Università di Pisa
Dipartimento di Informatica

Technical Report: TR–00–10

A New Distributed Model to Control and Coordinate a Set of Autonomous Mobile Robots: The Corda Model

Giuseppe Prencipe
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, 56100 - Pisa, Italy
e-mail: prencipe@di.unipi.it

August 17, 2000

A New Distributed Model to Control and Coordinate a Set of Autonomous Mobile Robots: The CORDA Model

Giuseppe Prencipe
Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, 56100 - Pisa, Italy
e-mail: `prencipe@di.unipi.it`

August 17, 2000

Abstract

Over the past few years, the focus of robotic design has been moving from a scenario where few, specialized (and expensive) units were used to solve a variety of tasks, to a scenario where many, general purpose (and cheap) units were used to achieve some common goal. Consequently, part of the focus has been to better understand how to efficiently coordinate and control a set of such "simpler" mobile units. Studies can be found in different disciplines, from engineering to artificial life: a shared feature of the majority of these studies has been the design of algorithms based on heuristics, without mainly being concerned with correctness and termination of such algorithms. Few studies have focused on trying to formally model an environment constituted by mobile units, studying which kind of capabilities they must have in order to achieve their goals; in other words, to study the problem from a *computational* point of view. This paper focuses on one of these studies, based on a model first introduced in [6]. First, its main features are described. Then, the main differences from a previous model [1, 14] (the only one, to our knowledge, that analyzes the problem of coordinating and controlling a set of autonomous, mobile units from this point of view), are highlighted, showing the novelty of this approach.

1 Introduction

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 several reasons: the advantages that can arise from a distributed and parallel solution to the given problems, such as a faster computation; the ability to perform tasks which are unable to be executed by a single agent; increased fault tolerance; and, the decreased cost through simpler individual robot design. On the other hand, the main concern in such a system is to find an efficient way to coordinate and control the mobile units, in order to exploit to the utmost the presence of many elements moving independently.

Leading research has been conducted in recent years in different fields. In the engineering area we can cite the Cellular Robotic System (CEBOT) of Kawaguchi *et al.* [8], the Swarm

Intelligence of Beni *et al.* [3], and the Self-Assembly Machine ("fructum") of Murata *et al.* [10]. In the AI community there has been a number of remarkable studies: social interaction leading to group behavior by Matarić [9]; selfish behavior of cooperative robots in animal societies by Parker [11]; and primitive animal behavior in pattern formation by Balch and Arkin [2].

The shared 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. In contrast, we want to analyze an environment populated by a set of autonomous, mobile robots, aiming to identify the algorithmic limitations of what they can do. In other words, we want to study the problem from a *computational point of view.* Specifically, we present a model, CORDA (COordination and control of a set of Robots in a totally Distributed and Asynchronous environment), that appeared for the first time[1] in [6], which has as its primary objective to describe a set of simple mobile units, which have no central control, hence move independently from each other, which are totally asynchronous, and which execute the same deterministic algorithm in order to achieve some goal. The robots we model are quite *weak* and simple, but this simplicity allows us to formally highlight by an algorithmic and computational viewpoint the minimal capabilities they must have in order to accomplish basic tasks and produce interesting interactions. Furthermore, it allows us to better understand the power and limitations of the distributed control in an environment inhabited by mobile agents, hence to formally prove what can be achieved under the "weakness" assumptions of CORDA, that will be described later in more detail (see [7] for more detailed motivations).

An investigation with an algorithmic flavor has been undertaken within the AI community by Durfee [5], who argues in favor of limiting the knowledge that an intelligent robot must possess in order to be able to coordinate its behavior with others. The work of Suzuki and Yamashita [1, 13, 14], however, is the 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, operating under several assumptions on the power of the individual robot. Although the model of Suzuki *et al.* (which we will refer to as SYm) and CORDA share some features, they differ in some aspects –specifically in the way the asynchronicity is modeled– that render the two models quite different. In this paper we highlight these differences, proving that the algorithms designed on SYm do not work on CORDA. Moreover, it will be shown that the features that render CORDA different from SYm render it more realistic, that is, in our opinion, it better models how a set of autonomous, mobile robots behave in a totally asynchronous environment.

In Section 2, SYm and CORDA are described, highlighting the features that render the two models different. In Section 3, we show that the algorithms designed in SYm do not work in CORDA, highlighting why some of the assumptions in CORDA better reflect a totally asynchronous environment populated by a set of mobile units, hence the novelty of this approach. Finally, in Section 4 we draw some conclusions and present suggestions for further study.

2 SYm vs. CORDA

The two models discussed in this paper share some basic features. The robots are modeled as units with computational capabilities, which are able to freely move in the plane. They

[1]This is the first time, however, that the model is baptized CORDA.

are viewed as points, and they are equipped with sensors that let them observe the positions of the other robots in the plane. Depending on whether they can observe all the plane or just a portion of it, two different models can arise: *Unlimited* and *Limited Visibility* model (each robot can see only whatever is at most at distance V from it). The robots are *anonymous*, meaning that they are a priori indistinguishable by their appearances, and they do not have any kind of identifiers that can be used during the computation. They are *asynchronous* and no central control is allowed. Each robot has its own *local view* of the world. This view includes a local Cartesian coordinate system with origin, unit of length, and the *directions* of two coordinate axes, identified as X axis and Y axis, together with their *orientations*, identified as the positive and negative sides of the axes. 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. They execute, however, the same deterministic algorithm, which takes in input the positions of the robots in the plane observed at a time instant t, and returns a destination point towards which the executing robot moves. The algorithm is *oblivious* if the new position is determined only from the positions of the others at t, and not on the positions observed in the past[2]; otherwise, it is called *non oblivious*. Moreover, there are no explicit means of communication: the 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.

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 in the studies done in [6, 14], 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 can also lead to some advantages. For example, avoiding the ability to remember what has been computed in the past gives the system the nice property of self-stabilization [7, 14].

During its life, each robot cyclically executes three *phases*: (i) it observes the positions of the others in the world, (ii) it computes its next destination point, and (iii) it moves towards the point it just computed. As already stated, the robots execute these phases *asynchronously*, without any central control: in this feature the two models drastically differ. In fact, in SYm phases (i) to (iii) are executed atomically, while this assumption is dropped in CORDA. In the following we better describe how the asynchronicity is approached in the two models.

2.1 The *atomicity of a cycle* of SYm

In this section we better describe how the movement of the robots is modeled in SYm [1, 14]. 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*, but this case is not interesting for the purpose of this paper.

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

[2]We also refer to the robots as *oblivious* because of this feature of the algorithms they execute.

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[3]), 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 $p_i(t+1) = p$, where p is the point returned by ψ. The maximum distance that r_i can move in one step is bounded by a distance $\epsilon_i > 0$ (this implies that every robot can travel at least a distance $\epsilon = \min\{\epsilon_1, \ldots, \epsilon_n\} > 0$). The reason for such a constant is to simulate a continuous monitoring of the world by the robots.

Thus, r_i executes the three phases (i)–(iii) *atomically*, 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*). If we call *cycle* the sequence of the phases (i), (ii) and (iii) described before, we have what we can call *atomicity of a cycle*. In our opinion, these assumptions poorly model the way a set of autonomous, mobile, and *asynchronous* robots interact and coordinate in order to accomplish some given task.

2.2 The *asynchronicity within a step* of CORDA

The first (small) difference between the two models, is that in CORDA there is the possibility that two robots, even if considered points, realize when they collide. That is, when a robot bumps into another one, it simply stops on the same position of the other robot. This feature, even if present in a model where the robots are seen as points, has been introduced because it renders CORDA closer to a model where each robot has a dimension, hence to the real world. The "bumping" assumption lets us easily solve a problem that Suzuki *et al.* showed to be not solvable in an oblivious way in SYm: the *gathering problem*[4], when we have only two robots [14]. In this problem, the two robots are asked to gather in a not-prefixed point in the plane in a finite number of cycles. To solve this problem in CORDA, it is sufficient that a robot, after having observed, simply moves towards the other robot it sees.

Similarly to SYm, each robot repeatedly executes four phases. A robot is initially in a waiting state (*Wait*); at any point in time, asynchronously and independently from the other robots, it observes the environment in its area of visibility (*Observe*), it calculates its destination point based only on the current locations of the observed robots (*Compute*), it then moves towards that point (*Move*) and goes back to a waiting state. The phases are described more formally in the following.

1. **Wait** The robot is idle. A robot cannot stay infinitely idle.

2. **Observe** The robot observes the world by taking a snapshot of the positions of all other robots with respect to its local coordinate system. Each robot r is viewed as a point, and therefore its position in the plane is given by its coordinates. In addition, the 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*. If, on the other hand, a robot can recognize that there is

[3]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.

[4]This problem is called *Point Formation Problem* by Suzuki *et al.*, [1, 14].

more than one fellow on the positions where it is, we say that it can detect a *weak multiplicity*.

3. **Compute** The robot performs a *local computation* according to its deterministic, oblivious algorithm. The result of the computation can be a destination point or `do_nothing()`.

4. **Move** If the result of the computation was `do_nothing()`, the robot does not move; otherwise it moves, along any curve it likes, towards the point computed in the previous phase. 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 along a curve, but it cannot know how far it will go in the current cycle, because it can stop anytime during its movement.

A computational cycle is defined as the sequence of the *Wait-Observe-Compute-Move* phases; the "life" of a robot is then a sequence of computational cycles.

In addition, we have the following assumptions on the behavior of a robot:

A1 (No Infinite Sleep) A robot can not Wait indefinitely.

A2 (Minimal Distance) For each robot r, an arbitrary small constant $\delta_r > 0$ is fixed. It represents the minimum distance the robot r travels in the Move phase, when the result of the computation is not `do_nothing()`. If the computed destination point is closer than δ_r, it will reach it. Clearly, each robot whose computation's result is not `do_nothing()`, travels at least a distance $\delta = \min_r \delta_r$.

Another secondary difference we can point out, is related to the way a robot moves, and to Assumption A2: 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 ϵ_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.

The main difference between the two models is, as stated before, in the way the asynchronicity is regarded. In CORDA the environment is *totally 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 phase of a given cycle are made[5]. 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: we have *asynchronicity within a cycle* [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 phase, it is possible that the movement it will perform will not be "coherent" with what it observed, since, during the Compute phase, other robots can have moved. On

[5]For the sake of uniformity, we assume that the Observe phase can also last an unpredictable, but finite, amount of time, even if its result is a snapshot of the robots' positions.

the other hand, we believe that, in this way, the asynchronicity of a system constituted by a set of autonomous, mobile robots, is better modeled.

3 The Novelty of CORDA

In this section we show 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.

3.1 The oblivious case

In the following we will analyze the oblivious case, showing that the algorithms in SYm do not work in CORDA, in both the limited and unlimited setting. The problem we consider is the *gathering problem*: the robots are asked to gather in a not-prefixed point in the plane in a finite number of cycles. This is the only problem, to our knowledge, solved with an oblivious algorithm in SYm[1, 14]. An algorithm is said *to solve* the problem if it lets the robots gather in a point, given any *initial configuration*. An initial configuration is the set of robots' positions when the computation starts, one position per robot, with no position occupied by more than one robot.

3.1.1 The unlimited visibility setting

An algorithm for solving the gathering problem in the unlimited visibility setting is presented in [14]. The idea is the following. Starting from distinct initial positions, the robots are moved in such a way that eventually there will be exactly one position, say p, that two or more robots occupy. Once such a situation has been reached, all the robots move towards p. In the following we report the oblivious algorithm described in [14] to let the robots achieve a situation where p is determined.

Algorithm 1 (Point Formation Algorithm in SYm**, Unlimited Visibility setting [14]).**

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 $|\overline{p_1p_2}| = |\overline{p_1p_3}| \neq |\overline{p_2p_3}|$, then the robot at p_1 moves toward the foot of the perpendicular drop from its current position to $\overline{p_2p_3}$ 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 $|\overline{p_1p_2}| > |\overline{p_1p_3}| > |\overline{p_2p_3}|$, then the robot at p_3 moves toward the foot of the perpendicular drop from its current position to $\overline{p_1p_2}$ 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.

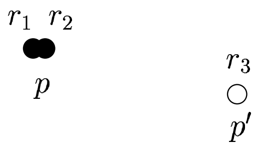


Figure 1: Initial situation with three robots ($n = 3$) for the proof of Theorem 3.1.

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 $\epsilon > 0$ 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 $\epsilon > 0$ in one step, if part 2.3 continues to hold,then eventually the radius of C_t becomes at most ϵ. 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.

It is clear that such a strategy works only if the robots in the system have the ability to detect the multiplicity. In SYm this capability is never mentioned, but it is clearly used implicitly. It is possible, however, to prove that such a capability is indeed necessary to solve the problem. In fact, we have the following:

Theorem 3.1. *In* CORDA, *there exists no deterministic oblivious algorithm that solves the gathering problem in a finite number of cycles for a set of n robots that, given a point $\overline{p}$ in the plane, can not distinguish the multiplicity of $\overline{p}$ (that is, if there is more than one robot on $\overline{p}$).*

Proof. Let $\mathcal{A}$ be a deterministic oblivious algorithm that solves the gathering problem in CORDA, and let us assume that, given a point $\overline{p}$ in the plane, the robots can not distinguish the multiplicity of $\overline{p}$. Moreover, let us suppose that at the beginning robots $r_1, \ldots, r_{n-1}$, from now on the *black* robots, lie all together on a point p of the plane, and r_n, the *white* robot, lies on a point p' of the plane, with $p \neq p'$ (in figure 1 the case $n = 3$ is pictured.

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).

Note that since the robots cannot distinguish the multiplicity, we have that at the beginning each robot can see only one other robot in the system (we can even suppose common knowledge on n). Therefore, we have that at the beginning all the robots have the same view of the world, namely each of them sees exactly one other robot in the plane. Moreover, let us suppose that the Observe, Compute and Move phases last the same amount of time, say ρ.

In the following we will describe an adversary *Adv* that schedules the movements of the robots in such a way that, whatever is the strategy of $\mathcal{A}$, it will never let the robots to gather in a point. The adversary chooses which robot to *activate*, that is, which one must start an Observe phase, and when to activate it. We note that if a robot is activated at time t, at time $t + 3\rho$ it will be in the Wait phase again. Moreover, *Adv* decides the distance a robot travels in the Move phase. In particular, the adversary follows the rules described below.

Let p and p' respectively be the points where the black robots lie and r_n lies. We have:

*Adv*1 If activating one black robot, it does not compute p' as destination point by executing $\mathcal{A}$, then activate all the black robots, and move them a distance δ (see Assumption A2 in Section 2.2).

*Adv*2 Otherwise,

- *Adv*2.1 activate $n-2$ black robots, and move them until they reach the destination point they compute;
- *Adv*2.2 after the activated black robots reach the destination point (that is after 3ρ time), activate r_n, and move it to the destination point it computes;
- *Adv*2.3 after another 3ρ time, activate the last black robot, and move it to the destination point it computes.

First, we note that if after choosing *Adv*1 none of the black robots move, not even the white robot would move, since $\mathcal{A}$ is deterministic, and r_n has the same view of the world as the black robots; namely it sees only one other robot in the world. Hence $\mathcal{A}$ would not solve the problem. Thus, as long as the adversary chooses *Adv*1, the black robots move *all together* a distance δ, *without reaching* p', while the white one is still (note that while they are moving, the white robot is in Wait, therefore it can not see the black ones while they move). They move *all together without reaching* p' because (I) $\mathcal{A}$ is deterministic and they are activated all at once; (II) there is a black robot that, if activated, would not reach the position occupied by r_n; (III) all the black robots share the same view of the world, since they all lie on p and see only one other robot in the system, namely the white one; (IV) the Observe, Compute, and Move phases each last ρ time.

Moreover, since $\mathcal{A}$ solves the problem by hypothesis, in a finite number of cycles the rule *Adv*2 is chosen by the adversary, otherwise the black robots would never reach r_n. Thus, when *Adv*2 is chosen for the first time, $n-2$ black robots are activated all together. Since this rule is chosen because there is a black robot that, if activated, would compute the position occupied by r_n as destination point, and by (III) and (IV) above, we have that, after 3ρ time, the $n-2$ activated black robots will reach p' (while they are moving, everybody else is in Wait, therefore they can not be seen while moving). After that, in *Adv*2.2, the white robot is activated. r_n has the same view of the world that the black robots that moved in $Adv2.1$ had. Specifically, the white robot sees only one robot, the last black robot on p (while the black robots in $Adv2.1$ saw only the white one). Since $\mathcal{A}$ is deterministic, r_n must compute the same thing computed by the black robots in $Adv2.1$; that is, it must decide to reach the other robot it sees, hence it computes p as destination point. Therefore, by (IV), after 3ρ time the white robot reaches p. Finally, the last black robot (still on p) is activated. Once again, it has the same view of the world that its fellow robots had in $Adv2.1$, therefore it computes p' as destination point, and reaches it in 3ρ time.

In conclusion, if $Adv2.1$ is chosen at time t, at time $t + 9\rho$ we have that all the black robots went where the white robot was at t, and the white robot went where the black robots were at t. Hence the black and white robots simply switched positions, and are once again in the same situation they were in at the beginning. Therefore, the adversary Adv can choose a scheduling of the robots such that, when they should gather in one point by executing $\mathcal{A}$, they do not. This leads to a contradiction. □

We note that the proof makes use of the assumption that a robot is not even able to recognize the multiplicity of the position where it is. However, even if we assume that a robot can distinguish a *weak multiplicity*, that is if there is more than one robot on the position where it is, Theorem 3.1 still holds. In fact, it is sufficient to assume to have two white robots on p', and to change the second rule of the adversary as follows:

$Adv'2$ Otherwise,

- $Adv'2.1$ activate $n-3$ black robots, and move them to the destination point they compute;
- $Adv'2.2$ after the activated black robots reach the destination point (that is after 3ρ time), activate the white robots, and move them to the destination point they compute;
- $Adv'2.3$ after other 3ρ time, activate the last black robot, and move it to the destination point it computes.

Moreover, the same result can be proven in SYm.

Corollary 3.1. *In* SYm*, there exists no deterministic oblivious algorithm that solves the gathering problem in finite time for a set of $n > 3$ robots*[6] *that, given a point $\overline{p}$ in the plane, cannot distinguish the multiplicity of $\overline{p}$, or that can distinguish the weak multiplicity.*

Proof. Let us assume that the robots, given a point $\overline{p}$ in the plane, cannot distinguish the multiplicity of $\overline{p}$, and that there exists a deterministic oblivious algorithm $\mathcal{A}$ that solves the gathering problem in SYm. The only difference with respect to the proof of Theorem 3.1 is in the upper bound ϵ_i a robot r_i can travel in each step (see Section 2.1). Let $\epsilon = \min_i \epsilon_i$, and let us define the adversary in the following way:

$Adv''1$ If there exists no black robot that, if activated, would compute p' as destination point by executing $\mathcal{A}$, then activate all the black robots, and move them a distance ϵ.

$Adv''2$ If there exists at least one black robot that, if activated, would compute p' as destination point by executing $\mathcal{A}$, but the distance between p and p' following the path given by $\mathcal{A}$ is $> \epsilon$, then activate all the black robots, and move them a distance ϵ.

[6]In [14], it is proven that there exists no oblivious algorithm that solves the problem when $n = 2$, under the assumption that two robots never collide.

Adv''3 Otherwise,

- *Adv*''3.1 activate $n-2$ black robots, and move them until they reach the destination point they compute;
- *Adv*''3.2 after the activated black robots reach the destination point (that is after one time step), activate r_n, and move it to the destination point it computes;
- *Adv*''3.3 activate the last black robot, and move it to the destination point it computes.

Considering that in SYm a cycle is atomic (as explained in Section 2.1), and following the proof of Theorem 3.1, the first part of the corollary follows.

To prove the second part of the corollary (the robots are supposed to distinguish a weak multiplicity), it is sufficient to assume to have two white robots, and to change *Adv*''3 as follows:

Adv'''3 Otherwise,

- *Adv*'''3.1 activate $n-3$ black robots, and move them until they reach the destination point they computed;
- *Adv*'''3.2 after the activated black robots reach the destination point (that is after one time step), activate the white robots, and move them to the destination point they compute;
- *Adv*'''3.3 activate the last black robot, and move it to the destination point it computes.

□

Finally, we are ready to state the main result of this section. In the following theorem we prove that Algorithm 1 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 having two robots in the system on the same point.

Theorem 3.2. *Algorithm 1 does not solve the gathering problem in* CORDA, *in the unlimited visibility setting.*

Proof. Let us suppose to have 4 robots that at the beginning are on a circle C, as pictured in Figure 2, Step 1. 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.

Step 1 At the beginning the four robots are in distinct positions, on a circle C. r_1 and r_2 enter the Observe phase, while the others are in Wait. After having observed, both of them enter the Compute phase, and let us assume that the robot in p_2 is computationally very slow (or, alternatively, that p_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 phase. r_1 starts moving towards the center, while r_2 is still in Compute, and r_3 and r_4 are in Wait.

Step 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 phase of its first cycle, and r_3 and r_4 are in Wait.

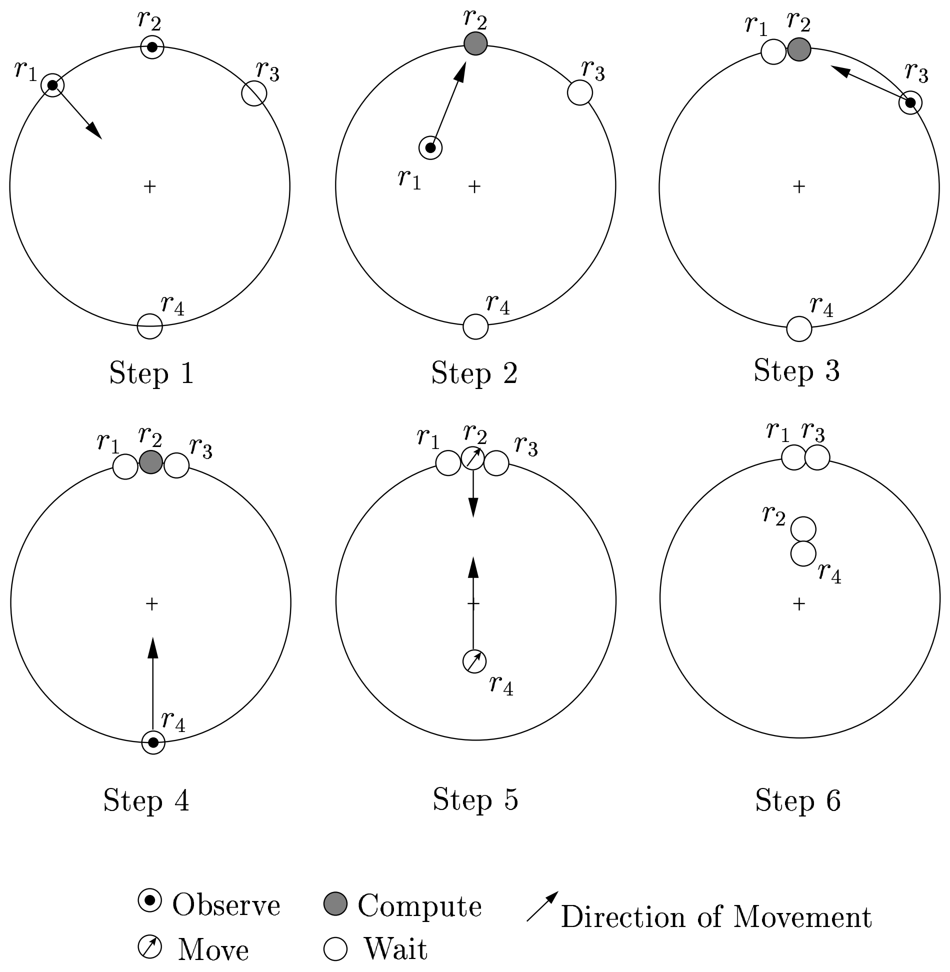


Figure 2: Proof of Theorem 3.2. The dotted circles indicate the robots in the Observe phase; the grey ones the robots in the Compute phase; the circle with an arrow inside are the robots that are moving; the white circles represent the robots in Wait. The arrows indicate the direction of the movement computed in the Compute phase.

Step 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 Observe phase, 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.

Step 4 r_3 reaches r_1 and r_2 on p, hence it ends its execution, since the algorithm is assuming that p is the gathering point. r_1 is in Wait, r_2 still in its first Compute phase, and r_4 starts its first Observe phase, decides to move towards p, and starts moving.

Step 5 While r_4 is on its way towards p, r_2 ends its first Compute phase. Since the computation is done according to what it observed in its previous Observe phase (Step 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.

Step 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" [14], is violated.

□

Remark 1. We note that in Step 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 that the robots stop because they bump, the proof is still valid. Otherwise, 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 Observe phase 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 3.2 still holds.

Finally, it is easy to see that the theorem is still valid even if we simulate a continue monitoring of the world by the robots, as in SYm, by introducing a maximum distance each robot can travel in a Move phase.

It follows from the proof of the previous theorem, and from Remark 1, that the real difference between the two models is in the way the asynchronicity is modeled: in SYm each cycle is atomic, while in CORDA each phase in a given cycle can last an unpredictable, but finite, amount of time; that is, we have *asynchronicity within a cycle*. Such a hypothesis clearly renders the analysis and design of algorithms more difficult, but in our opinion it better models a totally asynchronous environment populated by a set of mobile units.

3.1.2 The limited visibility setting

In [1], an algorithm to solve the gathering problem in the limited visibility setting is presented. In the following we shortly describe it[7].

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 [1] (*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 $c_i(t)$.

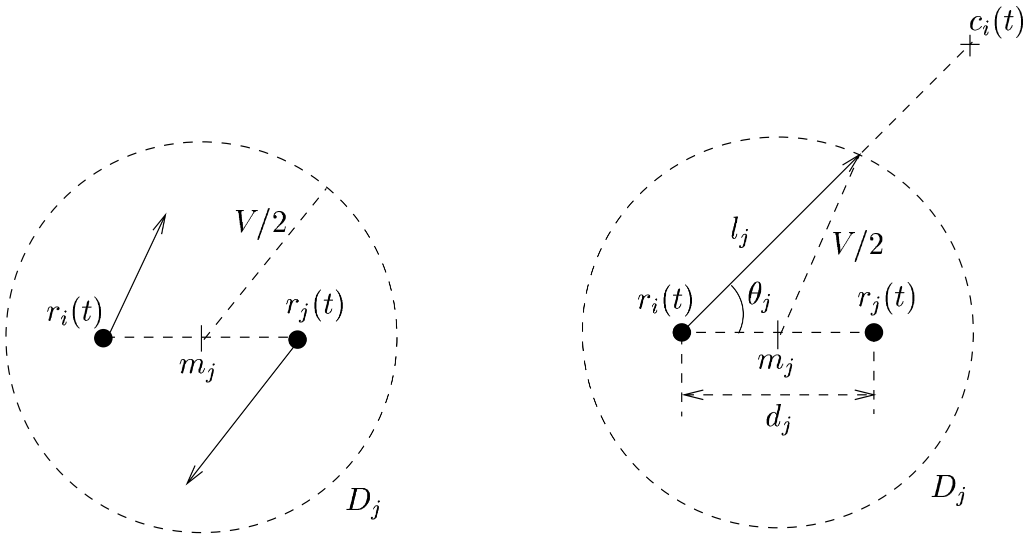
Every time a robot r_i becomes active, the algorithm moves r_i toward $c_i(t)$, but only over a certain distance $MOVE$. Specifically, if r_i does not see any robot other than itself,

[7]The notation used in [1], is slightly different from that used in [14]. We will use in the following the notation used in the original paper.

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. $dist(r_i(t), x) \leq \sigma$, where $dist(a, b)$ indicates the distance between two points. This condition follows from the restriction on the maximum distance a robot can move in one phase[8].

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.a).



a. b.

Figure 3: The algorithm for the gathering problem in SYm, limited visibility setting.

The algorithm is described more formally in the following (refer to Figure 3.b).

Algorithm 2 (Point Formation Algorithm in SYm, **Limited Visibility setting [1]).**

```
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. $GOAL = dist(r_i(t), c_i(t))$,
```

[8]σ is the name used in [1] for both the ϵ described in Section 2.1 and used in [14].

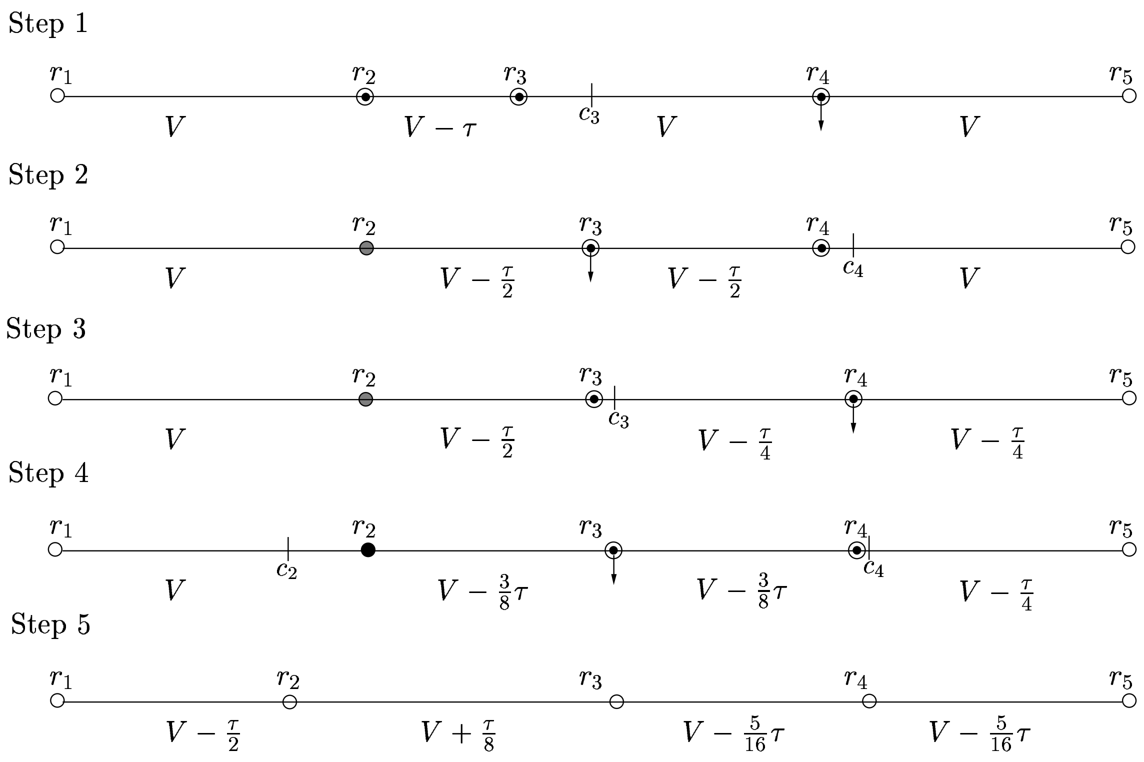


Figure 4: Proof of Theorem 3.3. The symbols used for the robots are the same as in Figure 2. The vertical arrows mean that a robot decided not to move ($Move = 0$). A robot r_i moves always towards the center c_i of the smallest enclosing circle of all the robots it can see.

5. $MOVE = \min\{GOAL, LIMIT\}$,
6. $x =$ point on $\overline{r_i(t)c_i(t)}$ at distance $MOVE$ from $r_i(t)$.

In [1] 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 [1].

Theorem 3.3. *Algorithm 2 does not solve the gathering problem in* CORDA, *in the limited visibility setting.*

Proof. Let us suppose to have at the beginning 5 robots on a straight line, as shown in Figure 4. Moreover, let τ be a constant such that $\delta = \tau/16$, where δ is the constant introduced in Section 2.2. At the beginning, we have the following *visibility situation*: r_1 can see r_2 ($\overline{r_1r_2} = V$), r_2 can see r_1 and r_3 ($\overline{r_2r_3} = V-\tau$), r_3 can see r_2 and r_4 ($\overline{r_3r_4} = V$), r_4 can see r_3 and r_5 ($\overline{r_4r_5} = V$), 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.

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

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

$$r_4\text{:} \qquad \mathit{Goal} = 0 \;\; \Rightarrow \mathit{Move} = 0$$

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

Step 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 Observe and Compute again, as follows:

$$r_3\text{:} \qquad \mathit{Goal} = 0 \;\; \Rightarrow \mathit{Move} = 0$$

$$r_4\text{:} \qquad \left\{ \begin{array}{l} \mathit{Goal} = \mathit{dist}(r_4, c_4) = \left|\frac{V+V-\frac{\tau}{2}}{2} - V + \frac{\tau}{2}\right| = \frac{\tau}{4} \\ \mathit{Limit} = \min\{-\frac{V-\frac{\tau}{2}}{2} + \frac{V}{2}, V\} = \frac{\tau}{4} \end{array} \right. \Rightarrow \mathit{Move} = \frac{\tau}{4}$$

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

Step 3 After the movement of the previous step, the visibility situation is still unchanged, that is, the proximity graph is still connected. r_3 and r_4 enter their third Observe and Compute phases.

$$r_3\text{:} \qquad \left\{ \begin{array}{l} \mathit{Goal} = \mathit{dist}(r_3, c_3) = \left|\frac{V-\frac{\tau}{2}+V-\frac{\tau}{4}}{2} - V + \frac{\tau}{2}\right| = \frac{\tau}{8} \\ \mathit{Limit} = \min\{-\frac{V-\frac{\tau}{2}}{2} + \frac{V}{2}, \frac{V-\frac{\tau}{4}}{2} + \frac{V}{2}\} = \frac{\tau}{4} \end{array} \right. \Rightarrow \mathit{Move} = \frac{\tau}{8}$$

$$r_4\text{:} \qquad \mathit{Goal} = 0 \;\; \Rightarrow \mathit{Move} = 0$$

r_3 and r_4 move again. The other robots are in the same phases as in the previous step.

Step 4 The proximity graph is still connected. r_3 and r_4 Observe and Compute again (this is their fourth cycle).

$$r_3\text{:} \qquad \mathit{Goal} = 0 \;\; \Rightarrow \mathit{Move} = 0$$

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

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

$$r_2\text{:} \qquad \left\{ \begin{array}{l} \mathit{Goal} = \mathit{dist}(r_2, c_2) = \left|\frac{V+V-tau}{2} - V\right| = \frac{\tau}{2} \\ \mathit{Limit} = \min\{V, -\frac{V-\tau}{2} + \frac{V}{2}\} = \frac{\tau}{2} \end{array} \right. \Rightarrow \mathit{Move} = \frac{\tau}{2}$$

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

Step 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 step. So, the invariant that "robots that are mutually visible at t remain within distance V of each other thereafter" asserted in [1] is violated. Therefore, the theorem follows.

□

Once again, we note that what renders the algorithm useless in CORDA is the substantial difference of how the asynchronicity is modeled.

Remark 2. In [1], Step 5 of Algorithm 2 takes into account the maximum distance σ a robot can travel in each cycle[8], namely:

5. $MOVE = \min\{GOAL, LIMIT, \sigma\}$.

We did not mention it in the proof of the previous theorem because there is no similar upper bound in CORDA. Theorem 3.3, however, is still valid even if we assume such a constant. In fact, it is sufficient to define τ such that $\tau < \sigma$, and that $\delta = \tau/16$.

3.2 The non oblivious case

In [14], the problem of characterizing "the class of geometric patterns that the robots can form" is analyzed. The authors present a non oblivious algorithm to solve the problem, leaving as an open problem the existence of an oblivious algorithm. We reiterate that an oblivious algorithm takes in input all the positions of all the robots since the beginning of the computation; that is, each robot has the capability to remember where its fellows have been since the beginning.

The solution they propose is based on one basic operation, the *broadcast*: each robot r_i wants to communicate (broadcast) to all the others the direction of a line l_i it privately chooses, that passes through its initial position. In order to do so, each robot r_i keeps moving along l_i in the established direction, until it observes that every r_j, $j \neq i$ has changed positions at least twice (i.e., until r_i sees r_j at three or more distinct positions). When this happens, every r_j, $j \neq i$, can compute the direction of l_i from the (at least two) distinct positions in which it must have observed r_i. Moreover, they introduced a technique, that will not be described here, that allows the robots to determine which robots is broadcasting which direction (remember that the robots are anonymous, see Footnote 3).

The authors use the broadcast twice to let each r_i communicate to everyone else its initial position; that is, to discover the initial configuration (called "distribution" in [14]) of the robots in the system. In order to do so, each r_i broadcasts the positive direction of its X axis; it returns straight to its initial position; it broadcasts the positive direction of its Y axis; and finally it returns straight to its initial position. After this, every robot can easily discover the initial positions of all the other robots.

In the following, we show that the technique adopted to broadcast the initial configuration does not work in the asynchronicity-within-a-step assumption of CORDA.

Theorem 3.4. *The non oblivious broadcast-technique, adopted in [14] to discover the initial distribution of the robots, does not work in* CORDA.

Proof. Let us suppose to have n robots, and that all of them enter the Observe phase. Therefore, every r_i sees the initial positions of r_j, $1 \leq j \neq i \leq n$. After the Observe phase, all the robots enter the Compute phase and the Move phase. Let us suppose, however, that r_1 is really slow, therefore all the others execute two cycles while r_1 is still in its first Move. This implies that, since in CORDA a robot can be seen while it is moving, $r_2, \ldots, r_n$ see r_1 in two different positions, therefore they start broadcasting their respective Y axes. When r_1 enters its second Observe phase, it sees all the other robots already broadcasting their Y axes, while it thinks that they are still broadcasting their X axes (since r_1 has not yet seen all the others change positions at least twice). Hence, r_1 computes the wrong initial configuration of the robots in the system. □

4 Conclusions

In this paper we presented a model introduced for the first time in [6, 7], CORDA, consisting of a set of autonomous, anonymous, memoryless, mobile robots - features that render the robots rather "weak". Moreover, we analyzed the main differences with SYm [1, 14], the only other model, to our knowledge, that focuses on the problem of studying the algorithmic problems that arise in an asynchronous environment populated by a set of autonomous, anonymous, mobile units that are requested to accomplish some given task. In particular, 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. We feel that the *asynchronicity within a cycle* modeled in CORDA better describes the way a set of independently-moving units operate in a totally asynchronous environment; hence the motivation to further investigate coordination problems in a distributed, asynchronous environment using CORDA.

The purpose of our study is to gain a better understanding of the power of the distributed control from an algorithmic point of view. Specifically, we want to understand what kind of goals such a set of robots can achieve, and what are the minimal requirements and capabilities that they must have in order to do so.

Until now, the pattern formation problem is the only problem that has been extensively studied using CORDA [6, 7, 12]: the robots are required to form some pattern they are given in input. Other interesting problems that are usually studied in robotics and that could be analyzed in CORDA are, for example, the *flocking* problem (given an initial formation of the robots, they have to keep it while moving towards some destination); the *following* problem (a robot must follow another one); the *intruder* problem (all the robots in the environment must chase and "catch" a robot that is "different" from all the others).

Other issues which merit further research, regard the operating capabilities of the robots modeled in CORDA. In fact, it would be interesting to look at models where robots have different capabilities. For instance, we could use a totally *non-oblivious* model, that is, one with an unlimited amount of memory that each robot could use. Alternatively, we could equip the robots 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; or how it would affect the self-stability property of the oblivious algorithms [7].

Other features we could add to our model which would inspire further study include giving a dimension to the robots and adding stationary obstacles to the environment, thus adding the possibility of collision between robots or between moving robots and obstacles. Furthermore, we could also study how the robots can use some kind of direct communication, and we could introduce different kinds of robots that move in the environment (as in the *intruder* problem described above).

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.

Acknowledgment

I would like to thank Paola Flocchini and Nicola Santoro for the discussions and comments that helped with the writing of this paper.

References

[1] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. A Distributed Memoryless Point Convergence Algorithm for Mobile Robots with Limited Visibility. *IEEE Trans. on Robotics and Automation*, 15(5):818–828, 1999.

[2] T. Balch and R. C. Arkin. Behavior-based Formation Control for Multi-robot Teams. *IEEE Trans. on Robotics and Automation*, 14(6), December 1998.

[3] G. Beni and S. Hackwood. Coherent Swarm Motion Under Distributed Control. In *Proc. DARS'92*, pages 39–52, 1992.

[4] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative Mobile Robotics: Antecedents and Directions. In *Int. Conf. on Intel. Robots and Sys.*, pages 226–234, 1995.

[5] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *ICMAS*, pages 406–413, 1995.

[6] 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 *ISAAC '99*, pages 93–102, 1999.

[7] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IVS*, 2000.

[8] Y. Kawauchi and M. Inaba and. T. Fukuda. A Principle of Decision Making of Cellular Robotic System (CEBOT). In *Proc. IEEE Conf. on Robotics and Autom.*, pages 833–838, 1993.

[9] M. J Matarić. *Interaction and Intelligent Behavior.* PhD thesis, MIT, May 1994.

[10] S. Murata, H. Kurokawa, and S. Kokaji. Self-Assembling Machine. In *Proc. IEEE Conf. on Robotics and Autom.*, pages 441–448, 1994.

[11] L. E. Parker. On the Design of Behavior-Based Multi-Robot Teams. *Journal of Advanced Robotics*, 10(6), 1996.

[12] G. Prencipe. Achievable Patterns by an Even Number of Autonomous Mobile Robots. Technical Report TR–00–11, Università di Pisa, August 2000.

[13] K. Sugihara and I. Suzuki. Distributed Algorithms for Formation of Geometric Patterns with Many Mobile Robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[14] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam J. Comput.*, 28(4):1347–1363, 1999.