Theory Comput. Systems **38**, 539–558 (2005)
DOI: 10.1007/s00224-005-1101-1

Theory of Computing Systems

The Effect of Synchronicity on the Behavior of Autonomous Mobile Robots*

Giuseppe Prencipe

Dipartimento di Informatica, Università di Pisa,
largo Bruno Pontecorvo, 3-56127 Pisa, Italy
prencipe@di.unipi.it

Abstract. Over the past few years, the focus of robotic design has been moving from a scenario where a few specialized (and expensive) units were used to solve a variety of tasks, to a scenario where many general purpose (and cheap) units are used to achieve some common goal. Consequently, part of the focus has been to understand better how to coordinate and control a set of such "simpler" mobile units efficiently. Studies can be found in different disciplines, from engineering to artificial life: a shared feature of the majority of these works has been the design of algorithms based on heuristics, with no main concern on their correctness and termination. Few researchers have focused on trying to model formally an environment constituted by mobile units, analyzing 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. In this paper we do a direct comparison between two models, ATOM and CORDA, introduced in two studies leading in this direction. First their main features are described, and then the main differences are highlighted, showing the relationship between the class of problems solvable in the two models.

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 a certain predetermined task is accomplished 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;

* Parts of this paper have appeared in preliminary form in [20].

the ability to perform tasks that cannot be accomplished 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.

Several studies in different fields have been conducted in this direction. In the engineering area we can cite the Cellular Robotic System (CEBOT) of Kawaguchi et al. [16], the Swarm Intelligence of Beni and Hackwood [4], and the Self-Assembly Machine ("fructum") of Murata et al. [18]. In the AI community there have been a number of remarkable studies: social interaction leading to group behavior by Matarić [17]; selfish behavior of cooperative robots in animal societies by Parker [19]; and primitive animal behavior in pattern formation by Balch and Arkin [3].

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.

A different approach is 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, the approach is to study the problem from a *computational point of view*.

Typical problems that have been studied in this perspective are taken directly from those usually studied in robotics and engineering. For instance, we can cite the Arbitrary Pattern Formation problem, where the robots are asked to form in finite time, a pattern they are given as input [10], [25] (e.g., a circle [7]); the Gathering problem, where the robots are asked to gather in a point of the plane not known in advance [1], [2], [5], [12], [25]; the Flocking problem, where the robots are required to follow a leader unit wherever it goes, while keeping a formation given them as input, and not knowing beforehand the path the leader will follow [14]; the Intruder problem, where all robots in the environment are required to chase and capture an intruder that sneaks through a restricted area they are patrolling [13].

This paper deals with two studies leading in this direction (the only ones, to our knowledge, that analyze the problem of coordinating and controlling a set of autonomous, mobile units from this point of view). The first study was started by Suzuki and Yamashita [23]–[25], and then continued in [2] and [22]. It gives a nice and systematic account on the algorithmics of different tasks for the robots, operating under several assumptions on the power of the individual robot. The second is by Flocchini et al. [10], [12]: they present a model (that we refer to as CORDA—COordination and control of a set of ROBOTS in a DISTRIBUTED and ASYNCHRONOUS environment) that has as its primary objective the description of a set of totally asynchronous mobile units, which have no central control (i.e., move independently from each other) and that execute the same deterministic algorithm in order to achieve some goal. In both studies the modeled robots are rather *weak* and simple, but this simplicity allows the authors to highlight formally from an algorithmic and computational viewpoint the minimal capabilities the robots must have in order to accomplish basic tasks and produce interesting interactions. Furthermore, it allows us to understand better the power and limitations of the distributed control in an environment inhabited by mobile agents, hence to prove formally what can be achieved under the "weakness" assumptions of the models, that will be described later in more detail.

An investigation with an algorithmic flavor has been undertaken within the AI community by Durfee [9], 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.

Although the model of Suzuki and Yamashita (which we refer to as ATOM) and CORDA share some features, they differ in some aspects that render the two models quite different. In this paper we do a direct comparison between the two models, highlighting these differences, focusing in particular on the different approach in modeling the asynchronicity of the environment in which the robots operate.

In Section 2 ATOM and CORDA are described, highlighting the features that render the two models different. In Section 3 we show that the class of problems solvable in CORDA is strictly contained in the class of problems solvable in ATOM; furthermore, we present a set of conditions that allow us to establish whether an algorithm designed in ATOM to solve a given problem can be used in CORDA to solve the same problem. In Section 4 we present a case study that shows an example on one of those conditions. Finally, in Section 5 we draw some conclusions and present open problems and suggestions for further study.

2. Modeling Autonomous Mobile Robots

In this section we describe the approaches used in ATOM and CORDA to model the control and coordination of a set of autonomous mobile robots. In particular, we first present the common features of the two models, and successively present in detail the *instantaneous actions* of ATOM, and the *full asynchronicity* of CORDA, that model the behavior of the robots.

2.1. *Common Features*

The two models discussed in this paper share some basic features. Consider a distributed system populated by a set of n autonomous mobile robots, denoted by $r_1, \ldots, r_n$, that are modeled as devices with computational capabilities[1] which are able to move freely 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 set of absolute positions[2] on the plane occupied by the robots at a given time instant is called a *configuration of the robots*.

The behavior of a robot is a cycle of sensing, computing, moving, and being inactive. In particular, each robot is capable of sensing the positions of other robots in its surrounding, performing local computations on the sensed data, and moving towards the computed destination. The local computation is done according to a deterministic algorithm that takes as 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.

[1] To our knowledge, nothing is ever mentioned on the computational power of the modeled robots. For the purpose of this paper, they can be considered as Turing-equivalent machines.

[2] That is, with respect to an inertial reference frame.

The local view of each robot 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 made on the robots' capabilities, and on the amount of information that they share and use during the accomplishment of the assigned task. In particular:

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

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*.

(Agr) 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 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). We refer to this scenario as *no agreement* on the local coordinate systems (see Figure 1(c)). 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 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 talk of *total agreement* on the local coordinate systems.

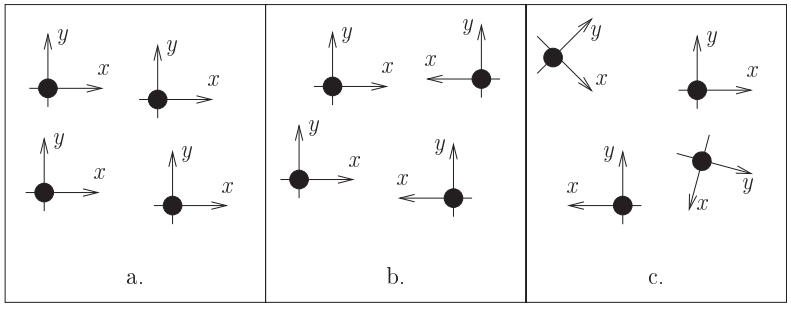

Fig. 1. (a) *Total agreement* on the local coordinate systems. (b) *Partial agreement* on one axis direction and orientation. (c) *No agreement* on the local coordinate systems.

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 North–South axis, but there is no general agreement on where East or West is (see Figure 1(b)): we talk of *one axis direction and orientation agreement* (or shortly *partial agreement*).

(Obl) The robots can access local memory to store different amount of information regarding the positions in the plane of their fellows. In particular, if the robots can only store the robots' positions retrieved in the last observation, we have *oblivious* robots. In contrast, if the robots can store all the positions retrieved since the beginning of the computation, we have *non-oblivious* robots. We also refer to the algorithm the robots execute as *oblivious* or *non-oblivious*, depending on the adopted model.

Note that *the robots always have common knowledge* [15] *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.[3]

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 robots' positions in the plane, and taking a deterministic decision accordingly. 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).

Finally, a critical feature regards the way the robots act during the computation; that is, the timing of the operations executed by each robot during its life. In particular:

(Time) If the amount of time spent in observation,[4] in computation, in movement, and in inaction is finite but otherwise unpredictable, then we say that the robots are *fully asynchronous*. In particular, the robots do not (need to) have a common notion of time. Each robot executes its actions at unpredictable time instants. This setting is adopted in CORDA.

In contrast, if the robots execute their activities (observation, computation, movement, and waiting) in an atomic and instantaneous fashion (that is, the amount of time spent in each activity of each cycle is negligible), we say that the robots are *atomically synchronized*. This setting is adopted in ATOM.

We discuss in more detail the implications of time settings in the following. Clearly, depending on the adopted settings, these basic features allow us to model different levels of power of the robots. For instance, the oblivious, asynchronous, and with no

[3] Note that the non-obliviousness feature does not imply the possibility of a robot finding out which robot corresponds to which position it stored, since the robots are anonymous.

[4] That is, activating the sensors and receiving their data.

agreement model renders the robots particularly weak, especially considering the current engineering technology. However, as already noted, the main interest of this kind of study is to approach the problem of coordinating and controlling a set of mobile units from a *computational* point of view. In fact, the general idea proposed in this work is to look for the minimum power to give to the robots so that they can solve a given task; hence, to analyze formally the strengths and weaknesses of the distributed control. Furthermore, this simplicity also leads to some advantages. For example, making do without the ability to remember what has been observed in the past (obliviousness) could give the system the nice property of self-stabilization [6], [8], [25].

2.2. *The* Full Asynchronicity *of* CORDA

In CORDA the robots are *fully asynchronous*: the amount of time spent in observation, in computation, in movement, and in inaction is finite but otherwise unpredictable. In particular, the robots do not have a common notion of time, and each robot executes its actions at unpredictable time instants.

During its life, each robot cyclically executes four *states*: (i) initially it is inactive—*Wait*, (ii) asynchronously and independently from the other robots, it observes the positions of the other robots in its area of visibility—*Look*, (iii) it computes its next destination point by executing the algorithm (the same for all robots)—*Compute*, and (iv) it moves towards the point it just computed—*Move*; after the move it goes back to a waiting state. As already stated, the robots execute these states *asynchronously*, without any central control: in this feature CORDA drastically differs from ATOM (as we will see in Section 2.3).

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

(i) **Wait** The robot is idle. A robot cannot stay indefinitely idle (see Assumption A2 below). At the beginning all robots are in *Wait*.

(ii) **Look** The robot observes the world by activating its sensors which will return a *snapshot* of the positions of all other robots within the visibility range 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 in its local coordinate system: this set forms 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 taken at a time smaller than or equal to t.

(iii) **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 robots, and the result of the *Compute* state is a *destination point*. If the robots are oblivious, then $\mathcal{A}$ can access only the set of robots' positions retrieved during the last *Look*; otherwise, $\mathcal{A}$ can access the information relative to all the positions of the robots observed since the beginning of the computation.

(iv) **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; 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 A1 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.[5] The amount 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 but unpredictable. In addition, no time assumption within a state is made. 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.

This assumption of *asynchronicity within a cycle* is important in a totally asynchronous environment, since each robot has enough time to perform its local computation; furthermore, in this way it is also possible to model different motorial speeds of the robots. If the robots move according to this time setting, we say that they move according to an *asynchronous activation schedule*. Furthermore,

Definition 2.1. An algorithm $\mathcal{A}$ *solves* a problem P if the robots, activated according to *any* asynchronous activation schedule, reach in a finite number of cycles a configuration such that the task defined by P is accomplished.

A 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 2(a), the first part of the *Look* is executed between time t_1 and t_2, the snapshot is taken at time t_2, and the third part is executed between time t_2 and t_3. In the following we assume that the first and third parts have null length. This is not a loss of generality: in fact, the first part can be thought of as part of the previous *Wait* state, and the third part as part of the following *Compute* state (as shown in Figure 2(b)). Therefore, each *Look* consists only of the snapshot: if r is executing a *Look* at time t, then $\mathbb{V}_r(t)$ is the snapshot retrieved at t.

In the model there are only two limiting assumptions about space and time. The first one refers to space; namely, the distance traveled by a robot during a computational cycle.

Assumption A1 (Distance). *The distance traveled by a robot r in a move is not infinite. Furthermore, 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 at least δ_r.*

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

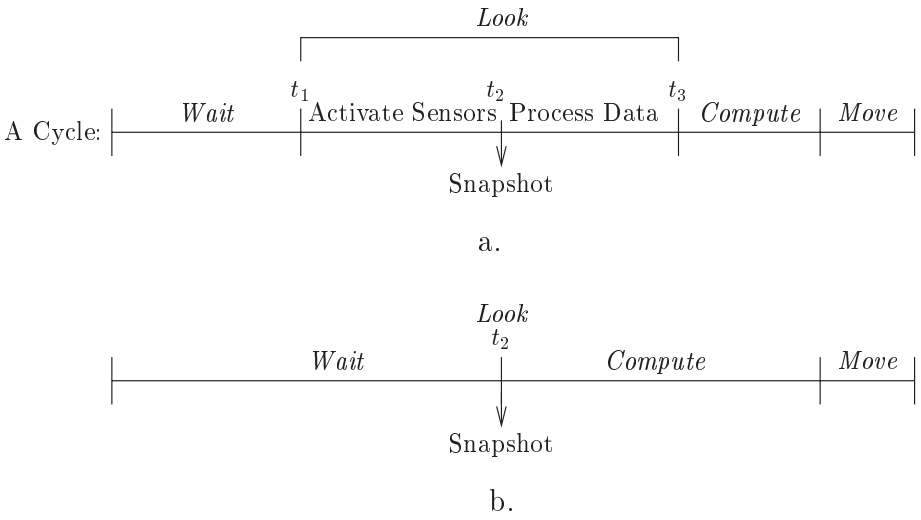

Fig. 2. (a) An example of a cycle. In particular, the three parts that constitute the *Look* state (between time t_1 and t_3) are put in evidence. (b) Without loss of generality, we can assume that the *Look* state starts with the snapshot, since the time spent in activating the sensors can be thought as part of the previous *Wait*.

As no other assumptions on space exist, the distance traveled by a robot in a cycle is unpredictable. The reason for introducing δ_r is to ensure progress in the movement of the robots; in other words, if r aims to reach a destination point p, Assumption A1 ensures that r will reach p in a finite number of cycles. Without such an assumption, it would be impossible to prove the termination of the algorithms designed in CORDA in a finite number of cycles: in fact, following a classical Zenonian argument, without Assumption A1 r could move at each cycle towards p always by half of its distance from p.

Similarly, to prove that the algorithms designed in CORDA terminate in finite time, the following assumption on the length of a computational cycle is made.

Assumption A2 (Computational Cycle). *The amount of time required by a robot r to complete a computational cycle is not infinite. Furthermore, there exists a constant $\varepsilon_r > 0$ 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.

2.3. *A Different Approach*: *The* Instantaneous Actions *of* ATOM

As highlighted in the previous section, one important feature of CORDA is that the agents in the environment act following *fully asynchronous* behavior. The main aspect where

ATOM and CORDA drastically differ is in the way the robots interact, specifically in the way time is modeled. In fact in ATOM, similarly to CORDA, the cycle of life of each robot is to (i) 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.

However, the robots operate in an *atomic* fashion, i.e., they perform their actions always together: for instance, at a given time t, there cannot exist two distinct robots that are performing a *Look* and a *Move*, respectively. Such a setting has been introduced and studied in [2] and [25].

In particular, discrete time $0, 1, \ldots$ is used to describe the behavior of the robots in the system, and a cycle "is regarded as an atomic instantaneous event, and thus a robot observes other robots (when a cycle begins) only when they are stationary" [2]. 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 we say that the robots are *strongly synchronized*.[6] Let A_t be the set of active robots at time t; furthermore, let us call the sequence $A_0, A_1, \ldots$ an *atomic activation schedule*. In the following we denote this sequence by ActATOM, the position of robot r_i at time instant t by $p_i(t)$, and the algorithm every robot uses by ψ. For any $t \geq 0$, if r_i is inactive, then $p_i(t+1) = p_i(t)$; otherwise

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

where p is the point returned by ψ. Depending on whether the model is oblivious or not, ψ takes as input only the positions of the robots retrieved at the last *Look*, or all the configurations retrieved since the beginning. An algorithm ψ *solves* a problem P if the robots, activated following any atomic activation schedule, reach in a finite number of cycles a configuration such that the task defined by P is accomplished.

Thus, from (1), r_i executes the three states *atomically* and *instantaneously*, in the sense that a robot that is active and observes at t, has already reached its destination point p at $t+1$, and no fellow robot can see it *while* it is moving (or, alternatively, the movement is *instantaneous*).

The maximum distance that r_i can move in one cycle has as an upper bound the local unit measure of r_i, that corresponds to some physical distance[7] $\sigma_i > 0$ (if the destination is closer than σ_i, the robot can clearly move less). The reason for such a constant is to simulate continuous monitoring of the world by the robots. However, in [2] and [25] this limit is stated as a restriction on ψ that must be written in such a way as never to return a destination point at a distance greater than σ_i from $p_i(t)$.

2.4. *Final Remarks*

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

[6] In [2] and [25] the authors refer to this case simply as *synchronous*.

[7] In [2] and [25], this constant is denoted by ε_i.

time instants. Moreover, no assumptions on the cycle time of each robot, and on the time each robot takes 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.[8] 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 performs is not "coherent" with the current configuration (i.e., the configuration it observed at the time of the *Look* and the configuration at the time of the *Move* can differ), 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.

A further remark regards the distance each robot can travel during a cycle. As already mentioned, in ATOM such a distance is bounded by σ_i. 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). The only assumption in CORDA is that there is a lower bound on such a distance: when a robot r moves, it moves at least some positive, small constant δ_r.

3. *Instantaneous Action* versus *Full Asynchronicity*

The two time settings presented in the previous section (asynchronous for CORDA and atomic for ATOM) are drastically different. In this section we discuss in more detail the features that render the two models different; in particular, we focus on the time feature of the two models introduced so far, highlighting the relationships between the two time settings.

The first question we address is: Can an algorithm designed in ATOM be *executed* in CORDA, and vice versa? The difference between an algorithm designed in ATOM and one for CORDA relies on the amount of distance that each robot is allowed to travel in each cycle. In fact, to our knowledge and as mentioned in Sections 2.3 and 2.4, in all the algorithms designed for ATOM there is 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, then the algorithm explicitly returns a point at most at distance σ_i. We call (for ATOM) an algorithm that imposes such a limitation on the movements of the robots *valid*.

In contrast, such a limitation is not required in CORDA. Hence, it follows that any valid ATOM algorithm can be executed in CORDA with no changes. Clearly, this does not implies that a valid algorithm that solves a given problem in ATOM solves the same problem in CORDA; it simply means that a valid algorithm can also be *run* in CORDA with no modifications.

[8] Note that this does not mean that the observing robot can distinguish a moving robot from a non-moving one.

On the other hand, an algorithm $\mathcal{A}$ designed for CORDA needs some transformation in order to become valid: it is sufficient to give as input to the algorithm of robot r_i also the maximum amount of distance it can travel at each cycle, σ_i (note that σ_i is known to the robot, since it is its local unit measure, see Section 2.3). In particular, let p_i be the position of r_i when it executes $\mathcal{A}$ at a given time, and let $dest_i$ be the destination point returned by this computation. Then the transformed valid algorithm $\mathcal{A}_v$ returns a point p on the segment $[p_i, dest_i]$ such that $dist(p_i, p) = \min(\sigma_i, dist(p_i, p))$. Note that, by the way the *Move* state is defined in CORDA, if $\mathcal{A}$ solves a given problem P in CORDA, then $\mathcal{A}_v$ will still solve P in CORDA; shortly, we will also prove that $\mathcal{A}_v$ solves the problem in ATOM.

The other major difference between the two models regards the way time is modeled. The asynchronous setting in CORDA is very lazy: in fact, no restriction is imposed on the timing of the robots' activities. In particular, given a problem $\mathcal{P}$, and an algorithm $\mathcal{A}$ that solves it in such a time setting, the robots are able to solve P independently from the duration of their cycles, and from the duration of the states in each cycle. This means that the robots are able to solve P, even if they move in a *synchronized* fashion: that is, if all the activities of the robots are done at the same time. In other words, all robots start a cycle together at the same time, and each state of a cycle lasts the same amount of time. Notice that this time setting is still different from the atomic setting, where each cycle is executed atomically, and the only choice of a schedule is on the activation of a robot.

Hence, we have just introduced a third time setting: the *synchronous*, that is somehow between the asynchronous and the atomic time setting. More formally, a *synchronous activation schedule* is an asynchronous schedule with the following constraints (refer to the example in Figure 3):

1. all robots start their executions together, say at time $t = 0$;
2. each robot can either execute a *normal cycle* (NC) or a *long wait* (LW). In particular, given a constant $\rho > 0$, a normal cycle is a cycle that lasts 4ρ time, and where each state lasts exactly ρ time; in a long wait a robot is simply in *Wait* for 4ρ time.

It follows by the above definition and by Definition 2.1, that any problem that can be solved in the asynchronous time setting is solved even if the robots move according to a synchronous schedule.

Now we are ready to address the following question: What is the relationship between the synchronous and the atomic time setting? That is, if a problem can be solved in the

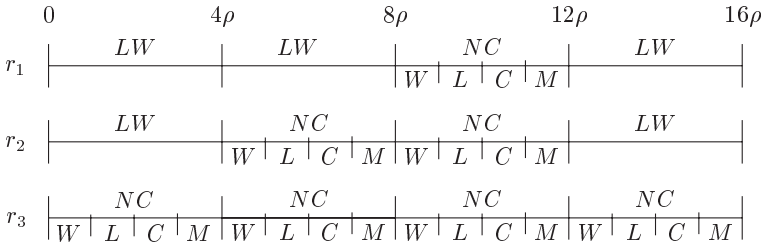

Fig. 3. An example of a synchronous activation schedule for three robots.

synchronous time setting, can it be solved in the atomic one? The following lemma gives the answer.

Lemma 3.1. *Given a deterministic algorithm $\mathcal{A}$ designed in* CORDA, *any synchronous activation schedule Sync$\mathcal{F}$ can be transformed into an atomic activation schedule* ActATOM *such that if the robots, starting from the same initial configuration, are in a configuration $\mathbb{G}_t$ at time t when they execute $\mathcal{A}$ according to Sync$\mathcal{F}$, they are in $\mathbb{G}_t$ at time t when they execute $\mathcal{A}$ according to* ActATOM.

Proof. As noted above, $\mathcal{A}$ can be transformed into a valid algorithm $\mathcal{A}_v$. By executing $\mathcal{A}_v$ according to *Sync$\mathcal{F}$*, a robot performs only NCs or LWs, and each NC or LW always starts at a time that is multiple of 4ρ (see Figure 3). ActATOM is built from *Sync$\mathcal{F}$* as follows: if at time $t = 4 \cdot k \cdot \rho$, for $k \geq 0$, robot r_i executes an NC (resp., LW), then r_i is active (resp., inactive) in $\mathcal{A}_{t/4\rho}$.

Let us choose $\rho = \frac{1}{4}$, and execute $\mathcal{A}_v$ according to both *Sync$\mathcal{F}$* and ActATOM. The proof is by induction: by hypothesis, the robots start from the same initial configuration at time 0 in both executions; hence, the robots are in $\mathbb{G}_0$ in both executions.

Let us assume that at time t the robots are in the same configuration $\mathbb{G}_t$ in both executions. In *Sync$\mathcal{F}$* the robots that execute an NC observe at time $t + \frac{1}{4}$, while in ActATOM the active robots observe at time t; hence, in both executions, the robots observe the same configuration. Since $\mathcal{A}_v$ is deterministic, and in *Sync$\mathcal{F}$* no robot moves between time t and $t + \frac{3}{4}$, in both executions the robots are in $\mathbb{G}_{t+1}$ at time $t + 1$, and the lemma follows. □

An example of the above lemma is given in Figure 4. In the following theorem, we prove that any problem that can be solved in CORDA can be solved in ATOM . We denote by $\mathfrak{C}$ and $\mathfrak{Z}$ the class of problems that are solvable in the asynchronous and the atomic setting, respectively. We can state the following:

Theorem 3.1. $\mathfrak{C} \subseteq \mathfrak{Z}$.

Proof. Let $\mathcal{A}$ be an algorithm that solves a given problem in CORDA. By definition, $\mathcal{A}$ solves the problem only if the robots reach a final configuration by executing $\mathcal{A}$ according to *any* asynchronous activation schedule; hence, according to a synchronous one, *Sync$\mathcal{F}$*. By using the transformation offered by Lemma 3.1, the theorem follows. □

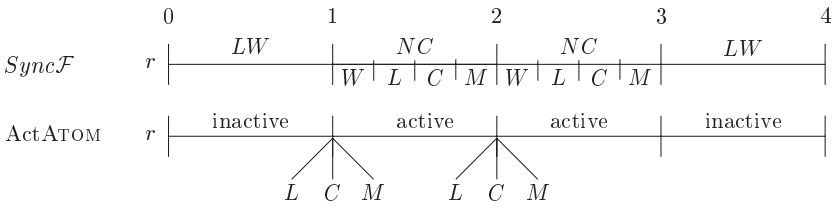

Fig. 4. Example of the idea behind Lemma 3.1.

The vice versa is not true. To show this result, we introduce a problem that can be solved in the atomic setting, but not in the asynchronous one.

Definition 3.1 (Movement Awareness). The *Movement Awareness* problem $\mathcal{MA}$ is divided in two subtasks $\mathcal{T}_1$ and $\mathcal{T}_2$. In $\mathcal{T}_1$ each 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 has observed r_j in at least three different positions, and after r_j has observed r_i in at least three different positions, for all $j \neq i$.

Note that $\mathcal{MA}$ is solvable only in the non-oblivious setting: in fact, a robot can switch from $\mathcal{T}_1$ to $\mathcal{T}_2$ only if it can remember the previous positions occupied by the other robots.

Lemma 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 $\widetilde{\mathcal{A}}$ that correctly solves $\mathcal{MA}$ in the asynchronous setting. The generic robot r_i 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_i decides to switch to $\mathcal{T}_2$. By hypothesis, $\widetilde{\mathcal{A}}$ solves $\mathcal{MA}$ under any schedule. Let us take $j \neq i$ and consider the particular asynchronous activations schedule such that r_j starts its first *Move* state at time $t' < t$, and finishes it at time $2t$ (that is, at time t, r_j is still executing its first cycle). Then $\mathcal{MA}$ is not correctly solved, since r_j has not started its second cycle at time t yet, hence r_j has not observed r_i in at least three different positions yet, a contradiction. Notice that the proof does not depend on any particular feature of $\widetilde{\mathcal{A}}$, hence it is valid for any $\widetilde{\mathcal{A}}$, proving that no such algorithm can exist. □

An algorithm similar to the one used in [25] to discover the initial configuration ("distribution") of the robots in the system, can be used to solve $\mathcal{MA}$ in the atomic and non-oblivious setting. Namely, each robot starts moving along the direction it locally chooses, e.g., the direction of its local y axis. When a robot r_i observes another robot r_j in at least three different positions, r_i moved at least twice. Moreover, since the actions are instantaneous, r_i can correctly deduce that r_j observed at least three times, hence that r_j observed r_i in at least three different positions. However, a problem can arise: since the robots are indistinguishable, if $n > 2$, then r_i might not be able to determine how r_j has moved, given the configurations at two distinct time instants. A solution to this problem is offered in [25]: each robot r_i memorizes the distance $a_i > 0$ to its nearest neighbor when it becomes active for the first time and moves at most distance $a_i/2^{k+1}$ in the kth move. In this way, r will remain in the interior of the $a_i/2$-neighborhood of its initial position, and thus every robot can correctly determine which robot has moved to which position. Therefore, r_i can correctly start $\mathcal{T}_2$ when it observes all $r_j \neq r_i$ in at least three different positions.

Hence, we can state the following:

Lemma 3.3. *$\mathcal{MA}$ is solvable in* ATOM *in the non-oblivious setting.*

From the above results, we can finally state the following:

Theorem 3.2. $\mathfrak{C} \subset \mathfrak{Z}$.

A question that arises is: Does it happen in the oblivious case? Unfortunately, we do not yet have an answer. Our conjecture, however, is that the result stated in Theorem 3.2 also holds 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 cannot tell if r' moved since the last cycle or not: every observation is like the first one. 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 might not be "coherent" with the configuration at the time of the *Move*. This clearly does not happen in ATOM, where the possible states a robot can be in are executed atomically.

Theorem 3.1 tells us that any problem P solvable in the asynchronous time setting can be solved in the atomic one: we first take a solution algorithm for P in CORDA, and then we transform it into an algorithm $\mathcal{A}_v$ valid in ATOM. Vice versa, Theorem 3.2 states that not all problems that are solvable in ATOM are solvable in CORDA. Since it is usually easier to think in terms of the atomic model, in the following we want to focus on the solution algorithms that solve problems in ATOM. As already noted in Section 3, algorithms designed in ATOM can be run in CORDA too. So we ask the following: Given an algorithm $\mathcal{A}$ that solves a given problem P in ATOM, when can this algorithm be used to solve the same problem in CORDA? What are the conditions that allow $\mathcal{A}$ to solve the same problem in the asynchronous model too?

The first observation regards the "coherence" of the computations with respect to the snapshots. In particular, as already highlighted, $\mathcal{A}$ must correctly handle computations that are done on "outdated" observations. More precisely, $\mathcal{A}$ does not work in CORDA if

(1) it assumes that, given any robot r_i, the configuration of robots does not change between the time the snapshot has been taken and the time when the computation starts, in any of the cycles r_i executes.

Note that (1) implies that the algorithm will not work if

(1A) it assumes that when a robot makes its observation, all the other robots are still; or, alternatively, that no moving robot can be observed *while* in movement;
(1B) it assumes that, for all robots that at a given time $t \geq 0$ are computing, $\mathcal{A}$ takes as input the same configuration of robots.

An example of (1A) is given by the proof of Lemma 3.2, where $\widetilde{\mathcal{A}}$ implicitly assumes that no robot can be seen while it moves.

The second important aspect that $\mathcal{A}$ must be capable of facing is the "coherence" of the movements with respect to the computations. More precisely, $\mathcal{A}$ does not work in

CORDA if

(2) it assumes that, given any robot r_i, the configuration of robots does not change between the time the computation starts and the time the movement begins, in any of the cycles r_i executes.

In the next section we present a case study that shows an example of an algorithm that solves the Gathering problem assuming (2), hence does not solve the problem in CORDA.

4. Case Study: Oblivious Gathering with Unlimited Visibility

In the *Gathering* problem in the unlimited visibility setting, the robots are asked to gather in a not predetermined point in the plane, denoted by $\mathfrak{p}$, in a finite number of cycles.

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 any configuration in which all the robots occupy distinct positions.

An oblivious algorithm to solve this problem has been presented in [2] and [25] for ATOM (called Algorithm 5 in the Appendix); we show that the given solution assumes Condition (2) described in previous section, hence does not work in CORDA.

The intuition behind the algorithm 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 (the *gathering* point). 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 (i.e., if on a given position in the plane there is more than one robot). In [2] and [25] this capability is never mentioned, but it is clearly used implicitly. However, it is necessary in order to solve the problem, as shown in [21].

Theorem 4.1. *Algorithm* 5 *does not correctly locate a unique gathering point when run in* CORDA, *in the unlimited visibility setting*.

Proof. The key to proving the theorem is to show that the correctness of Algorithm 5 relies on Condition (2). In particular, we give an initial configuration of the robots and describe an asynchronous activation schedule that leads to having two gathering points in the plane, if Condition (2) is violated.

Let us suppose we have four robots r_i, $i = 1, 2, 3, 4$, that at the beginning are on a circle $\mathcal{C}$, with r_2 and r_4 occupying the ending points of a diameter of $\mathcal{C}$ (as pictured in Figure 5, Cycle 1). In the following the positions of the robots are indicated by p_i, $i = 1, 2, 3, 4$. A possible schedule of the robots is described in what follows:

Cycle 1. At the beginning the four robots are in distinct positions on a circle $\mathcal{C}$. r_1 and r_2 enter the *Look* state, while the others are in *Wait*. After having observed, r_1 and r_2 enter the *Compute* state: we assume that r_2 is computationally slower than r_1. Therefore, r_1 decides to move towards the center of $\mathcal{C}$ (Part 2.3 of Algorithm 5), while r_2 is still 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*.

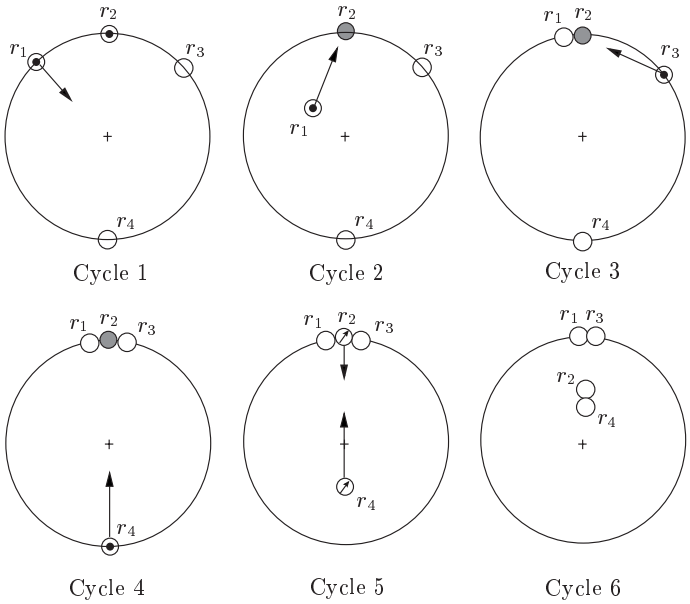

Fig. 5. Proof of Theorem 4.1. The dotted circles indicate the robots in the *Look* state; the grey ones the robots in the *Compute* state; the circle with an arrow inside are the robots that are moving; the white circles represent the robots in *Wait* . The arrows indicate the direction of the movement computed in the *Compute* state.

Cycle 2. r_1 is inside $\mathcal{C}$, while the other robots are still on $\mathcal{C}$. Now r_1 observes again (already in its second cycle) and, according to Part 2.1 of the algorithm, decides to move towards 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 one robot 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 is 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 $\mathcal{C}$ (Part 2.3 of the algorithm), thus violating Condition (2). r_2 starts moving towards the center of $\mathcal{C}$ after r_4 passes over the center of $\mathcal{C}$ in its move towards p; r_1 is in *Wait*.

Cycle 6. r_2 and r_4 are moving in opposite directions on the same diameter of $\mathcal{C}$, and they stop exactly on the same point p'.

At this time there are *two* gathering points in the plane, namely p and p' with $p \neq p'$. Therefore, Algorithm 5 does not correctly locate a unique gathering point. □

5. Conclusions

In this paper we discussed two models, ATOM [2], [25] and CORDA [10]–[12], whose main focus is to study 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. These studies want to gain a better understanding of the power of the distributed control from an algorithmic point of view; specifically, the goal is 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. To our knowledge, these are the only approaches to the study of the control and coordination of mobile units in this perspective.

We showed that the different way in which the asynchronicity is modeled in ATOM and CORDA is the key feature that renders the two models different: in ATOM the robots operate executing *instantaneous actions*, while in CORDA *full asynchronicity* is modeled, and the robots execute their states in a finite, but otherwise unpredictable, amount of time. In particular, we showed that $\mathfrak{C} \subset \mathfrak{Z}$ in the non-oblivious setting, where $\mathfrak{C}$ and $\mathfrak{Z}$ denote the class of problems that are solvable in CORDA and ATOM, respectively. One open issue is to prove this result also in the oblivious setting.

Finally, we presented a set of conditions that allow us to recognize whether an algorithm designed in ATOM to solve a given problem can be used to solve the same problem in CORDA. We have also shown that the ATOM algorithm presented in [25] to solve the oblivious gathering in the unlimited visibility setting, violates one of those conditions, hence does not work in the asynchronous environment modeled by CORDA.

We feel that the approach used in CORDA better describes the way a set of independently moving robots interact in real robotic applications, where the asynchronicity of robots' actions is of great importance; this would motivate further investigations in coordination problems in a distributed, asynchronous environment using the *fully asynchronous* approach. Issues that merit further research concern the operating capabilities of the modeled robots. In fact, it would be interesting to look at models where robots have different capabilities. For instance, it could be studied how the robots can use some kind of direct communication; or the robots could be equipped with just a bounded amount of memory (*semi-obliviousness*), and the relationship between amount of memory and solvability of the problems could be analyzed, or how semi-obliviousness would affect the self-stabilization property of the oblivious algorithms [6].

Other features that could 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.

The relationship between memory and the ability of robots to complete given tasks, dimensional robots, obstacles in the environment that limit the visibility and that moving robots must avoid or push aside, suggest that the algorithmic nature of distributed coordination of autonomous, mobile robots merits further investigation.

Acknowledgments

I thank Vincenzo Gervasi, Nicola Santoro, Anna Bernasconi, Valentina Ciriani, and the anonymous referees for the discussions and comments that helped with the writing of this paper.

Appendix. Oblivious Gathering in ATOM

In this appendix we report the oblivious algorithm described in [25] that lets the robots gather in a point in ATOM, in the unlimited visibility setting. In particular, we report the part of the algorithm that lets the robots achieve a configuration where a unique point p with multiplicity greater than one is determined. When such a configuration is achieved, all the robots will gather on p. Note that the algorithm works for $n \geq 3$ robots. In fact, in [25], it has been shown that the problem is not solvable for $n = 2$ robots.

Algorithm 1 (Point Formation Algorithm in ATOM, Unlimited Visibility [25]).

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 towards 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 towards 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.

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 cycle, 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. $n \geq 4$; $\mathcal{C}_t$ 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 $\mathcal{C}_t$, then r moves towards the position of any robot, say r', on the circumference of $\mathcal{C}_t$ 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 $\mathcal{C}_t$, then these robots move towards the center of $\mathcal{C}_t$ while all other robots remain stationary (so that the center of $\mathcal{C}_t$ 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 $\mathcal{C}_t$, then every robot moves towards the center of $\mathcal{C}_t$. Since all robots can move up to at least a constant distance $\sigma > 0$ in one cycle, if Part 2.3 continues to hold, then eventually the radius of $\mathcal{C}_t$ 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 $\mathcal{C}_t$ or (ii) there is exactly one robot r at the center of $\mathcal{C}_t$, and therefore there is a

robot that is not on $\mathcal{C}_{t'}$ (and the problem is reduced to Part 2.1 or Part 2.2) since a cycle passing through r and a point on $\mathcal{C}_t$ intersects with $\mathcal{C}_t$ at most at two points.

References

[1] N. Agmon and D. Peleg. Fault-Tolerant Gathering Algorithms for Autonomous Mobile Robots. In *Proceedings of the* 15*th ACM–SIAM Symposium on Discrete Algorithms* (*SODA*), pages 1063–1071, 2004.

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

[3] T. Balch and R. C. Arkin. Behavior-Based Formation Control for Multi-Robot Teams. *IEEE Transactions on Robotics and Automation*, 14(6), December 1998.

[4] G. Beni and S. Hackwood. Coherent Swarm Motion under Distributed Control. In *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems* (*DARS*), pages 39–52, 1992.

[5] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the Gathering Problem. In *Proceedings of the* 30*th International Colloquium on Automata*, *Languages and Programming* (*ICALP*), 2003.

[6] X. A. Debest. Remark About Self-Stabilizing Systems. *Communication of the ACM*, 238(2):115–117, February 1995.

[7] X. Défago and A. Konagaya. Circle Formation for Oblivious Anonymous Mobile Robots with No Common Sense of Orientation. In *Proceedings of the Workshop on Principles of Mobile Computing*, pages 97–104, 2002.

[8] S. Dolev. *Self-Stabilization*. The MIT Press, Cambridge, MA, 2000.

[9] E. H. Durfee. Blissful Ignorance: Knowing Just Enough to Coordinate Well. In *Proceedings of the International Conference on Multi-Agent Systems* (*ICMAS*), pages 406–413, 1995.

[10] 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 of the* 10*th Annual International Symposium on Algorithms and Computation* (*ISAAC*), pages 93–102. LNCS 1741. Springer-Verlag, Berlin, December 1999.

[11] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *Proceedings of the IEEE Intelligent Vehicles Symposium* (*IV*), pages 480–485, 2000.

[12] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Autonomous Mobile Robots with Limited Visibility. In *Proceedings of the* 18*th International Symposium on Theoretical Aspects of Computer Science* (*STACS*), pages 247–258. LNCS 2010. Springer-Verlag, Berlin, February 2001.

[13] V. Gervasi and G. Prencipe. Robotic Cops: the Intruder Problem. In *Proceedings of the* 2003 *IEEE Conference on Systems*, *Man and Cybernetics* (*SMC*), pages 2284–2289, October 2003.

[14] V. Gervasi and G. Prencipe. Coordination without Communication: the Case of the Flocking Problem. *Discrete Applied Mathematics*, 143:203–223, 2004.

[15] 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* (*PODC*), pages 50–61, 1984.

[16] Y. Kawauchi, M. Inaba, and T. Fukuda. A Principle of Decision Making of Cellular Robotic System (CEBOT). In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 833–838, 1993.

[17] M. J Matarić. Interaction and Intelligent Behavior. Ph.D. thesis, MIT, May 1994.

[18] S. Murata, H. Kurokawa, and S. Kokaji. Self-Assembling Machine. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 441–448, 1994.

[19] L. E. Parker. On the Design of Behavior-Based Multi-Robot Teams. *Journal of Advanced Robotics*, 10(6):547–578, 1996.

[20] G. Prencipe. Instantaneous Actions vs. Full Asynchronicity: Controlling and Coordinating a Set of Autonomous Mobile Robots. In *Proceedings of the VII Italian Conference on Theoretical Computer Science* (*ICTCS*), pages 185–190, October 2001.

[21] G. Prencipe. On the Feasibility of Gathering by Autonomous Mobile Robots. In *Proceedings of the* 12*th International Colloquium on Structural Information and Communication Complexity* (*SIROCCO*), pages 246–261. LNCS 3499. Springer-Verlag, Berlin, 2005.

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

[23] 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, 1993.

[24] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots—Formation and Agreement Problems. In *Proceedings of the Third Colloquium on Structural Information and Communication Complexity* (*SIROCCO*), pages 313–330, 1996.

[25] I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *SIAM Journal of Computing*, 28(4):1347–1363, 1999.

Received July 31, 2002, *and in final form November* 9, 2004. *Online publication June* 29, 2005.