TuringMobile: A Turing Machine of Oblivious Mobile Robots with Limited Visibility and its Applications

Giuseppe A. Di Luna
Aix-Marseille University and LiS Laboratory, Marseille, France
giuseppe.diluna@lif.univ-mrs.fr

Paola Flocchini
University of Ottawa, Ottawa, Canada
paola.flocchini@uottawa.ca

Nicola Santoro
Carleton University, Ottawa, Canada
santoro@scs.carleton.ca

Giovanni Viglietta[1]
JAIST, Nomi City, Japan
johnny@jaist.ac.jp

Abstract

In this paper we investigate the computational power of a set of mobile robots with limited visibility. At each iteration, a robot takes a snapshot of its surroundings, uses the snapshot to compute a destination point, and it moves toward its destination. Each robot is punctiform and memoryless, it operates in $\mathbb{R}^m$, it has a local reference system independent of the other robots' ones, and is activated asynchronously by an adversarial scheduler. Moreover, robots are non-rigid, in that they may be stopped by the scheduler at each move before reaching their destination (but are guaranteed to travel at least a fixed unknown distance before being stopped).

We show that despite these strong limitations, it is possible to arrange $3m+3k$ of these weak entities in $\mathbb{R}^m$ to simulate the behavior of a stronger robot that is rigid (i.e., it always reaches its destination) and is endowed with k registers of persistent memory, each of which can store a real number. We call this arrangement a *TuringMobile*. In its simplest form, a TuringMobile consisting of only three robots can travel in the plane and store and update a single real number. We also prove that this task is impossible with fewer than three robots.

Among the applications of the TuringMobile, we focused on Near-Gathering (all robots have to gather in a small-enough disk) and Pattern Formation (of which Gathering is a special case) with limited visibility. Interestingly, our investigation implies that both problems are solvable in Euclidean spaces of any dimension, even if the visibility graph of the robots is initially disconnected, provided that a small amount of these robots are arranged to form a TuringMobile. In the special case of the plane, a basic TuringMobile of only three robots is sufficient.

2012 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, I.2.9 Robotics

Keywords and phrases Mobile Robots, Turing Machine, Real RAM

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Related Version Full Version on ArXiv: https://arxiv.org/pdf/1709.08800

[1] [Contact Author. Address: 1-50-D-21 Asahidai, Nomi City, Ishikawa Prefecture 923-1211, Japan. Phone: +81 80-8691-6839.]

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

1.1 Framework and Background.

The investigations of systems of autonomous mobile robots have long moved outside the boundaries of the engineering, control, and AI communities. Indeed, the computational and complexity issues arising in such systems are important research topics within theoretical computer science, especially in distributed computing. In these theoretical investigations, the robots are usually viewed as computational entities that live in a metric space, typically $\mathbb{R}^2$ or $\mathbb{R}^3$, in which they can move. Each robot operates in "Look-Compute-Move" (LCM) cycles: it observes its surroundings, it computes a destination within the space based on what it sees, and it moves toward the destination. The only means of interaction between robots are observations and movements: that is, communication is *stigmergic*. The robots, identical and outwardly indistinguishable, are *oblivious*: when starting a new cycle, a robot has no memory of its activities (observations, computations, and moves) from previous cycles ("every time is the first time").

There have been intensive research efforts on the computational issues arising with such robots, and an extensive literature has been produced in particular in regard to the important class of *Pattern Formation* problems [8, 20, 22, 23, 29, 30] as well as for *Gathering* [1, 2, 4, 9, 10, 12, 13, 15, 11, 21, 25] and *Scattering* [6, 24]; see also [7, 14, 31]. The goal of the research has been to understand the minimal assumptions needed for a team (or swarm) of such robots to solve a given problem, and to identify the impact that specific factors have on feasibility and hence computability.

The most important factor is the power of the adversarial scheduler that decides when each activity of each robot starts and when it ends. The main adversaries (or "environments") considered in the literature are: *synchronous*, in which the computation cycles of all active robots are synchronized, and at each cycle either all (in the fully synchronous case) or a subset (in the semi-synchronous case) of the robots are activated, and *asynchronous*, where computation cycles are not synchronized, each activity can take a different and unpredictable amount of time, and each robot can be independently activated at each time instant.

An important factor is whether a robot moving toward a computed destination is guaranteed to reach it (*rigid* robot), or it can be stopped on the way (*non-rigid*) at a point decided by an adversary. In all the above cases, the power of the adversaries is limited by some basic fairness assumption. All the existing investigations have concentrated on the study of (a-)synchrony, several on the impact of rigidity, some on other relevant factors such as agreement on local coordinate systems or on their orientation, etc.; for a review, see [19].

From a computational point of view, there is another crucial factor: the visibility range of the robots, that is, how much of the surrounding space they are able to observe in a Look operation. In this regard, two basic settings are considered: *unlimited visibility*, where the robots can see the entire space (and thus all other robots), and *limited visibility*, when the robots have a fixed visibility radius. While the investigations on (a-)synchrony and rigidity have concentrated on all aspects of those assumptions, this is not the case with respect to visibility. In fact, almost all research has assumed unlimited visibility; few exceptions are the algorithms for Convergence [4], Gathering [16, 17, 21], and Near-Gathering [25] when the visibility range of the robot is limited. The unlimited visibility assumption clearly greatly simplifies the computational universe under investigation; at the same time, it neglects the more general and realistic one, which is still largely unknown.

Let us also stress that, in the existing literature, all results on oblivious robots are for $\mathbb{R}^1$ and $\mathbb{R}^2$; the only exception is the recent result on plane formation in $\mathbb{R}^3$ by semi-synchronous rigid robots with unlimited visibility [31]. No results exist for robots in higher dimensions.

1.2 Contributions.

In this paper we contribute several constructive insights on the computational universe of oblivious robots with limited visibility, especially asynchronous non-rigid ones, in any dimension.

TuringMobile

The first and main contribution is the design of a "moving Turing Machine" made solely of asynchronous oblivious non-rigid robots in $\mathbb{R}^m$ with limited visibility, for any $m \geq 2$. More precisely, we show how to arrange $3m + 3k$ identical non-rigid oblivious robots in $\mathbb{R}^m$ with a visibility radius of $V + \varepsilon$ (for any $\varepsilon > 0$) and how to program them so that they can collectively behave as a single rigid robot in $\mathbb{R}^m$ with k persistent registers and visibility radius V would. This team of identical robots is informally called a *TuringMobile*. We obtain this result by using as fundamental construction a basic component, which is able to move in $\mathbb{R}^2$ while storing and updating a single real number. Interestingly, we show that 3 agents are necessary and sufficient to build such a machine. The TuringMobile will then be built by arranging multiple copies of this basic component. Notably, the robots that constitute a TuringMobile need only be able to compute arithmetic operations and square roots.

A TuringMobile is a powerful construct that, once deployed in a swarm of robots, can act as a rigid leader with persistent memory, allowing the swarm to overcome many handicaps imposed by obliviousness, limited visibility, and asynchrony. As examples we present a variety of applications in $\mathbb{R}^m$, with $m \geq 2$.

There is a limitation to the use of a TuringMobile when deployed in a swarm of robots. Namely, the TuringMobile must be always recognizable (e.g., by its unique shape) so that other robots cannot interfere by moving too close to the machine, disrupting its structure. This limitation can be overcome when the robots of the TuringMobile are visibly distinguishable from the others. However, this requirement is not necessary for all applications, but is only required when we want to perfectly simulate a rigid robot with memory.

We remark that we do not discuss how robots can self-assemble into a TuringMobile. We only focus on how the machine can be designed when we can freely arrange some robots. In the case of robots with unlimited visibility, a TuringMobile can be self-assembled, provided that the initial configuration of the robots is asymmetric. In the case of limited visibility, self-assembling a TuringMobile is more delicate. However, we argue that assuming the presence of our TuringMobile is analogous to assuming the presence of a certain number of distinguished robots: self-assembling a TuringMobile is possible if these distinguished robots are all visible to each other and arranged in an asymmetric configuration.

Applications

We propose several applications of our TuringMobile. First of all, the TuringMobile can explore and search the space. We then show how it can be employed to solve the long-standing open problem of (Near-)Gathering with limited visibility in spite of an asynchronous non-rigid scheduler and disagreement on the axes, a problem still open without a TuringMobile. Interestingly, the presence of the TuringMobile allows Gathering to be done even if the initial visibility graph is disconnected (this does not change the fact that there are cases in which Gathering is impossible, as remarked in [4, 21]). Finally we show how the arbitrary Pattern Formation problem can be solved under the same conditions (asynchrony, limited visibility, possibly disconnected visibility graph, etc.).

The paper is organized as follows: In Section 2 we give formal definitions, introducing mobile robots with or without memory as *oracle semi-oblivious real RAMs*. In Section 3 we illustrate our implementation of the TuringMobile. In Section 4 we show how to apply the TuringMobile to solve fundamental problems. Due to space constraints, the proof of correctness of our TuringMobile implementation, several technical parts of the paper, and additional figures can be found in the full paper [18].

2 Definitions and Preliminaries

2.1 Oracle Semi-Oblivious Real RAMs

Real random-access machines. A *real RAM*, as defined by Shamos [26, 28], is a random-access machine [3] that can operate on real numbers. That is, instead of just manipulating and storing integers, it can handle arbitrary real numbers and do infinite-precision operations on them. It has a finite set of internal *registers* and an infinite ordered sequence of *memory cells*; each register and each memory cell can hold a single real number, which the machine can modify by executing its program.[2]

A real RAM's instruction set contains at least the four arithmetic operations, but it may also contain k-th roots, trigonometric functions, exponentials, logarithms, and other analytic functions, depending on the application. The machine can also compare two real numbers and branch depending on which one is larger.

The initial contents of the memory cells are the *input* of the machine (we stipulate that only finitely many of them contain non-zero values), and their contents when the machine halts are its *output*. So, each program of a real RAM can be viewed as a partial function mapping tuples of reals into tuples of reals.

Oracles and semi-obliviousness. We introduce the *oracle semi-oblivious real RAM*, which is a real RAM with an additional "ASK" instruction. Whenever this instruction is executed, the contents of all the memory cells are replaced with new values, which are a function of the numbers stored in the registers.

In other words, the machine can query an external oracle by putting a question in its k registers in the form of k real numbers. The oracle then reads the question and writes the answer in the machine's memory cells, erasing all pre-existing data. The term "semi-oblivious" comes from the fact that, every time the machine invokes the oracle, it "forgets" everything it knows, except for the contents of the registers, which are preserved.[3]

▶ Remark. In spite of their semi-obliviousness, these real RAMs with oracles are at least as powerful as Turing Machines with oracles.

2.2 Mobile Robots as Real RAMs

Mobile robots. Our oracle semi-oblivious real RAM model can be reinterpreted in the realm of *mobile robots*. A mobile robot is a computational entity that lives in a metric space, typically $\mathbb{R}^2$ or $\mathbb{R}^3$. It can observe its surroundings and move within the space based on what it sees. The same space may be populated by several mobile robots and static objects.

---

[2] Nonetheless, the constant operands in a real RAM's program cannot be arbitrary real numbers, but have to be integers.

[3] Observe that, in general, the machine cannot salvage its memory by encoding its contents in the registers: since its instruction set has only analytic functions, it cannot injectively map a tuple of arbitrary real numbers into a single real number.

To compute its next destination point, a mobile robot executes a real RAM program with input a representation of its local view of the space. After moving, its entire memory is erased, but the content of its k registers is preserved. Then it makes a new observation; from the observation data and the contents of the registers, it computes another destination point, and so on. If $k = 0$, the mobile robot is said to be *oblivious*.

The actual movement of a mobile robot is controlled by an external *scheduler*. The scheduler decides how fast the robot moves toward its destination point, and it may even interrupt its movement before the destination point is reached. If the movement is interrupted midway, the robot makes the next observation from there and computes a new destination point as usual. The robot is not notified that an interruption has occurred, but it may be able to infer it from its next observation and the contents of its registers. For fairness, the scheduler is only allowed to interrupt a robot after it has covered a distance of at least δ in the current movement, where δ is a positive constant. This guarantees, for example, that if a robot keeps computing the same destination point, it will reach it in a finite number of iterations. If $\delta = \infty$, the robot always reaches its destination, and is said to be *rigid*.

Mobile robots, revisited. A mobile robot in $\mathbb{R}^m$ with k registers can be modeled as an oracle semi-oblivious real RAM with $2m + k + 1$ registers, as follows.

- m *position registers* hold the absolute coordinates of the robot in $\mathbb{R}^m$.
- m *destination registers* hold the destination point of the robot, expressed in its local coordinate system.
- 1 *timestamp register* contains the time of the robot's last observation.
- k *true registers* correspond to the registers of the robot.

As the RAM's execution starts, it ignores its input, erases all its registers, and executes an "ASK" instruction. The oracle then fills the RAM's memory with the robot's initial position p, the time t of its first observation, and a representation of the geometric entities and objects surrounding the robot, as seen from p at time t.

The RAM first copies p and t in its position registers and timestamp register, respectively. Then it executes the program of the mobile robot, using its true registers as the robot's registers and adding $m + 1$ to all memory addresses. This effectively makes the RAM ignore the values of p and t, which indeed are not supposed to be known to the mobile robot.

When the robot's program terminates, the RAM's memory contains the output, which is the next destination point p', expressed in the robot's coordinate system. The RAM copies p' into its destination registers, and the execution jumps back to the initial "ASK" instruction.

Now the oracle reads p, p', and t from the RAM's registers (it ignores the true registers), converts p' in absolute coordinates (knowing p and the orientation of the local coordinate system of the robot) and replies with a new position p'', a timestamp $t' > t$, and observation data representing a snapshot taken from p'' at time t'. To comply with the mobile robot model, p'' must be on the segment pp', such that either $p'' = p'$ or $\overline{pp''} \geq \delta$. The execution then proceeds in the same fashion, indefinitely.

Note that in this setting the oracle represents the scheduler. The presence of a timestamp in the query allows the oracle to model dynamic environments in which several independent robots may be moving concurrently (without a timestamp, two observations from the same point of view would always be identical).

Snapshots and limited visibility. In the mobile robot model we consider in this paper, an observation is simply an instantaneous *snapshot* of the environment taken from the robot's position. In turn, each entity and object that the robot can see is modeled as a dimensionless point in $\mathbb{R}^m$. A mobile robot has a positive *visibility radius* V: it can see a point in $\mathbb{R}^m$ if and only if it is located at distance at most V from its current position. If $V = \infty$, the robot is said to have *unlimited visibility*.

As we hinted at earlier in this section, a mobile robot has its own local reference system in which all the coordinates of the objects in its snapshots are expressed. The origin of a robot's local coordinate system always coincides with the robot's position (hence it follows the robot as it moves), and its orientation and handedness are decided by the scheduler (and remain fixed). Different mobile robots may have coordinate systems with a different orientation or handedness. (However, when two robots have the same visibility radius, they also implicitly have the same unit of distance.)

So, a snapshot is just a (finite) list of points, each of which is an m-tuple of real numbers.

Simulating memory and rigidity. The main contribution of this paper, loosely speaking, is a technique to turn non-rigid oblivious robots into rigid robots with persistent memory, under certain conditions. More precisely, if $3m + 3k$ identical non-rigid oblivious robots in $\mathbb{R}^m$ with a visibility radius of $V + \varepsilon$ (for any $\varepsilon > 0$) are arranged in a specific pattern and execute a specific algorithm, they can collectively act in the same way as a single rigid robot in $\mathbb{R}^m$ with $k > 0$ persistent registers and visibility radius V would. This team of identical robots is informally called a *TuringMobile*.

We stress that the robots of a TuringMobile are *asynchronous*, that is, the scheduler makes them move at independent arbitrary speeds, and each robot takes the next snapshot an arbitrary amount of time after terminating each move. The robots are also *anonymous*, in that they are indistinguishable from each other, and they all execute the same program.

Although our technique is fairly general and has a plethora of concrete applications (some are discussed in Section 4), a "perfect simulation" is achieved only under additional conditions on the scheduler or on the environment (see Section 3.2).

3 Implementing the TuringMobile

3.1 Basic Implementation

We will first describe how to construct a basic version of the TuringMobile with just three oblivious non-rigid robots in $\mathbb{R}^2$. This TuringMobile can remember a single real number and rigidly move in the plane by fixed-length steps: its layout is sketched in Figure 1. In Section 3.2, we will show how to combine several copies of this basic machine to obtain a full-fledged TuringMobile.

Position at rest. The elements of the basic TuringMobile are three: a *Commander* robot, a *Number* robot, and a *Reference* robot, located in C, N, and R, respectively. These robots have the same visibility radius of $V + \varepsilon$, where $\varepsilon \ll V$, and there is always a disk of radius ε containing all three of them. When the machine is "at rest", $\angle NRC$ is a right angle, the distance between C and R is some fixed value $d \ll \varepsilon$, and the distance between R and N is approximately $2d$. More precisely, N lies on a segment QQ' of length λ, where $\lambda \ll d$ is some fixed value, such that Q has distance $2d - \lambda/2$ from R and Q' has distance $2d + \lambda/2$ from R.

Representing numbers. The distance between the Reference robot and the Number robot when the TuringMobile is at rest is a representation of the real number r that the machine is currently storing. One possible technique is to encode the number r as $\overline{RN} = 2d + \arctan(r) \cdot \lambda/\pi$ and to decode it as $r = \tan\left((\overline{RN} - 2d) \cdot \pi/\lambda\right)$. However, there are also more complicated methods that use only arithmetic functions (see the full paper [18]).

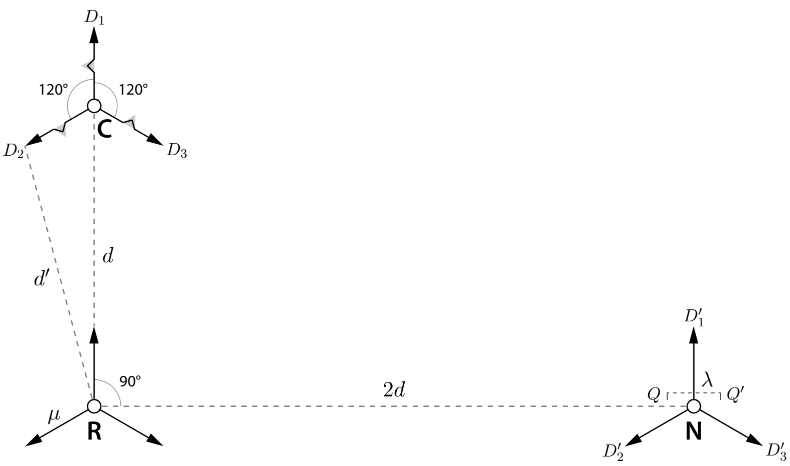

Figure 1 Basic TuringMobile at rest, not drawn to scale (μ and λ should be smaller)

Movement directions. The Commander's role is to decide in which direction the machine should move next, and to initiate the movement. When the machine is at rest, the Commander may choose among three possible *final destinations*, labeled D_1, D_2, and D_3 in Figure 1. The segments CD_1, CD_2, and CD_3 all have the same length μ, with $\lambda \ll \mu \ll d$, and form angles of 120° with one another, in such a way that D_1 is collinear with R and C.

Around the center of each segment CD_i there is a *midway triangle* τ_i, drawn in gray in Figure 1. This is an isosceles triangle of height λ whose base lies on CD_i and has length λ as well. When the Commander decides that its final destination is D_i, it moves along the segment CD_i, but it takes a detour in the midway triangle τ_i, as we will explained shortly.

Structure of the algorithm. Algorithm 1 is the program that each element of the basic TuringMobile executes every time it computes its next destination point.

Since the robots are anonymous, they first have to determine their roles, i.e., who is the Commander, etc. (line 1 of the algorithm). We make the assumption that there exists a disk of radius ε containing only the TuringMobile (close to its center) and no other robot. Using the fact that the two closest robots must be the Commander and the Reference robot and that the two farthest robots must be the Commander and the Number robot, it is then easy to determine who is who (these properties will be preserved throughout the execution, as proved in the full paper [18]).

Once it has determined its role, each robot executes a different branch of the algorithm (cf. lines 2, 13, and 23). The general idea is that, when the Commander realizes that the machine is in its rest position, it decides where to move next, i.e., it chooses a final destination D_i. This choice is based on the number r stored in the machine's "memory" (i.e., the number encoded by $\overline{RN}$), the relative positions of the visible robots external to the machine, and also on the application, i.e., the specific program that the TuringMobile is executing.

When the Commander has decided its final destination D_i, the entire machine moves by the vector $\overrightarrow{CD_i}$, and the Number robot also updates its distance from the Reference robot to represent a different real number r'. Again, this number is computed based on the number r the machine was previously representing, the relative positions of the visible robots external to the machine, and the specific program: in general, the new distance between N and Q is a function f of the old distance. When all this is done, the machine is in its rest position again, so the Commander chooses a new destination, and so on, indefinitely.

Algorithm 1 Basic TuringMobile in $\mathbb{R}^2$

1: Identify *Commander*, *Number*, *Reference* (located in C, N, R, respectively)
2: **if** I am *Commander* **then**
3: Compute *Virtual Commander* C' (based on R and N) and points A_i, S_i, S'_i, B_i, D_i
4: **if** I am in C' **then** Choose final destination D_i and move to A_i
5: **else if** $\exists i \in \{1,2,3\}$ s.t. I am on segment $C'A_i$ but not in A_i **then** Move to A_i
6: **else if** $\exists i \in \{1,2,3\}$ s.t. I am in A_i **then**
7: Move to point P on segment $S_iS'_i$ such that $\overline{PS_i} = f(\overline{NQ})$
8: **else if** $\exists i \in \{1,2,3\}$ s.t. I am in triangle $A_iS_iS'_i$ but not on segment $S_iS'_i$ **then**
9: Move to the intersection of segment $S_iS'_i$ with the extension of line A_iC
10: **else if** $\exists i \in \{1,2,3\}$ s.t. I am on $S_iS'_i$ **and** $\overline{NQ} = \overline{CS_i}$ **then** Move to B_i
11: **else if** $\exists i \in \{1,2,3\}$ s.t. I am in triangle $B_iS_iS'_i$ but not in B_i **then** Move to B_i
12: **else if** $\exists i \in \{1,2,3\}$ s.t. I am on segment B_iD_i but not in D_i **then** Move to D_i
13: **else if** I am *Number* **then**
14: **if** $\overline{CR} = d + \mu$ **or** $\overline{CR} = d'$ **then**
15: Compute *Virtual Commander* C' (based on C and R) and points D'_i
16: **if** $\overline{CR} = d + \mu$ **and** I am not in D'_1 **then** Move to D'_1
17: **else if** $\overline{CR} = d'$ **and** $\angle NRC > 90°$ **and** I am not in D'_2 **then** Move to D'_2
18: **else if** $\overline{CR} = d'$ **and** $\angle NRC < 90°$ **and** I am not in D'_3 **then** Move to D'_3
19: **else**
20: Compute *Virtual Commander* C' (based on R and N) and points S_i, S'_i
21: **if** $\exists i \in \{1,2,3\}$ s.t. C is on segment $S_iS'_i$ **then**
22: Move to point P on segment QQ' such that $\overline{PQ} = \overline{CS_i}$
23: **else if** I am *Reference* **then**
24: **if** *Commander* and *Number* are not tasked to move (based on the above rules) **then**
25: γ = circle centered in C with radius d
26: γ' = circle with diameter CN
27: Move to the intersection of γ and γ' closest to R

Coordinating movements. Note that it is not possible for all three robots to translate by $\overrightarrow{CD_i}$ at the same time, because they are non-rigid and asynchronous. If the scheduler stops them at arbitrary points during their movement, after the structure of the machine has been destroyed, they will be incapable of recovering all the information they need to resume their movement (recall that they are oblivious and they have to compute a destination point from scratch every time).

To prevent this, the robots employ various coordination techniques. First the Commander moves to the middle triangle τ_i, and precisely to its base vertex A_i, as shown in Figure 2(a) (cf. line 5 of Algorithm 1). Then it positions itself on the altitude $S_iS'_i$, in such a way as to indicate the new number r' that the machine is supposed to represent. That is, the Commander picks the point on $S_iS'_i$ at distance $f(\overline{NQ})$ from S_i (lines 6 and 7). Even if it is stopped by the scheduler before reaching such a point, it can recover its destination by drawing a ray from A_i to its current position and intersecting it with $S_iS'_i$ (lines 8 and 9).

When the Commander has reached $S_iS'_i$, it waits to let the Number robot adjust its position on the segment QQ' to match that of the Commander on $S_iS'_i$, as in Figure 2(b) (lines 21 and 22). This effectively makes the Number robot represent the new number r'. Note that the Number robot can do this even if it is stopped by the scheduler several times during its march, because the Commander keeps reminding it of the correct value of r': since r' depends on the old number r, the Number robot would be unable to re-compute r' after it has forgotten r. Once the Number robot has reached the correct position on QQ', the Commander starts moving again (line 10) and finally reaches D_i while the other robots wait, as in Figure 2(c) (lines 11 and 12).

When the Commander has reached D_i, the Number robot realizes it and makes the corresponding move (lines 14–18) while the other two robots wait. The destination point of the Number robot is D'_i, as shown in Figure 1. Finally, when the Number robot is in D'_i, the Reference robot realizes it and makes the final move to bring the TuringMobile back into a rest position (lines 23–27).

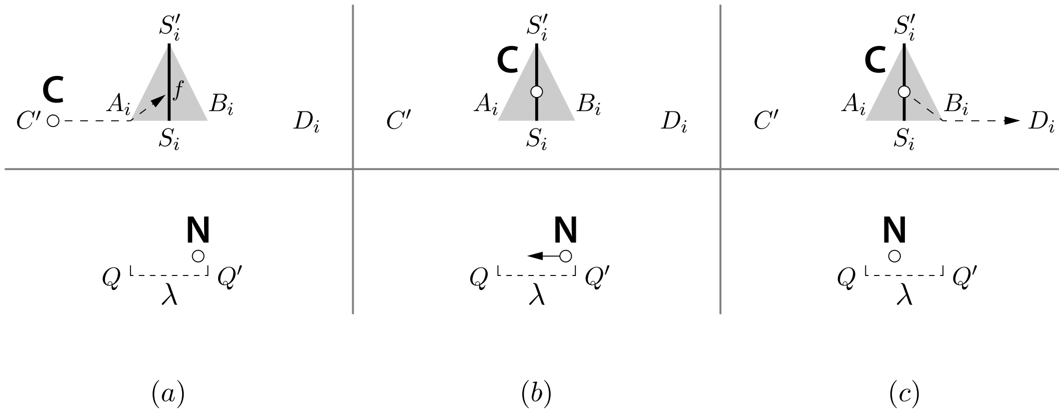

Figure 2 Coordinated movement of the Commander and the Number robot

Computing the Virtual Commander. After the Commander has left its rest position and is on its way to D_i, the TuringMobile loses its initial shape, and identifying the D_i's and the midway triangles becomes non-trivial. So, the robots try to guess where the Commander's original rest position may have been by computing a point called the *Virtual Commander* C'.

Assuming that the Reference and Number robots have not moved from their rest positions, the Virtual Commander is easily computed: draw the line ℓ through R perpendicular to RN; then, C' is the point on ℓ at distance d from R that is closest to C. Once we have C', we can construct the points D_i with respect to C' (in the same way as we did in Figure 1 with respect to C). This technique is used by Algorithm 1 at lines 3 and 20.

In the special case where the Commander has reached its final destination D_i and the Reference robot has not moved from its rest position (but perhaps the Number robot has moved), the Virtual Commander can also be computed. This situation is recognized because the distance between the Commander and the Reference robot is either maximum (i.e., $d+\mu$) or minimum (i.e., $d' = \sqrt{d^2 + \mu^2 - d\mu}$), as Figure 1 shows. If the distance is maximum, then C must coincide with D_1; otherwise, C coincides with D_2 (if the angle $\angle NRC$ is obtuse) or D_3 (if the angle $\angle NRC$ is acute). Since we know the position of R and one of the D_i's, it is then easy to determine the other D_i's. This technique is used at line 15.

The Reference robot's behavior. To know when it has to start moving, the Reference robot executes Algorithm 1 from the perspective of the Commander and the Number robot: if neither of them is supposed to move, then the Reference robot starts moving (line 24).

We have seen that the Number robot can determine its destination D'_i solely by looking at the positions of C and R, which remain fixed as it moves. For the Reference robot the destination point is not as easy to determine, because the distance between C and N varies depending on what number is stored in the TuringMobile.

However, the Reference robot knows that its move must put the TuringMobile in a rest position. The condition for this to happen is that its destination point be at distance d from C (line 25) and form a right angle with C and N (line 26). There are exactly two such points in the plane, but one of them has distance much greater than μ from R, and hence the Reference robot will pick the other (line 27).

As the Reference robot moves toward such a point, all the above conditions must be preserved, due to the asynchronous and non-rigid nature of the robots. This is not a trivial requirement, and a proof that it is indeed fulfilled is in the full paper [18].

3.2 Complete Implementation

We have shown how to implement a basic component of the TuringMobile in $\mathbb{R}^2$ consisting of three robots: a Commander, a Number, and a Reference. The basic component is able to rigidly move by a fixed distance μ in three fixed directions, 120° apart from one another. It can also store and update a single real number.

Planar layout. We can obtain a full-fledged TuringMobile in $\mathbb{R}^2$ by putting several tiny copies of the basic component side by side. For the machine to work, we stipulate that there exists a disk of radius σ that contains all the robots constituting the TuringMobile and no extraneous robot, where $\sigma \ll \varepsilon$. The distance between two consecutive basic components of the TuringMobile is roughly s, where $d \ll s \ll \sigma$. This makes it easy for the robots to tell the basic components apart and determine the role of each robot within its basic component.

Since a basic component of the TuringMobile is a scalene triangle, which is chiral, all its members implicitly agree on a clockwise direction even if they have different handedness. Similarly, all robots in the Turing Mobile agree on a "leftmost" basic component, whose Commander is said to be the *Leader* of the whole machine.

Coordinated movements. All the basic components of the TuringMobile are always supposed to agree on their next move and proceed in a roughly synchronous way. To achieve this, when all the basic components are in a rest position, the Leader decides the next direction among the three possible, and executes line 4 of Algorithm 1. Then all the other Commanders see where the Leader is going, and copy its movement.

When all the Commanders are in their respective A_i's, they execute line 7 of the algorithm, and so on. At any time, each robot executes a line of the algorithm only if all its homologous robots in the other basic components of the TuringMobile are ready to execute that line or have already executed it; otherwise, it waits. When the last Reference robot has completed its movement, the machine is in a rest position again, and the coordinated execution repeats with the Leader choosing another direction, etc.

Simulating a non-oblivious rigid robot. Let a program for a rigid robot $\mathcal{R}$ in $\mathbb{R}^2$ with k persistent registers and visibility radius V be given. We want the TuringMobile described above to act as $\mathcal{R}$, even though its constituting robots are non-rigid and oblivious.

Our TuringMobile consists of $2 + k$ basic components, each dedicated to memorizing and updating one real number. These $2 + k$ numbers are the x coordinate and the y coordinate of the destination point of $\mathcal{R}$ and the contents of the k registers of $\mathcal{R}$. We will call the first two numbers the *x variable* and the *y variable*, respectively.

When the TuringMobile is in a rest position, its x and y variables represent the coordinates of the destination point of $\mathcal{R}$ relative to the Leader of the machine. Whenever the TuringMobile moves by μ in some direction, these values are updated by subtracting the components of an appropriate vector of length μ from them. Of course, this update is computed by the Commanders of the first two basic components of the machine, which communicate it to their respective Number robots, as explained in Section 3.1.

Let P be the destination point of $\mathcal{R}$. Since the TuringMobile can only move by vectors of length μ in three possible directions, it may be unable to reach P exactly. So, the Leader always plans the next move trying to reduce its distance from P until this distance is at most 2σ (this is possible because $\mu \ll d \ll \sigma$).

When the Leader is close enough to P, it "pretends" to be in P, and the TuringMobile executes the program of $\mathcal{R}$ to compute the next destination point. Recall that the visibility radius of $\mathcal{R}$ is V, and that of the robots of the TuringMobile is $V + \varepsilon$. Since $\sigma \ll \varepsilon$, each member of the TuringMobile can therefore see everything that would be visible to $\mathcal{R}$ if it were in P, and compute the output of the program of $\mathcal{R}$ independently of the other members. The only thing it should do when it executes the program of $\mathcal{R}$ is subtract the values of the x and y variables to everything it sees in its snapshot, discard whatever has distance greater than V from the center, and of course discard the robots of the TuringMobile and replace them with a single robot in the center. Then, the robots that are responsible for updating the x and y variables add the relative coordinates of the new destination point of $\mathcal{R}$ to these variables. Similarly, the robots responsible for updating the k registers of $\mathcal{R}$ do so.

Restrictions. The above TuringMobile correctly simulates $\mathcal{R}$ under certain conditions. The first one is that, if all robots are indistinguishable, then no robot extraneous to the TuringMobile may get too close to it (say, within a distance of σ of any of its members). This kind of restriction cannot be dispensed with: whatever strategy a team of oblivious robots employs to simulate a single non-oblivious robot's behavior is bound to fail if extraneous robots join the team creating ambiguities between its members. Nevertheless, the restriction can be removed if the members of a TuringMobile are distinguishable from all other robots.

Another difficulty comes from the fact that, if the TuringMobile is made of more than one basic component and its Commanders are all in their respective A_i's and ready to update the values represented by the machine, they may get their screenshots at different times, due to asynchrony. If the environment moves in the meantime, the screenshots they get are different, and this may cause the machine to compute an incorrect destination point or put inconsistent values in its simulated registers.

There are several possible solutions to this problem: we will only mention two trivial ones. We could assume the Commanders to be *synchronous*, that is, make the scheduler activate them in such a way that all of them take their screenshots at the same time. This way, all Commanders get compatible screenshots and compute consistent outputs. Another possible solution is to make the TuringMobile operate in an environment where everything else is static, i.e., no moving entities are present other than the TuringMobile's members.

We stress that these restrictions make sense if a perfect simulation of $\mathcal{R}$ is saught. As we will see in Section 4, there are several other applications of the TuringMobile technique where no such restriction is required.

Higher dimensions. Let us now generalize the above construction of a planar TuringMobile to $\mathbb{R}^m$, for any $m \geq 2$. We start with the same TuringMobile $\mathcal{M}$ with $2+k$ basic components laid out on a plane $\gamma \subset \mathbb{R}^m$. Since $\mathcal{M}$ has only two basic components for the x and y variables, we will add $m-2$ basic components to it, positioned as follows.

Let vectors v_1 and v_2 be two orthonormal generators of γ, and let us complete $\{v_1, v_2\}$ to an orthonormal basis $\{v_1, v_2, \ldots, v_m\}$ of $\mathbb{R}^m$. Now, for all $i \in \{3, 4, \ldots, m\}$, we make a copy of the basic component of $\mathcal{M}$ containing the Leader, we translate it by $s \cdot v_i$, and we add it to the TuringMobile (s is the same value used in the construction of the planar TuringMobile at the beginning of Section 3.2). Note that the Leader of this new TuringMobile $\mathcal{M}'$ is still easy to identify, as well as the plane γ when $\mathcal{M}'$ is at rest.

Clearly, m basic components allow the machine to record a destination point in $\mathbb{R}^m$, as opposed to $\mathbb{R}^2$. Additionally, the positions of the basic components with respect to γ give the machine an m-dimensional sense of direction (see the full paper [18] for further details).

▶ **Theorem 1.** *Under the aforementioned restrictions, a rigid robot in $\mathbb{R}^m$ with k persistent registers and visibility radius V can be simulated by a team of $3m + 3k$ non-rigid oblivious robots in $\mathbb{R}^m$ with visibility radius $V + \varepsilon$.* ◀

4 Applications

In this section we discuss some applications of the TuringMobile. We also prove that the basic TuringMobile constructed in Section 3.1 is minimal, in the sense that no smaller team of oblivious robots can accomplish the same tasks.

4.1 Exploring the Plane

The first elementary task a basic TuringMobile in $\mathbb{R}^2$ can fulfill is that of *exploring* the plane. The task consists in making all the robots in the TuringMobile see every point in the plane in the course of an infinite execution. We first assume that the three members of the TuringMobile are the only robots in the plane. Later in this section, we will extend our technique to other types of scenarios and more complex tasks.

▶ **Theorem 2.** *A basic TuringMobile consisting of three robots in $\mathbb{R}^2$ can explore the plane.*

Proof. Recall that a basic TuringMobile can store a single real number r and update it at every move as a result of executing a real RAM program with input r. In particular, the TuringMobile can count how many times it has moved by simply starting its execution with $r = 0$ and computing $r := r + 1$ at each move.

Moreover, the Commander chooses the direction of the next move (in the form of a point D_i, see Figure 1) by executing another real RAM program with input r. If r is an integer, the Commander can therefore compute any Turing-computable function on r, and so it can decide to move to D_1 the first time, then to D_2 twice, then to D_3 three times, to D_1 four times, and so on. This pattern of moves is illustrated in Figure 3, and of course it results in the exploration of the plane, because the visibility radius of the robots is much greater than the step μ. ◀

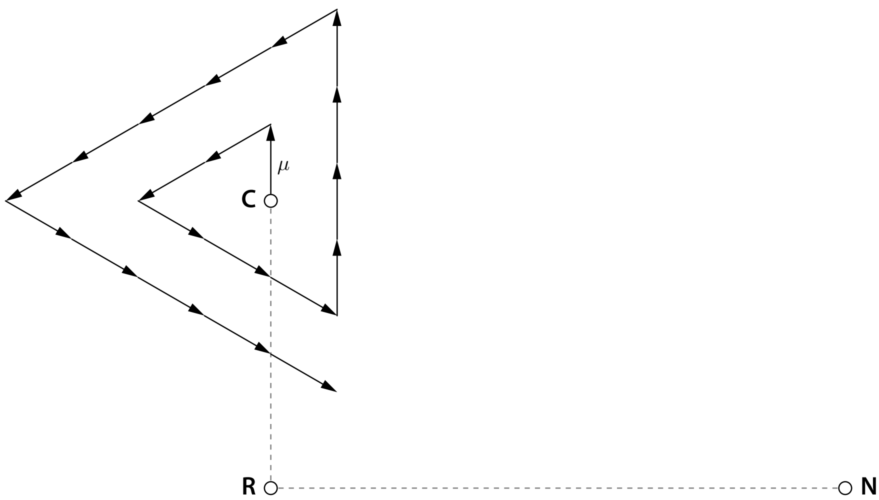

Figure 3 Exploration of the plane by a basic TuringMobile

4.2 Minimality of the Basic TuringMobile

We can use the previous result to prove indirectly that our basic TuringMobile design is minimal, because no team of fewer than three oblivious robots in $\mathbb{R}^2$ can explore the plane.

▶ **Theorem 3.** *If only one or two oblivious identical robots with limited visibility are present in $\mathbb{R}^2$, they cannot explore the plane, even if the scheduler lets them move synchronously and rigidly.*

Proof. Assume that a single oblivious robot is given in $\mathbb{R}^2$. Since it always gets the same snapshot, it always computes the same destination point in its local coordinate system, and so it always translates by the same vector. As a consequence, it just moves along a straight ray, and therefore it cannot explore the plane.

Let two oblivious robots be given, and suppose that their local coordinate systems are oriented symmetrically. Whether the robots see each other or not, if they take their snapshots simultaneously, they get identical views, and so they compute destination points that are symmetric with respect to O. If they keep moving synchronously and rigidly, O remains their midpoint. So, if the robots have visibility radius V, they see each other if and only if they are in the circle γ of radius $V/2$ centered in O.

Let O be the midpoint of the robots' locations, and consider a Cartesian coordinate system with origin O. Without loss of generality, when the robots do not see each other, they move by vectors $(1, 0)$ and $(-1, 0)$, respectively. Let ξ be the half-plane $y \geq V$, and observe that ξ lies completely outside γ.

It is obvious that the robots cannot explore the entire plane if neither of them ever stops in ξ. The first time one of them stops in ξ, it takes a snapshot from there, and starts moving parallel to the x axis, thus never seeing the other robot again, and never leaving ξ. Of course, following a straight line through ξ is not enough to explore all of it. ◀

4.3 Near-Gathering with Limited Visibility

The exploration technique can be applied to several more complex problems. The first we describe is the *Near-Gathering* problem, in which all robots in the plane must get in the same disk of a given radius ε (without colliding) and remain there forever. It does not matter if the robots keep moving, as long as there is a disk of radius ε that contains them all.

It is clear that solving this problem from every initial configuration is not possible, and hence some restrictive assumptions have to be made. The usual assumption is that the initial visibility graph of the robots be connected [21, 25]. Here we make a different assumption: there are three robots that form a basic TuringMobile somewhere in the plane, and each robot not in the TuringMobile has distance at least ε from all other robots. (Actually we could weaken this assumption much more, but this simple example is good enough to showcase our technique.)

Say that all robots in the plane have a visibility radius of $V \gg \varepsilon$, and that the TuringMobile moves by $\mu \ll \epsilon$ at each step. The TuringMobile starts exploring the plane as above, and it stops in a rest position as soon as it finds a robot whose distance from the Commander is smaller than $V/2$ and greater than ε. On the other hand, if a robot is not part of the TuringMobile, it waits until it sees a TuringMobile in a rest position at distance smaller than $V/2$. When it does, it moves to a designated area $\mathcal{A}$ in the proximity of the Commander. Such an area has distance at least $3d$ from the Commander, so no confusion can arise in the identification of the members of the TuringMobile. If several robots are eligible to move to $\mathcal{A}$, only one at a time does so: note that the layout of the TuringMobile itself gives an implicit total order to the robots around it. Observe that the robots cannot form a second TuringMobile while they move to $\mathcal{A}$: in order to do so, two of them would have to move to $\mathcal{A}$ at the same time and get close enough to a third robot. Once they enter $\mathcal{A}$, the robots position themselves on a segment much shorter than d, so they cannot possibly be mistaken for a TuringMobile.

Once the eligible robots have positioned themselves in $\mathcal{A}$, the TuringMobile resumes its exploration of the plane, and the robots in $\mathcal{A}$ copy all its movements. Now, if the total number of robots in the plane is known, the TuringMobile can stop as soon as all of them have joined it. Otherwise, the machine simply keeps exploring the plane forever, eventually collecting all robots. In both cases, the Near-Gathering problem is solved.

4.4 Pattern Formation with Limited Visibility

Suppose the robots are exactly n, and they are tasked to form a given *pattern* consisting of a multiset of n points: this is the *Pattern Formation* problem, which becomes the *Gathering* problem in the special case in which the points are all coincident. For this problem, it does not matter where the pattern is formed, nor does its orientation or scale.

Again, the Pattern Formation problem is unsolvable from some initial configurations, so we make the same assumptions as with the Near-Gathering problem. The algorithm starts by solving the Near-Gathering problem as before. The only difference is that now there is a second tiny area $\mathcal{B}$, attached to $\mathcal{A}$ (and still far enough from the TuringMobile), which the robots avoid when they join $\mathcal{A}$. This is because this second area will be used to form the pattern.

Since n is known, the TuringMobile knows when it has to interrupt the exploration of the plane because all robots have already been found. At this point, the robots switch algorithm: one by one, they move to $\mathcal{B}$ and form the pattern. This task is made possible by the presence of the TuringMobile, which gives an implicit order to all robots, and also unambiguously defines an embedding of the pattern in $\mathcal{B}$. So, each robot is implicitly assigned one point in $\mathcal{B}$, and it moves there when its turn comes.

If $n = 3$ or $n = 4$, there are uninteresting ad-hoc algorithms to do this: so, let us assume that $n \geq 5$. The first to move are the robots in $\mathcal{A}$: this part is easy, because they all lie on a small segment, which already gives them a total order. The robots only have to be careful enough not to collide with other robots before reaching their final positions.

When this part is done, there are at least two robots in $\mathcal{B}$, all of which have distance much smaller than d from each other. Then the members of the TuringMobile join $\mathcal{B}$ as well, in order from the closest to the farthest. Each of them chooses a position in $\mathcal{B}$ based on the robots already there and the remnants of the TuringMobile. Moreover, the members of the TuringMobile that have not started moving to $\mathcal{B}$ yet cannot be mistaken for robots in $\mathcal{B}$, because they are at a greater distance from all others (and vice versa).

Note that, when the last robot leaves the TuringMobile and joins $\mathcal{B}$, it is able to find its final location because there are already at least four robots there, which provide a reference frame for the pattern to be formed. When this last robot has taken position in $\mathcal{B}$, the pattern is formed.

4.5 Higher Dimensions

Everything we said in this section pertained to robots in the plane. However, we can generalize all our results to robots in $\mathbb{R}^m$, for $m \geq 2$. Recall that, at the end of Section 3.2, we have described a TuringMobile for robots in $\mathbb{R}^m$, which can move within a specific plane γ exactly as a bidimensional TuringMobile, but can also move back and forth by μ in all other directions orthogonal to γ.

Now, extending our results to $\mathbb{R}^m$ actually boils down to exploring the space with a TuringMobile: once we can do this, we can easily adapt our techniques for the Near-Gathering and the Pattern Formation problem, with negligible changes.

There are several ways a TuringMobile can explore $\mathbb{R}^m$: we will only give an example. Consider the exploration of the plane described at the beginning of this section, and let P_i be the point reached by the Commander after its ith move along the spiral-like path depicted in Figure 3 (P_0 is the initial position of the Commander).

Our m-dimensional TuringMobile starts exploring γ as if it were $\mathbb{R}^2$. Whenever it visits a P_i for the first time, it goes back to P_0. From P_0, it keeps making moves orthogonal to γ until it has seen all points in $\mathbb{R}^m$ whose projection on γ is P_0 and whose distance from P_0 is at most i. Then it goes back to P_0, moves to P_1, and repeats the same pattern of moves in the section of $\mathbb{R}^m$ whose projection on γ is P_1. It then does the same thing with P_2, etc. When it reaches P_{i+1} (for the first time), it goes back to P_0, and proceeds in the same fashion. By doing so, it explores the entire space $\mathbb{R}^m$.

Note that this algorithm only requires the TuringMobile to count how many moves it has made since the beginning of the execution: thus, the machine only has to memorize a single integer. The direction of the next move according to the above pattern is then obviously Turing-computable given the move counter.

5 Conclusions

We have introduced the TuringMobile as a special configuration of oblivious non-rigid robots that can simulate a rigid robot with memory. We have also applied the TuringMobile to some typical robot problems in the context of limited visibility, showing that the assumption of connectedness of the initial visibility graph can be dropped if a unique TuringMobile is present in the system. Our results hold not only in the plane, but also in Euclidean spaces of higher dimensions.

The simplest version of the TuringMobile (Section 3.1) consists of only three robots, and is the smallest possible configuration with these characteristics (Theorems 2 and 3). Our generalized TuringMobile (Section 3.2), which works in $\mathbb{R}^m$ and simulates k registers of memory, consists of $3m + 3k$ robots (Theorem 1). We believe we can decrease this number to $m + k + 3$ by putting all the Number robots in the same basic component and adopting a more complicated technique to move them. However, minimizing the number of robots in a general TuringMobile is left as an open problem.

Our basic TuringMobile design works if the robots have the same radius of visibility, because that allows them to implicitly agree on a unit of distance. We could remove this assumption and let each of them have a different visibility radius, but we would have to add a fourth robot to the TuringMobile for it to work (as well as keep the TuringMobile small compared to *all* these radii).

Recall that, in order to encode and decode arbitrary real numbers we used the α function and its inverse, which in turn are computed using the arctan and the tan functions. However, using transcendental functions is not essential: we could achieve a similar result by using only comparisons and arithmetic operations. The only downside would be that such a real RAM program would not run in a constant number of machine steps, but in a number of steps proportional to the value of the number to encode or decode. With this technique, we would be able to dispense with the trigonometric functions altogether, and have our robots use only arithmetic operations and square roots to compute their destination points.

References

1 C. Agathangelou, C. Georgiou, and M. Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *32nd ACM Symposium on Principles of Distributed Computing (PODC)*, 250–259, 2013.

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

3 A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts, 1974.

4 H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

5 B. Bérard, P. Lafourcade, L. Millet, M. Potop-Butucaru, Y. Thierry-Mieg, and S. Tixeuil. Formal verification of mobile robot protocols. *Distributed Computing*, 29(6):459–487, 2016.

6 Q. Bramas and S. Tixeuil. The random bit complexity of mobile robots scattering. *International Journal of Foundations of Computer Science*, 28(2): 111–134, 2017.

7 D. Canepa, X. Défago, T. Izumi, and M. Potop-Butucaru. Flocking with oblivious robots. In *18th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 94–108, 2016.

8 S. Cicerone, G. Di Stefano, and A. Navarra. Asynchronous pattern formation: the effects of a rigorous approach. *arXiv:1706.02474*, 2017.

9 S. Cicerone, G. Di Stefano, and A. Navarra. Minimum-traveled-distance gathering of oblivious robots over given meeting points. In *10th International Symposium on Algorithms and Experiments for Sensor Systems (Algosensors)*, 57–72, 2014.

10 M. Cieliebak, P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

11 R. Cohen and D. Peleg. Convergence properties of the gravitational algorithm in asynchronous robot systems. *SIAM Journal on Computing*, 34(6):1516–1528, 2005.

12 P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Certified universal gathering in $\mathbb{R}^2$ for oblivious mobile robots. In *30th International Symposium on Distributed Computing (DISC)*, 187–200, 2016.

13 P. Courtieu, L. Rieg, S. Tixeuil, and X. Urbain. Impossibility of gathering, a certification. In *Information Processing Letters*, 115(3):447–452, 2015.

14 S. Das, P. Flocchini, N. Santoro, and M. Yamashita. Forming sequences of geometric patterns with oblivious mobile robots. *Distributed Computing*, 28(2):131–145, 2015.

15 X. Défago, M. Gradinariu, S. Messika, P. Raipin-Parvédy, and S. Dolev. Fault-tolerant and self-stabilizing mobile robots gathering. In *20th International Symposium on Distributed Computing (DISC)*, 46–60, 2006.

16 B. Degener, B. Kempkes, P. Kling, and F. Meyer auf der Heide. Linear and competitive strategies for continuous robot formation problems. *ACM Transactions on Parallel Computing*, 2(1): 2:1–2:18, 2015.

17 B. Degener, B. Kempkes, T. Langner, F. Meyer auf der Heide, P. Pietrzyk, and R. Wattenhofer. A tight runtime bound for synchronous gathering of autonomous robots with limited visibility. In *23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 139–148, 2011.

18 G. Di Luna, P. Flocchini, N. Santoro, and G. Viglietta. TuringMobile: a turing machine of oblivious mobile robots with limited visibility and its applications. *arXiv:1709.08800*, 2017.

19 P. Flocchini, G. Prencipe, and N. Santoro. *Distributed Computing by Oblivious Mobile Robots.* Morgan & Claypool, 2012.

20 P. Flocchini, G. Prencipe, N. Santoro, and G. Viglietta. Distributed computing by mobile robots: Uniform circle formation. *Distributed Computing*, 2016, to appear, doi:10.1007/s00446-016-0291-x.

21 P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Gathering of asynchronous robots with limited visibility. *Theoretical Computer Science*, 337(1–3):147–168, 2005.

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

23 N. Fujinaga, Y. Yamauchi, S. Kijima, and M. Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

24 T. Izumi, M. Gradinariu Potop-Butucaru, and S. Tixeuil. Connectivity-preserving scattering of mobile robots with limited visibility In *12th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 319-331, 2010.

25 L. Pagli, G. Prencipe, and G. Viglietta. Getting close without touching: Near-Gathering for autonomous mobile robots. *Distributed Computing*, 28(5):333–349, 2015.

26 F. P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, Berlin and New York, 1985.

27 H. Rogers, Jr. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, 1967.

28 M. I. Shamos. *Computational Geometry.* Ph.D. thesis, Department of Computer Science, Yale University, 1978.

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

30 M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theoretical Computer Science*, 411(26–28):2433–2453, 2010.

31 Y. Yamauchi, T. Uehara, S. Kijima, and M. Yamashita. Plane formation by synchronous mobile robots in the three-dimensional Euclidean space. *Journal of the ACM* 64(3): 16:1–16:43, 2017.