Computing by Mobile Robotic Sensors

Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro

Abstract The research areas of *mobile robotic sensors* lie in the intersection of two major fields of investigations carried out by quite distinct communities of researchers: *autonomous robots* and *mobile sensor networks*. Robotic sensors are micro-robots capable of locomotion and sensing. Like the sensors in wireless sensor networks, they are *myopic*: their sensing range is *limited*. Unlike the sensors in wireless sensor networks, robotic sensors are *silent*: they have no direct communication capabilities. This means that synchronization, interaction, and communication of information among the robotic sensors can be achieved solely by means of their sensing capability, usually called *vision*. In this Chapter, we review the results of the investigations on the computability and complexity aspects of systems formed by these *myopic* and *silent* mobile sensors.

1 Introduction

1.1 Distributed Computing and Mobile Entities

In distributed computing the research focus is on the computational and complexity issues of systems composed of autonomous computational entities interacting with each other (e.g. to solve a problem, to perform a task).

Paola Flocchini
School of Information Technology and Engineering, University of Ottawa, e-mail: flocchin@site.uottawa.ca

Nicola Santoro
School of Computer Science, Carleton University e-mail: santoro@scs.carleton.ca

Giuseppe Prencipe
Dipartimento di Informatica, University of Pisa e-mail: prencipe@di.unipi.it

While traditionally the entities have been assumed to be *static*, recent advances in a variety of fields, ranging from robotics to artificial intelligence to software engineering to networking, have motivated the distributed computing community to address the situation of *mobile* entities. Indeed recently an increasing number of investigations are being carried out on the computational and complexity issues arising in systems of autonomous mobile entities located in a spatial universe $\mathcal{U}$. The entities have storage and processing capabilities, exhibit the same behavior (i.e., execute the same protocol), and can move in $\mathcal{U}$ (their movement is constrained by the nature of $\mathcal{U}$).

Depending on the nature of $\mathcal{U}$, there are two basic settings in which autonomous mobile entities are being investigated. The first setting, sometimes called *graph world* or *discrete universe*, is when the universe is a simple graph; this is for example the case of mobile agents in communication networks (e.g., [15, 26, 36]). The second setting, called sometimes *continuous universe*, is when $\mathcal{U}$ is a region of the 2D (or 3D) space. This is for example the case of robotic swarms, mobile sensor networks, mobile robotic sensors, etc. (e.g., [1, 13, 16, 21, 31, 34, 35, 38, 45, 47, 48, 56, 66, 75, 79, 81, 82, 83]). In both settings, the research concern is on determining what tasks can be performed by such entities, under what conditions, and at what cost. In particular, a central question is to determine what minimal hypotheses allow a given problem to be solved.

In the continuous setting two major research areas can be distinguished, their difference resting on the types of assumptions made, carried out by quite distinct communities of investigators:

- *autonomous robots*, an established and mature research field (which includes swarm robotics and robotic networks), investigated mainly by researchers in robotics, control, artificial intelligence, and more recently by algorithmic researchers;
- *mobile sensor networks*, a new and emerging research field whose investigations are carried out mostly as an extension of the more traditional (static) sensor networks.

These two areas have overlapping boundaries, and their intersection is a region of surprising and interesting convergence of research interests. An important such region is the area of *mobile robotic sensors*, the topic of this Chapter.

1.2 Robots, Sensors, and Mobility

The addition of motorial capabilities to a computational entity not only empowers the entity in non trivial ways, but it also, and more importantly, empowers the system employing such entities. This empowerment takes many forms and displays different aspects. This is particularly evident in the case of

wireless sensor networks. Indeed empowering the sensors with mobility allows the network to perform tasks and solve problems which would be impossible to do with static sensors. Indeed, mobile sensors have gained attention lately as important tools for a wide range of applications and tasks, such as search and rescue, exploration and mapping, evaluation of civil infrastructure, military operations.

The first proposals for the use of mobility in sensor networks have been for *exogenous* solutions: mobile robots are introduced into the network of static sensors to augment the capacities of the system or to simplify the management of the network (e.g., to repair failed sensors, to redeploy sensors so to improve overall coverage, to gather information from the robots, etc.), and this research still continues (e.g., see [5, 6, 27, 54, 67, 84, 86]).

At the same time, a large number of investigators have suggested and studied the use of *endogenous* mobility in sensor networks: *mobile sensor networks*, in which the sensors have processing power, wireless communication, and motion capabilities. The sensors operate in a totally distributed way, moving under their own control, and reacting to the inputs received from the environment where they operate, thus creating computational systems capable of interacting with the physical environment (e.g., see [11, 14, 44, 45, 46, 50, 57, 58, 61, 71, 82, 88]).

At this point, the research results start to merge with the rich existing literature on *autonomous robots*, in particular with that of systems of *microrobots* and of *robotic sensors*, studied also from the control and the computing point of view (e.g., [1, 7, 13, 18, 21, 28, 31, 34, 38, 56, 66, 76, 82, 83]). These investigations differ greatly from each other depending on the assumptions they make. Major differences exist depending on whether: the entities' actions are synchronized (e.g., [13, 47, 81, 83]) or no timing assumptions exist (e.g., [31, 35, 53]); the sensors have persistent memory (e.g., [13, 43, 83]) or are oblivious (e.g., [35, 46, 79]); the sensors have the computational power of Turing machines (e.g., [81, 83]) or are simple Finite State machines (e.g., [4, 16, 30, 47]); the visibility/communication range is limited (e.g., [31, 43, 38, 47, 79]) or extends to the entire region (e.g., [1, 10, 12, 35, 83]).

1.3 Mobile Robotic Sensors

The crucial difference between systems of *robotic sensors* and wireless *sensor networks* is the following. In *sensor networks*, regardless of whether mobile or static, the entities are endowed with both *sensing* and (wireless) *communication* capabilities. However, of these two capabilities, only the latter - wireless communication - is used for synchronization, interaction, and communication of information among the sensors and within the network.

On the contrary, *robotic sensors* are generally endowed solely with *sensing* capability. In other words, they are *silent*: they have no direct communication

capabilities. This means that synchronization, interaction, and communication of information among the sensors and within the network can be achieved solely by means of their sensing capability, usually called *vision*. This characteristic is indeed the same one assumed in most of the traditional research on autonomous robots[1].

The lack of direct means of communication has many computational drawbacks; indeed, the fact that robotic sensors must rely solely on their sensing capabilities for all their interactions is a severe limitations. It does however have one advantage, in the determination of an entity's neighbours. In fact, in systems of robotic sensors, the determination of one's neighbours is done by sensing capabilities (e.g., vision): any sensor in the sensing radius is detected even if inactive, and thus no other mechanisms are needed. On the other hand, in traditional wireless sensor networks, determination of the neighbours is achieved by radio communication; since an inactive sensor does not participate in any communication, the simple activity of determining one's neighbours, to be completed, requires the use of randomization or the presence of sophisticated synchronization and scheduling mechanisms (e.g., [63, 69, 74]).

Another important difference with mobile sensor networks is that robotic sensors, like most autonomous robots, are often equipped with a much larger energy reserve, or have self-charging capability (e.g., on board PV or ability to plug into the power grid to recharge their batteries). Hence energy is a concern but not as crucial as in mobile sensor networks.

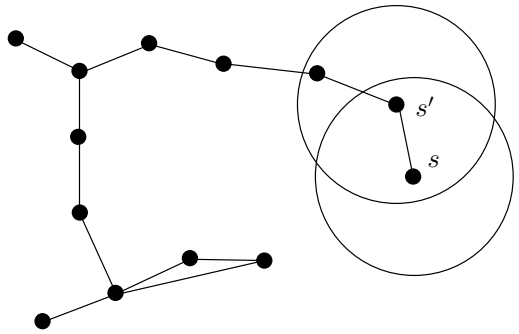

Fig. 1 Limited visibility: a sensor can only see sensors that are within its radius of visibility.

The key feature that robotic sensors share with mobile sensor networks is that they are *myopic*: their sensing range is *limited* (see Figure 1). Precisely this feature constitutes the key difference between robotic sensors and tra-

[1] The notable exceptions are the *robotic networks* studied in the control community that assume and use direct communication, e.g. [7, 65].

ditional models of autonomous robots and micro-robots. In fact, algorithmic robotic research usually assume *unlimited visibility*: the entities are capable of determining the location of all other regardless of their position in the region, e.g. [1, 9, 12, 19, 35, 52, 53, 70, 77, 81, 83, 91]. Additional differences between robotic sensors and traditional models of autonomous robots and micro-robots robotic sensors are that usually the robots are more powerful (both memory-wise and computationally) than sensors, and typically there is no requirement for the robots to reach a state of static equilibrium (e.g., in most cases the swarm just converges towards a desired formation or pattern).

Summarizing, robotic sensors are

1. *mobile*, like mobile sensor networks and autonomous robots;
2. *silent*, like traditional autonomous robots;
3. *myopic*, like sensor networks.

The purpose of this Chapter is to present and discuss the research efforts on systems of robotic sensors. It is organized as follows. In Section 2 we will present in details the computational model and introduce the formalism and terminology. We will then review the research results on computing by mobile robotic sensors. The investigations have been focusing on three fundamental problems: *Self Deployment*, *Pattern Formation*, and *Gathering*; they will be discussed in Sections 3, 4 and 5, respectively.

2 Modeling Mobile Robotic Sensors

2.1 Capabilities

The system is composed of a set $\mathbb{S} = \{s_1, \ldots, s_n\}$ of n mobile robotic sensors operating in a spatial universe $\mathbb{U} \subseteq \mathbb{R}^2$.

A *mobile robotic sensor* (or simply *sensor*) $s \in \mathbb{S}$ is modeled as a computational unit: it has its own local memory and it is capable of performing local computations.

A sensor is endowed with sensorial capabilities and it can perceive the spatial environment $\mathbb{U}$ and the sensors in it, within a fixed distance $v > 0$, called *visibility radius*. Each sensor has its own local coordinate system: 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. However, the local coordinate systems of the sensors might not be consistent with each other.

Each sensor is endowed with motorial capabilities; it can turn and move in any direction. A move may stop before the sensor reaches its destination, e.g. because of limits to its motion energy; however, it is assumed that the distance traveled in a move by s is not infinitesimally small (unless it brings

the sensor to its destination): there exists a constant $\delta_s > 0$, such that, if the destination is closer than δ_s, s will reach it; otherwise, s will move towards it by at least δ_s. Note that, without this assumption, it would be impossible for s to ever reach its destination, following a classical Zenonian argument. In the following, we shall use $\delta = \min_s \delta_s$.

The sensors are *silent*: they have no means of direct communication of information to other sensors. Thus, any communication occurs in a totally implicit manner, by observing the other sensors' positions. Let $s(t)$ denote the position of sensor s at time t; when no ambiguity arises, we shall omit the temporal indication.

The sensors are *autonomous* (i.e., without a central control) and *identical* (i.e., they execute the same protocol). They might be *anonymous* (i.e., a priori indistinguishable by their appearance and without identifiers that can be used during the computation).

2.2 Behavior

At any point in time, a sensor is either *active* or *inactive*. When *active*, a sensor s performs the following three operations, each in a different state:

1. (State *Locate*) It observes the spatial environment $\mathbb{U}$ and the sensors in it, within its visibility radius $v > 0$. As a result, it determines, in its own coordinate system, a snaphshot of the positions of the sensors in its *circle of visibility* at that time. The *circle of visibility* of s at time t is the surrounding circle of s in its most recent *Locate*. The *Locate* state can be assumed, without loss of generality, to be istantaneous.[2]
2. (State *Compute*) It performs a local computation, according to an algorithm (the same for all sensors) that takes in input the result of its previous *Locate*, and returns a destination point. Hence, the algorithm, the same for all sensors, will specify which operations a sensor must perform whenever it is active.
3. (State *Move*) It moves towards the computed destination point; if the destination point is the current location, the sensor stays still. A move may stop before the sensor reaches its destination, e.g. because of limits to the sensor's motion energy.

When *inactive* a sensor is in *Sleep* state:

4. (State *Sleep*) It is idle and does not perform any operation.

Summarizing, the sensors operate in a continuous *Locate-Compute-Move-Sleep* life cycle.

[2] Any time spent to activate its sensors (before the snapshot is taken) and to process the information retrieved with the snapshot, will be charged to the *Sleep* and to the *Compute* state, respectively.

2.3 Synchronization

Depending on the degree of synchronization among the life cycles of different sensors, three sub-models are traditionally identified: *synchronous*, *semi-synchronous*, and *asynchronous*.

In the *synchronous* model (SYNC), the cycles of all sensors are fully synchronized: the sensors become active all at the same time and each operation of the life cycle is performed by all sensors simultaneously. Alternatively, there is a global clock tick reaching all sensors simultaneously, and a sensor's cycle is an instantaneous event that starts at a clock tick and ends by the next. As a consequence, no sensor will ever be observed while moving. This model is used e.g. in [38, 21, 83].

In the *semi-synchronous* model (SSYNC), there is a global clock tick reaching all sensors simultaneously, and a sensor's activities are an instantaneous event that starts at a clock tick and ends by the next. Hence, also in this model, no sensor will ever be observed while moving. However, at each clock tick, some sensors might not become active. The unpredictability of which sensors become active at a clock tick is restricted by the fact that at every clock tick at least one sensor is active, and every sensor becomes active infinitely often. This model, sometimes called ATOM, is used e.g. in [1, 9, 12, 13, 19, 21, 83].

In the *asynchronous* model (ASYNC), there is no global clock and the sensors do not have a common notion of time. Furthermore, the duration of each activity (or inactivity) is finite but unpredictable. As a result, sensors can be seen while moving, and computations can be made based on obsolete observations. For example (see Figure 2), sensor s in transit towards its destination, is seen by r; however, s is not aware of r's existence and, if it starts the next cycle before r starts moving, s will continue to be unaware of r. This (realistic but more difficult) model, sometimes called CORDA, is used e.g. in [9, 35, 33, 34, 52, 53, 70].

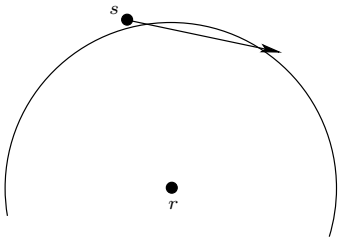

Fig. 2 When s starts moving (the left end of the arrow), r and s do not see each other. While s is moving, r enter state *Locate* and sees s; however, s is still unaware of r. After s passes the visibility circle of r, it is still unaware of r.

2.4 Memory

In addition to its programs, each sensor has a local working memory, or *workspace*, used for computations and to store different amount of information (e.g., regarding the location of its neighbours) obtained during the cycles. Two submodels have been identified, depending on whether or not this workspace is persistent.

In the *oblivious* model, all the information contained in the workspace is *cleared* at the end of each cycle. In other words, the sensors have *no* memory of past actions and computations, and the computation is based solely on what determined in the current cycle. The importance of obliviousness comes from its link to *self-stabilization* and *fault-tolerance*. This model, sometimes improperly called *memoryless*, is used e.g. in [9, 12, 13, 19, 35, 34, 53].

In the *persistent memory* model, all the information contained in the workspace is *legacy*: unless explicitly erased by the sensor, it will persist thoughout the sensor's cycles. This model is commonly used for both wireless sensor networks and micro-robots.

An additional important parameter is the *size* of the persistent workspace. Noticeable are the two extreme cases. One extreme is the *unbounded memory* case, where no information is ever erased; hence sensors can remember all past computations and actions (e.g., see [81, 83]). The other extreme is when the size of the workspace is constant; in this case, the sensors are just *Finite-State Machines* (e.g., [4, 16, 47]) .

3 Self Deployment

3.1 Introduction

The first important problem faced with sensor systems is the effective deployment of the sensors within the spatial universe $\mathbb{U}$, assumed to be *finite*. The deployment must usually satisfy some optimization criteria with respect to the space $\mathbb{U}$ (e.g., uniformity, maximum coverage). In case of static sensors, they are usually deployed by external means, either carefully (e.g., manually installed) or randomly (e.g., dropped by an airplane); in the latter case, the distribution of the sensors may not satisfy the desired optimization criteria.

If the sensing entities are *mobile*, as in the case of mobile sensor networks, vehicular networks, and robotic sensor networks, they are potentially capable to position themselves in appropriate locations without the help of any central coordination or external control, a task called *Self Deployment*.

In this section we consider some of the problems and issues we must face to achieve such a rather complex task; indeed, designing localized algorithms

for efficient and effective deployment of the mobile entities is a challenging research issue.

Some of the initial proposals on the deployment of mobile sensors were still based on centralized approaches, e.g. employing a powerful cluster head to collect the initial location of the mobile sensors and determine their target location [92]. The current research efforts are on the development of local protocols that allow the sensors to move from an initial random configuration to a uniform one acting in a purely local, decentralized, distributed fashion. An essential requirement is clearly that the sensors will reach a state of *static equilibrium*, that is the self-deployment will be completed within finite time. How this task can be efficiently accomplished continues to be the subject of extensive research in the mobile sensor networks community (e.g., see [43, 44, 45, 46, 58, 62, 72, 87, 88]). Similar questions have been posed in terms of *scattering* or *coverage* in cooperative mobile robotics and swarm robotics (e.g., [8, 47]), as well as in terms of the *formation* problem for those entities (e.g. [9, 13, 19, 35, 33, 53, 81, 83, 85]).

The existing self-deployment protocols differ greatly from each other depending on the assumptions they make; for example some require the sensors to be deployed one at a time [16, 45, 47], while others require prespecified destinations for the sensors [62]. However, sensors are usually dispersed in the environment all together, more or less at the same time, with no a-priori knowledge of where their final location should be. Actually, unlike the case of ad-hoc networks, for small sensors localization is very hard, so it can not be generally assumed that the sensors know where they are.

The self-deployment problem has been investigated with the goal to cover the area so to satisfy some optimization criteria, typically to maximize the coverage (e.g., see [43, 44, 45, 46, 58, 62, 72, 87, 88]). For example, in [88] the problem is to maximize the sensor coverage of the target area minimizing the time needed to cover the area, while in [72] the additional constraint is a minimum requirement on the degree of all nodes. Typically, distributed self-deployment protocols first discover the existence of *coverage holes* (the area not covered by any sensor) in the target area based on the sensing service required by the application. After discovering a coverage hole, the protocols calculate the target positions of these sensors, that is the positions where they should move.

All of these solutions to the self-deployment problem require direct communication between sensors, hence can not be employed by robotic sensors. It is also interesting to observe that, even with communication, none of the existing self-deployment proposals is capable of providing a complete uniform coverage. This impossibility is hardly surprising since those protocols are *generic*, that is they must work in any environment regardles of its topology or structure. This fact opens a series of interesting questions, first of all whether it is possible for the sensors to self-deploy achieving uniform coverage in specific environments (e.g., corridors, grids, rims). The next important question is on the capabilities and a priori knowledge needed by the sensors

to achieve this goal; in other words, how "weak" the sensors can be and still be able to uniformly self-deploy. In particular, the focus of this section is on conditions for self-deployment of mobile robotic sensors so to obtain uniform coverage of specific spaces $\mathbb{U}$.

3.2 Uniform Deployment On Linear Borders

The first spatial universe $\mathbb{U}$ considered is possibly the simplest: a *linear border* or *corridor*, along which the sensors are required to place themselves evenly.

A corridor can be viewed as a *line* $\mathcal{L}$, on which the sensors are initially located at random distinct points. From an external point of view, the sensors can be ordered based on their position on the line from left to right; without loss of generality, let s_1 be the leftmost sensor and let s_i be to the right of s_{i-1}, $2 \leq i \leq n$. This order is clearly unknown to the sensors.

The goal is for the sensors to self deploy evenly in the segment of the line delimited by the positions of the leftmost and rightmost sensors s_1 and s_n (that, alternatively, could represent some perimeter marks rather than sensors).

Assuming that each sensor s_i is capable of viewing its neighbours s_{i-1} and s_{i+1} if they exist, the self deployment algorithm, by Cohen and Peleg [13], is remarkably simple:

Protocol CORRIDOR SPREAD (for sensor s_i)

- If no other sensor is seen on the left or on the right, then do nothing;
- Otherwise, move to point $x = \frac{1}{2}\ (s_{i+1} + s_{i-1})$.

With sensors that are anonymous, oblivious, and with no common coordinate system, the above protocol *converges* to a uniform deployment in the SSYNC (and thus also in the SYNC) model.

Let us show the idea of the convergence proof in the SYNC model. Note that, since the sensors operate in one-dimension, any coordinate system will give the same resulting destination. Therefore, in order to analyze the protocol, an external global coordinate system is used, of which the sensors have clearly no knowledge. In the following, the coordinate system where $s_0(t) = 0$ and $s_{n-1}(t) = 1$ is chosen as the global coordinate system. The goal is to spread the sensors uniformly; that is, at the end, sensor s_i should occupy position $\frac{i}{n-1}$. Let $\mu_i[t]$ be the *shift* of the s_i's location at time t from its final position. According to the protocol, the position of sensor s_i changes from $s_i(t)$ to

$$s_i(t+1) = \tfrac{1}{2}\ (s_{i-1}(t) + s_{i+1}(t))$$

for $2 \leq i \leq n-1$, while sensors s_1 and s_n never move. Therefore, the shifts changes with time as

$$\mu_i[t+1] = \tfrac{1}{2}\ (\mu_{i+1}[t] + \mu_{i-1}[t])$$

Consider the following *progress* measure

$$\psi[t] = \Sigma_{i=1}^{i=n}\ \mu^2[t]$$

Then

Lemma 1. *$\psi[t]$ is a decreasing function of t unless the robots are already equally spread.*

Finally,

Theorem 1. *In the* Sync *model, every $O(n^2)$ cycles, $\psi[t]$ is at least halved; furthermore, the sensors converge to equidistant positions.*

The idea of the convergence prove in Ssync is similar; in fact, first a non-decreasing quantity is defined, and its monotonicity proven. Then, by relating this quantity to the non-constant terms of the cosine series, it is proven that it decreases by a constant factor on every round, proving convergence.

Theorem 2. *In the* Ssync *model, anonymous, oblivious sensors on a line $\mathcal{L}$ with no common coordinate system* converge *to uniform deployment.*

3.3 Uniform Deployment Along Circular Borders

Consider next an important class of spatial regions, that of *circular borders* or *circular rims*. Deployment in these spaces occurs for example when the sensors have to surround a dangerous area and can only move along its outer perimeter. This situation is modeled by describing the space $\mathbb{U}$ as a *ring* $\mathcal{C}$. Starting from an initial arbitrary placement on the ring, the sensors must within finite time position themselves along the ring at (approximately) equal distance; see Figure 3.

The self-deployment problem along a ring is related to the well studied problem in the field of swarm robotics of *uniform circle formation* [9, 19, 22, 23, 53, 77, 85]. In this problem (discussed in more details in Section 4.2), the robots are required to uniformly place themselves on the circumference of a circle not determined in advance (i.e., the entities do not know the location of the circle to form). The main difference between the uniform circle formation and the self-deployment problem in the ring is that in uniform circle formation the entities can freely move on the two dimensional plane in which they have to form a ring; in contrast, our sensors can move only *on* the ring, which is the entire environment.

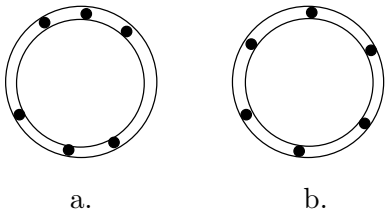

Fig. 3 Starting from an initial arbitrary placement (a), the sensors must move to a uniform cover of the ring (b).

Let $\mathbb{S} = \{s_1, \ldots, s_n\}$ be the n sensors initially arbitrarily placed on the ring $\mathcal{C}$ (see Figure 3). Initially no two sensors are placed at the same location; the algorithms should avoid *collisions*, i.e. having two sensors simultaneously occupying the same point; without loss of generality, let s_i be the sensor immediately before s_{i+1} in the clockwise direction, with s_n preceding s_1. Let $d_i(t)$ be the distance between sensor s_i and sensor s_{i+1} at time t; when no ambiguity arises, we will omit the time and simply indicate the distance as d_i. Let $d = L/n$, where L denotes the length of the ring $\mathcal{C}$. The sensors have reached an *exact* self-deployment at time t if $d_i(t) = d$ for all $1 \leq i \leq n$. Given $\epsilon > 0$, the sensors have reached an ϵ-*approximate* self-deployment at time t if $d - \epsilon \leq d_i(t) \leq d + \epsilon$ for all $1 \leq i \leq n$.

An algorithm $\mathcal{A}$ correctly solves the *exact* (resp. ϵ-*approximate*) self-deployment problem if, in any execution of $\mathcal{A}$ by the sensors in $\mathcal{C}$, regardless of their initial position in $\mathcal{C}$, there exists a time t' such that the sensors have reached an *exact* (resp. ϵ-*approximate*) self-deployment at time t' and are in a quiescent state.

3.3.1 Impossibility Without Orientation

There is a strong negative result for the SSYNC (and thus for the ASYNC) model. In fact, *exact* self-deployment is actually *impossible* if the sensors do not share a *common orientation* of the ring; notice that this is much less a requirement than having global coordinates or sharing a common coordinate system. This impossibility result, by Focchini, Prencipe and Santoro [31], holds even if the sensors have unlimited memory of the past computations and actions (i.e., unlimited persistent memory, see Section 2.4), and their visibility radius is unlimited.

Theorem 3. *Let the sensors be on a ring $\mathcal{C}$. In absence of common orientation of $\mathcal{C}$, there is no deterministic exact self-deployment algorithm even if the sensors have unbounded persistent memory, their visibility radius in unlimited. and the scheduling is* SSYNC.

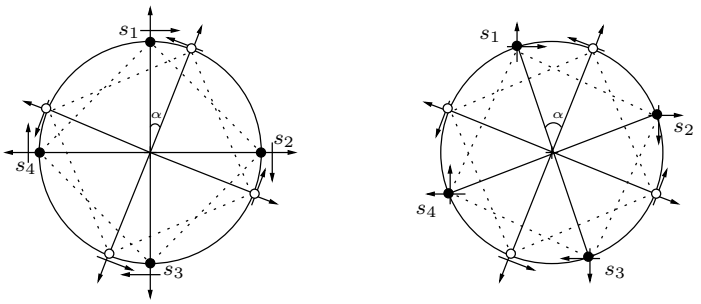

Fig. 4 (a) An example of starting configuration for the proof of Theorem 3. The black sensors are in S_1, while the white ones in S_2. (b) Theorem 3: the adversary moves only sensors in S_1.

To see why this is the case, consider the following setting. Let n be even; partition the sensors in two sets, $S_1 = \{s_1, \ldots, s_{n/2}\}$ and $S_2 = S \setminus S_1$, and place the sensors of S_1 and S_2 on the vertices of two regular $(n/2)$-gons on *Circle*, rotated of an angle $\alpha < 360°/n$. Furthermore, all sensors have their local coordinate axes rotated so that they all have the same view of the world (refer to Figure 4.a for an example). In other words, the sensors in S_1 share the same orientation, while those in S_2 share the opposite orientation of $\mathcal{C}$. Denote a configuration with such properties by $Y(\alpha)$. A key property of $Y(\alpha)$ is the following.

Property 1. Let the system be in a configuration $Y(\alpha)$ at time step t_i.

1. If activating only the sensors in S_1, *no* exact self-deployment on $\mathcal{C}$ is reached at time step t_{i+1}, then also activating only the ones in S_2 *no* exact self-deployment on $\mathcal{C}$ would be reached at time step t_{i+1}; furthermore, in either case the system would be in a configuration $Y(\alpha')$ for some $\alpha' < 360°/n$
2. If activating only the sensors in S_1 an exact self-deployment on $\mathcal{C}$ is reached at time step t_{i+1}, then also activating only the sensors in S_2 an exact self-deployment on $\mathcal{C}$ would be reached at time step t_{i+1}.
3. If activating only the sensors in S_1 an exact self-deployment on $\mathcal{C}$ is reached at time step t_{i+1}, then activating both sets no exact self-deployment on $\mathcal{C}$ would be reached at time step t_{i+1}, and the system would be in a configuration $Y(\alpha')$ for some $\alpha' < 360°/n$.

Using this property is easy to design an Adversary that will force any self-deployment $\mathcal{A}$ to never succeed in solving the problem: the Adversary will choose $Y(\alpha)$ as the initial configuration, and behave as follows:

(Step a) If activating only the sensors in S_1 no exact self-deployment on $\mathcal{C}$ is reached: then activate all sensors in S_1, while all sensors in S_2 are inactive;

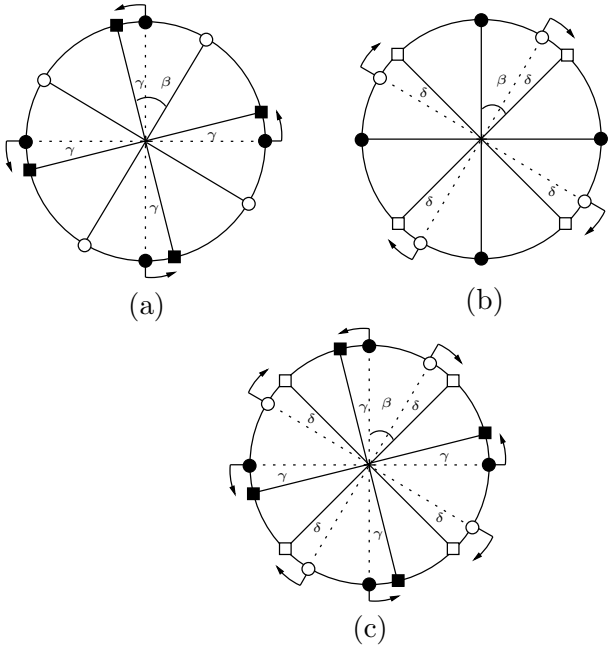

Fig. 5 Theorem 3. (a) If only the sensors in S_1 are activated at t, all sensors would be uniformly placed at time $t+1$, with $\beta+\gamma = 45°$. (b) If only the sensors in S_2 are activated at t, all sensors would be uniformly placed at time $t+1$, with $\beta+\gamma = 45°$. (c) Therefore, if all sensors would be activated at t, they would not be in an exact self-deployment on $\mathcal{C}$, having $\gamma+\beta+\delta \neq 2\pi/n = 45°$. In all figures, the squares represent the destination of the active sensors.

otherwise, activate all sensors. Go to (b).

(Step b) If activating only the sensors in S_2 no exact self-deployment on $\mathcal{C}$ is reached: then activate all sensors in S_2, while all sensors in S_1 are inactive; otherwise, activate all sensors. Goto (a).

By Property 1, if the configuration at time $t_i \geq t_0$ is $Y(\alpha)$ for some $\alpha < 360°/n$, then regardless of whether the Adversary executes step (a) or (b), the resulting configuration is $Y(\alpha')$ for some $\alpha' < 360°/n$, and hence *no* exact self-deployment on $\mathcal{C}$ is reached at time step t_{i+1}. Hence, there exists an infinite execution of $\mathcal{A}$ in which no exact self-deployment will ever be reached. The alternating between steps (a) and (b) by the Adversary ensures the feasibility of this execution: every sensor will in fact become active infinitely often.

Recently, the impossibility without orientation has been announced to hold also for the stronger SYNC model [29].

Since the impossibility result of Theorem 3 holds in absence of common orientation of the ring, the focus will now be on *oriented* rings; two cases will be considered, depending on whether or not the desired final distance d is known to the sensors.

3.3.2 Exact Deployment

Faced with this strong negative result of Theorem 3, the interesting question becomes under what restrictions the self-deployment problem can be solved with an exact algorithm. Since the impossibility result holds in absence of common orientation of the ring, consider the problem in *oriented* rings.

In an oriented ring, if the desired final distance d is known or computable (e.g., both the number or sensors and the length of the ring are known), *exact* self-deployment is indeed possible. This positive result holds even if the sensors are oblivious and asynchronous, provided their visibility radius is at least $2d$.

The algorithm by Focchini, Prencipe and Santoro [31] proving this result is very simple:

> Protocol RING - KNOWN INTERDISTANCE (for sensor s_i)
>
> - Locate clockwise at distance $2d$. Let d_i be the distance to s_{i+1} (if visible, else $d_i = 2d$).
> - If $d_i \leq d$ do not move.
> - If $d_i > d$ move clockwise and place yourself at distance d from s_{i+1} (if visible, else at distance d from current location).

Like in other cases (e.g., [12, 13]), the difficulty is not in the protocol but in the proof of its correctness. Using this protocol, and observing that the algorithm operates in ASYNC, we have

Theorem 4. *Let the sensors share a common orientation of the ring $\mathcal{C}$, and be able to locate to distance $2d$. If they know d, then* exact *self-deployment is possible even if the sensors are oblivious and the scheduling is* ASYNC.

3.3.3 ϵ-Approximate Deployment

In an oriented ring, if the sensors do *not* know the desired final distance d, then ϵ-approximate self-deployment is still possible for any $\epsilon > 0$; also in this case, the protocol works even for the weakest sensors: oblivious and asynchronous, provided their visibility radius is greater than $2d$.

Also in this case the proof is provided by a simple protocol [31]: sensors asynchronously and independently locate in both directions at distance v,

then they position themselves in the middle between the closest observed sensor (if any).

Protocol RING - UNKNOWN INTERDISTANCE (for sensor s_i)

- Locate around at distance v. Let d_i be the distance to next sensor, d_{i-1} the distance to the previous (if no sensor is visible clockwise, $d_i = v$, analogously for counterclockwise).
- If $d_i \leq d_{i-1}$ do not move.
- If $d_i > d_{i-1}$ move to $\frac{d_i+d_{i-1}}{2} - d_{i-1}$ clockwise.

This algorithm converges to a uniform deployment. The crucial property is that

Property 2. For any $\epsilon > 0$ there exists a time t, such that $\forall t' > t, \forall i$: $|d_i(t') - d| \leq \epsilon$.

Hence, by adding to the protocol a test on whether both d_i and d_{i-1} are within ϵ from d (in which case no move is performed by s_i), it follows that ϵ-*approximate* self-deployment is possible even if the scheduling is ASYNC, if the sensors share a common orientation of the ring $\mathcal{C}$ and are able to locate to distance $v > 2d$.

The strategy used by the protocol described here is *go-to-half*. Interestingly it was shown by Dijkstra ([25] pp. 34–35) that in an unoriented ring *go-to-half* does *not* converge, and hence can not be used for approximate self-deployment in an unoriented ring. However, a different strategy, *go-to-quater*, does converge in an unoriented ring [19, 77], and can thus be used for ϵ-*approximate* self-deployment in an unoriented ring with unknown d, yielding the following more powerful result:

Theorem 5. *Let the sensors in the ring $\mathcal{C}$ be able to locate to distance $v > 2d$. Then* ϵ-approximate *self-deployment is possible even if the sensors are oblivious, the scheduling is* ASYNC*, and the ring is not oriented.*

3.4 Uniform Deployment in Rectangular Spaces

3.4.1 Problem Definition and Notation

The next class of spaces $\mathbb{U}$ considered is that of *rectangular spaces*, that is spaces delimited by a rectangular *border* B. The sensors are capable of detecting any part of B within their visibility radius. Assume there is a local sense of orientation: each sensor has a consistent notion of "up-down" and "left-right" (e.g., as provided by a compass), where the "up-down" axis is parallel to the longest side of the border.

A rectangular space of size $L \times W$ can be logically subdivided into equal sized square of size d^2 by considering a $(l+1) \times (w+1)$ rectangular *grid* $\mathcal{G}$,

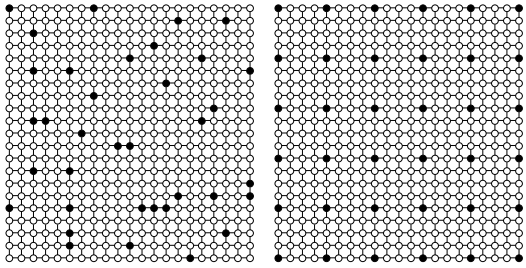

Fig. 6 A random initial configuration, and a uniform deployment.

where $l = L/d$ and $w = W/d$ and the distance between neighbouring nodes is d, and the visibility radius is $v \geq 2d$. The uniform self-deployment problem in rectangular spaces thus consists of reaching an equilibrium configuration where the sensors are evenly placed among the grid points (see Fig 6).

Let us indicate by $(0, 0)$ the left-most lower corner of $\mathcal{G}$ and by (i, j) the node belonging to column i and row j. For simplicity, assume $n = (k + 1)^2$, that $\mathcal{G}$ is a square grid with $l = w = k \cdot d$, and that the sensors are initially arbitrarilly located at distinct grid points. In this case, an equilibrium configuration consists of nodes $(i \cdot d, j \cdot d)$, with $i, j \in [0, k]$ hosting exactly one sensor each (see Fig 6). The goal is to design a collision-free protocol for positioning the sensors at those grid points. Note that the (i, j) coordinates of the grid nodes are global and thus *not known* to the sensors, which use only relative coordinates. The common orientation will be modeled by assuming that the edges of the grid are consistently labeled *Up*, *Down*, *Left* and *Right*, and edge labels are visible to the sensors.

3.4.2 Uniform Deployment Protocol

The uniform deployment algorithm SCATTER is a set of local rules for the robots designed by Barriére *et al.* [4]. Each sensor has a *state* variable belonging to a set of states $\{-1, 0, 1, 2, 3, 4\}$ (initialized to -1) which determines the set of rules to be followed, which solely depend on the robot's current state, its position, and the positions of the robots within its visibility radius.

Upon startup, the execution of the algorithm is logically divided in three *phases*: *Cleaning*, *Collecting*, and *Deploying*. Waking up for the first time (in state -1), each sensor determines what phase to start, depending on its relative position. If a sensor is at the *left upper corner* it directly enters state 3 (i.e., it starts *Deploying*). If a robot is on the *left or bottom border*, it enters state 0 (i.e., it starts *Cleaning*). Otherwise, it enters state 1 (i.e., it starts *Collecting*).

- *Cleaning:* robots (if any) on the left and bottom borders move leaving those nodes empty. (it is performed only by robots in state 0).

- *Collecting:* robots move towards the left-upper corner of the grid (it is performed by robots in states 1 and 2).
- *Deploying:* robots follow a distinguished path on the grid, called *snake-path*, eventually occupying their final positions (it is performed by robots in states 3 and 4).

Note that, due to asynchrony, at any point in time robots could be performing actions belonging to different phases. The asynchronous execution of the various phases requires special care in order to insure that robots continue to progress in their phase avoiding collisions and the creation of deadlocks.

Cleaning. The *Cleaning* phase is performed only by robots in state 0 and the goal is to have the robots move from the left and bottom borders toward the interior of the grid. This is done by having robots on the left border move down (if the down node is empty) and robots on the bottom border move up (if the top node is empty) or right (if the top node is occupied but the right node is empty). When moving up from the bottom border, a sensor enters state 1 and starts the *Collecting* phase.

Collecting. The *Collecting* phase is performed by robots in states 1 or 2 and the goal of the robots is to arrive to the left-upper corner. To avoid conflicts with robots possibly already in the *Deploying* phase, during the *Collecting* phase a sensor should not consider robots that are on the left border. The general rule for a sensor in the Collecting phase is to go up towards the upper-left corner (i.e., highest priority is given to movements up). However, depending on the neighboring conditions, to guarantee progress and avoid deadlocks, robots might have to move also left or right as described below.

Whenever a sensor (in state 1 or 2) has an empty upper node it goes *up* and stays in (or enters) state 1. If a sensor in state 1 is on the right neighbor of the upper left corner and the upper left corner is empty, it goes *left* and enters state 3 (*Deploying* phase). If a sensor in state 1 has all the visible nodes in its same column above itself occupied, and the left and bottom-left nodes are empty, it goes *left* and stays in state 1. Finally, a sensor (in state 1 or 2) goes *right* and stays in (or enters) state 2 if within its visibility radius all the following conditions are satisfied: all the nodes not below nor right are occupied, the lower right node is empty, all the nodes in its same row on the right are empty, at least one of the the above and right nodes is empty, none of the robots in the above and right nodes at distance strictly less than $2d$ can go up. In all other cases a sensor stays in (or enters) state 1.

The movements of robots in the Collecting phase allow them to accumulate in a convenient shape around the upper left corner.

Deploying. This phase is executed by robots in state 3 and 4. A sensor starts this phase when it arrives to the upper left corner and enters state 3. In this phase the robots move on the snake-path (see Figure 7) and eventually stop in their final positions, that is the nodes $(i \cdot d, j \cdot d)$, with $0 \leq i, j \leq k$, called *final nodes*.

Let k be odd; the snake-path is the path $n_0, n_1, \ldots n_{(K-1)d}$, that starts at $n_0 = (0, n)$, ends at $n_{(K-1)d} = (d, n)$ and passes through every final node as

shown in Figure 7. By a slight modification of this path and, consequently, of the algorithm, the snake-path can be defined for k even.

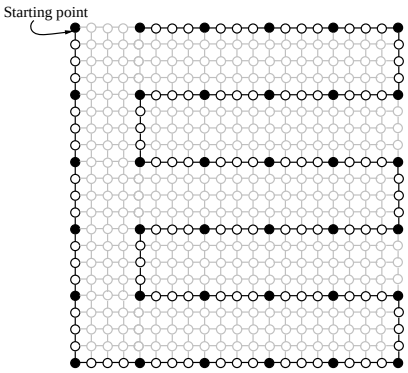

Fig. 7 The snake-path for $N = 21^2$, $K = 36$,$k = 5$ $d = 4$.

Because of asynchrony, complications may arise. For example, if a sensor in the *Deploying* phase enters in contact with robots that are still in the *Collecting* phase, the sensor has to wait before continuing on the snake-path. Special care has to be taken and it can be shown that deadlock sare avoided and progress is guaranteed.

A sensor s enters state 3 when it reaches the left-upper corner. In this state, s follows the left border moving only if it sees above another sensor at distance lower than d and if the lower node is empty. If the sensor is in the left upper corner it only moves if the right node is occupied, insuring in this way that there is at least one node in the *Collecting* phase. When the last node enters the *Deploying* phase it will stay at the upper left corner and eventually the other nodes will position themselves at distance d from each other. When in state 3, a sensor s is following the left border of the grid. It enters state 4 when it reaches the bottom-left corner. A sensor in state 4 follows the snake-path, from the left-bottom corner. A sensor s in the *Deploying* phase might see in the rows above some robots still performing the *Collecting* phase (the presence of these robots can be detected because of their "wrong" positions). In this case s does not move. If s does not see any sensor out of the snake-path but there are no robots in the $d-1$ preceding nodes or the next node is occupied, then it waits. Otherwise s moves forward on the snake-path. When the last node enters the *Deploying* phase it will stay at the upper left corner and eventually all other nodes will stop at their final position.

The correctness of the algorithm is proven by showing that no deadlocks can occur, and that progress is guaranteed. Notice that, although described for a square grid, the protocol works in any rectangular grid, provided the di-

rection of the largest dimension is known. Since protocol SCATTER terminates within finite time with a uniform scattering, we have:

Theorem 6. *The uniform deployment in rectangular spaces $\mathcal{G}$ can be solved without any collisions by sensors each having a constant amount of memory and a discrete visibility radius $2d$ in the* ASYNC *model.*

3.5 Incremental Deployment and Filling

3.5.1 Incremental Deployment as Filling

The task of uniform self deployment is usually performed *after* the sensors have entered the space $\mathbb{U}$. This is because typically the sensors enter the space, all at once or in groups, without attention to the desired final placement criterion[3], in a process called *injection*. Indeed, this is true in many situations especially in the case of very simple sensors. However the separation between injection and self deployment does not always occur. In fact, there are applications where the sensing entities are injected into the system one at a time, from one or few entry points; this is particularly true in the case of more complex (and/or delicate) sensorial entities (e.g., to avoid damage). In these situations, instead of having two separate processes, injection and self-deployment, the focus is on achieving the final goal directly, in a single process, called *incremental deployment*.

Howard, Mataric, and Sukhatme [45] proposed an incremental deployment algorithm for mobile robotic networks. Under the assumptions of global coordination, location awareness, and nodal visibility, that algorithm deploys robots one-at-a-time from a single entry point (*door*), and maintains a line of sight relationship between robots. They assume every sensor is equipped with an ideal localization sensor; however, localization is very hard, especially for small sensors, so it can not be generally assumed that the sensors know where they are. A very important and interesting mechanism they use is a logical orthogonal grid, superimposed on the space, that divides the space into cells, transforming the continuous space into an orthogonal cellular space. Let us stress that orthogonal spaces are interesting of their own, because they can be used to model indoor and urban environment; furthermore the discretization of a continuous space into a cellular space is a rather common process used in a variety of contexts.

This approach has the additional advantage of reducing the problem of *incremental deployment* of an unknown arbitrary space $\mathbb{U}$ to the problem of *filling* an unknown cellular space. In the *Filling* problem, the mobile entities have to occupy all the cells of an unknown cellular space, entering through one or more designated entry points called *doors*; within finite time, the entities

[3] For example, in some applications, sensors are dropped from the air.

must reach a quiescent state, with exactly one entity in each cell. If two sensors are in the same cell at the same time then there is a *collision*. The algorithm executed by the sensors should avoid collisions (e.g., to prevent damage to the sensor or its sensory equipment).

The reduction to the filling problem is obtained by superimposing on the space $\mathbb{U}$ a logical *orthogonal grid* of the appropriate size; this will divide the space into cells (boundary cells might not be all within $\mathbb{U}$). Notice that the resulting cellular space $\mathcal{M}$ is orthogonal, i.e. polygonal with sides either parallel or perpendicular to one another (e.g., see Figure 8). The space can be completed to become a bicolored cellular rectangle[4] A, where each cell, called *pixel* is colored *white* if it is part of $\mathcal{M}$, *black* otherwise (see Figure 8).

At this point, to achieve an *incremental deployment* in the unknown arbitrary space $\mathbb{U}$ it is sufficient to perform a *filling* of the unknown orthogonal space $\mathcal{M}$; that is, filling the white pixels of A.

The problem of *filling* unknown orthogonal space $\mathcal{M}$ has been investigated by Hsiang et al. [47] for mobile sensors, and by Das, Mesa and Santoro [16] for robotic networks.

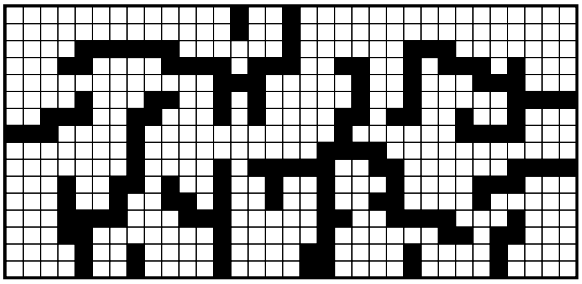

Fig. 8 A orthogonal space $\mathcal{M}$ (white cells) to be filled by the sensors and its enclosing cellular rectangle A.

In the study of Hsiang et al. [47], the sensors enter $\mathcal{M}$ from one or more doors. Their results are based on an ingenious follow-the-leader technique where each sensor communicates with the one following it and instructions to move are communicated from predecessor to successor. The sensors are anonymous but they need some persistent memory to remember whether or not is a leader and the direction of its movement. Since the algorithm uses only $O(1)$ bits of working memory in total, computationally the sensors can be just *finite-state machines*. In addition to requiring explicit communication, the solution of [47] assumes that the sensors operate in the SYNC model, which allows perfect coordination and synchronization between the sensors.

[4] A is the smallest cellular rectangle enclosing $\mathcal{M}$

3.5.2 Filling by Robotic Sensors

For *robotic sensors* (where no direct communication exists), the filling problem of orthogonal spaces (and thus the incremental deployment problem) have been investigated by Das, Mesa, Santoro [16]. First of all they proved that the sensors must have some *persistent* memory of the past, for solving the filling problem successfully.

Theorem 7. *The filling problem can not be solved by oblivious sensors, even if they have unbounded visibility. This result holds even if there is only a single door and the model is* SSYNC.

Thus some persistent memory is required. Indeed with just a constant amount of persistent memory, filling can be done in the case of a single door. This result is obtained in [16] with visibility radius of one in the SSYNC model

Let the bicolored cellular rectangle A containing $\mathcal{M}$, be formed of pixels $p_{i,j}$, $1 \leq i \leq l$, $1 \leq j \leq c$; let *discrete* visibility radius of 1 mean that the sensor sees all eight neighboring cells (i.e., $v \geq \sqrt{2}q$, where q is the cell length). To understand the protocol, the structure of $\mathcal{M}$ will be represented by a graph $G = (V, E)$ defined as follow: First partition each column into segments of consecutive white pixels ended by a black pixel in both extremes and numbered from top to down. Each segment is a node of G. Denote by $v_j^k \in V$ the node corresponding to the $k-th$ segment of column j, and by d_j^k the bottom-most pixel of the segment v_j^k. There is an edge $(v_j^k, v_{j'}^{k'}) \in E$ if and only if: (a) $j = j' + 1$ or $j = j' - 1$ and (b) there is a pixel $p_{i,j'} \in v_{j'}^{k'}$ neighbor to d_j^k or there is a pixel $p_{i,j} \in v_j^k$ neighbor to $d_{j'}^{k'}$.

It is easy to verify that the graph G so obtained is an acyclic connected graph (i.e. a tree). If there is an edge $(v_j^k, v_{j'}^{k'})$ such that the bottom-most pixel $d_j^k = p_{i,j}$ of v_j^k is a neighbor of the pixel $p_{i,j'} \in, v_{j'}^{k'}$, we say that $p_{i,j}$ is the ***entry point*** from v_j^k to $v_{j'}^{k'}$ and $p_{i,j'}$ is the ***entry point*** from $v_{j'}^{k'}$ to v_j^k.

The idea of the algorithm (FILLING - SINGLE DOOR) is to move the robots along the paths in G, starting from the node containing the door. Since the sensor can see the eight neighboring pixels, it can determine when it has reached an *entry point*. Let
$block^+(p_{i,j}) \equiv (p_{i,j}$ is empty $) \wedge ((p_{i-1,j}$ is **black**$) \vee (p_{i-1,j+1}$ is **black**$)))$
and let
$block^-(p_{i,j}) \equiv (p_{i,j}$ is empty $) \wedge ((p_{i-1,j}$ is **black**$) \vee (p_{i-1,j-1}$ is **black**$)))$.

Protocol Filling - Single Door
Meta-Rule: A sensor never backtracks.
Sensor s in pixel $p_{i,j}$:
If ($p_{i+1,j}$ is empty) **Then** s moves to $p_{i+1,j}$.
Else If ($p_{i-1,j}$ is empty) **Then** s moves to $p_{i-1,j}$.
Else If (($block^-(p_{i,j-1})$)) **Then** s moves to $p_{i,j-1}$.
Else If (($block^-(p_{i,j+1})$)) **Then** s moves to $p_{i,j+1}$.
Else s does not move.

Since algorithm Single Door is collision free, it terminates in finite time, and completely fills the space [16], we have the following

Theorem 8. *The filling problem for any space orthogonal space $\mathcal{M}$ with a single door, can be solved without any collisions by sensors each having a constant amount of memory and a discrete visibility radius* 1 *in the* Ssync *model.*

In the case of multiple doors, there are other strong limitations: the sensors must have a discrete visibility radius of at least two and they should not be indistinguishable [16]. Thus, for the problem to be solvable at all, sensors entering the space from different doors must be distinguishable, i.e., have different colors, and each sensor must have discrete visibility radius of at least two. Indeed, under this assumption, the problem can be solved. However, the algorithm in this case is more complex that the one when there is only a single door.

The idea of the algorithm, presented in [16], is as follows. Sensors coming from different doors (i.e. sensors of different colors) follow distinct paths in G and these paths do not intersect. In other words, the algorithm ensures that the cells visited by sensors of color c_i are occupied by sensors of the same color (and never by sensors of any other color). To achieve this, a sensor before moving to a pixel $p_{i,j}$ needs to determine if this pixel was visited by sensors of another color; fortunately, this can be done. Hence

Theorem 9. *The filling problem for any orthogonal space $\mathcal{M}$ with multiple doors, can be solved without any collisions by sensors (with distinct color for distinct doors) each having a constant amount of memory and a discrete visibility radius* 2 *in the* Ssync *model.*

4 Pattern Formation

The *pattern formation* problem is one of the most important coordination problem for robotic systems. The geometric pattern to be formed is a set of points (given by their Cartesian coordinates) in the plane, initially known by the entities. Initially the entities are in arbitrary positions, with the only

requirement that no two entities are in the same position, and that, of course, the number of points prescribed in the pattern and the number of entities are the same. The robots are said to *form the pattern* if, at the end of the computation, the positions of the robots coincide, in everybody's local view, with the points of the pattern. Depending on the application, the formed pattern may be *translated*, and/or *rotated*, and/or *scaled*, and/or *flipped* into its mirror position with respect to the initial pattern. In particular, the pattern formation problem is said to be *scale-free* if the formed pattern can be an arbitrarily scaled version of the input pattern.

The pattern formation problem is practically relevant because, if the robots can form a given pattern, they can agree on their respective roles in a subsequent, coordinated action. For this reason, it has been extensively investigated in the literature on autonomous robots (thus, without using communication); e.g., see [3, 9, 19, 20, 35, 51, 53, 81, 83, 85, 89]. It has also been studied in systems using direct wireless communication (e.g., [37, 40, 56]).

The basic research questions are which patterns can be formed, and how they can be formed. In this section, we review the existing results on pattern formation by *mobile robotic sensors*, that is by mobile entities that are silent and myopic. The spatial universe $\mathbb{U}$ is assumed to be the 2D space, and it is assumed that initially the sensors are arbitrarily dispersed in $\mathbb{U}$ but the visibility graph is connected.

4.1 Forming Scale-Free Patterns

Almost all protocols for pattern formation of silent autonomous robots assume *unlimited visibility*; in particular, they use the fact that each sensor can see all the other robots (e.g., [3, 9, 19, 20, 35, 51, 53, 81, 83, 85, 89, 90]). Thus, these protocols can not be employed directly by robotic sensors, which by definition have a limited sensing range.

However, those same algorithms can be effectively used if the formed pattern can be an arbitrarily scaled version of the input pattern, i.e. for the *scale-free* pattern formation problem. This can be achieved by the following two-steps strategy:

GATHER & FORM

1. Every sensors gets within the visibility range of every other sensor.
2. The sensors execute the relevant pattern formation protocol that assumes unlimited visibility.

The first step of this strategy requires solving the problem called *Near Gathering* [32]: starting from an initial arbitrary distribution in $\mathbb{U}$, the sensors, avoiding any collision, must within finite time reach a static equilibrium in

which they are all mutually visible and on distinct locations; that is, there exists a time t when all sensors are in a state of static equilibrium and, for any two sensors s and r, $0 < |s(t) - r(t)| \leq v$. This problem is closely related to the *Rendezvous* or *Gathering* problem that will be discussed in details in Section 5.

Once the first step has been performed, then the appropriate unlimited-visibility pattern formation protocol can be started. In particular, the algorithms for *arbitrary pattern formation* (e.g., [35, 83, 90]) can be used by the sensors to form any input pattern. There are some provisos. In particular, to start the second step, a sensor must know that the execution of the first step has been completed; that is, all sensors are within its visibility range. However this implies that the number n of sensors must be known to the sensors. Another important point is that global mutual visibility, once reached in the first step, must be maintained throughout the execution of the second step. This necessary condition might not be of trivial enforcement; for example there are some pattern formation algorithms that require, during their execution, some entities to move away from the others at a distance that (because of the limited visibility range of the sensors) might bring them out of the range of some other sensors.

In other words, the execution of the unlimited-visibility pattern formation protocol must be carefully planned, tailored to the requirements of the pattern being formed, taking into accounts the movements of the sensors required by the algorithm: a non trivial task. To date, no generic protocol is available for scale-free pattern formation by robotic sensors.

4.2 Circle Formation

A particular pattern extensively studied in literature is the *circle*: the sensors, starting from arbitrary positions in the plane, have to arrange themselves in a circle of a given diameter D. Observe that this pattern formation problem is not scale-free, and thus requires the agreement of the robots on the same unit distance.

If the sensors must be arranged at regular intervals on the boundary of a circle the problem is also called *uniform circle formation*. This kind of formation can be usefully deployed in surveillance tasks: the sensors are placed on the border of the area (or around the target) to surveil.

One of the first discussion on circle formation by a group of mobile entities was by Debest [17], who introduced it as an illustration of self-stabilizing distributed algorithms. He discussed the problem, but did not provide an algorithm.

The uniform circle formation problem was first studied by Sugihara and Suzuki [81]. They presented an heuristic that allowed the sensors to form an approximation of a circle having a given diameter $D \leq v$; it works without

requiring common coordinate systems, the sensors can be oblivious, and the scheduling ASYNC. For sensor s, let $s_f(t)$ and $s_c(t)$ denote the position of the farthest and of the closest sensors at time t, respectively; and let $\epsilon > 0$ be an arbitrarily small predefined quantity. The protocol is rather simple:

Protocol CIRCLE CONVERGENCE (for sensor s at time t)

1. If $|s_f(t) - s(t)| > 2D$ then move towards $s_f(t)$.
2. If $|s_f(t) - s(t)| < 2D - \epsilon$, then move away from $s_f(t)$.
3. If $2D - \epsilon \leq |s_f(t) - s(t)| \leq 2D$, then move away from $s_c(t)$.

Experiments have shown that sometimes the sensors converge towards a conguration similar to a *Reuleaux triangle* rather than a circle. Successively, the protocol has been improved by Tanaka [85], that proposed a new solution that produces a better approximation of the circle.

For the simpler SYNC model, a protocol that allows oblivious sensors without common coordinate system to converge towards a uniform placement on a circle has been recently proposed by Lee *et al* as part of their investigation on forming concentric circles [55].

There are many other protocols for uniform circle formation [9, 19, 20, 23, 22, 53, 77]. For instance, the problem has been studied in the SSYNC setting by Défago and Konagaya [19], with anonymous and oblivious sensors, by presenting a solution that is a composition of two independent algorithms whereby the sensors first deterministically form a circle, and then converge to a situation in which all sensors are arranged uniformly on its boundary; simulation results of these studies have been presented in [77]. The solution in [19] is, however, computationally expensive: in fact, it involves the use of Voronoi diagrams, necessary to avoid the very specic possibility in which at least two robots share at some time the same position and also have total agreement on the coordinate system. Based on this observation, in [9] it is presented a new algorithm that avoids these expensive calculations; unfortunately, their solution relies on the simplifying assumption that sensors must not be located on the same radius, that radically changes the difficulty of the problem. Katreniak in [53], employing anonymous and oblivious sensors with no common coordinate system, handles to task of forming a *biangular circle*, when the number of sensors is even: the sensors place themselves on the rim of a circle, each pair of adjacent sensors on the rim of the circle form with its center either an angle α or an angle β, and the angles alternate; the sensors act under ASYNC scheduling. When the number of sensors is odd, the sensors achieve the uniform circle. A solution that does not work for any number of robots has also been presented in [23], where the proposed oblivious algorithm works for a prime number of sensors in the semi-synchronous model. Dieudonné *et al.* [22] build upon the work of Katreniak [53], and extend it for the case with an even number of sensors; the algorithm solves the problem in nite time for any number n of sensors, except when $n = 4, 6$ and 8, under the SSYNC schedule. Also, here the sensors have the ability to reach exactly in

one step their computed destination, and cannot stopping on the way. This assumption was later dropped in [24]; however, the algorithm in [24] still does not work for $n = 4$. Successively, Défago and Souissi presented in [20] an algorithm by which sensors deterministically form a circle in a finite number of steps, and then asymptotically converge toward a situation in which they are positioned at regular intervals on the boundary of this circle, again under the SSYNC schedule. In contrast with the analogous two-parts solutions previously presented in [19], here the two parts are combined in a single and simpler algorithm. However, even if this solution works for any number of sensors, it only converges to the uniform circle formation, in contrast with the solution in [24].

There are two major problems with all these solutions [9, 19, 20, 23, 22, 53, 77]. The first problem is that these protocols assume *unlimited visibility*; hence they can not be used directly by robotic sensors, by definition myopic. This means that, first of all, robotic sensors can use these protocols only if the sensors are all mutually visible (i.e., the visibility graph is complete) and continue to be so throughout the execution of the protocol. The second problem is that these protocols are for *scale-free* circle formation; however, the problem we are facing has a fixed scale (given by the diameter D).

To overcome the first problem, the sensors can obtain an initial global mutually visibility by first performing a *Near Gathering* [32], and then execute the protocol (i.e., using the strategy GATHER&FORM discussed in Section 4.1); the difficulty of ensuring maintenance of global mutual visibility during execution clearly depends on which of those protocols is being used. To overcome the second problem, once a uniform circle formation has been obtained, an additional step is needed to *scale* the circle to the required dimension; this is not difficult provided $v \geq D$.

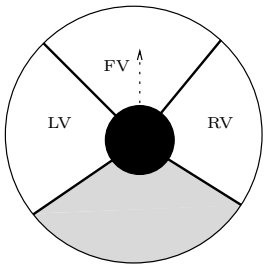

Fig. 9 The vision model for the emergent approach to circle formation. The black circle represents the robots; the dark area is the blind zone of the robot.

The problem of arranging robotic sensors in circular shapes has been studied by Miyamae, Ichikawa and Hara [68], considering robotic sensors whose vision is not only limited (i.e., within the visibility range v) but also *directional*. In fact, the vision function of each sensor detects another sensor within

distance v with the center of the sensor assumed to be the origin and the direction of movement the reference angle (0°); however the detection occurs only within three areas: forward (FV), and its left (LV) and right (RV) sides; the backward area is that not detecting a sensor (see Figure 9),

Furthermore, the sensors can only detect the presence of other sensors within their visibility areas, and not the exact number of sensors in their surrounding. In particular, each sensor can distinguish two scenarios for the forward area: zero sensors (FV= 0), or ≥ 1 sensors (FV= 1); for the left area, each sensor can distinguish three scenarios: zero sensors (LV= 00), one sensor (LV= 01), or more than one sensor (RV= 10); symmetrically, three scenarios can be detected for the right area as well. Based on these simple information, each sensor acts as described in the following protocol.

Protocol EMERGENT CIRCLE (for sensor s_i)

1. If FV= 0, LV= 01, and RV= 00, then turn left.
2. If FV= 0, LV= 00, and RV= 01, then turn right.
3. If FV= 0, LV= 01, and RV= 01, then turn to the last previous direction.
4. For the others scenarios, proceed straight.

The emergent behavior of the sensors following these simple rules has been analyzed by computer simulations in [68]. The experimental results show that the formation of the circle depends on the number of sensors and the front and side view angles of local vision, demonstrating that the front view angle must be between 15° and 75°, while the side view angles between 60° and 120°. Another interesting observation arising from the simulations is that the circle formation rate decreases the larger the number of sensors is; that is, an excessive number of sensors affects negatively the formation process.

5 Gathering

In systems of mobile entities, one of the most basic coordination and synchronization task is that of *gathering*: the entities, placed in arbitrary positions in $\mathbb{U}$, must congregate at a single location (the choice of the location is not predetermined). This fundamental problem is also called *rendezvous* or *homing*. If the entities are seen as points, the gathering problem is the one of having all entities move to the same point, that is forming the special pattern *point*; hence the problem is sometimes called *point formation*.

The gathering problem has been extensively investigated both experimentally and theoretically in the *unlimited visibility* setting, that is assuming that the entities are capable to sense ("see") the entire space (e.g., see [1, 10, 12, 18, 35, 49, 81, 83]).

In general, and more realistically, sensors can sense only a surrounding within a radius of bounded size (refer to the example depicted in Figure 1). This setting, which is the one in which robotic sensors operate, is understandably more difficult; for example, a sensor might not even know the total number of sensors nor where they are located if outside its radius of visibility. Not surprisingly, not many algorithmic results are known (e.g., [2, 3, 34, 59, 60, 79]). They are reviewed in this section, according to the scheduling model assumed: ASYNC, SSYNC, and SYNC.

5.1 Asynchronous Gathering

The most difficult setting for the gathering problem is clearly the *asynchronous* one, where little or no timing assumptions are made. In the literature, there are only few algorithms tackling asynchrony when gathering robotic sensors [34, 60].

In the investigation of Lin, Morse and Anderson [60], a limited form of asynchrony is considered where the time required by the *Wait*, *Locate*, and *Compute* states is bounded by a globally predefined amount, while the time spent in the *Move* state by sensor i is bounded by a locally predefined quantity (i.e., not necessarily the same for each sensor). This form of asynchrony lies in between the ASYNC model and the SSYNC model. The resulting solution allows a set of non-oblivious sensors with limited visibility to *converge* towards a single point.

The fully asynchronous model ASYNC is considered only in the investigation of Flocchini *et al* [34]. They show that the availability of *orientation*[5] allows a set of anonymous oblivious sensors with limited visibility to *gather* at a single point in finite time. This result holds not only allowing each activity and inactivity of the sensors to be totally unpredictable (but finite) in duration, but also making their movement towards a destination unpredictable (but not infinitesimally small) in length. In the rest of this subsection, we look at this result in more details.

Let *Right* be the rightmost vertical axis where some sensor initially lie. The idea of the algorithm is to make the sensors move towards *Right*, in such a way that, after a finite number of steps, they will reach it and gather at the bottommost position occupied by a sensor at that time.

Let s perform a *Locate* operation at time t; as a result, it has available its circle of visibility $\mathcal{C}_t(s)$ with the positions of all the sensors in it at time t. The algorithm describes the computation that s will now do with this input. Different destination points will be computed depending on the positions of the sensors in its circle of visibility; once the computation is completed, s

[5] i.e., agreement on axes and directions (positive vs. negative) of a common coordinate system, but not necessarily on the origin nor on the unit distance.

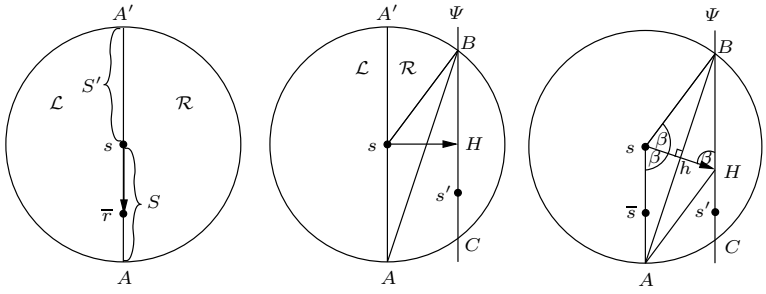

Fig. 10 (a) Notation used in the Gathering Algorithm. (b) Horizontal move. (c) Diagonal move.

starts moving towards its destination (but it may stop before the destination is reached). Informally,

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

To describe the diagonal movement in details we need to introduce some notation (refer to Figure 10). Let $\overline{AA'}$ be the vertical diameter of $\mathcal{C}_t(s)$ with A' the top and A the bottom end point; let $\mathcal{R}_s$ denote the topologically open region (with respect to $\overline{AA'}$) inside $\mathcal{C}_t(s)$ and to the right of s and let $S = \overline{sA}$ and $S' = \overline{sA'}$, where both S' and S are topologically open on the s side (i.e., s belongs neither to S' nor to S). Let Ψ be the vertical axis of the sensor in $\mathcal{R}_s$, if any, nearest to s with respect to its projection on the horizontal axis. We are now ready to describe the details of the diagonal movement routine:

```
Diagonal_Movement(Ψ)
  B := upper intersection between C_t(s) and Ψ;
  C := lower intersection between C_t(s) and Ψ;
  A := point on S at distance v from s;
  2β = AŝB;
  If β < 60° Then
    (B, Ψ) := Rotate(s, B);
  H := Diagonal_Destination(Ψ, A, B);
  Move(H).
```

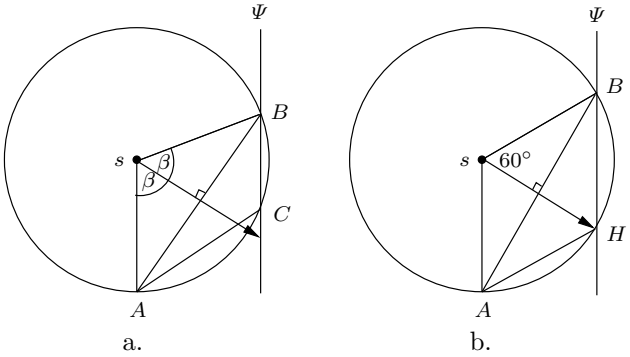

Fig. 11 Routine `Rotate()`: in (a), $\beta < 60°$; in (b) the scenario after `Rotate()` has been executed.

where `Rotate()` and `Diagonal_Destination()` are as follows.

- `Rotate`(s, B) rotates the segment $\overline{sB}$ in such a way that $\beta = 60°$ and returns the new position of B and Ψ. This choice of angle ensures that the destination point is not outside the circle. (see Figure 11).
- `Diagonal_Destination`(Ψ, A, B) computes the destination of s in the following way: the direction of s's movement is given by the perpendicular to the segment $\overline{AB}$; the destination of s is the point H on the intersection of the direction of its movement and of the axis Ψ.

The correctness of the algorithm is proven by first showing that the sensors which are initially visible will stay visible until the end of the computation, and then that the robots' movement leads to non-infinitesimally-small progress towards gathering thus concluding that all sensors will gather in a point on *Right* in finite time. We then have:

Theorem 10. *In the* Async *model a set of anonymous oblivious sensors in* $\mathbb{R}^2$*, endowed with orientation, can* gather *at a single point in finite time.*

Notice that the proposed algorithm does *not* assume that the sensors have the capability of *multiplicity detection* (i.e., the ability to determine in the sensing phase if more than one sensor is in a given location).

As mentioned, the above algorithm requires an agreement on the coordinate system. The problem of creating such an agreement in Async has been studied by Samiloglu, Gazi, and Bugra Koku, who have proposed several strategies that experimentally have been observed to converge towards a common orientation [78].

Probabilistic protocols for gathering in absence of agreement on the coordinate systems have been proposed and experimental analyzed by Soysal, Bahçeci, and Şahin [80].

5.2 Semi Synchronous Gathering

In the SSYNC model, the gathering problem has also been tackled by Ando *et al* [2]. In contrast with the setting considered in Section 5.1, here the sensors do not have any kind of common orientation. However, the compass has been traded with the semi-synchronicity of the sensors. Moreover, with the solution of [2], the robots only *converge* towards a gathering point.

Let $P(t) = \{s_1(t), \ldots, s_n(t)\}$ denote the set of the n sensors' positions at time t. Also, let $S_i(t)$ denote the set of sensors that are within distance v from s_i at time t; that is, the set of sensors that are visible from s_i (note that $s_i \in S_i(t)$). $SC_i(t)$ denotes the smallest enclosing circle of the set $\{s_j(t) | s_j \in S_i(t)\}$ of the positions of the sensors in $S_i(t)$ at t; let $c_i(t)$ be the center of $SC_i(t)$.

The algorithm is described below (refer also to Figure 12).

SEMI-SYNCH GATHERING ALGORITHM

1. If $S_i(t) = \{s_i\}$, then $x = s_i(t)$.
2. $\forall s_j \in S_i(t) \setminus \{s_i\}$,
   - 2.1. $d_j = dist(s_i(t), s_j(t))$,
   - 2.2. $\theta_j = c_i(t)\widehat{s_i(t)s_j(t)}$,
   - 2.3. $l_j = (d_j/2)\cos\theta_j + \sqrt{(v/2)^2 - ((d_j/2)\sin\theta_j)^2}$,
3. $LIMIT = \min_{s_j \in S_i(t)\setminus\{s_i\}}\{l_j\}$,
4. $GOAL = dist(s_i(t), c_i(t))$,
5. $MOVE = \min\{GOAL, LIMIT, \sigma\}$,
6. $x =$ point on $[s_i(t)c_i(t)]$ at distance $MOVE$ from $s_i(t)$.

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

1. $dist(s_i(t), x) \leq \sigma$. Note that this means that the sensors agree on an arbitrary small constant $\sigma > 0$, a-priori known, and they use it to bound the distance traveled by a sensor in one step.
2. For every sensor $s_j \in S_i(t)$, x lies in the disk D_j whose center is the midpoint m_j of $s_i(t)$ and $s_j(t)$, and whose radius is $v/2$. This condition ensures that s_i and s_j will still be visible after the movement of s_i (and possibly of s_j, see Figure 12.a).

The correctness proof is based on the following reasoning: First, two sensors that are connected in the visibility graph at time t, will stay connected at time $t+1$. In fact, if $s_i(t)$ and $s_j(t)$ are connected, then $s_j(t) \in S_i(t)$ and $s_i(t) \in S_j(t)$ and then, by definition of $LIMIT$, both $s_i(t+1)$ and $s_j(t+1)$

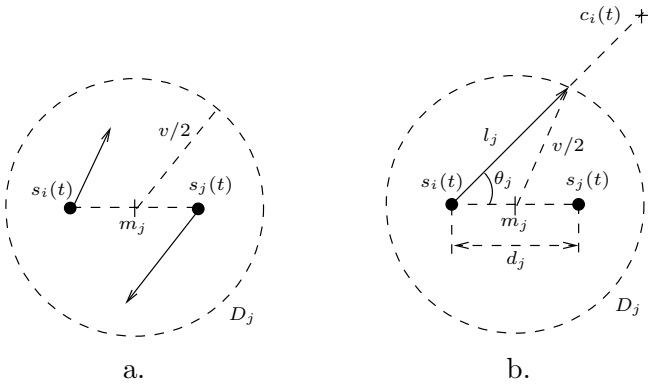

Fig. 12 The algorithm for the gathering problem in SSYNC.

lie inside the disc with center m_j (Figure 12.a). Second, let $CH(t)$ be the convex hull of the sensors at time t, for any $t \geq t_0$, $CH(t+1) \subseteq CH(t)$ leading to the proof that $CH(t)$ converges to a point. Then we have:

Theorem 11. *In the* SSYNC *model, a set of anonymous oblivious sensors in* $\mathbb{R}^2$ *can converge to a gathering point.*

The problem has also been examined in the same model SSYNC when there are are inaccuracies or faults. In particular, Gordon, Wagner and Bruckstein have investigated the case when the sensors cannot accurately measure the distance from their neighbors, and hence cannot rely on this information [42, 41]. The gathering problem has also been examined with respect to the availability of compasses. As discussed in Section 5.1, the presence of reliable compasses allows the sensors to gather in finite time even in the ASYNC model (Theorem 10) (and thus also in the SSYNC model). The problem of gathering when compasses are unstable for some arbitrary long periods has been studied by Soussi, Défago and Yamashita [79]. They proved that, in the SSYNC model, the sensors will gather in finite time, provided that the compasses stabilize eventually.

5.3 Fully Synchronous Gathering

There have been several investigations on the gathering problem with sensors operating in the fully synchronous scenario, i.e., in model SYNC [39, 59, 64, 91]. The starting point of these investigations is the convergence protocol of Ando *et al* [2], described in Section 5.2, operating in the SSYNC and thus in the SYNC models. Like [2], these protocols work for oblivious sensors with no common coordinate system, and they all converge towards a unique point; unlike [2], they are only for the SYNC model.

Lin, Morse, and Anderson [59] propose and analyse a family of convergence algorithms for gathering in the plane based on [2]. In [64] variants of the general strategy described in [59] are developed: Martínez considers the presence of noisy measurements of neighbors [64].

In all the above investigations on gathering, as well as those in the ASYNC and SSYNC models discussed before, the universe $\mathbb{U}$ in which the gathering was taking place was (implicitly) assumed to be either the entire plane or a *convex* region of the plane.

The case when the sensors operate in a *non-convex* region (see Figure 13), of which they have no map, has been considered only by Ganguli, Cortés, and Bullo in [39]. In such a space, two sensors s and s' are said to be *mutually visible* at time t if not only their distance is at most v but also the segment connecting their positions at time t is completely contained in $\mathbb{U}$;. for instance, sensors s and s' in Figure 13 are within distance v, but they are not mutually visible. The approach used to solve the problem is that of computing a set of constraints that the sensors have to follow when moving so that (a) the mutual visibility graph stays connected during the movements, and (b) the distances between sensor strictly decrease at each time step.

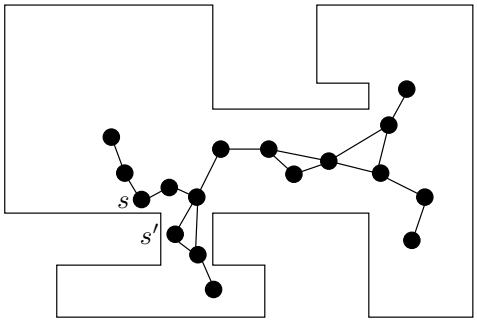

Fig. 13 An example of non-convex environment for the gathering problem in [39]. The edges between sensors represent the edges of the visibility graph.

The first set of constraints is derived from those by Ando *et al* [2] discussed in section 5.2, and guarantees that condition (a) is met; in particular, it imposes that if two sensors s_i and s_j are mutually visible at time t, they stay connected at time $t+1$: let $p_i = s_i(t)$ and $p_j = s_j(t)$ be the positions of sensors s_i and s_j at time t, respectively, then s_i and s_j are allowed to move inside the ball B of radius $\frac{v}{2}$ centered in the mid-point of p_i and p_j (see Figure 12). Clearly, since the sensors operate in a non-convex environment, the sensors are limited to move inside any convex area contained in the intersection between B and $\mathbb{U}$: in [39] the *Constraint Set Generator Algorithm* is given, to allow the computation of such a convex area by any pair of mutually visible sensors.

The overall idea of the gathering algorithm, called the *Perimeter Minimizing Algorithm*, can be summarized as follows: at each time step t, each sensor computes all convex areas resulting from executing the *Constraint Set Generator Algorithm* for all its neighbors in the v-range visibility graph at t; the area where it is allowed to move in order to verify condition (a) is therefore the intersection of all these areas. The sensor moves now towards the circumcenter of its allowed moving zone, i.e. the center of the smallest circle enclosing this area. It can be proven that this choice satisfies condition (b) above.

Theorem 12. *In the* Sync *model, a set of anonymous oblivious sensors in operating in a non-convex environment can converge to a gathering point.*

The behaviour of this protocol has been analyzed experimentally in [39] also when (i) the sensors operate asynchronously, or (ii) the sensors can introduce distance and direction errors in both sensing and moving, or (iii) the sensors are modeled as disks; the results give an indication of the robustness of the protocol with respect to those three factors.

5.4 Coalescence

An interesting problem related to the gathering is *Coalescence*: arbitrarily dispersed (and possibly isolated) mobile sensors must independently search for their fellow sensors with the goal of being all within a given distance. Note that if that distance is not greather than v, this is precisely the *Near Gathering* problem whose goal is to form a single connected visibility graph.

The coalescence problem has been investigated by Poduri and Sukhatme in [73]. They consider sensors performing an independent random search of the other sensors according to the following set of rules.

Coalescence

1. An isolated sensor
   a. uniformly chooses a direction of movement θ in $[0, 2\pi)$.
   b. moves following the chosen direction with a constant speed for a constant distance.
2. When two sensors "meet" (i.e., they are within distance v) they form a single cluster: they stay connected to each other and move together following the same (random) path.
3. When two clusters meet they coalesce to form a single cluster.

Given this set of rules, considering a fully synchronous scenario (i.e., the Sync model), the main question addressed in [73] is how long will it take

the sensors to coalesce into a single connected visibility graph. Clearly the *spread* of the clusters plays an important role: in fact, if the sensors in each cluster remain spread out, the disconnected sensors have higher probability of being discovered. That is, here the focus is on connectivity rather than on colocation. They show analytically that coalescence time has an exponential distribution which is function of the number of sensors, spread, communication range, and size of the domain. Also, as the number of sensors increases, coalescence time decreases as $O(\frac{1}{\sqrt{n}})$ and $\Omega(\frac{1}{n}\log n)$. Simulation experiments support the analytical results, suggesting that the lower bounds derived analytically for coalescence time is tight.

6 Conclusions and Open Problems

Several research questions are still open. With respect to *gathering*, the outstanding open problem is whether it is possible to gather when the robotic sensors are asynchronous, oblivious and without common orientation.

For *self-deployment*, the foremost open problem is the determination of whether knowledge of d is indeed necessary for exact self-deployment in an *oriented ring*. Should this be the case, the research goal becomes to determine which is the "weakest" additional assumption (e.g., a priori knowledge, capability) that would make exact self-deployment possible. A more general and challenging open problem is to find additional sensors' capabilities that would enable the existence of an asynchronous exact self-deployment protocol in *unoriented rings*.

The impact that sensorial errors and inaccuracies have on the correctness of the algorithms should be studied in detail. New algorithms are needed for different assumptions on the visibility power of the sensors; for instance, the accuracy of the sensors' ability to detect the other sensors' positions might decrease with the distance. Another important concern is clearly the one of the presence of possible faulty sensors. The fault-tolerant issues have been recently addressed in [1, 52, 79], but only in the unlimited visibility setting.

Finally, an open and important research direction is to identify meaningful efficiency parameters and study the computational complexity of the problem. In fact, in all existing investigations, the complexity of the solutions has never been an issue; indeed, there is an absence of cost measures.

Slightly faulty snapshots, obstacles that limit the visibility and that moving sensors must avoid or push aside, sensors that appear and disappear from the scene, as well as precise cost measures, clearly suggest that the algorithmic nature of distributed coordination of autonomous, mobile robotic sensors is far from been completed and further investigations are clearly needed.

Acknowledgments. This work has been supported in part by the Natural Sciences and Engineering Research Council of Canada, under Discovery Grants, and by PRIN 2008 - MadWeb.

References

[1] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. *SIAM Journal on Computing*, 36:56–82, 2006.

[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] H. Ando, I. Suzuki, and M. Yamashita. Formation and agreement problems for synchronous mobile robots with limited visibility. In *Proceedings of IEEE Symposium of Intelligent Control*, pages 453–460, 1995.

[4] L. Barrière, P. Flocchini, E. Mesa-Barrameda, and N. Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science*, to appear, 2010.

[5] M. A. Batalin and G. S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Telecommunication Systems*, 26(2):181–196, 2004.

[6] D. Bhadauria and V. Isler. Data gathering tours for mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3868–3873, 2009.

[7] F. Bullo, J. Cortes, and S. Martinez. *Distributed Control of Robotic Networks*. Princeton University Press, 2009.

[8] Y. U. Cao, A. S. Fukunaga, A. B. Kahng, and F. Meng. Cooperative mobile robotics: Antecedents and directions. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 226–234, 1995.

[9] I. Chatzigiannakis, M. Markou, and S. Nikoletseas. Distributed circle formation for anonymous oblivious robots. In *Proceedings of* 3^{rd} *International Workshop on Experimental and Efficient Algorithms (WEA)*, pages 159 –174, 2004.

[10] M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Solving the gathering problem. In *Proceedings of* 30^{th} *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1181–1196, 2003.

[11] J. Clark and R. Fierro. Mobile robotic sensors for perimeter detection and tracking. *ISA Transactions*, 46(1):3–13, 2007.

[12] R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34:1516–1528, 2005.

[13] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399:71–82, 2008.

[14] K. Dantu, M. Rahimi, H. Shah, S. Babel, A. Dhariwal, and G.S. Sukhatme. Robomote: enabling mobility in sensor networks. In *Proceedings of* 4^{th} *International Symposium on Information Processing in Sensor Networks (IPSN)*, pages 404– 409, 2005.

[15] S. Das, P. Flocchini, S. Kutten, A. Nayak, and N. Santoro. Map construction of unknown graphs by multiple agents. *Theoretical Computer Science*, 385(1-3):34–48, 2007.

[16] S. Das, E. Mesa-Barrameda, and N. Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In *Proceedings of* 4^{th} *International Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSOR)*, pages 25–140, 2008.

[17] X. A. Debest. Remark about self-stabilizing systems. *Communication of the ACM*, 2(38):115–177, 1995.

[18] X. Défago, M. Gradinariu, S. Messika, and P. R. Parvédy. Fault-tolerant and self-stabilizing mobile robots gathering. In *Proceedings of* 20^{th} *International Symposium on Distributed Computing (DISC)*, pages 46–60, 2006.

[19] X. Défago and A. Konagaya. Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In *Proceedings of Workshop on Principles of Mobile Computing*, pages 97–104, 2002.

[20] X. Défago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theoretical Computer Science*, 396(1-3):97–112, 2008.

[21] Y. Dieudonné, S. Dolev, F. Petit, and M. Sega. Deaf, dumb, and chatting asynchronous robots. In *Proceedings of* 13^{th} *International Conference on Principles of Distributed Systems (OPODIS)*, LNCS 5923, pages 71–85, 2009.

[22] Y. Dieudonné, O. Labbani-Igbida, and F. Petit. Circle formation of weak mobile robots. *ACM Transactions on Autonomous and Adaptive Systems*, 3(4), 2008.

[23] Y. Dieudonné and F. Petit. Circle formation of weak robots and lyndon words. *Information Processing Letters*, 4(104):156—162, 2007.

[24] Y. Dieudonné and F. Petit. Swing words to make circle formation quiescent. In *Proceedings of* 14^{th} *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS 4474, pages 166—179, 2007.

[25] E. W. Dijkstra. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982.

[26] S. Dobrev, P. Flocchini, G. Prencipe, and N. Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Computing*, 19(1):1–19, 2006.

[27] M. Dunbabin, P. Corke, I. Vasilescu, and D. Rus. Data muling over underwater wireless sensor networks using an autonomous underwater

vehicle. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2006.

[28] A. Efrima and D. Peleg. Distributed models and algorithms for mobile robot systems. In *Proceedings of* 33^{rd} *Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, LNCS 4362, pages 70–87, 2007.

[29] Y. Elor and A. M. Bruckstein. Multi-agent deployment and patrolling on a ring graph. Technical Report CIS-2009-16, Computer Science Department, Technion, Israel, 2009.

[30] N. Fatès. Solving the decentralized gathering problem with a reaction-diffusion-chemotaxis scheme. *Swarm Intelligence*, (to appear), 2010.

[31] P. Flocchini, G. Prencipe, and N. Santoro. Self-deployment algorithms for mobile sensors on a ring. *Theoretical Computer Science*, 402(1):67–80, 2008.

[32] P. Flocchini, G. Prencipe, and N. Santoro. Near gathering of weak robots with limited visibility: Algorithms and applications. Technical report, Carleton University, 2010.

[33] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Pattern formation by autonomous robots without chirality. In *Proceedings of* 8^{th} *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 147–162, 2001.

[34] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous mobile robots with limited visibility. *Theoretical Computer Science*, 337:147–168, 2005.

[35] P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.

[36] P. Fraigniaud, L. Gasieniec, D. Kowalski, and A. Pelc. Collective tree exploration. *Networks*, 48:166–177, 2006.

[37] J. Fredslund and M. J. Matarić. A general algorithm for robot formations using local sensing and minimal communication. *IEEE Transactions on Robotics and Automation*, 18(5):837–846, 2002.

[38] A. Ganguli, J. Cortes, and F. Bullo. Visibility-based multi-agent deployment in orthogonal environments. In *Proceedings of American Control Conference*, pages 3426–3431, 2007.

[39] A. Ganguli, J. Cortés, and F. Bullo. Multirobot rendezvous with visibility sensors in nonconvex environments. *IEEE Transactions on Robotics*, 25(2):340–352, 2009.

[40] S. Gilbert, N. Lynch, S. Mitra, and T. Nolte. Self-stabilizing robot formations over unreliable networks. *ACM Transactions on Autonomous and Adaptive Systems*, 4(3):1–29, 2009.

[41] N. Gordon, Y. Elor, and A. M. Bruckstein. Gathering multiple robotic agents with crude distance sensing capabilities. In *Proceedings of* 6^{th} *International Conference on Ant Colony Optimizatin and Swarm Intelligence*, LNCS 5217, pages 72–83, 2008.

[42] N. Gordon, I. A. Wagner, and A. M. Bruckstein. Gathering multiple robotic a(ge)nts with limited sensing capabilities. In *Proceedings of* 2^{nd} *International Conference on Ant Colony Optimization and Swarm Intelligence*, LNCS 3172, pages 142–153, 2004.

[43] N. Heo and P. K. Varshney. A distributed self spreading algorithm for mobile wireless sensor networks. In *Proceedings of IEEE Wireless Communication and Networking Conference*, volume 3, pages 1597–1602, 2003.

[44] N. Heo and P. K. Varshney. Energy-efficient deployment of intelligent mobile sensor networks. *IEEE Transactions on Systems, Man, and CyberNetics - Part A: Systems and Humans*, 35(1):78–92, 2005.

[45] A. Howard, M. J. Mataric, and G. S. Sukhatme. An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots*, 13(2):113–126, 2002.

[46] A. Howard, M. J. Mataric, and G. S. Sukhatme. Mobile sensor network deployment using potential fields. In *Proceedings of* 6^{th} *International Symposium on Distributed Autonomous Robotics Systems (DARS)*, pages 299–308, 2002.

[47] T.-R. Hsiang, E. Arkin, M. A. Bender, S. Fekete, and J. Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Proceedings of* 5^{th} *Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 77–94, 2002.

[48] Y. Ikemoto, Y. Hasegawa, T. Fukuda, and K. Matsuda. Gradual spatial pattern formation of homogeneous robot group. *Information Sciences*, 171(4):431–445, 2005.

[49] D. Jung, G. Cheng, and A. Zelinsky. Experiments in realising cooperation between autonomous mobile robots. In *Proceedings of* 5*th International Symposium on Experimental Robotics (ISER)*, pages 513–524, 1997.

[50] A. Kansal, W. Kaiser, G. Pottie, M. Srivastava, and G. S. Sukhatme. Reconfiguration methods for mobile sensor networks. *ACM Transactions on Sensor Networks*, 3(4):22–23, 2007.

[51] M. Kasuya, N. Ito, N. Inuzuka, and K. Wada. A pattern formation algorithm for a set of autonomous distributed robots with agreement on orientation along one axis. *Systems and Computers in Japan*, 37(10):89–100, 2006.

[52] Y. Katayama, Y. Tomida, H. Imazu, N. Inuzuka, and K. Wada. Dynamic compass models and gathering algorithms for autonomous mobile robots. In *Proceedings of* 14^{th} *Colloquium on Structural Information and Communication Complexity (SIROCCO)*, LNCS 4474, 2007.

[53] B. Katreniak. Biangular circle formation by asynchronous mobile robots. In *Proceedings of* 12^{th} *International Colloquium on Structural and Communication Complexity (SIROCCO)*, LNCS 3499, pages 185–199, 2005.

[54] O. Kosut, A. Turovsky, J. Sun, M. Ezovski, G. Whipps, and L. Tong. Integrated mobile and static sensing for target tracking. In *Proceedings of Military Communications Conference (MILCOM)*, pages 1–7, 2007.

[55] G. Lee, S. Yoon, N. Y. Chong, and H. Christensen. A mobile sensor network forming concentric circles through local interaction and consensus building. *Journal of Robotics and Mechatronics*, 21(4):469–477, 2009.

[56] J. Lee, S. Venkatesh, and M. Kumar. Formation of a geometric pattern with a mobile wireless sensor network. *Journal of Robotic Systems*, 21(10):517–530, 2004.

[57] X. Li, H. Frey, N. Santoro, and I. Stojmenovic. Focused coverage by mobile sensor networks. In *Proceedings of* 6^{th} *IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, pages 466–475, 2009.

[58] X. Li and N. Santoro. An integrated self-deployment and coverage maintenance scheme for mobile sensor networks. In *Proceedings of* 2^{nd} *International Conference on Mobile Ad-Hoc and Sensors Networks (MSN)*, pages 847–860, 2006.

[59] J. Lin, A.S. Morse, and B.D.O. Anderson. The multi-agent rendezvous problem. part 1: The synchronous case. *SIAM Journal on Control and Optimization*, 46(6):2096–2119, 2007.

[60] J. Lin, A.S. Morse, and B.D.O. Anderson. The multi-agent rendezvous problem. part 2: The asynchronous case. *SIAM Journal on Control and Optimization*, 46(6):2120–2147, 2007.

[61] B. Liu, P. Brass, O. Dousse, P. Nain, and D. Towsley. Mobility improves coverage of sensor networks. In *Proceedings of* 6^{th} *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 300–308, 2005.

[62] L. Loo, E. Lin, M. Kam, and P. Varshney. Cooperative multi-agent constellation formation under sensing and communication constraints. *Cooperative Control and Optimization*, pages 143–170, 2002.

[63] N. Lynch, S. Mitra, and T. Nolte. Motion coordination using virtual nodes. In *Proceedings of* 44^{th} *IEEE Conference on Decision and Control*, 2005.

[64] S. Martínez. Practical multiagent rendezvous through modified circumcenter algorithms. *Automatica*, 45(9):2010–2017, 2009.

[65] S. Martínez, F. Bullo, J. Cortes, and E. Frazzoli. On synchronous robotic networks -parts i and ii. *IEEE Transactions on Automatic Control,*, 52(12):2199–2226, 2007.

[66] E. Martinson and D. Payton. Lattice formation in mobile autonomous sensor arrays. In *Proceedings of International Workshop on Swarm Robotics (SAB)*, pages 98–111, 2004.

[67] Y. Mei, C. Xian, S. Das, Y.C. Hu, and Y.-H. Lu. Sensor replacement using mobile robots. *Computer Communications*, 30(13):2615–2626, 2007.

[68] T. Miyamae, S. Ichikawa, and F. Hara. Emergent approach to circle formation by multiple autonomous modular robots. *Journal of Robotics and Mechatronics*, 21(1):3–11, 2009.

[69] S. E. Nikoletseas. Models and algorithms for wireless sensor networks (smart dust). In *Proceedings of* 32^{nd} *Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, LNCS 3831, pages 64–83, 2006.

[70] Y. Oasa, I. Suzuki, and M. Yamashita. A robust distributed convergence algorithm for autonomous mobile robots. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 287–292, 1997.

[71] M.R. Pac, A. M. Erkmen, and I. Erkmen. Scalable self-deployment of mobile sensor networks: A fluid dynamics approach. In *Proceedings of IEEE/RSJ International Conference Intelligent Robots and Systems*, pages 1446–1451, 2006.

[72] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *Proceedings of IEEE International Conference on Robotic and Automation*, pages 165–173, 2004.

[73] S. Poduri and G. S. Sukhatme. Achieving connectivity through coalescence in mobile robot networks. In *Proceedings of* 1^{st} *ACM International Conference on Robot Communication and Coordination (RoboComm)*, volume 318, pages 1–6, 2007.

[74] O. Powell, P. Leone, and J. Rolim. Energy optimal data propagation in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 67(3):302–317, 2007.

[75] G. Prencipe. The effect of synchronicity on the behavior of autonomous mobile robots. *Theory Of Computing Systems*, 38:539–558, 2005.

[76] G. Prencipe and N. Santoro. Distributed algorithms for mobile robots. In *Proceedings of* 5^{th} *IFIP International Conference on Theoretical Computer Science (TCS)*, 2006.

[77] S. Samia, X. Défago, and T. Katayama. Convergence of a uniform circle formation algorithm for distributed autonomous mobile robots. In *Proceedings of Journés Scientifiques Francophones (JSF), Tokio, Japan*, 2004.

[78] A. T. Samiloglu, V. Gazi, and A. Bugra Koku. Comparison of three orientation agreement strategies in self-propelled particle systems with turn angle restrictions in synchornous and asynchronous settings. *Asian Journal of Control*, 10(2):212–232, 2008.

[79] S. Souissi, X. Défago, and M. Yamashita. Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. *ACM Transactions on Autonomous and Adaptive Systemse*, 4(1):1–27, 2009.

[80] O. Soysal, E. Bahçeci, and E. Şahin. Aggregation in swarm robotic systems: Evolution and probabilistic control. *Turkish Journal Electrical Engineering*, 15(2):199–225, 2007.

[81] K. Sugihara and I. Suzuki. Distributed algorithms for formation of geometric patterns with many mobile robots. *Journal of Robotics Systems*, 13:127–139, 1996.

[82] S. Susca, S. Martinez, and F. Bullo. Monitoring enviromental boundaries with a robotic sensor network. *IEEE Transactions on Control Systems Technology*, 16(2):288–296, 2008.

[83] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM Journal on Computing*, 28(4):1347–1363, 1999.

[84] T. Suzuki, R. Sugizaki, K. Kawabata, Y. Hada, and Y. Tobes. Deployment and management of wireless sensor network using mobile robots for gathering environmental information. In *Proceedings of* 9^{th} *International Symposia on Distributed Autonomous Robotic Systems (DARS)*, pages 63–72, 2009.

[85] O. Tanaka. Forming a circle by distributed anonymous mobile robots. Technical report, Department of Electrical Engineering, Hiroshima University, Japan, 1992.

[86] O. Tekdas, J.H. Lim, A. Terzis, and V. Isler. Using mobile robots to harvest data from sensor fields. *IEEE Wireless Communications*, 16(1):22–28, 2009.

[87] G. Wang, G. Cao, P. Berman, and T. La Porta. A bidding protocol for deploying mobile sensors. *IEEE Transactions on Mobile Computing*, 6(5):563–576, 2007.

[88] G. Wang, G. Cao, and T. La Porta. Movement-assisted sensor deployment. *IEEE Transactions on Mobile Computing*, 5(6):640–652, 2006.

[89] P. K. C. Wang. Navigation Strategies for Multiple Autonomous Mobile Robots Moving in Formation. *Journal of Robotic Systems*, 8(2):177–195, 1991.

[90] M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, (to appear), 2010.

[91] J. Yu, M. LaValle, and D. Liberzon. Rendezvous without coordinates. In *Proceedings of* 47^{th} *IEEE Conference on Decision and Control*, pages 1803–1808, 2008.

[92] Y. Zou and K. Chakrabarty. Sensor deployment and target localization in distributed sensor networks. *ACM Transactions on Embedded Computing Systems*, 3(1):61–91, 2004.