Instantaneous Actions vs. *Full Asynchronicity*: Controlling and Coordinating a Set of Autonomous Mobile Robots

Giuseppe Prencipe

Dipartimento di Informatica, Università di Pisa
Corso Italia, 40, 56100 - Pisa, Italy
prencipe@di.unipi.it

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 two of these studies [1,6,14] (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). First, their main features are described. Then, the main differences are highlighted, showing the relationship between the class of problems solvable in the two models.

Keywords: Mobile Robots, Distributed Coordination, Distributed Models, Computability.

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

A. Restivo, S. Ronchi Della Rocca, L. Roversi (Eds.): ICTCS 2001, LNCS 2202, pp. 154–171, 2001.

coordinate and control the mobile units, in order to exploit to the utmost the presence of many elements moving independently.

Several studies have been conducted in recent years in different fields. In the engineering area we can cite the Cellular Robotic System (CEBOT) of Kawaguchi *et al.* [9], the Swarm Intelligence of Beni *et al.* [3], and the Self-Assembly Machine ("fructum") of Murata *et al.* [11]. In the AI community there has been a number of remarkable studies: social interaction leading to group behavior by Matarić [10]; selfish behavior of cooperative robots in animal societies by Parker [12]; 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.

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*. 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 is by Suzuki *et al.* [1,13,14]. 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. The second is by Flocchini *et al.* [6,8]: they present a model (that we will refer to as Corda – COordination and control of a set of Robots in a totally Distributed and Asynchronous environment), that 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. In both studies, the modeled robots are rather *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 the models, 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.

Although the model of Suzuki *et al.* (which we will refer to as SYm) and Corda share some features, they differ in some aspects that render the two models quite different. In this paper we highlight these differences, focusing in particular on the different approach in modeling the asynchronicity of the envi-

ronment in which the robots operate, and showing that the algorithms designed on SYm do not work in general on CORDA.

In Section 2.1, SYm 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 SYm. In Section 4, we present a case study: we analyze the *oblivious gathering* problem, showing that the algorithmic solutions designed for SYm do not work in CORDA. 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 present the approaches used in SYm and CORDA to model the control and coordination of a set of autonomous mobile robots. In particular, we first present the common features in the two models, and successively present in detail the *instantaneous action* of SYm, and the *full asynchronicity* of CORDA, that model the interactions between the robots.

2.1 Common Features

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 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[1]; otherwise, it is called *non oblivious*. Moreover, there are no

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

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 is in three *states*: (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 states (i) to (iii) are executed atomically (instantaneously), while this assumption is dropped in CORDA. In the following we better describe how the asynchronicity is approached in the two models.

2.2 The *Instantaneous Actions* 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 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[2]), 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 is then capable of traveling 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 states (i)–(iii) *instantaneously*, in the sense that a robot that is active and observes at t, has already reached its destination

[2] 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.

point p at $t + 1$. Therefore, 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*).

2.3 The *Full Asynchronicity* of CORDA

Similarly to SYm, each robot repeatedly executes four states. 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 (*Look*), 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 states are described more formally in the following.

1. **Wait** The robot is idle. A robot cannot stay infinitely idle.
2. **Look** The robot observes the world by activating its sensors which will return a snapshot of the positions of all other robots with respect to its local coordinate system. Each robot 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 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 algorithm. The result of the computation can be a destination point or a *null movement* (i.e., the robot decides to not move).
4. **Move** If the result of the computation was a *null movement*, the robot 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 A2 below). Hence, the robot can only go towards its goal, but it cannot know how far it will go in the current cycle, because it can stop anytime during its movement [3].

A computational cycle is defined as the sequence of the *Wait-Look-Compute-Move* states; 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(Computational Cycle) The amount of time required by a robot r to complete a computational cycle is not infinite, nor infinitesimally small.

A2(Distance) The distance traveled by a robot r in a *Move* is not infinite. Furthermore, it is not infinitesimally small: there exists an arbitrarily small constant $\delta_r > 0$, such that if the result of the computation is not a *null*

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

movement and the destination point is closer than δ_r, r will reach it; otherwise, r will move towards it of at least δ_r. In the following, we shall use $\delta = \min_r \delta_r$.

Therefore, 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 *fully asynchronous*, in the sense that there is no common notion of time, and a robot observes the environment at unpredictable time instants. Moreover, no assumptions on the cycle time of each robot, and on the time each robot elapses to execute each state of a given cycle are made. It is only assumed that each cycle is completed in finite time, and that the distance traveled in a cycle is finite. Thus, each robot can take its own time to compute, or to move towards some point in the plane: in this way, it is possible to model different computational and motorial speeds of the units. Moreover, every robot can be seen *while* it is moving by other robots that are observing. This feature renders more difficult the design of an algorithm to control and coordinate the robots. For example, when a robot starts a *Move* state, it is possible that the movement it will perform will not be "coherent" with what it observed, since, during the *Compute* state, other robots can have moved.

3 *Instantaneous Action* vs. *Full Asynchronicity*

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

Let us first introduce the definition of a valid *activation schedule* for an algorithm in CORDA.

Definition 1. *Given an algorithm $\mathcal{A}$, an activation schedule for $\mathcal{A}$ in* CORDA *is defined as a function $\mathcal{F}(t) =< \mathbb{W}(t), \mathbb{L}(t), \mathbb{C}(t), \mathbb{M}(t) >$, where $\mathbb{W}(t)$ is a set of pairs (r, t'), such that*

1. *r is a robot that is in the* Wait *state at time t,*
2. *$t' > t$, and*

3. in $\mathbb{W}(t)$ *there is at most one pair per each robot in the system*

($\mathbb{L}(t)$*,* $\mathbb{C}(t)$*, and* $\mathbb{M}(t)$ *are defined similarly for the* Look*,* Compute*, and* Move *states, respectively).*

Definition 2. *An activation schedule is* valid*, if the following conditions hold: (i)* $(r, t') \in \mathbb{W}(t) \Rightarrow \forall\, t \leq t'' < t'$*,* $(r, t') \in \mathbb{W}(t'')$ *(a similar condition applies also for* $\mathbb{L}(t)$*,* $\mathbb{C}(t)$*, and* $\mathbb{M}(t)$*); (ii) for all* t*,* $\mathbb{W}(t)$*,* $\mathbb{L}(t)$*,* $\mathbb{C}(t)$*, and* $\mathbb{M}(t)$ *constitute a partition of all the robots in the system.*

An algorithm $\mathcal{A}$ correctly solves a problem $\mathcal{P}$ in Corda, if, given any valid activation schedule for $\mathcal{A}$, the robots accomplish the task defined by $\mathcal{P}$ in a finite number of cycles. Let us denote by $\mathfrak{C}$ and $\mathfrak{Z}$ the class of problem that are solvable in Corda and SYm, respectively. We are now ready to show that SYm is at least as powerful as Corda, that is $\mathfrak{C} \subseteq \mathfrak{Z}$.

Theorem 1. *Any algorithm that correctly solves a problem* $\mathcal{P}$ *in* Corda*, correctly solves* $\mathcal{P}$ *also in* SYm*.*

Proof. Let $\mathcal{A}$ be an algorithm that solves a given problem $\mathcal{P}$ in Corda. In order to prove that $\mathcal{A}$ solves $\mathcal{P}$ also in SYm, we show that any execution of $\mathcal{A}$ in SYm corresponds to an activation schedule in Corda. Hence, since by hypothesis $\mathcal{A}$ correctly solves $\mathcal{P}$ in Corda, the theorem follows.

Let us execute $\mathcal{A}$ in SYm, and let $\mathcal{E}(\overline{t})$ be the set of robots that are active at time $\overline{t}$. Therefore, all the robots $\mathcal{E}(\overline{t})$ finish to execute their cycle at time $\overline{t}+1$. The activation schedule $\mathcal{F}(t)$, for all $\overline{t} \leq t < \overline{t}+1$, in Corda for $\mathcal{A}$ corresponding to the portion of the execution of $\mathcal{A}$ in SYm starting at time $\overline{t}$ and ending at time $\overline{t}+1$, is defined as follows (see Figure 1). If $r \in \mathcal{E}(\overline{t})$, then for all $\overline{t} \leq t < t_1$, $(r, t_1) \in \mathbb{L}(t)$; for all $t_1 \leq t < t_2$, $(r, t_2) \in \mathbb{C}(t)$; for all $t_2 \leq t < t_3$, $(r, t_3) \in \mathbb{M}(t)$; and for all $t_3 \leq t < \overline{t}+1$, $(r, \overline{t}+1) \in \mathbb{W}(t)$. Otherwise, for all $\overline{t} \leq t < \overline{t}+1$, $(r, \overline{t}+1) \in \mathbb{W}(t)$. In other words, all the robots in $\mathcal{E}(\overline{t})$ start their *Look* state, while all the others are in *Wait*. Moreover, all these robots execute their three states perfectly synchronized, so that they start their next cycle all together. Inductively, $\mathcal{F}(t)$, for all $\overline{t}+1 \leq t < \overline{t}+2$, corresponding to the next cycle (from time $\overline{t}+1$ to $\overline{t}+2$) of the execution of $\mathcal{A}$ in SYm is constructed.

Therefore, any execution of $\mathcal{A}$ in SYm corresponds to a valid activation schedule for $\mathcal{A}$ in Corda. Since by hypothesis $\mathcal{A}$ correctly solves $\mathcal{P}$ on Corda, the robots will correctly accomplish their task in SYm, and the theorem follows.

Corollary 1. *Any problem that can be solved in* Corda*, can be solved in* SYm*; hence* $\mathfrak{C} \subseteq \mathfrak{Z}$*.*

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

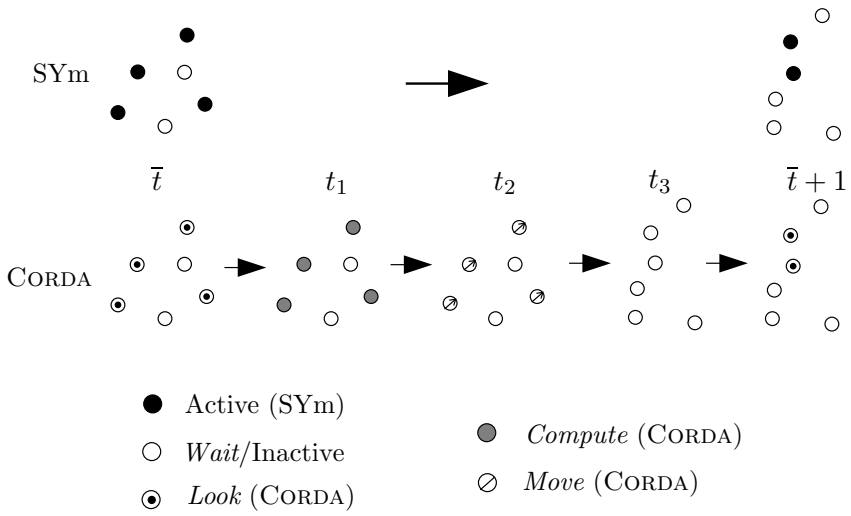

Fig. 1. The activation schedule defined in Theorem 1

Definition 3 (Movement Awareness). *The* Movement Awareness *problem* $\mathcal{MA}$ *is divided in two subtasks* $\mathcal{T}_1$ *and* $\mathcal{T}_2$. *In* $\mathcal{T}_1$, *robot* r_i, $1 \leq i \leq n$, *simply moves along a direction it chooses arbitrarily;* r_i *can start* $\mathcal{T}_2$ *only after it observed* r_j *in at least three different positions, and after* r_j *observed* r_i *in at least three different positions, for all* $j \neq i$.

Theorem 2. *There exists no algorithm that solves* $\mathcal{MA}$ *in* CORDA *in the non oblivious setting.*

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

An algorithm similar to the one used in [14] to discover the initial configuration ("distribution") of the robots in the system, can be used to solve in SYm $\mathcal{MA}$. Namely, each robot starts moving along the direction it locally chooses, e.g. the direction of its local y axis. When a robot r observes another robot r' in at least three different positions, r moved at least twice. Moreover, since in SYm the actions are instantaneous, r can correctly deduce that r' observed at least twice, hence that r' observed r in at least three different positions.

Therefore, r can correctly start $\mathcal{T}_2$ when it observes all $r' \neq r$ in at least three different positions. Hence, we can state the following

Theorem 3. *$\mathcal{MA}$ is solvable in* SYm*, in the non oblivious setting.*

Corollary 2. $\mathfrak{C} \subset \mathfrak{Z}$.

A question that arises is: what does it happen in the oblivious case? Unfortunately, we do not yet have an answer. Our conjecture, however, is that the result stated in Corollary 2 holds also in the oblivious case. In the non oblivious setting, the fact that in Corda a robot can be seen by its fellows *while* it is moving is crucial to prove $\mathfrak{C} \subset \mathfrak{Z}$. This is not the case in the oblivious setting. In fact, since the robots have no memory of robots' positions observed in the past, every time a robot r observes another robot r', r can not tell if r' moved since last cycle or not, and every observation is like the first one (that is every time r observes, is like the execution begins). Hence, we believe that the key to prove $\mathfrak{C} \subset \mathfrak{Z}$ in the oblivious case is related to the fact that in Corda the positions of the robots between a *Look* and a *Compute* can change, hence the computation can be done on "outdated" data. In other words, if r executes the *Look* at time t and the *Compute* at time $t' > t$, the set of robots' positions at t and at t' can be clearly different; hence r computes its destination point on the old data sensed at time t, implying that the movement will not be "choerent" with what it observed at time t. This clearly does not happen in SYm, where the possible states a robot can be in are executed instantaneously.

4 Case Study: Oblivious Gathering

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

The problem we consider is the *gathering problem*: the robots are asked to gather in a not predetermined point in the plane in a finite number of cycles. An algorithm is said *to solve* the gathering 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. This is the only problem, to our knowledge, solved with an oblivious algorithm in SYm [1,14]. In the following, we will analyze both the unlimited and limited visibility setting.

4.1 The Unlimited Visibility Setting

An algorithm for solving the gathering problem in SYm in the unlimited visibility setting (called Algorithm 1 in Appendix A.1) is presented in [14]. The idea is as follows. Starting from distinct initial positions, the robots are moved in such a way that eventually there will be exactly one position, say p, that two or more robots occupy. Once such a situation has been reached, all the robots move towards p. It is clear that such a strategy works only if the robots in the

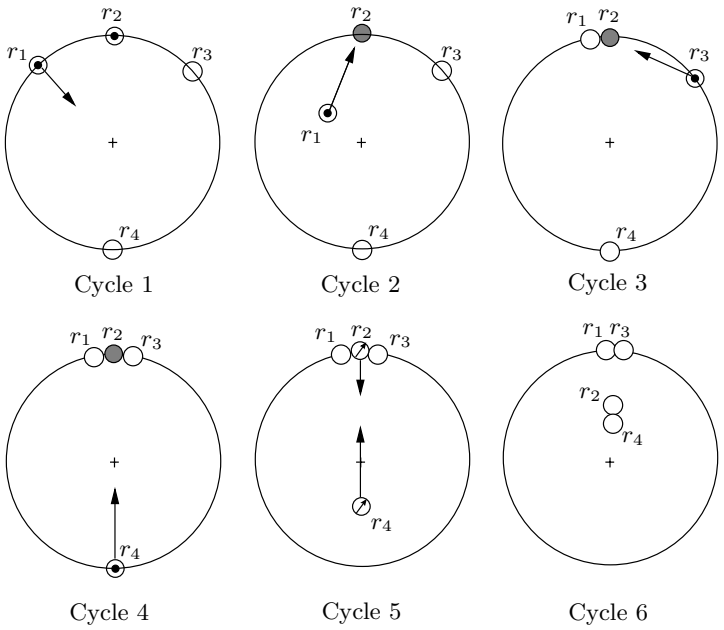

Fig. 2. Proof of Theorem 4. The symbols used for the robots are the same as in Figure 1. The dotted circles indicate the robots in the *Look* state; the grey ones the robots in the *Compute* state; the circle with an arrow inside are the robots that are moving; the white circles represent the robots in *Wait*. The arrows indicate the direction of the movement computed in the *Compute* state

system have the ability to detect the multiplicity. In SYm this capability is never mentioned, but it is clearly used implicitly.

Theorem 4. *Algorithm 1 does not solve the gathering problem in* Corda, *in the unlimited visibility setting.*

Proof. In order to show that Algorithm 1 does not solve the gathering problem in Corda, we give an initial configuration of the robots and describe an activation schedule that leads to having two points in the plane with multiplicity greater than two, thus violating the invariant proven for Algorithm 1, that "eventually there will be exactly one position that two or more robots occupy" [14].

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

Cycle 1 At the beginning the four robots are in distinct positions, on a circle C. r_1 and r_2 enter the *Look* state, while the others are in *Wait*. After having

observed, both of them enter the *Compute* state, and let us assume that r_2 is computationally very slow (or, alternatively, that r_1 is very fast). Therefore, r_1 decides to move towards the center of C (part 2.3 of Algorithm 1), while r_2 is stuck in its *Compute* state. r_1 starts moving towards the center, while r_2 is still in *Compute*, and r_3 and r_4 are in *Wait*.

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

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

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

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

Cycle 6 r_2 and r_4 are moving in opposite directions on the same diameter of C, and they stop exactly on the same point p' (in CORDA a robot can stop before reaching its final destination). There are two points in the plane, namely p and p' with $p \neq p'$, with two robots on each. Therefore, the invariant proven for Algorithm 1, that "eventually there will be exactly one position that two or more robots occupy" [14], is violated.

Remark 1. We note that in Cycle 6 we made use of the possibility that a robot stops before reaching the destination point it computed. The proof, however, works even if we do not assume this; that is, if r_2 and r_4 do not stop before reaching their respective destination points. In fact, if we assume, as in SYm, that the robots simply cross each other without stopping, if (i) the crossing happens in a point $p' \neq p$, and (ii) r_1 enters its 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 4 still holds.

4.2 The Limited Visibility Setting

In [1], an algorithm to solve the gathering problem in SYm in the limited visibility setting (called Algorithm 2 in Appendix A.2) is presented. We recall that, in this setting a robot can see only whatever is at distance V from it. In the following we shortly describe it.

Let us denote by $r_i(t)$ the position of robot r_i at time instant t. The set $P(t) = \{r_1(t), \ldots, r_n(t)\}$ then denotes the set of the robots' positions at t. Define $G(t) = (R, E(t))$, called the *Proximity Graph* at time t, by $(r_i, r_j) \in E(t) \leftrightarrow dist(r_i(t), r_j(t)) \leq V$, where $dist(p, q)$ denotes the Euclidean distance between points p and q. It can be proven that, if the proximity graph is not connected at the beginning, the robots can not gather in a point [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, then r_i does not move. Otherwise, the algorithm chooses x to be the point on the segment $\overline{r_i(t)c_i(t)}$ that is closest to $c_i(t)$ and that satisfies the following conditions:

1. $dist(r_i(t), x) \leq \sigma$. An arbitrary small constant $\sigma > 0$ is fixed, and it is assumed that the distance a robot can travel in one state is bounded by σ (similarly to the ϵ introduced in Section 2.2).
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).

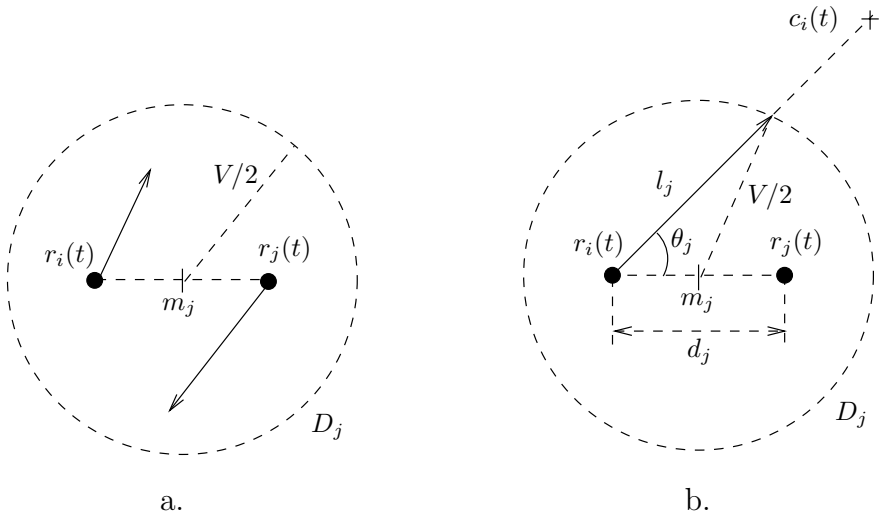

Fig. 3. The algorithm for the gathering problem in SYm, limited visibility setting

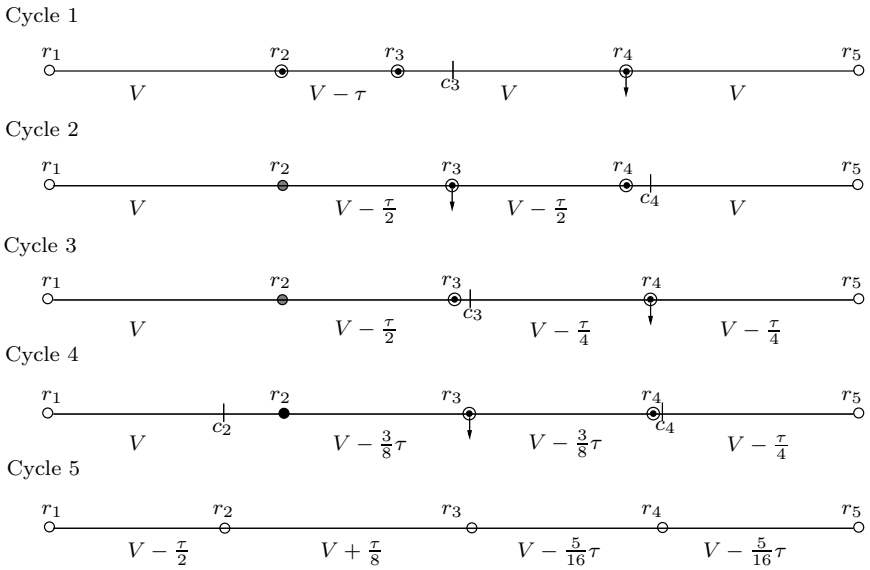

Fig. 4. Proof of Theorem 5. The symbols used for the robots are the same as in Figure 2. A vertical arrow means that a robot decided not to move ($Move = 0$). A robot r_i moves always towards the center c_i of the smallest enclosing circle of all the robots it can see

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

In [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 5. *The algorithm presented in [1] does not solve the gathering problem in* CORDA*, in the limited visibility setting.*

Proof. In order to show that Algorithm 2 does not solve the gathering problem in CORDA, we give an initial configuration of the robots and describe an activation schedule 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].

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 $\tau \leq \sigma$ and $\delta = \tau/16$, where δ is the constant introduced in the Assumption A2 in Section 2.3. At the beginning,

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Cycle 5 The distance between r_2 and r_3 is $V + \tau/8 > V$; so r_2 and r_3 can not see each other anymore, breaking the proximity graph connectivity that we had at the beginning of the cycle. So, the invariant that "robots that are mutually visible at t remain within distance V of each other thereafter" asserted in [1] is violated. Therefore, the theorem follows.

5 Conclusions

In this paper we discussed two models, SYm [1,14], and CORDA [6,7,8], 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 SYm and CORDA, is the key feature that renders the two models different: in SYm the robots operate executing *instantaneous actions*, while in CORDA *full asynchronicity* is modeled, and the robots elapses finite, but otherwise unpredictable, amount of time to execute their states. In particular, we showed that $\mathfrak{C} \subset \mathfrak{Z}$ in the non oblivious setting. Therefore, one open issue is to prove this result also in the oblivious setting.

We feel that the approach used in CORDA better describes the way a set of independently-moving units operates in a totally asynchronous environment; hence the motivation to further investigate coordination problems in a distributed, asynchronous environment using the *fully asynchronous* approach. Issues which merit further research, regard the operating capabilities of the robots modeled. In fact, it would be interesting to look at models where robots have different capabilities. For instance, we could equip the robots with just a bounded amount of memory (*semi-obliviousness*), and analyze the relationship between amount of memory and solvability of the problems, or how it would affect the self-stability property of the oblivious algorithms [7].

Other features that 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, where all the robots in the environment must chase and "catch" a "designated" robot).

Relationship between memory and ability of the 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 would like to thank Paola Flocchini, Nicola Santoro and Vincenzo Gervasi 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. 154, 155, 157, 162, 164, 165, 166, 168, 170, 171
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. 155
3. G. Beni and S. Hackwood. Coherent Swarm Motion Under Distributed Control. In *Proc. DARS'92*, pages 39–52, 1992. 155
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. 155
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. 154, 155, 157, 168
7. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Distributed Coordination of a Set of Autonomous Mobile Robots. In *IEEE Intelligent Veichle 2000*, pages 480–485, 2000. 155, 157, 168
8. P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of Asynchronous Mobile Robots with Limited Visibility. In *STACS 2001*, volume 2010 of *Lecture Notes in Computer Science*, pages 247–258, 2001. 155, 168
9. 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. 155
10. M. J. Matarić. *Interaction and Intelligent Behavior*. PhD thesis, MIT, May 1994. 155
11. S. Murata, H. Kurokawa, and S. Kokaji. Self-Assembling Machine. In *Proc. IEEE Conf. on Robotics and Autom.*, pages 441–448, 1994. 155
12. L. E. Parker. On the Design of Behavior-Based Multi-Robot Teams. *Journal of Advanced Robotics*, 10(6), 1996. 155
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. 155
14. I. Suzuki and M. Yamashita. Distributed Anonymous Mobile Robots: Formation of Geometric Patterns. *Siam J. Comput.*, 28(4):1347–1363, 1999. 154, 155, 157, 161, 162, 163, 164, 168, 170, 171

Appendix

A Oblivious Gathering in SYm

In this appendix, we report the oblivious algorithms described in [1,14] that let the robots gather in a point in SYm, in both the unlimited and limited visibility settings.

A.1 Unlimited Visibility

In the following we report the oblivious algorithm described in [14] that lets the robots achieve a configuration where a unique point p with multiplicity greater than one is determined.

Algorithm 1 (Point Formation Algorithm in SYm, Unlim. Visib.[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.
- **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.

A.2 Limited Visibility

In the following we report the oblivious algorithm described in [14] that lets the robots gather in a point (refer to Figure 3.b).

Algorithm 2 (Point Formation Algorithm in SYm**, Lim. Visib.** [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. $GOAL = dist(r_i(t), c_i(t))$,
5. $MOVE = \min\{GOAL, LIMIT, \sigma\}$,
6. $x =$ point on $\overline{r_i(t)c_i(t)}$ at distance $MOVE$ from $r_i(t)$.