative to a point representing the leader.

Since we need to model robots that "continuously" move, we assume that the robots are never in *Wait*, and that the time spent in looking and computing is negligible compared to the time spent in moving.

Moreover, no one of the followers knows in advance the path that the leader will follow, nor it can derive it at run-time. Their only task is to observe where the leader and the other followers are, reach an agreement — without communicating — on how and where to form the pattern in the plane, and move to positions such that the flock is formed and maintained.

7.3 The Flocking Problem

In this section we give a formal definition of a family of problems that we call collectively the *Flocking Problem*. In particular, we propose two variants of the problem, and characterize through several metrics the degree of acceptability of an approximate solution.

7.3.1 Definitions

Definition 7.3.1 (Formation). A *formation* $\mathbb{F} = \{p_1, \ldots, p_{n-1}, p_L\}$ is a configuration with a distinguished point, p_L. We call the distinguished point the *leader* of the formation, and the remaining points the *followers* of the formation.

We call *radius* of a formation $\mathbb{F}$, denoted by $R_{\mathbb{F}}$ the maximum distance between the designated point p_L and the other points in $\mathbb{F}$:

$$R_{\mathbb{F}} = \max_{i=1\ldots n-1} dist(p_L, p_i).$$ ◁▷

Configurations are used to model the positions of a set of vehicles, and also to express the set of points that constitute the desired formation. The formation whose

points are the current positions of the vehicles (included the leader) is called the (current) *fleet* (denoted by $\mathbb{E}$), while the formation given in input to the robots, and whose points represent the desired position of the vehicles once the flock is formed, is called the *pattern* (denoted by $\mathbb{P}$).

In order to assess the degree of success of the flocking, we need a measure of how well the current fleet approximate the desired pattern. We introduce such a measure in the following

Definition 7.3.1 ($\mathcal{D}$-distance). Given two configurations $C = \{c_1, \ldots, c_n\}$ and $G = \{g_1, \ldots, g_n\}$, we define the $\mathcal{D}$ distance among them as follows:

$$\mathcal{D}(C, G) = \min_{\pi \in \Pi} \sum_{i=1}^{|C|} dist(c_i, g_{\pi(i)})$$

where Π is the set of all the possible permutations of $1 \ldots |C|$.

Moreover, we define $\rho_i(C, G) = dist(c_i, g_{\pi'(i)})$, where π' is the permutation such that $\mathcal{D}(C, G) = \sum_{i=1}^{|C|} dist(c_i, g_{\pi'(i)})$. As a shorthand, we will write simply ρ_i when the configurations are evident in the context. ◁▷

As an aside, we note that other kind of measures can be used. For instance, the maximum of the sum of the distances, or its average, or the sum of the squares of the distances. Our experiments, however, have been tested according to the measure defined in Definition 7.3.1.

We can consider that the desired pattern is reached when the vehicles place themselves in the desired shape, with no regard for the orientation, or we can ask, in addition, for a specific orientation (typically, corresponding to the current heading of the leader). These two alternatives are defined formally in the following

Definition 7.3.2 (Target). Given a pattern $\mathbb{P}$ and a fleet $\mathbb{E}$, we call an *undirected target* of the vehicles any formation that is obtained by translating $\mathbb{P}$ so that its leader point coincides with the leader of $\mathbb{E}$, and rotated by an arbitrary angle. We denote such a formation with $\mathcal{T}_{\mathbb{P},\mathbb{E}}$.

Given, in addition, an angle θ, we call the *directed target* of the vehicles the particular undirected target that is rotated by θ. We denote this formation with $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ We call the followers' positions in a target the *slots* of the target. ◁▷

Notice that, given a pattern and the position and heading of the leader, there are infinite undirected targets (Figure 7.1.a), but only one directed target (Figure 7.1.b) — that, naturally, is also an undirected target.

Since in our model the leader is constantly moving, while the followers only execute discrete cycles, it is impossible for them to exactly form and maintain the

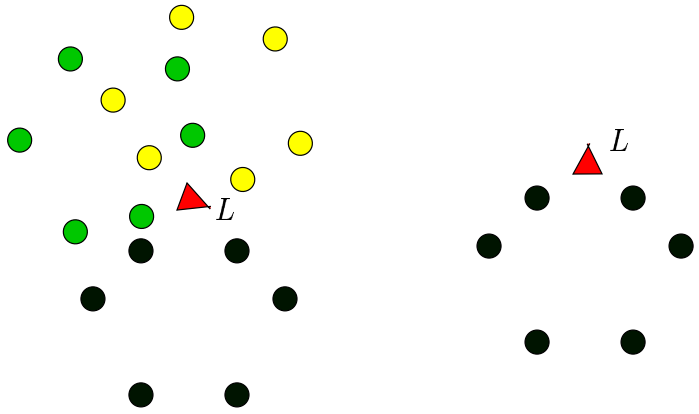


Figure 7.1: Undirected and directed targets. The triangle represents the leader.

pattern at any desired time. To take this effect into account, we introduce two distinct notions for *exactly* and *approximately* forming the pattern in the definitions below.

Definition 7.3.2 (Exact Formation). Given a fleet $\mathbb{E}$ and a pattern $\mathbb{P}$, we say that the followers in $\mathbb{E}$ exactly form an undirected target $\mathcal{T}_{\mathbb{P},\mathbb{E}}$ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) = 0.$$

Moreover, if ψ is the heading of the leader, we say that the followers exactly form the desired pattern if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi}_{\mathbb{P},\mathbb{E}}) = 0.$$

◁▷

We extend the above definitions to the case in which the formation is not kept exactly.

Definition 7.3.3 (Approximate Formation). An undirected target is formed up to ξ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) \leq \xi.$$

Analogously, the directed target is formed up to ξ if

$$\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi}_{\mathbb{P},\mathbb{E}}) \leq \xi.$$

◁▷

Finally, we can introduce our formal definition of the Flocking Problem.

Definition 7.3.4 (The Flocking Problem). Let $f_1, \ldots, f_{n-1}$ be a group of vehicles according to our model, and let L be an additional distinguished leader vehicle, with heading ψ_L, whose positions constitute a formation $\mathbb{E}$, and let $\mathbb{P}$ be a pattern given in input to $f_1, \ldots, f_{n-1}$. The vehicles solve the exact (resp. approximate) *Flocking Problem* if, starting from an arbitrary formation at time t_0, $\exists t_1 \geq t_0$ such that, $\forall t \geq t_1$ the vehicles exactly (resp. up to ξ) form a certain formation. More specifically, four variants of the flocking problem exists:

- exact undirected flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) = 0$
- exact directed flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi_L}_{\mathbb{P},\mathbb{E}}) = 0$
- approximate undirected flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}_{\mathbb{P},\mathbb{E}}) \leq \xi$
- approximate directed flocking: $\mathcal{D}(\mathbb{E}, \mathcal{T}^{\psi_L}_{\mathbb{P},\mathbb{E}}) \leq \xi$

◁▷

The exact flocking variants cannot be solved in our model, since we assume that the leader moves continuously and arbitrarily, while the followers only have discrete opportunities for observing the position of the leader and adjust their course accordingly. Hence, while exact flocking can be considered as an ideal reference problem, in the following we will concentrate on the two variants of approximate flocking.

Notice however that, since we characterize through ξ the degree of approximation, and since we will give conditions that relate ξ to the features of the vehicles, an arbitrarily good approximation can be obtained.

7.3.2 Conditions

In order for the problem to be solvable, a number of conditions must be met. Let v_L and ω_L be the maximum linear and angular velocity of the leader, respectively, and let v_f be the maximum linear velocity of follower f. Firstly, the leader must not be too fast, otherwise the followers will not be able to maintain the formation. Formally,

$$v_L < \min_i v_{f_i}. \tag{7.1}$$

Moreover, the slots must not move too fast for the followers, as a consequence of the leader changing direction; thus, also the angular velocity of the leader must be limited:

$$v_L + \omega_L \cdot R_{\mathbb{P}} < \min_i v_{f_i}. \tag{7.2}$$

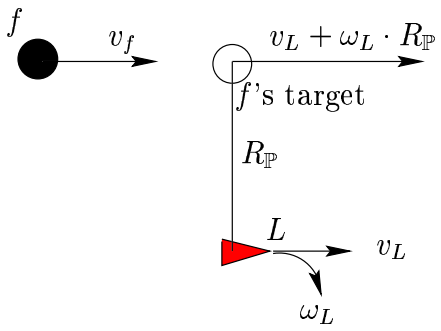


Figure 7.2: Limitations on the linear and angular velocity of the leader.

In fact, in the worst possible case the leader is moving away from a follower f while at the same time turning so that the tangential velocity of the points in $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ that f is trying to reach is maximal (see Figure 7.2).

Secondly, in order to maintain the flocking up to ξ, the time spent in a *Move* by a follower must not be too long. Otherwise, the leader could change direction and move away from a follower in the time between two consecutive *Look*s, without the follower having a chance to correct its course. Formally, let

$$k^f = (v_L + \omega_L \cdot R_{\mathbb{P}}) \max_j \tau_f^M(j)$$

where $\tau_f^M(j)$ is the duration of the *Move* phase of the j-th cycle of the follower f. In the above definition, k^f is the maximum distance that a point in $\mathcal{T}^{\theta}_{\mathbb{P},\mathbb{E}}$ may travel during the longest *Move* phase of the follower f. Since we want that the overall $\mathcal{D}$-distance remains bounded by ξ, the following condition must be met:

$$\sum_i k^{f_i} \leq \xi \tag{7.3}$$

The condition above is overly restrictive, though, since we consider the maximum duration for the *Move* of all the followers. We can state a less restrictive (but still only sufficient) condition by considering the duration of a *Move* at a specified point in time. In detail, let $\bar{\tau}_f(t)$ be the time between t and the beginning of the *Move* of the follower f that is being executed at time t (see Figure 7.3). Then,

$$\forall t, \sum_i (v_L + \omega_L \cdot R_{\mathbb{P}}) \bar{\tau}_{f_i}(t) \leq \xi \tag{7.4}$$

As a last remark, we note that, since we want the vehicles to form a specific formation and *to keep that* formation while moving, it is necessary for the followers to agree on a common unit measure. Otherwise (i.e. if the input pattern can be scaled), the formation would be formed even if, instead of following the leader,

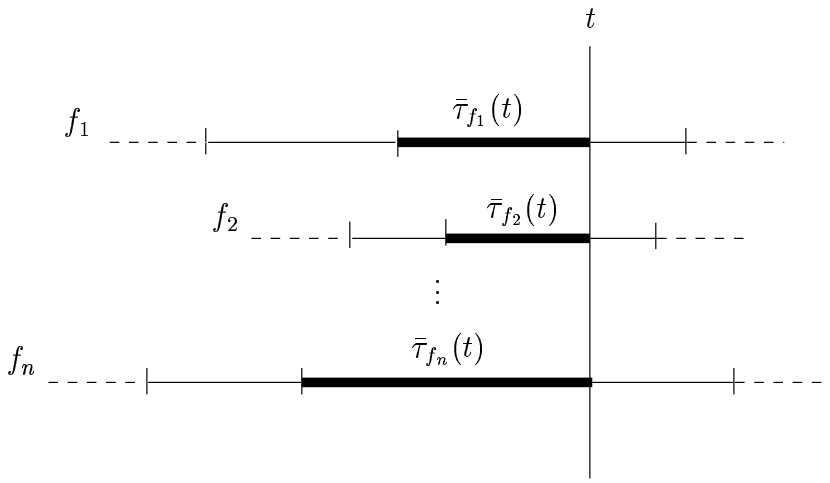


Figure 7.3: Definition of $\bar{\tau}_f$.

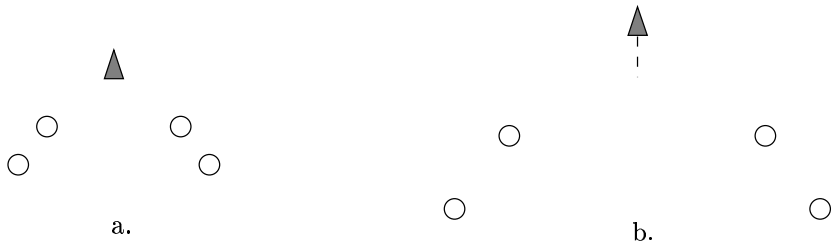


Figure 7.4: The triangle represents the leader, and the circles the positions of $\mathbb{P}$. If the vehicles did not have a common unit distance, then they could scale $\mathbb{P}$, and the flock would not follow L while it moves.

the followers would simply "scale" the input formation (that is, we would have a situation where the vehicles would form a pattern that becomes bigger and bigger as the leader goes farther away, see Figure 7.4). Hence,

Observation 7.3.1. In order for the problem to be significant, $f_1 \ldots, f_{n-1}$ must have common knowledge on the unit measure. ◇

7.4 Basic Algorithm for the Flocking Problem

In this section we present an algorithm to solve the approximate directed flocking problem that works in the general setting where there is no agreement on the local coordinate systems. Every follower f_i is given in input a pattern $\mathbb{P}$ described as a set $p_1, \ldots p_{|\mathbb{P}|}$ of points, relatives to the leader vehicle, L; we clearly assume to have

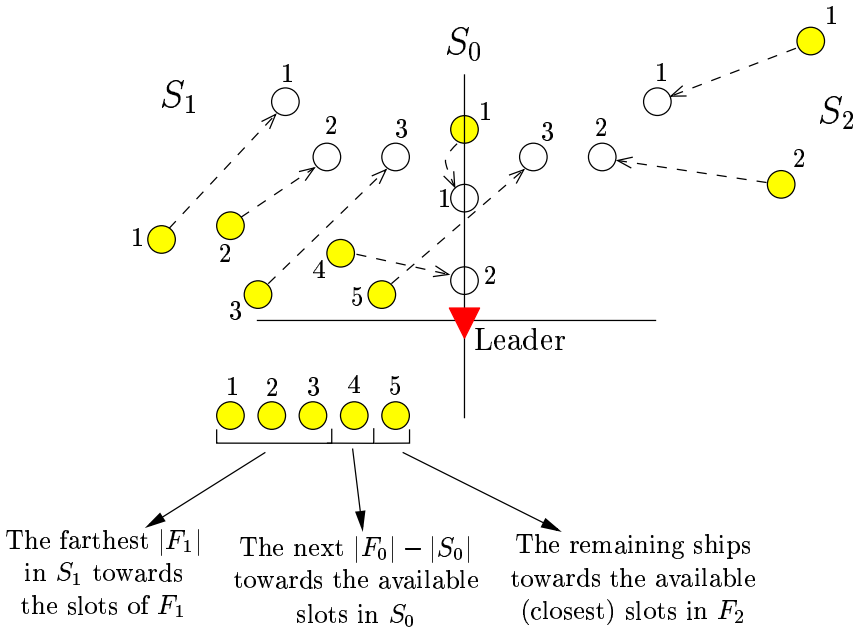


Figure 7.5: Examples of the behavior of Algorithm 8. The filled triangle is the leader, the filled circles are the followers, and the empty circles the *slots* the followers want to reach.

$|\mathbb{P}|-1$ followers arbitrarily placed on distinct positions at the beginning (this defines a *valid* initial configuration for this problem).

The intuition behind the basic algorithm is described in the following (see also Figure 7.5). First, the generic follower f computes the baricenter B of the followers' positions (Line 1), by executing `Baricenter(`$Followers$`)` ($Followers$ and L are the positions of the followers and of the leader retrieved in the previous *Look* state, respectively), and a shared vertical axis Y given by the line passing through L and B, and oriented according to $\overrightarrow{BL}$ can be derived: this is accomplished by `Get_Y_axis(`L, B`)`, that returns the axis Y that all the followers will use to agree on orienting themselves in the plane.[2] Then S_0, S_1 and S_2, containing respectively vehicles whose positions are exactly on Y, to its left, and to its right, are computed (according to the local concept of *left/right of* Y).

At this point, f executes `Final_Positions(`$\mathbb{P}, L, Y$`)` (Line 6), that rotates the points in $\mathbb{P}$, assuming that the leader is moving according to the direction and orientation of Y, and translates them into the observed leader's position. The positions returned by this routine are the *slots* that the followers will try to reach. After having computed the baricenter B_F of the *slots* in Line 7, these positions are

[2]If $L = B$, the followers can simply wait for the leader to move away from B, or for some fellow follower that is already moving to break the tie.

Algorithm 8 The Basic Flocking Algorithm

Input: The pattern $\mathbb{P}$ to be kept, relative to the leader. *Followers* and L are the positions of $f_1, \ldots, f_{n-1}$ and of the leader retrieved in the last *Look* state, respectively. *me* represents the robots executing the algorithm.

$B :=$ `Baricenter(`$Followers$`)`;
$Y :=$ `Get_Y_axis(`L, B`)`;
$S_0 :=$ {Robots On Y};
$S_1 :=$ {Robots On The Left Of Y};
$S_2 :=$ {Robots On The Right Of Y};
$F :=$ `Final_Positions(`$\mathbb{P}, L, Y$`)`;
$B_F :=$ `Baricenter(`F`)`;
$F_0 :=$ {Final Positions On Y};
$F_1 :=$ {Final Positions On The Left Of Y};
$F_2 :=$ {Final Positions On The Right Of Y};
For All $j = 0, 1, 2$ **Do**
    `Sort(`F_j, L, B_F`)`;
    `Sort(`S_j, L, B`)`;
End For
Case *me* in
- S_1
    $k :=$ `Rank(`me, S_1`)`;
    If $k \leq |F_1|$ **Then**
        `Move(`k-th Position In F_1`)`.
    Else If $k \leq |F_1| + |F_0| - |S_0|$ **Then**
        $H :=$ {Robots In S_1 Whose Rank $> |F_1|$}$\cup$
            {Robots In S_2 Whose Rank $> |F_2|$};
        `Sort(`H, L, B`)`;
        $k' :=$ `Rank(`me, H`)`;
        $p :=$ $(k' + |S_0|)$-th *slot* in F_0;
        `Move(`p`)`.
    Else
        `Move(`$(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th Position In F_2`)`.
- S_2
    /* This case is symmetric to the previous one */
- S_0
    $k :=$ `Rank(`me, S_0`)`;
    If $k \leq |F_0|$ **Then**
        `Move(`k-th Position In F_0`)`.
    Else If $|S_1| \leq |S_2|$ **Then**
        `Move(`$(|F_1| - |S_1| + (k - |F_0|))$-th Position In F_1`)`.
    Else
        `Move(`$(|F_2| - |S_2| + (|k - |F_0|))$-th Position In F_2`)`.

partitioned in three subsets: those exactly on Y (F_0, Line 8), to the left of Y (F_1, Line 9), and to its right (F_2, Line 10). Then, F_j, $j = 0, 1, 2$, are sorted in decreasing order with respect to the distances from L and B_F (Line 12), and S_j, $j = 0, 1, 2$, are sorted in decreasing order with respect to the distances from L and B (Line 13). These sorting operations are done by `Sort(`A, l, b`)`, where A is the array (set of points) to be sorted. In particular, after the sorting, it is guaranteed that

$$\begin{aligned} \forall i, j,\ i < j \quad \Rightarrow \quad & (dist(l, p_i) > dist(l, p_j)) \vee \\ & (dist(l, p_i) = dist(l, p_j) \wedge dist(b, p_i) > dist(b, p_j)), \end{aligned}$$

where p_i and p_j are points in A. Next, the rank k of f in the subset it belongs to is computed, by `Rank(`$me, \cdot$`)`.

Now, if f is the k-th follower in S_1, and $k \leq |S_1|$, then it moves towards the k-th position in F_1 (Line 19; a similar argument applies if f is in S_2, see Line 34). Otherwise, if there are positions *available* in S_0 (i.e., $|F_0| > |S_0|$ and $k - |F_1| \leq |F_0| - |S_0|$), f is directed towards S_0. In particular, Line 21 computes the set H containing the vehicles in S_1 and S_2 whose rank is respectively bigger than $|F_1|$ and $|F_2|$; H is then sorted, and the rank k' of f in H is computed in Line 24. Then, f is directed towards the $(k' + |S_0|)$-th *slot* in F_0 (Lines 25–26), that is towards a *slot* in F_0 that is not a target of vehicles in S_0 (refer to Lines 31–38 to see how vehicles in S_0 choose their targets). If no position in S_0 is available, f moves towards the $(|F_2| - (k - |F_1| - |F_0| + |S_0|) + 1)$-th position in F_2, that is towards one of the *slots* in F_2 that are not a destination point of either a vehicle in S_1 whose rank is smaller than k, or of a vehicle in S_2 (Line 28).

If f is in S_0, and its rank k is smaller than $|F_0|$, then it simply moves towards the k-th *slot* in F_0 (Line 34). Otherwise, it chooses to move towards the side that has fewer vehicles (note that if $|S_1| = |S_2|$, then it chooses to move towards a slot in F_1). In Figure 7.5, an example of how the followers chose their *slots* is depicted.

Finally, `Move(`p`)` moves the executing vehicle towards p, and terminates the current cycle. As already pointed out in Section 3.1, in general the vehicle does not reach p in one *Move* (the distance it travels is finite, but unpredictable). Clearly, since the vehicles can not remember p in the next cycle (obliviousness), this implies that it is possible that f changes its destination point in the next cycle, because its ranking can change.

Since the vehicles are memoryless, they can not be sure of the direction of movement of L. They only assume that the leader is going away from B (i.e. according to $\overrightarrow{BL}$). Furthermore, the followers assume that the direction of movement of L is given by the axis passing through B and L, and oriented from B towards L, hence they can reach an agreement on Y. They can not, however, reach in general a similar agreement on X, that is on an axis orthogonal to Y that would let them agree on

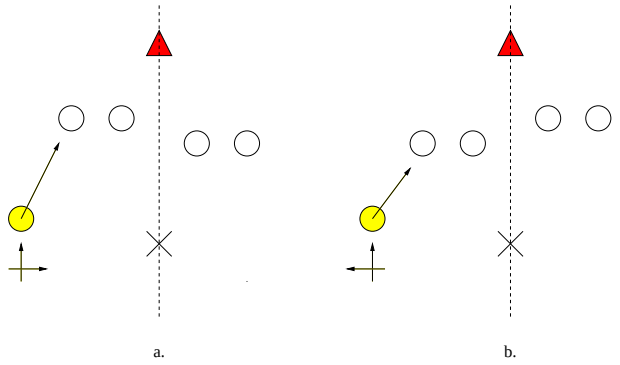


Figure 7.6: With a non-lb-symmetric pattern, the destination point of a unit depends on the chirality of the local coordinate system.

the concept of *left* and *right*. Hence, the basic algorithm applies only to formations that are symmetric with respect to the direction of movement of L.

7.4.1 Applicability of the Algorithm

The basic algorithm statistically solves the approximate undirected flocking problem. Furthermore, it statistically solves the approximate directed flocking problem, provided that

1. the pattern is centered on the leader and directed according to the $\overrightarrow{BL}$ axis;
2. the pattern $\mathbb{P} = \{p_1, \ldots, p_n, L\}$ is symmetric with respect to the axis passing through L and the baricenter of the other points in $\mathbb{P}$ (we call this kind of patterns *lb-symmetric*), and
3. the pattern contains at most 1 point lying on this axis.

In fact, if the pattern is not lb-symmetric, the target *slot* of a follower robot depends on whether the follower is in S_1 or S_2, and thus on whether it should try to form the *semi-patterns* F_1 or F_2. However, this assignment depends on the chirality of the local coordinate system (see Figure 7.6), and thus the followers cannot generally reach an agreement on the formation to keep.[3] Notice that if the pattern is lb-symmetric, the choice still cannot be made, but it becomes unrelevant since the two semi-patterns are indistinguishable.

Also, if the pattern is lb-symmetric but has more than one position lying on the $\overrightarrow{BL}$ axis (see Figure 7.7), two vehicles could be ranked equal in step 24 of

[3]In particular, no agreement can be reached if all the vehicles occupy positions that are symmetric with respect to the $\overrightarrow{BL}$ axis, while the pattern is not symmetric. See Section 7.6.4 for a more detailed discussion on this case and for possible solutions.

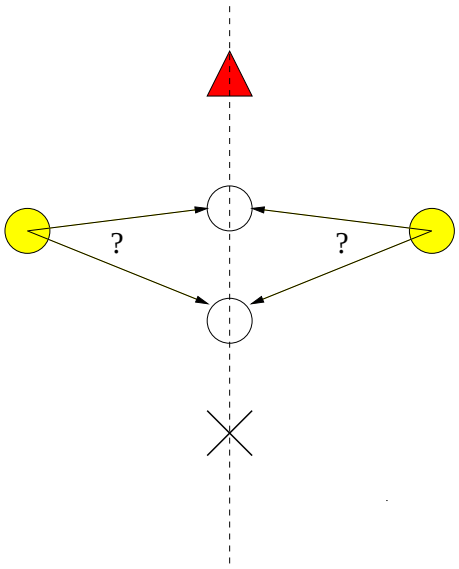


Figure 7.7: More than 1 position on the axis.

the algorithm. In this case, the two vehicles would select the same slot on F_0 as destination, preventing the correct formation of the pattern. Notice that for any number $m > 2$ of slots on the axis, the same problem could happen when $m - 2$ slots are taken by as many vehicles, while the two remaining vehicles could occupy symmetric positions as shown in Figure 7.7.

7.4.2 Analysis of Experiments

To measure how far the vehicles are at time t from the aimed formation, we use the following functions:

- $\Delta_e(t) = \mathcal{D}(\mathbb{E}(t), \mathcal{T}^{\phi_B(t)}_{\mathbb{P},\mathbb{E}(t)})$ the distance from the estimated formation, obtained from the position of the baricenter at time t. In particular, $\mathbb{E}(t)$ denotes the positions of the robots at time t, and $\mathcal{T}^{\phi_B(t)}_{\mathbb{P},\mathbb{E}(t)}$ is the directed target obtained by translating the leader of $\mathbb{P}$ onto the leader of $\mathbb{E}(t)$, and by rotating $\mathbb{P}$ of an angle $\phi_B(t)$ such that the baricenter of $p_1, \ldots, p_{n-1}$ lies on the line passing through the leader of $\mathbb{E}(t)$ and the baricenter of the followers in $\mathbb{E}(t)$; and
- $\Delta_r(t) = \mathcal{D}(\mathbb{E}(t), \mathcal{T}^{\psi_L(t)}_{\mathbb{P},\mathbb{E}(t)})$ the distance from the real formation, obtained taking into account the actual heading of the leader at time t, $\psi_L(t)$.

In Figure 7.11, some plots of Δ_e and Δ_r are reported, relative to computer simulations run with six followers trying to keep a wedge shaped formation, four in a line, ten in a spread formation (all shown in Figure 7.8), and with random formations.

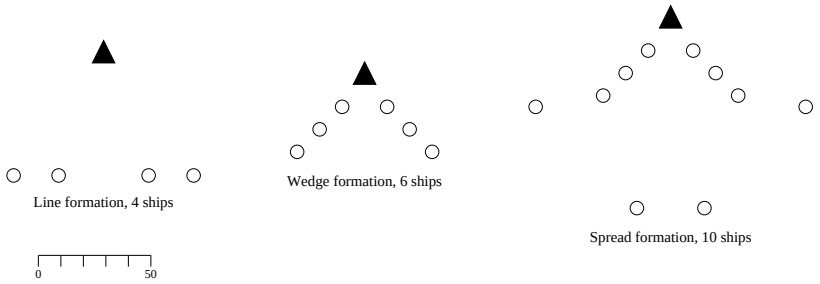


Figure 7.8: Fleet formations used in the simulations.

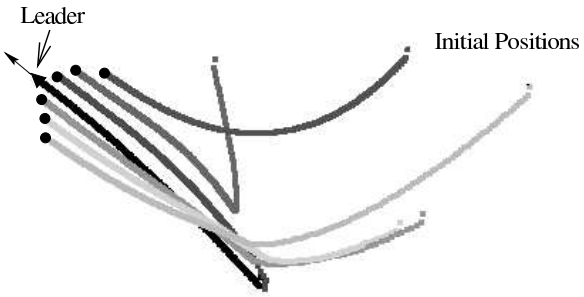


Figure 7.9: Trace of the vehicles while forming and keeping a wedge shaped formation.

In all cases, the simulations[4] started with all the vehicles (leader and followers) randomly placed in a square area with a side of 512 units. Each vehicle also had a random direction and orientation of the axes for its local coordinate system. Each follower had a random velocity between 0.5 and 5.0 units/move, while the leader's speed was determined in accordance with the limitations stated in Section 7.3.2. The fixed formations used in the experiments are shown in Figure 7.8; random formations were obtained by randomly choosing from two to eight symmetric points in the area delimited by the points (-150,-50) and (-50,+50), where (0,0) represents the leader and the axes are oriented so that x coincides with the leader's direction.

The leader's course was determined as follows: at all times, the leader would move forward according to its velocity. At each move, with a probability of 1/20, the leader could start turning to its left or right with an angular speed limited again according to Section 7.3.2. If already turning, with the same probability the leader could stop and continue its course as a straight line. Figures 7.9 and 7.10 show the courses of the vehicles in two simulation runs.

In Figure 7.11, it can be observed how convergence to the estimated formation (Δ_e) is obtained on average in less than half the time needed to reach the real

[4]The simulator has been written by Vincenzo Gervasi - University of Pisa.

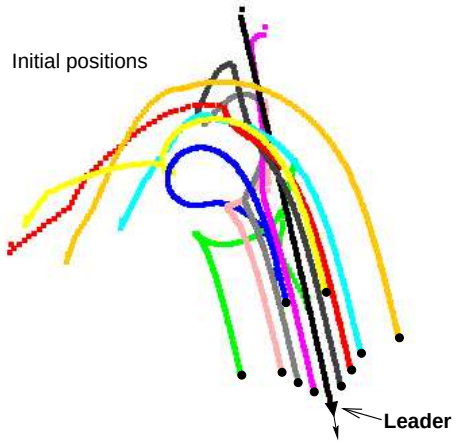


Figure 7.10: Trace of the vehicles while forming and keeping the spread formation with ten vehicles. Note the circular trajectory of the vehicles at the beginning, while trying to align the formation with the course of the leader.

formation. Figure 7.11 also reveals other interesting phenomena. In the Δ_r graph for the wedge formation, we can observe a "plateau" caused by the vehicles forming the pattern in exactly the wrong direction, i.e. in front of the leader rather than behind it. This correctly formed, but incorrectly-headed pattern, is kept until the leader starts changing its direction, at which point all the followers rapidly reach their proper positions behind the leader. Related effects can also be observed in the Δ_r graphs for the other formation, in which instabilities in the distance are caused by the followers trying to catch up with change of directions of the leader.

All our experiments demonstrated that the algorithm is properly behaved, and in all cases the followers were able to assume the desired formation and to maintain it while following the leader vehicle along its route. Indeed, while Section 7.3.2 provides conditions under which the ability of the vehicles to follow the leader is guaranteed, even when those conditions were relaxed the followers were usually able to compensate for sudden turns or accelerations of the leader (as long as the effective speed of the leader remained, at least on average, lower than that of the slowest follower). As a further remark, we note that the obliviousness of the algorithm contributes to this result, since the followers do not base their computation on past leader's positions.

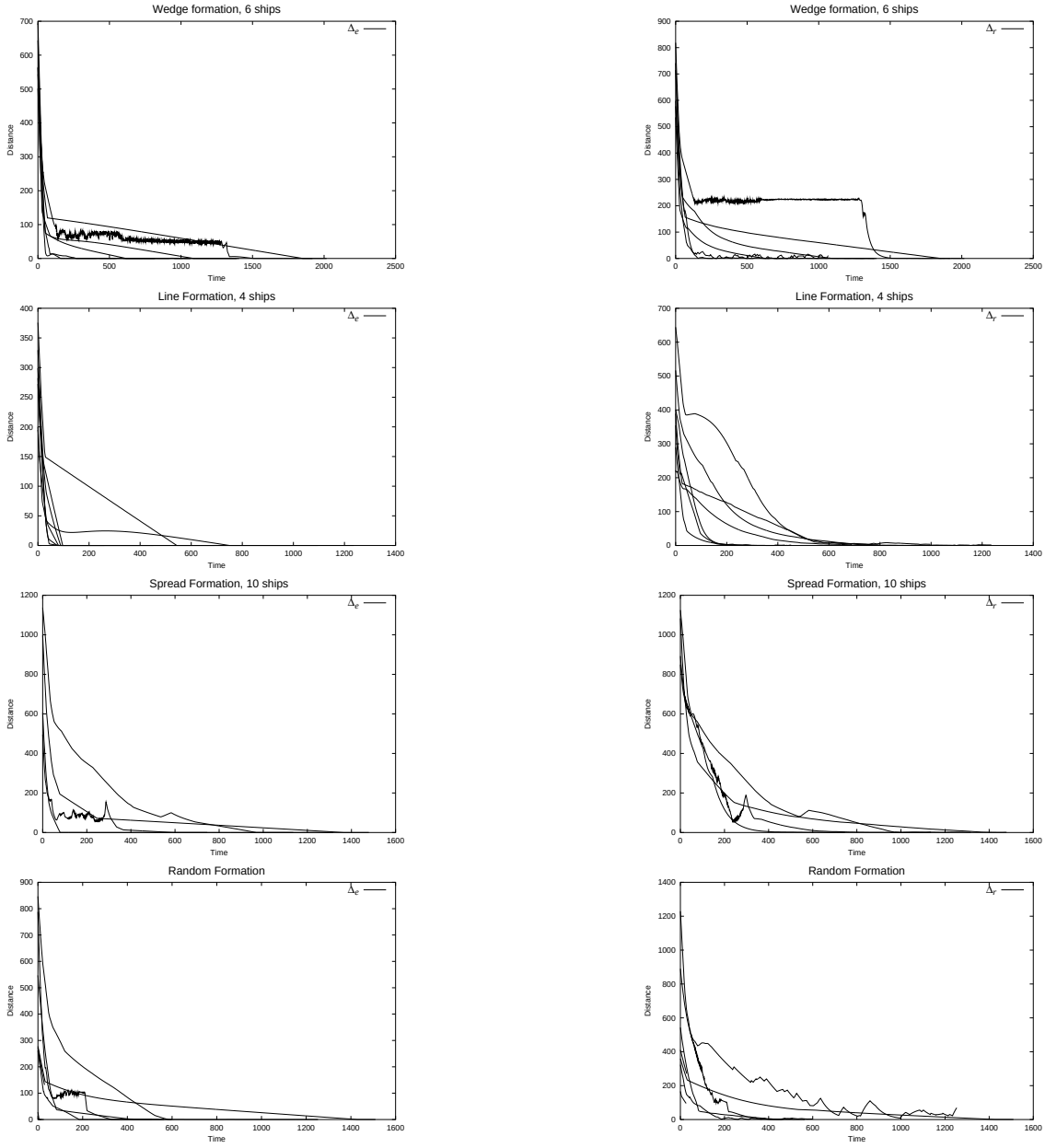


Figure 7.11: Some plots of the Δ_e and Δ_r, while forming a *wedge* of six vehicles, a *line* of four vehicles, a spread formation with ten vehicles, and random formations (with a number of vehicles variable between two and eight).

7.5 Extended Algorithms

The algorithm given above does not guarantee convergence, although simulations show that it provides statistical convergence in most cases. In this section we briefly discuss the problems with the basic algorithm, and provide two slightly more complex variations that solve these problems.

7.5.1 Problems with the Basic Algorithm

The basic algorithm suffers from a number of problems, and is subjected to somewhat restrictive conditions. In particular:

1. The basic algorithm converges rapidly only for approximate undirected flocking, while convergence in the case of approximate directed flocking is typically slower. This is caused by the followers' inability to observe the real heading of the leader (and by their obliviousness, since they cannot remember the previous position of the leader).

2. The followers can assume and maintain for an unpredictably long time a wrong formation. For instance, the followers can assume a formation that is specular to the correct one, and placed "in front" of the leader instead of behind it. In such a situation, as long as the leader maintains an heading that coincides with the $\overrightarrow{BL}$ axis, the followers will compensate any movement of the leader towards them by moving farther away, while keeping the formation on the wrong side and thus reproducing the same situation.

Problems 1 and 2 can be solved perfectly by observing the heading of the leader (i.e., by being able to distinguish the prow of the leader from the back), or — with a better approximation w.r.t. the baricenter — by having enough memory to store the previous position of the leader.

3. In certain situations, two or more vehicles could continue changing the *slots* they have to reach, causing instability and slowing down or impeding altogether the convergence of the algorithm.

In the following, we discuss some suggestions that can help in fixing some of the problems pointed out with Algorithm 8. Unfortunately, we did not implement this ideas yet; hence, we can not present evidence of the goodness of such strategies. This will be part of our near future work.

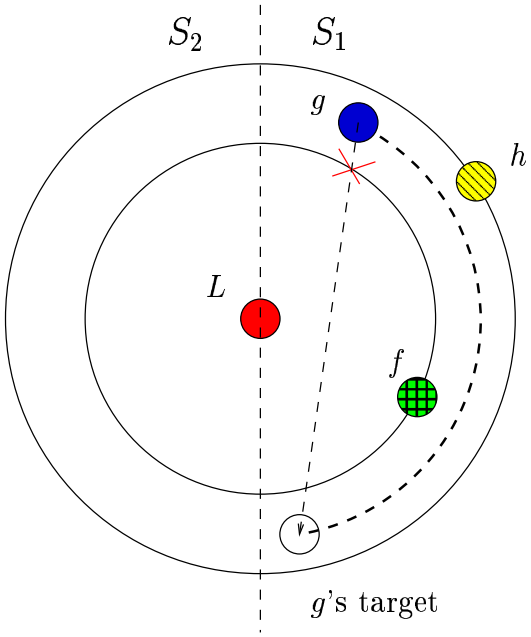


Figure 7.12: Strategy for the hula-hoop algorithm.

7.5.2 The Hula-Hoop Algorithm

A source of instability in the basic algorithm lies in the fact that the ranking of the vehicles in a *semi-space* (S_1 or S_2) can change during the execution. As a consequence, the assignment of the slots to the vehicles can change, and this in turn can cause sudden changes of direction of the followers. Although very rare, it is also theoretically possible that, in pursuing the new slots, two followers keep exchanging their ranking, so that the flock never stabilizes.

One way to solve this problem consists in ensuring that the initial rankings among the followers never change during the execution of the algorithm.

In the basic algorithm, the ranking is assigned based on the lexicographical ordering that is given by the distance from the leader and from the baricenter. Let f, g and h be the $(i-1)$-th, i-th, and $(i+1)$-th vehicle in the ranking, respectively (refer to Figure 7.12). The area in which the ranking of g does not change is given by the region delimited by the circles having as center the leader L, and as radius $dist(L, f)$ and $dist(L, h)$ respectively (the obvious extensions apply if g is the first or last vehicle in the ranking). We call this region the *stable space* of g.

To try to maintain a stable ranking, each follower has to remain always in its stable space. This entails:

1. stopping before crossing the stable space boundary if the target is outside the

stable space (waiting until the movements of other vehicles have changed the boundaries, possibly bringing the target inside the stable space), and

2. choosing curved trajectories instead of straight ones to reach a target that is inside the stable space when the straight trajectory would cross the boundaries.

While the above strategy increases the stability of the algorithm, it is not sufficient to guarantee it. In fact, it is possible that the movement of the leader, by changing the distances between the leader and the followers, changes the ranking of the followers even if they try to stay inside their stable spaces.

7.5.3 The Stripe Algorithm

A second variant of our basic algorithm increases the stability by changing the measure upon which the ranking is based rather than by changing the followers' strategies, as done with the hula-hoop algorithm.

The stripe variant uses a lexicographical ordering of the vehicles in a semi-space based on the distance from the Y axis, from the leader, and from the baricenter. More specifically, the `Sort(`A, l, b`)` routine in the basic algorithm is changed so that, after the sorting,

$$\begin{aligned} \forall i, j,\ i < j \quad \Rightarrow \quad & (dist(Y, p_i) > dist(Y, p_j)) \vee \\ & (dist(Y, p_i) = dist(Y, p_j) \wedge dist(l, p_i) > dist(l, p_j)) \vee \\ & (dist(Y, p_i) = dist(Y, p_j) \wedge dist(l, p_i) = dist(l, p_j) \wedge \\ & \qquad dist(b, p_i) > dist(b, p_j)) \end{aligned}$$

where $dist(Y, p)$ is the distance between the Y axis (passing through l and b) and the point p.

In this case, the stable space of a follower is defined as follows (refer to Figure 7.13). Let f, g and h be the $(i-1)$-th, i-th, and $(i+1)$-th vehicle in the ranking, respectively. The area in which the ranking of g does not change is given by the stripe parallel to the Y axis and delimited by the lines, also parallel to the Y axis and passing through f and h, respectively (again, the obvious extensions apply if g is the first or last vehicle in the ranking).

Also in this case, the followers must take care not to deliberately cross their stable space boundaries. However, since the stripes are oriented according to the (estimated) direction of the leader, there is lesser risk that the movements of the leader can change the ranking and thus introduce instabilities. On the other hand, this variant is more sensible to variations of the *heading* of the leader, especially when the distance between the followers and the leader is greater than a certain threshold,

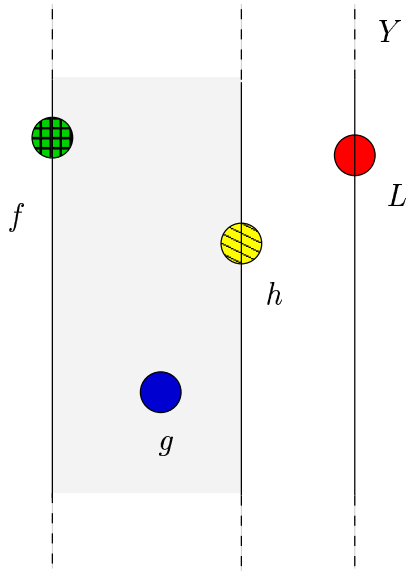


Figure 7.13: Stable spaces in the stripe algorithm.

since the stripes can swipe rapidly. However, this effect is only significant if the *real* heading can be observed. When the heading is estimated via the baricenter, sudden change of estimated heading are possible only when the baricenter is near the leader, and this in turn typically only happens when the follower are near the leader, thus rendering the effects of the swipe less important.

Notice however that there is no need to follow non-rectilinear trajectories. This variant of the algorithm is thus simpler than the hula-hoop one.

7.6 Discussion

As can be expected, the amount of shared knowledge among the followers and their computational and observational capabilities condition the type of problems that can be solved. In this section, we analyze how different combinations of shared knowledge, observation abilities and amount of memory influence the kind of flocking problems that can be solved by the followers.

7.6.1 Observability

Our model already assures us that the positions of the leader and of the followers are observable by any follower, and all the algorithms presented so far only rely on this simple capability. If, in addition to that, we can also observe the heading of the leader (e.g., by observing the direction of its prow), we can avoid relying on the

self-stabilization of a reference point (i.e., the baricenter) "behind" the leader, and lying along its direction of movement.

In particular, the followers can compute the *slots* to reach according exactly to the current direction of the leader, rather than approximating it based on the $\overrightarrow{BL}$ axis. Thus, it never happens that the followers stabilize on wrong directions (e.g., in front of the leader rather than behind it), and convergence is faster (actually, Δ_r coincides with Δ_e).

We do not discuss here whether being able to observe the direction of movement of the followers can improve[5] further the convergence.

7.6.2 Local Coordinate Systems

We distinguish four different levels of agreement on the direction and orientation of the coordinate axes.

No Agreement

The basic algorithm we presented in Section 7.4 does not assume any agreement among the followers on the direction and orientation of the axes. In this case, we have that:

- The basic algorithm and its variants in Section 7.5 solve the approximate undirected flocking problem if the input pattern is lb-symmetric and contains at most 1 point lying on the $\overrightarrow{BL}$ axis.

- They also solve the approximate directed flocking problem, assuming that the leader does not indefinitely keep the same direction. In this case convergence is slower than in the previous one.

- If the pattern is lb-symmetric, but contains more than 1 point on the $\overrightarrow{BL}$ axis, it may not converge.

- *Any* deterministic algorithm may not converge if the pattern is not lb-symmetric.

- A non-deterministic algorithm may reduce the probability of non-convergence to an arbitrarily small amount (see discussion in Section 7.6.4)

[5]This would allow some estimation of where the other followers will be at some future time.

One Axis Direction and Orientation Agreement

In the case in which the followers agree only the *direction* of a single axis[6] (and not on its orientation), they can use it and the $\overrightarrow{BL}$ axis to obtain a chirality (e.g., by assuming that the clockwise direction for angles goes from $\overrightarrow{BL}$ towards the acute angle between $\overrightarrow{BL}$ and the shared axis x). However, if one of the two shared axes coincides with the $\overrightarrow{BL}$ axis, no chirality can be obtained. Thus, in general, having agreement on one axis direction does not improve the capability of the vehicles to solve the flocking problem.

It should be noted, however, that even if the two axes coincide, the followers could simply wait (by executing a *Move* towards their current position) until a move by the leader or by a fellow follower breaks the tie.

Once a chirality is obtained, it can be used to establish a shared orientation on the common axis. Thus, having agreement on the orientation of the shared axis at the beginning of the computation does not improve the followers' capability to solve the problem.

Chirality

The followers can observe L and B, and assume that $\overrightarrow{BL}$ is a shared Y axis. Then, given the chirality, all the followers can agree that a shared X axis is oriented according to the clockwise direction and assume B as the origin. Thus, given the chirality a complete shared coordinate system can be established.

Two Axes

If all the followers agree on the direction and orientation of both axes, any tie condition can be broken. In particular, the followers can form non lb-symmetric patterns, and they can also form patterns with more than one slot on the Y axis.

7.6.3 Memory

So far we have discussed the case of oblivious algorithms, i.e. the robots cannot use any memory to store information about previous observations or decisions taken. In the following, the consequences of allowing bounded storage capabilities are discussed.

In particular, if the vehicles can store 1 position (e.g., two real numbers), the heading of the leader can be inferred by storing the position of the leader at the time of the last observation and considering the movement vector to the position

[6]Hence, they agree on the direction of both axes.

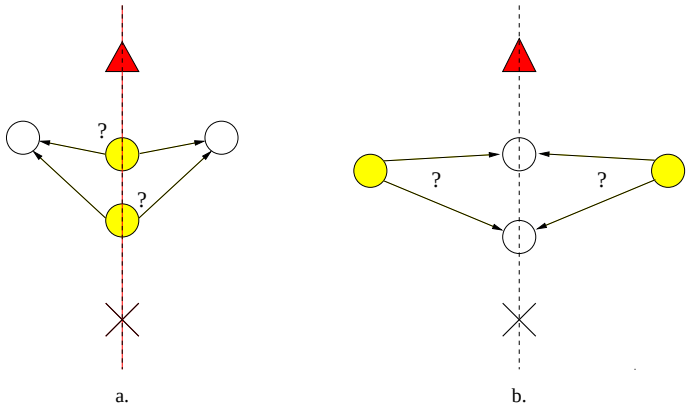


Figure 7.14: Cases of undeterminatedness with lb-symmetric formations with more than 1 point on the BL axis.

observed in the current cycle. In this case, what said in Section 7.6.1 about being able to observe the real heading of the leader applies.

It is interesting to observe that even with an unbounded amount of memory (non-oblivious algorithm), the followers cannot form patterns that are not lb-symmetric. In fact, consider the case in which the initial configuration of the vehicles is lb-symmetric, while the pattern to be formed is not. Given a follower v, we call its *buddy* v' the follower that occupies the position symmetric to that of v in the other semi-space. If, for each follower, its buddy has the same velocity and moreover their look-compute-move cycles are perfectly synchronized, i.e. they execute the algorithm according to a synchronous activation schedule, we end up in configurations that maintain the symmetry for an indefinitely long time.[7] Thus, any number of observations will not provide a means to break the symmetry, hence storing them does not help.

7.6.4 Randomization

If we can assume that the vehicles have the ability to take random choices (i.e., they are equipped with a proper entropy source), a larger class of problems can be solved. In particular, Algorithm 8 can be slightly modified in order to make it applicable also to formations that are not lb-symmetric or have more than one point lying on the $\overrightarrow{BL}$ axis.

In the latter case, two difficult cases arise, illustrated in Figure 7.14:

a. when two vehicles (or more) occupy positions lying exactly on the $\overrightarrow{BL}$ axis, and two (or more) symmetrical slots are available, one on each semi-space.

[7] Provided, naturally, that the algorithm employed by the followers is deterministic.

Lacking a shared direction for the x axis, the vehicles cannot agree on which of them should move towards the left-side slot and which towards the right-side one, and

b. when two vehicles (or more) occupy positions that are symmetric w.r.t. the $\overrightarrow{BL}$ axis, and two (or more) slots on the $\overrightarrow{BL}$ axis are available. Lacking a shared direction for the x axis, the vehicles cannot agree on which of them should move towards which slot.

In such cases, our basic algorithm always chooses a fixed slot as target: towards the available slot in the *local* F_1 in case a., towards the slot closest to the leader in case b. Obviously, this strategy does not produce an assignment of vehicles to slots that covers all the slots. In practice, this "tie" conditions are usually broken by differences in the velocity of the vehicles and by the asynchronicity of their cycles.

However, a stronger guarantee can be obtained by randomly choosing the target slot among the possible candidates in steps 29 and 40 of the algorithm. In the simplest case, involving only two slots, the probability of two vehicles always choosing the same target after k cycles is $1/2^k$. Even if the vehicles have the same velocity and synchronized cycles, as soon as they choose different target slots the symmetry is broken and the algorithm can continue normally.

Randomization can be used also in the case of non lb-symmetric formations. In this case, the algorithm for a follower f can

1. rank the semi-patterns according to some metric;
2. rank the semi-spaces according to the same metric;
3. assign the first semi-space to the first semi-pattern, and the second semi-space to the second semi-pattern;
4. rank the vehicles in the semi-space that contains f, and the slots in the corresponding semi-pattern;
5. choose a target for f in the semi-pattern corresponding to the semi-space that contains f, as done in the basic algorithm.

One way to compare configurations for the purpose of ranking semi-patterns and semi-spaces is defined by the following procedure:

`Ranking`

Input: Two configurations C and D with $|C| = |D|$, and two points L and B.
Output: $+1$ if C ranks higher than D, 0 if they rank equal, -1 otherwise.

```
Sort(C, L, B);
Sort(D, L, B);
i := 1;
While i ≤ |C| Do
    If (dist(c_i, L) > dist(d_i, L)) ∨ (dist(c_i, L) = dist(d_i, L) ∧ dist(c_i, B) >
    dist(d_i, B)) Then
        Return +1;
    If (dist(c_i, L) < dist(d_i, L)) ∨ (dist(c_i, L) = dist(d_i, L) ∧ dist(c_i, B) <
    dist(d_i, B)) Then
        Return −1;
    i := i + 1;
End While
Return 0;
```

Notice that the semi-patterns will always rank differently (since the pattern is asymmetric). It can happen, however, that the semi-spaces compare equals, that is, all the vehicles occupy positions that are symmetric w.r.t. the $\overrightarrow{BL}$ axis. In this case, the vehicles cannot agree on the assignment of semi-spaces to semi-pattern. Vehicles observing such a situation can, however, simply make a *Move* towards a random point in order to break the tie. As in the previous case, the probability of a tie continuing after k decisions is $1/2^k$.

7.7 Conclusions

In this chapter we have presented a novel algorithm for coordinating a set of non-communicating, asynchronous and memoryless vehicles into following a leader while keeping a fixed formation. The algorithm only assumes the vehicles share a common unit of distance, but no common sense of direction (i.e., a common coordinate system) is needed, nor any previous knowledge of the path the leader will follow. Moreover, the followers do not need to have an identity, and are not distinguishable in any way one from the other.

Indeed, the algorithm we propose exhibits remarkable robustness, and numeric simulations indicate that in most cases the formation is reached in a relatively short time and kept after that, as desired.

Chapter 8

Conclusions

A conclusion is the place where you got tired of thinking.
Martin H. Fischer

In this thesis, we provided a theoretical framework, CORDA, to model a distributed system whose entities are devices equipped with computational capabilities, with sensors, and able to freely move on a two dimensional plane. They are autonomous, anonymous, and with no means of direct communication - features that render them rather "weak". Their aim is to accomplish some given task all together and in finite time.

The purpose of this study is to gain a better understanding of the power of distributed control. In contrast with the majority of the research that deals with the study of a set of mobile agents that can independently move on a plane from an empirical point of view, in this thesis we approached the problem differently: from an **algorithmic** and **computational complexity** perspective. Namely, we aim to highlight under which conditions the agents are able to solve a given problem, and then design a distributed algorithm executed by all the agents that allows them to accomplish the given task in finite time. It is then clear that we intentionally make these extreme assumptions on the robots so we can better understand what kind of basic functionality a set of agents must have in order to accomplish a given task in a distributed fashion. By assuming the "weakest" agents, we can analyze in greater detail the strengths and weaknesses of the distributed control; furthermore, this approach allows us to highlight the set of agents' capabilities that are necessary to accomplish a certain task. The only other study, to our knowledge, that approaches the problem addressed in this thesis from this perspective is that of Suzuki *et. al.* In Chapter 3 we highlighted the main differences between the two approaches, showing that the different way the asynchronicity is modeled renders the algorithms designed in SYm useless in CORDA.

The research is still focusing on basic tasks such as *gathering*, *flocking*, *pattern formation*, *scattering*, etc. [9, 36, 52, 57, 78]. In the second part of this thesis we indeed studied three of these problems.

First, we have solved the problem of getting the robots to form an arbitrary pattern whose description is common to the agents in the system. We concluded that whether an arbitrary pattern is formable or not depends on the level of agreement the robots have on the orientation of the local coordinate systems. We have thus characterized the kind of patterns that can be formed with different amount of common knowledge to the agents in the system.

Second, we have analyzed the gathering problem for groups of autonomous mobile robots, and presented a gathering algorithm which solves the problem within *finite time*. Our solution operates in a *fully asynchronous* settings and can be achieved by robots with limited visibility, provided they have a common orientation.

Third, we studied the flocking problem, where the robots are asked to follow a leader while keeping a predetermined formation. We gave a rather simple algorithm that allows the robots to move while keeping symmetric formations; moreover, we tested our solution by means of computer simulations.

The results presented here are a further step towards a better understanding of the minimal robot capabilities enabling the collective resolution of a given global task. Furthermore, we have shown that from an algorithmic point of view, only the most fundamental aspects of mobile robot coordination are being understood Consequently, we are left with many issues regarding more detailed aspects of this area which merit further research. First, the problems left open in this thesis.

i. We showed in Chapter 3 the relationship between the class of problems solvable in SYm and the class of problems solvable in Corda. Part of the proofs, however, works in the non oblivious setting. Hence, it would be interesting to understand what kind of relationship exists between the two classes of problems in the totally oblivious scenario.

ii. Given that we have already shown that an arbitrary pattern cannot always be formed, it would be interesting to further investigate which patterns or classes of patterns could be formed and under which conditions. This would be beneficial as it would indicate the types of agreement which could be reached, and therefore the types of tasks which could be performed.

iii. The solution to the gathering problem with limited visibility proposed in Chapter 5 shows how one assumption ("orientation") has at least the same computational power for the gathering problem as another property ("instantaneous action"). This fact raises many intriguing questions, including: Are there

other (simpler) properties comparable to these two? How do we compare different properties? Another interesting problem is under which conditions (if any) this problem has a solution in the unlimited visibility setting. In fact, as already stated in Chapter 1, the gathering problem in this setting has a simple solution: all the robots can gather in the *Weber Point*, the point in the plane that, given n distinct points, minimizes the distances from them. Unfortunately, it has been proven that such a point is not computable. Our current solutions to this problem are not very easy and do not work for some particular initial configurations of the robots.

Furthermore, the results of this thesis open many interesting research directions, among them:

- the comparison of different solutions for the same set of robots. So far, in fact, no complexity measure has been proposed;
- the study of coordination problems under relaxed assumptions about the robots' capabilities. For instance, we could use a totally *non-oblivious* model, that is, one with an unlimited amount of memory that each agent could use. Alternatively, we could equip the agents with just a bounded amount of memory (*semi-obliviousness*), and see if this added "power" can be useful in solving problems otherwise unsolvable; if it could be used to design faster algorithms; and how it could affect the self-stabilizing feature of the oblivious algorithms;
- the investigation of simple tasks under different conditions of the environment. For instance, alterations we could make to our system which would inspire further study include adding dimension to the agents and stationary obstacles to the environment, thus adding the possibility of collision between robots or between moving robots and obstacles; or assuming uneven terrains. Furthermore, we could add the possibility that at any given time a robot may never see the world as it actually is, since the robots' cameras rotate slowly as they are taking a picture. We could also explore agents that have some kind of direct communication, and we could assume different kind of agents that move in the environment.
- the study of the impact that sensorial errors, possibly arising during the *Look* and the *Move* state, have on the overall correctness of the algorithms;
- contrary to robots in other research which has looked at modeling natural behaviors, our robots perform quite a complex computation in each step. It would therefore be of interest to explore in more detail the tradeoff between computation complexity and knowledge of the world;

- the design of new algorithms under different assumptions on the visibility power of the robots (e.g., when the accuracy of the robots' ability to detect the other robots' positions decreases with the distance);

- the study of the impact of *energy saving* issues on the solvability of the problems [72]. In fact, we can imagine that our agents are battery powered, hence they have limited life. Therefore, it would be interesting to analyze the aspects of the model where energy saving issues play some role. For instance, we can model sensors that consume more energy the farther they sense, and study the impact of this feature.

- the study of new problems like *scattering* (i.e., the robots start from the same location and their goal is to evenly scatter on the plane), *rescue* (i.e., the robots have to find a small object which is not initially visible), *exploration* (i.e., the robots have to gather information about the environment, with the purpose, for example, of constructing a map), and many more.

Slightly faulty snapshots, a limited range of visibility, obstacles that limit the visibility and that moving robots must avoid or push aside, as well as robots that appear and disappear from the scene clearly suggest that the algorithmic nature of distributed coordination of autonomous, mobile robots merits further investigation.

Bibliography

[1] E. Abravanel and H. Gingold. Learning Via Observation During the Second Year of Life. *Developmental Psychology*, 21(4):614–623, 1985.

[2] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility. *IEEE Transaction on Robotics and Automation*, 15(5):818–828, 1999.

[3] H. Ando, I. Suzuki, and M. Yamashita. Formation and Agreement Problems for Synchronous Mobile Robots with Limited Visibility. In *Proceedings IEEE International Symposium on Intelligent Control*, pages 453–460, August 1995.

[4] G. Antonoiu and P. K. Srimani. A Self-stabilizing Leader Election Algorithm for Tree Graphs. *Journal of Parallel and Distributed Computing*, 34:227–332, 1996.

[5] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[6] R. C. Arkin and T. Balch. *Cooperative Multiagent Robotic Systems*, chapter in *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, D. Kortenkamp and R. P. Bonasso and R. Murphy edition, 1998.

[7] J. B. Asendorpf and P. M. Baudonniere. Self-Awareness and Other-Awareness: Mirror Self-Recognition Synchronic Imitation Among Unfamiliar Peers. *Developmental Psychology*, 29(1):89–95, 1993.

[8] T. Balch and R. C. Arkin. Motor Schema-based Formation Control for Multiagent Robot Teams. In *International Conference on Multi Agent Systems (ICMAS)*, pages 10–16, 1995.

[9] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Transaction on Robotics and Automation*, 14(6), December 1998.

[10] A. Bandura. *Social Learning Theory.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1977.

[11] D. Barnes and J. Gray. Behaviour Synthesis for Co-operant Mobile Robot Control. In *International Conference on Control*, pages 1135–1140, 1991.

[12] R. Beckers, O. E. Holland, and J. L. Deneubourg. From Local Actions To Global Tasks: Stigmergy And Collective Robotics. In The MIT Press, editor, *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 1994.

[13] G. Beni and S. Hackwood. Coherent Swarm Motion Under Distributed Control. In *Proceedings DARS'92*, pages 39–52, 1992.

[14] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. Technical Report AI-TR-864, MIT, September 1985.

[15] R. A. Brooks. Elephant Don't Play Chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.

[16] R. A. Brooks. Intelligence Without Reason. Technical Report AI-TR-1293, MIT, April 1991.

[17] R. A. Brooks. New Approaches to Robotics. *Science*, 253:1227–1232, September 1991.

[18] R. A. Brooks. From Earwigs to Human. *Robotics and Autonomous Systems*, 20(2–4):291–304, June 1997.

[19] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative Mobile Robotics: Antecedents and Directions. In *IEEE/TSJ International Conference on Intelligent Robots and Systems*, pages 226–234, 1995. Yokohama, Japan.

[20] S. Chandrasekar and P. K. Srimani. A Self-stabilizing Algorithm to Synchronize Digital Clocks in a Distributed System. *Computers and Electrical Engineering*, 20(6):439–444, 1994.

[21] N. Chen, H. Yu, and S. Huang. A Self-stabilizing Algorithm for Constructing Spanning Trees. *Information Processing Letters*, 39:147–151, August 1991.

[22] Q. Chen and J. Y. S. Luh. Coordination and Control of a Group of Small Mobile Robots. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2315–2320, 1994. San Diego, CA.

[23] D. L. Cheney and R. M. Seyfarth. *How Monkeys See the World.* The University of Chicago Press, Chicago, 1990.

[24] E. J. Cockayne and Z. A. Melzak. Euclidean Constructibility in Graph-minimization Problems. *Mathematical Magazine*, 42:206–208, 1969.

[25] X. A. Debest. Remark About Self-stabilizing Systems. *Communication of the ACM*, 238(2):115–117, February 1995.

[26] D. C. Dennet. *The Intentional Stance.* The MIT Press, Cambridge, Massachusetts, 1987.

[27] J. Desai, J. Ostrowski, and V. Kumar. Control of Formations for Multiple Robots. In *IEEE International Conference on Robotics and Automation*, May 1998.

[28] E. W. Dijkstra. Self-stabilizing Systems in Spite of Distributed Control. *Communication of the ACM*, 17(11):643–644, November 1974.

[29] S. Dolev. *Self-Stabilization.* The MIT Press, 2000.

[30] S. Dolev, M. G. Gouda, and M. Schneider. Memory Requirements for Silent Stabilization. In *PODC96 Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 27–34, 1996.

[31] B. R. Donald, J. Jennings, and D. Rus. Towards a Theory of Information Invariants for Cooperating Autonomous Mobile Robots. Technical Report TR 93-1383, Department of Computer Science, Cornell University, September 1993.

[32] B. R. Donald, J. Jennings, and D. Rus. Analyzing Teams of Cooperating Mobile Robots. Technical Report TR 94-1429, Department of Computer Science, Cornell University, June 1994.

[33] B. R. Donald, J. Jennings, and D. Rus. Information Invariants for Distributed Manipulation. *The International Journal of Robotics Research*, 16(5), October 1997.

[34] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On Cooperation in Multi-Agent Systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[35] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *International Conference on Multi Agent Systems (ICMAS)*, pages 406–413, 1995.

[36] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Hard Tasks for Weak Robots: The Role of Common Knowledge in Pattern Formation by Autonomous Mobile Robots. In *Proceedings* 10^{th} *Annual International Symposium on Algorithms and Computation (ISAAC '99)*, volume LNCS 1741, pages 93–102, December 1999.

[37] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IEEE Intelligent Veichle Symposium (IV 2000)*, pages 480–485, 2000.

[38] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Pattern Formation by Autonomous Mobile Robots. *Interjournal of Complex Systems*, Article, 395, 2000. http://www.interjournal.org.

[39] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots With Limited Visibility. In *Proceedings* 18^{th} *International Symposium on Theoretical Aspects of Computer Science (STACS 2001)*, volume LNCS 2010, pages 247–258, February 2001.

[40] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Pattern Formation by Autonomous Robots Without Chirality. In *Proceedings VIII International Colloquium on Structural Information and Communication Complexity (SIROCCO 2001)*, pages 147–162, June 2001.

[41] S. Franklin. Coordination Without Communication. http://www.msci.memphis.edu/ franklin/coord.html, 1999.

[42] T. Fukuda, G. Iritani, T. Ueyama, and F. Arai. Optimization of Group Behavior on Cellular Robotic System in Dynamic Environment. In *International Conference on Robotics and Automation*, pages 1027–1032, 1994.

[43] T. Fukuda, Y. Kawauchi, and H. Asama M. Buss. Structure Decision Method for Self Organizing Robots Based on Cell Structures-CEBOT. In *Proceedings IEEE International Conference on Robotics and Automation*, volume 2, pages 695–700, 1989.

[44] T. Fukuda and S. Nakagawa. Approach to the Dynamically Reconfigurable Robotic System. *Journal of Intelligent and Robotic System*, 1:55–72, 1988.

[45] T. Fukuda and S. Nakagawa. Dynamically Reconfigurable Robotic System. *Proceedings IEEE International Conference on Robotics and Automation*, pages 1581–1586, 1988.

[46] V. Gervasi and G. Prencipe. Flocking by a Set of Autonomous Mobile Robots. Technical Report TR-01-24, Università di Pisa, October 2001.

[47] V. Gervasi and G. Prencipe. Need a Fleet? Use The Force! In *FUN With Algorithms 2 (FUN 2001)*, pages 149–164, May 2001.

[48] S. Gosh and M. H. Karaata. A Self-stabilizing Algorithm for Coloring Planar Graphs. *Distributed Computing*, 7:55–59, 1993.

[49] P. P. Grassè. La Theorie de la Stigmergie: Essai D'Interpretation des Termites Constructeurs. *Insect Society*, 6:41–83, 1959.

[50] J. Halpern and Y. Moses. Knowledge and Common Knowledge in a Distributed Environment. In *Proceedings of the* 3^{rd} *ACM Symposium on Principles of Distributed Computing*, pages 50–61, 1984.

[51] J. L. Welch J. E. Walter and N. M. Amato. Distributed Reconfiguration of Metamorphic Robot Chains. In *Proceedings of the* 19^{th} *ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC '00)*, pages 171–180, 2000.

[52] D. Jung, G. Cheng, and A. Zelinsky. Experiments in Realising Cooperation between Autonomous Mobile Robots. In *Fifth International Symposium on Experimental Robotics (ISER)*, June 1997. Barcelona, Catalonia.

[53] Y. Kawauchi and M. Inaba and. T. Fukuda. A Principle of Decision Making of Cellular Robotic System (CEBOT). In *Proceedings IEEE Conference on Robotics and Automation*, pages 833–838, 1993.

[54] Y. Kawauchi, M. Inaba, and T. Fukuda. A Principle of Distributed Decision Making of Cellular Robotic System (CEBOT). In *IEEE IRCA*, volume 3, pages 833–838, 1993.

[55] M. Nilsson M. J Matarić and K. T. Simsarian. Cooperative Multi-Robot Box-Pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 556–561, 1995.

[56] M. J Matarić. Minimizing Complexity in Controlling a Mobile Robot Population. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 830–835, 1992. Nice, France.

[57] M. J. Matarić. Designing Emergent Behaviors: From Local Interactions to Collective Intelligence. In *From Animals to Animats 2: Int. Conf. on Simulation of Adaptive Behavior*, pages 423–441. The MIT Press, 1993.

[58] M. J Matarić. *Interaction and Intelligent Behavior*. PhD thesis, MIT, May 1994.

[59] M. J Matarić. Behavior-Based Primitives for Articulated Control. In *From Animal to Animats 5, Fifth International Conference on Simulation of Adaptive Behavior*, pages 165–170. MIT Press, Cambridge, 1998.

[60] M. J. Matarić. Great Expectations: Scaling Up Learning by Embracing Biology and Complexity. In *NFS Workshop on Development and Learning*. Michigan State University, April 2000.

[61] D. McFarland. *Animal Behaviour: Psychobiology, Ethology and Evolution*. Longman, 1999. 3rd Edition.

[62] S. Murata, H. Kurokawa, and S. Kokaji. Self-Assembling Machine. In *Proceedings IEEE Conference on Robotics and Autom.*, pages 441–448, 1994.

[63] F. R. Noreils. An Architecture for Cooperative and Autonomous Mobile Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2703–2710, 1992.

[64] F. R. Noreils. Toward a Robot Architecture Integrating Cooperation between Mobile Robots: Application to Indoor Environment. *The International Journal of Robotics Research*, pages 79–98, 1993.

[65] Y. Oasa, I. Suzuki, and M. Yamashita. A Robust Distributed Convergence Algorithm for Autonomous Mobile Robots. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 287–292, October 1997.

[66] L. E. Parker. Adaptive Action Selection for Cooperative Agent Teams. In *Proceedings of Second International Conference on Simulation of Adaptive Behavior*, pages 442–450. MIT Press, 1992.

[67] L. E. Parker. On the Design of Behavior-Based Multi-Robot Teams. *Journal of Advanced Robotics*, 10(6), 1996.

[68] S. Premvuti and S. Yuta. Consideration on the Cooperation of Multiple Autonomous Mobile Robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 59–63, 1990.

[69] G. Prencipe. CORDA: Distributed Coordination of a Set of Autonomous Mobile Robots. In *Proceedings Fourth European Research Seminar on Advances in Distributed Systems (ERSADS 2001)*, pages 185–190, May 2001.

[70] G. Prencipe. Instantaneous Actions vs. Full Asynchronicity: Controlling and Coordinating a Set of Autonomous Mobile Robots. In *VII Italian Conference on Theoretical Computer Science (ICTCS 2001)*, pages 185–190, October 2001.

[71] M. Schneider. Self-stabilization. *ACM Computing Surveys*, 25(1), March 1993.

[72] G. J. Smit and P. J. M. Havinga. A survey of energy saving techniques for mobile computers, 1997. Internal Report, University of Twente.

[73] L. Steels. The Artificial Life Roots of Artificial Intelligence. *Artificial Life Journal*, 1:75–110, 1994.

[74] K. Sugihara and I. Suzuki. Distributed Motion Coordination of Multiple Mobile Robots. In *Proceedings* 5^{th} *IEEE International Symposium Intelligent Control*, pages 138–143, 1990.

[75] K. Sugihara and I. Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[76] I. Suzuki and M. Yamashita. Formation and Agreement Problems for Anonymous Mobile Robots. In *Proceedings of the* 31^{st} *Annual Conference on Communication, Control and Computing*, pages 93–102. University of Illinois, Urbana, IL, 1993.

[77] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots - Formation and Agreement Problems. In *Proceedings Third Colloquium on Structural Information and Communication Complexity (SIROCCO 1996)*, pages 313–330, 1996.

[78] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam Journal of Computing*, 28(4):1347–1363, 1999.

[79] P. K. C. Wang. Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[80] A. Weber. Theory of the Location Industries. *The University of Chicago Press*, 1937.

[81] E. Weiszfeld. Sur le Point Pour Lequel la Somme Des Distances de n Points Donnés Est Minimum. *Tohoku Mathematical*, 43:355–386, 1936.

[82] E. Welzl. Smallest Enclosing Disks (Balls and Ellipsoids). *Lecture Notes in Computer Science*, 555:359–370, 1991.

Appendix A

Oblivious Gathering in SYm

A.1 Unlimited Visibility

In order to prove that the algorithmic solution presented in [78] does not work in CORDA, in the following we report the oblivious algorithm described in [78] to let the robots achieve a situation where a unique point p with multiplicity greater than one is determined.

Algorithm 1 (Point Formation Algorithm in SYm, Unlim. Visib. [78]).

Case 1. $n = 3$; p_1, p_2, and p_3 denote the positions of the three robots.

1.1. If $n = 3$ and p_1, p_2, and p_3 are collinear with p_2 in the middle, then the robots at p_1 and p_3 move towards p_2 while the robot at p_2 remains stationary. Then eventually two robots occupy p_2.

1.2. If $n = 3$ and p_1, p_2, and p_3 form an isosceles triangle with $dist(p_1, p_2) = dist(p_1, p_3) \neq dist(p_2, p_3)$, then the robot at p_1 moves toward the foot of the perpendicular drop from its current position to $[p_2p_3]$ in such a way that the robots do not form an equilateral triangle at any time, while the robots at p_2 and p_3 remain stationary. Then eventually the robots become collinear and the problem is reduced to part 1.1.

1.3. If $n = 3$ and the lengths of the three sides of triangle p_1, p_2, p_3 are all different, say $dist(p_1, p_2) > dist(p_1, p_3) > dist(p_2, p_3)$, t hen the robot at p_3 moves toward the foot of the perpendicular drop from its current position to $[p_1p_2]$ while the robots at p_1 and p_2 remain stationary. Then eventually the robots become collinear and the problem is reduced to part 1.1.

1.4. If $n = 3$ and p_1, p_2, and p_3 form an equilateral triangle, then every robot moves towards the center of the triangle. Since all robots can move up

to at least a constant distance $\sigma > 0$ in one step, if part 1.4. continues to hold then eventually either the robots meet at the center, or the triangle they form becomes no longer equilateral and the problem is reduced to part 1.2 or part 1.3.

Case 2. $n \geq 4$; C_t denotes the smallest enclosing circle of the robots at time t.

2.1. If $n \geq 4$ and there is exactly one robot r in the interior of C_t, then r moves toward the position of any robot, say r', on the circumference of C_t while all other robots remain stationary. Then eventually r and r' occupy the same position.

2.2. If $n \geq 4$ and there are two or more robots in the interior of C_t, then these robots move toward the center of C_t while all other robots remain stationary (so that the center of C_t remains unchanged). Then eventually at least two robots reach the center.

2.3. If $n \geq 4$ and there are no robots in the interior of C_t, then every robot moves toward the center of C_t. Since all robots can move up to at least a constant distance $\sigma > 0$ in one step, if part 2.3 continues to hold,then eventually the radius of C_t becomes at most σ. Once this happens, then the next time some robot moves, say, at t', either (i) two or more robots occupy the center of C_t or (ii) there is exactly one robot r at the center of C_t, and therefore there is a robot that is not on $C_{t'}$ (and the problem is reduced to part 2.1 or part 2.2) since a cycle passing through r and a point on C_t intersects with C_t at most at two points.

A.2 Limited Visibility

In order to prove that the algorithmic solution presented in [3] does not work in CORDA, in the following we report the oblivious algorithm described in [3] to let the robots gather in a point (refer to Figure 3.12.b).

Algorithm 2 (Point Formation Algorithm in SYm, Lim. Visib. [3]).

1. If $S_i(t) = \{r_i\}$, then $x = r_i(t)$.
2. $\forall r_j \in S_i(t) - \{r_i\}$,
   - 2.1. $d_j = dist(r_i(t), r_j(t))$,
   - 2.2. $\theta_j = c_i(t)\widehat{r_i(t)}r_j(t)$,

2.3. $l_j = (d_j/2)\cos\theta_j + \sqrt{(V/2)^2 - ((d_j/2)\sin\theta_j)^2}$,

3. $LIMIT = \min_{r_j \in S_i(t) - \{r_i\}}\{l_j\}$,

4. $GOAL = dist(r_i(t), c_i(t))$,

5. $MOVE = \min\{GOAL, LIMIT, \sigma\}$,

6. $x =$ point on $[r_i(t)c_i(t)]$ at distance $MOVE$ from $r_i(t)$.