Title:	Memoryless gathering of mobile robotic sensors
Name:	Paola Flocchini[1]
Affil./Addr.	School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada
Keywords:	Gathering; Sensors aggregation; Rendezvous
SumOriWork:	2005; Flocchini, Prencipe, Santoro, Widmayer 1999; Ando, Oasa, Suzuki, Yamashita

Memoryless gathering of mobile robotic sensors

Paola Flocchini[1]

School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON, Canada

Years aud Authors of Summarized Original Work

2005; Flocchini, Prencipe, Santoro, Widmayer
1999; Ando, Oasa, Suzuki, Yamashita

Keywords

Gathering; Sensors aggregation; Rendezvous

Problem Definition

The Model: A *mobile robotic sensor* (or simply *sensor*) is modeled as a computational unit with sensorial capabilities: it can perceive the spatial environment within a fixed distance $V > 0$, called *visibility range*, it has its own local working memory, and it is capable of performing local computations [3; 4].

Each sensor is a point in its own local Cartesian coordinate system (not necessarily consistent with the others), of which it perceives itself as the centre. A sensor can move in any direction, but it may be stopped before reaching its destination, e.g. because of limits to its motion energy; however, it is assumed that the distance traveled in a move by a sensor is not infinitesimally small (unless it brings the sensor to its destination).

The sensors have no means of direct communication: any communication occurs in an implicit manner, by observing the other sensors' positions. Moreover, they are *autonomous* (i.e., without a central control) *identical* (i.e., they execute the same protocol), *anonymous* (i.e., without identifiers that can be used during the computation).

The sensors can be *active* or *inactive*. When *active*, a sensor performs a *Look-Compute-Move* cycle of operations: it first observes the portion of the space within its visibility range obtaining a snapshot of the positions of the sensors in its range at that time (*Look*); using the snapshot as an input, the sensor then executes the algorithm to determine a destination point (*Compute*); finally, it moves towards the computed destination, if different from the current location (*Move*). After that, it becomes *inactive* and stays idle until the next activation. Sensors are *oblivious*: when a sensor becomes active, it does not remember any information from previous cycles. Note that several sensors could actually occupy the same point; we call *multiplicity detection* the ability of a sensor to see whether a point is occupied by a single sensor or by more than one.

Depending on the degree of synchronization among the cycles of different sensors, three sub-models are traditionally identified: *synchronous*, *semi-synchronous*, and *asynchronous*. In the *synchronous* (FSYNC) and in the *semi-synchronous* (SSYNC) models, 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. In FSYNC, at each clock tick all sensors become active, while in SSYNC only a subset of the sensors might be active in each cycle. 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.

Let $S(t) = \{s_1(t), \ldots, s_n(t)\}$ denote the set of the n sensors' at time t. When no ambiguity arises, we shall omit the temporal index t. Moreover, with an abuse of notation we indicate by s_i both a sensor and its position. 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. At any point in time t, the sensors induce a *visibility graph* $G(t) = (N, E(t))$ defined as follows: $N = S$ and, $\forall r, s \in N$, $(r, s) \in E(t)$ iff r and s are at distance no more than the visibility range V.

The Problem: In this setting, one of the most basic coordination and synchronization task is *Gathering*: the sensors, initially placed in arbitrary distinct positions in a 2-dimensional space, must congregate at a single location (the choice of the location is not predetermined) within finite time. In the following, we assume $n > 2$. A problem closely related to *Gathering* is *Convergence*, where the sensors need to be arbitrarily close to a common location, without the requirement of ever reaching it. A special type of convergence (also called *Near-Gathering*, or *collision-less convergence*) requires the sensors to converge without ever colliding with each other.

Key Results

Basic Impossibility Results

First of all, neither *Convergence* nor *Gathering* can be achieved from arbitrary initial placements if the initial visibility graph $G(0)$ is not connected. So, in all the literature, $G(0)$ is always assumed to be connected. Furthermore, if the sensors have *neither* agreement on the coordinate system *nor* multiplicity detection, then *Gathering* is not solvable in SSYNC (and thus in ASYNC), regardless of the range of visibility and the amount of memory that they may have.

Theorem 1. *[8] In absence of multiplicity detection and of any agreement on the coordinate systems, Gathering is deterministically unsolvable in* SSYNC.

Given this impossibility result, the natural question is whether the problem can be solved with common coordinate systems.

Gathering with common coordinate systems

Assuming that the sensors agree on a common coordinate system, *Gathering* has been shown to be solvable under the weakest of the three schedulers (ASYNC) [2].

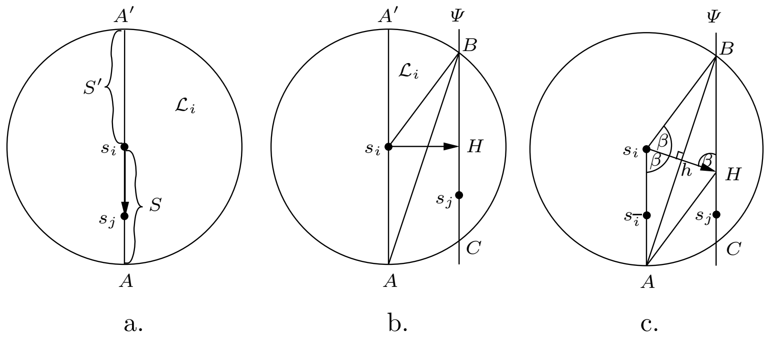

Fig. 1. From [4]: (a) Notation. (b) Horizontal move. (c) Diagonal move.

Let $\mathcal{R}$ be the rightmost vertical axis where some sensor initially lie. The idea of the algorithm is to make the sensors move towards $\mathcal{R}$, in such a way that, after a finite number of steps, they will reach it and gather at the bottom-most position occupied by a sensor at that time. Let the *Look* operation of sensor s_i at time t return $S_i(t)$. The computed destination point of s_i depends on the positions of the visible sensors. Once the computation is completed, s_i moves towards its destination (but it may stop before the destination is reached). Informally,

- If s_i sees sensors to its left or above on the vertical axis passing through its position (this axis will be referred to as *its vertical axis*), it does not move.
- If s_i sees sensors only below on its vertical axis, it moves down towards the nearest sensor.
- If s_i sees sensors only to its right, it moves horizontally towards the vertical axis of the nearest sensor.
- If s_i sees sensors both below on its vertical 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 we introduce some notation. (refer to Figure 1). Let $\overline{AA'}$ be the vertical diameter of $S_i(t)$ with A' the top and A the bottom end point; let $\mathcal{L}_i$ denote the topologically open region (with respect to $\overline{AA'}$) inside $S_i(t)$ and to the right of s_i and let $S = \overline{s_iA}$ and $S' = \overline{s_iA'}$, where neither S' and S include s_i. Let Ψ be the vertical axis of the horizontally closest sensor (if any) in $\mathcal{L}_i$.

```
Diagonal_Movement($\Psi$)
    $B :=$ upper intersection between $S_i(t)$ and $\Psi$;
    $C :=$ lower intersection between $S_i(t)$ and $\Psi$;
    $2\beta = A\widehat{s_i}B$;
    If $\beta < 60°$ Then
        $(B, \Psi) :=$ Rotate($s_i, B$);
    $H :=$ Diagonal_Destination($\Psi, A, B$);
    Move towards $H$.
```

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

- `Rotate(`s_i, B`)` rotates the segment $\overline{s_iB}$ in such a way that $\beta = 60°$ and returns the new position of B and Ψ. This angle choice ensures that the destination point is not outside the circle.
- `Diagonal_Destination(`Ψ, A, B`)` computes the destination of s_i as follows: the direction of s_i's movement is given by the perpendicular to the segment $\overline{AB}$; the

destination of s_i is the point H on the intersection of the direction of its movement and of the axis Ψ.

Theorem 2. *[2] With common coordinate systems, Gathering is possible in* Async.

Gathering has been shown to be possible in Ssync also when compasses are unstable for some arbitrary long periods, provided they have a common clockwise notion, and that they eventually stabilize, and assuming the total number of sensors is known [9].

Convergence and Near-Gathering

Convergence in Ssync. The impossibility result does not apply to the case of *Convergence*. In fact, it is possible to solve it in Ssync in the basic model (i.e., without common coordinate systems) [1].

Let $SC_i(t)$ denotes the smallest enclosing circle of the set $\{s_j(t)|s_j \in S(t)\}$ of positions of sensors in $S(t)$; let $c_i(t)$ be the center of $SC_i(t)$.

Every time s_i becomes active, it moves toward $c_i(t)$, but only up to a certain distance. Specifically, if s_i does not see any sensor other than itself, then s_i does not move. Otherwise, its destination is the point p on the segment $\overline{s_i(t)c_i(t)}$ that is closest to $c_i(t)$ and that satisfies the following condition: For every sensor $s_j \in S(t)$, p lies in the disk $\mathcal{C}_j$ whose center is the midpoint m_j of $s_i(t)$ and $s_j(t)$, and whose range 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.

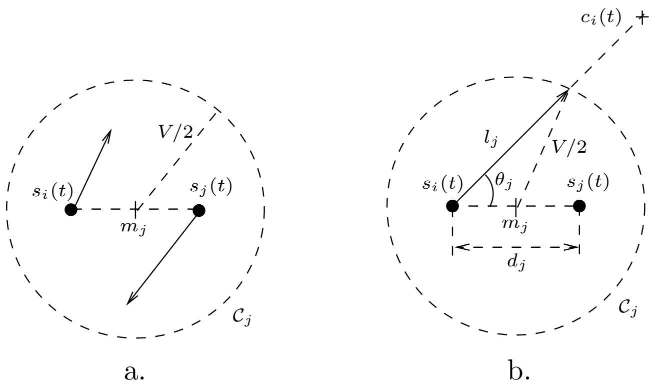

Fig. 2. From [4]: Notation for algorithm Convengence [1].

```
Convergence
Assumptions: Ssync.
1. If $S_i(t) = \{s_i\}$, then gathering is completed.
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. $D := \min\{goal, limit\}$,
6. $p :=$ point on $\overline{s_i(t)c_i(t)}$ at distance $D$ from $s_i(t)$.
7. Move towards $p$.
```

Theorem 3. *[1] Convergence is possible in* Ssync.

Convergence in Async**.** *Convergence* has been shown to be possible also in Async, but under special schedulers: *partial* Async [6] and *1-fair* Async [5]. In *partial* Async the time spent by a sensor performing the *Look*, *Compute* and *Sleep* operations is bounded by a globally predefined amount, and the time spent in the *Move* operation by a locally predefined amount; in 1-fair Async between two successive activations of each sensor, all the other sensors are activated at most once. Finally, *Convergence* has been studied also in presence of perception inaccuracies (radial errors in locating a sensor) and it has been show how to reach convergence in Fsync for small inaccuracies.

Near-Gathering. Slight modifications can make the algorithm of [1] described above collision-less, thus solving also the *Collision-less Convergence* problem (i.e., *Near-Gathering*) in Ssync. *Near-Gathering* can be achieved also in Async, with two additional assumptions [7]: 1) the sensors must partially agree on a common coordinate system (one axis is sufficient) and 2) the initial visibility graph must be *well-connected*, that is, the subgraph of the visibility graph that contains only the edges corresponding to sensors at distance *strictly smaller* than V must be connected.

Open Problems

The existing results for *Gathering* and *Convergence* leave several problems open. For example, *Gathering* in Ssync (and thus Async) has been proven impossible when neither multiplicity detection nor an orientation are available. While common orientation suffices, it is not known whether the presence of multiplicity detection alone is sufficient to solve the problem. Also, the impossibility result does not apply to Fsync; however no algorithm is known in such a setting that does not make use of orientation. Finally, it is not known whether *Convergence* (collision-less or not) is solvable in Async without additional assumptions: so far no algorithm has been provided, nor an impossibility proof.

Recommended Reading

1. Ando H, Oasa Y, Suzuki I, Yamashita M (1999) A distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation 15(5):818–828
2. Flocchini P, Prencipe G, Santoro N, Widmayer P (2005) Gathering of asynchronous mobile robots with limited visibility. Theoretical Computer Science 337:147–168
3. Flocchini P, Prencipe G, Santoro N (2011) Computing by mobile robotic sensors. Chapter 21 in *Theoretical Aspects of Distributed Computing in Sensor Networks*, S. Nikoletseas and J. Rolim Eds, Springer, ISBN 978-3-642-14849-1
4. Flocchini P, Prencipe G, Santoro N (2012) Distributed Computing by Oblivious Mobile Robots. Morgan & Claypool
5. Katreniak B (2011) Convergence with limited visibility by asynchronous mobile robots. In: 18^{th} Int. Colloquium on Structural Information and Communication Complexity (SIROCCO), pp 125–137
6. Lin J, Morse A, Anderson B (2007) The multi-agent rendezvous problem. part 2: The asynchronous case. SIAM Journal on Control and Optimization 46(6):2120–2147
7. Pagli L, Prencipe G, Viglietta G (2012) Getting close without touching. In: 19^{th} Int. Colloquium on Structural Information and Communication Complexity (SIROCCO), pp 315–326
8. Prencipe G (2007) Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384(2-3):222–231
9. Souissi S, Défago X, Yamashita M (2009) Using eventually consistent compasses to gather memory-less mobile robots with limited visibility. ACM Transactions on Autonomous and Adaptive Systemse 4(1):1–27