RoboCast: Asynchronous Communication in Robot Networks

Zohir Bouzid[1,⋆], Shlomi Dolev[2,⋆⋆],
Maria Potop-Butucaru[1], and Sébastien Tixeuil[1]

[1] Université Pierre et Marie Curie - Paris 6, France
[2] Ben Gurion University of the Negev, Israel

Abstract. This paper introduces the *RoboCast* communication abstraction. The RoboCast allows a swarm of non oblivious, anonymous robots that are only endowed with visibility sensors and do not share a common coordinate system, to asynchronously exchange information. We propose a generic framework that covers a large class of asynchronous communication algorithms and show how our framework can be used to implement fundamental building blocks in robot networks such as gathering or stigmergy. In more details, we propose a RoboCast algorithm that allows robots to broadcast their local coordinate systems to each others. Our algorithm is further refined with a local collision avoidance scheme. Then, using the RoboCast primitive, we propose algorithms for deterministic asynchronous gathering and binary information exchange.

1 Introduction

Existing studies in robots networks focus on characterizing the computational power of these systems when robots are endowed with visibility sensors and communicate using *only* their movements without relying on any sort of agreement on a global coordinate system. Most of these studies [1,5,4] assume oblivious robots (*i.e.* robots have no persistent memory of their past actions), so the "memory" of the network is implicit and generally deduced from the current positions of the robots. Two computation models are commonly used in robot networks: ATOM [9] and CORDA [7]. In both models robots perform in Look-Compute-Move cycles. The main difference is that these cycles are executed in a fully asynchronous manner in the CORDA model while each phase of the Look-Compute-Move cycle is executed in a lock step fashion in the ATOM model. These computation models have already proved their limitations. That is, the

⋆ Supported by DIGITEO project PACTOLE and the ANR projects R-DISCOVER and SHAMAN.

⋆⋆ Part of the research was done during a supported visit of Shlomi Dolev at LIP6 Universit Pierre et Marie Curie - Paris 6. Partially supported by Rita Altura trust chair in computer sciences, ICT Programme of the European Union under contract number FP7-215270 (FRONTS), and US Air Force European Office of Aerospace Research and Development, grant number FA8655-09-1-3016.

C. Lu, T. Masuzawa, and M. Mosbah (Eds.): OPODIS 2010, LNCS 6490, pp. 16–31, 2010.

deterministic implementations of many fundamental abstractions such as gathering or leader election are proved impossible in these settings without additional assumptions ([8]). The purpose of this paper is to study how the addition of *bounded* memory to each individual robot can increase the computational power of an *asynchronous* swarm of robots. We focus on an *all-to-all* communication primitive, called RoboCast, which is a basic building block for the design of any distributed system. A positive answer to this problem is the open gate for solving fundamental problems for robot networks such as gathering, scattering, election or exploration.

In robot networks, using motion to transmit information is not new [9,10,6]. In [9], Suzuki and Yamashita present an algorithm for broadcasting the local coordinate system of each robot (and thus build a common coordinate system) under the ATOM model. The algorithm heavily relies on the phase atomicity in each Look-Compute-Move cycle. In particular, a robot a that observes another robot b in four distinct positions has the certitude that b has in turn already seen a in at least two different positions. The situation becomes more intricate in the asynchronous CORDA model. Indeed, the number of different positions observed for a given robot is not an indicator on the number of complete cycles executed by that robot since cycles are completely uncorrelated. By contrast, our implementation of RoboCast is designed for the more general CORDA model and uses a novel strategy: the focus moves from observing robots in different positions to observing robots moving in different *directions*. That is, each robot changes its direction of movement when a particular stage of the algorithm is completed; this change allows the other robots to infer information about the observed robot.

Another non trivial issue that needs to be taken care of without explicit communication is *collisions avoidance*, since colliding robots could be confused due to indistinguishability. Moreover, robots may physically collide during their Move phase. One of the techniques commonly used to avoid collisions consists in computing a Voronoi diagram [2] and allowing robots to move *only* inside their Voronoi cells [5]. Since the Voronoi cells do not overlap with one another, robots are guaranteed to not collide. This simple technique works well in the ATOM model but heavily relies on the computation of the same Voronoi diagram by the robots that are activated concurrently, and thus does not extend to the CORDA model where different Voronoi diagrams may be computed by different robots, inducing possible collisions. Our approach defines a collision-free zone of movement that is compatible with the CORDA model constraints.

Applications of our RoboCast communication primitive include fundamental services in robot networks such as gathering and stigmergy. Deterministic gathering of two stateless robots has already been proved impossible when robots have no common orientation [9]. In [9], the authors also propose non-oblivious solutions for deterministic gathering in the ATOM model. Our RoboCast permits to extend this result to the CORDA model, using bounded memory and a limited number of movements. Recently, in [6], the authors extend the work of [9] to efficiently implement stigmergy in robot networks in the ATOM model.

Stigmergy is the ability for robots to exchange binary information that is encoded in the way they move. This scheme is particularly appealing for secure communication in robot networks, since *e.g.* jamming has no impact on robot communication capability. The RoboCast primitive allows to extend this mechanism to the CORDA model, with a collision-free stigmergy scheme.

Our contribution. We formally specify a robot network communication primitive, called RoboCast, and propose implementation variants for this primitive, that permit anonymous robots not agreeing on a common coordinate system, to exchange various information (*e.g.* their local coordinate axes, unity of measure, *rendez-vous* points, or binary information) using only motion in a two dimensional space. Contrary to previous solutions, our protocols all perform in the *fully asynchronous* CORDA model, use *constant* memory and a *bounded* number of movements. Then, we use the RoboCast primitive to efficiently solve some fundamental open problems in robot networks. We present a fully asynchronous deterministic gathering and a fully asynchronous stimergic communication scheme. Our algorithms differ from previous works by several key features: they are totally asynchronous (in particular they do not rely on the atomicity of cycles executed by robots), they make no assumption on a common chirality or knowledge of the initial positions of robots, and finally, each algorithm uses only a bounded number of movements. Also, for the first time in these settings, our protocols use CORDA-compliant collision avoidance schemes.

Roadmap. The paper is made up of six sections. Section 2 describes the computing model and presents the formal specification of the RoboCast problem. Section 3 presents our protocol and its complexity. The algorithm is enhanced in Section 4 with a collision-avoidance scheme. Using the Robocast primitive, Section 5 proposes algorithms for deterministic asynchronous gathering and binary information exchange. Finally, Section 6 provides concluding remarks. Due to space limitations, some proofs are given in a companion technical report [3].

2 Model

We consider a network that consists of a finite set of n robots arbitrarily deployed in a two dimensional space, with no two robots located at the same position. Robots are devices with sensing, computing and moving capabilities. They can observe (sense) the positions of other robots in the space and based on these observations, they perform some local computations that can drive them to other locations.

In the context of this paper, the robots are *anonymous*, in the sense that they can not be distinguished using their appearance and they do not have any kind of identifiers that can be used during the computation. In addition, there is no direct mean of communication between them. Hence, the only way for robots to acquire information is by observing their positions. Robots have *unlimited visibility*, *i.e.* they are able to sense the entire set of robots. We assume that robots are *non-oblivious*, *i.e.* they can remember observations, computations and motions

performed in previous steps. Each robot is endowed with a local coordinate system and a local unit measure which may be different from those of other robots. This local coordinate system is assumed to be fixed during a run unless it is explicitly modified by the corresponding robot as a result of a computation. We say in this case that robots *remember* their own coordinate systems. This is a common assumption when studying non-oblivious robot networks [9,6].

A *protocol* is a collection of n *programs*, one operating on each robot. The program of a robot consists in executing *Look-Compute-Move cycles* infinitely many times. That is, the robot first observes its environment (Look phase). An observation returns a snapshot of the positions of all robots within the visibility range. In our case, this observation returns a snapshot of the positions of *all* robots. The observed positions are *relative* to the observing robot, that is, they use the coordinate system of the observing robot. Based on its observation, a robot then decides — according to its program — to move or to stay idle (Compute phase). When a robot decides a move, it moves to its destination during the Move phase.

The local state of a robot is defined by the content of its memory and its position. A configuration of the system is the union of the local states of all the robots in the system. An *execution* $e = (c_0, \ldots, c_t, \ldots)$ of the system is an infinite sequence of configurations, where c_0 is the initial configuration of the system, and every transition $c_i \rightarrow c_{i+1}$ is associated to the execution of a non empty subset of *actions*. The granularity (or atomicity) of those actions is model-dependent and is defined in the sequel of this section.

A *scheduler* is a predicate on computations, that is, a scheduler defines a set of *admissible* computations, such that every computation in this set satisfies the scheduler predicate. A *scheduler* can be seen as an entity that is external to the system and selects robots for execution. As more power is given to the scheduler for robot scheduling, more different executions are possible and more difficult it becomes to design robot algorithms. In the remainder of the paper, we consider that the scheduler is fair and *fully asynchronous*, that is, in any infinite execution, every robot is activated infinitely often, but there is no bound on the ratio between the most activated robot and the least activated one. In each cycle, the scheduler determines the distance to which each robot can move in this cycle, that is, it can stop a robot before it reaches its computed destination. However, a robot r_i is guaranteed to be able to move a distance of at least δ_i towards its destination before it can be stopped by the scheduler.

We now review the main differences between the ATOM [9] and CORDA [7] models. In the ATOM model, whenever a robot is activated by the scheduler, it performs a *full* computation cycle. Thus, the execution of the system can be viewed as an infinite sequence of rounds. In a round one or more robots are activated by the scheduler and perform a computation cycle. The *fully-synchronous ATOM* model refers to the fact that the scheduler activates all robots in each round, while the regular *ATOM* model enables the scheduler to activate only a subset of the robots. In the CORDA model, robots may be interrupted by the scheduler after performing only a portion of a computation cycle. In particular,

phases (Look, Compute, Move) of different robots may be interleaved. For example, a robot a may perform a Look phase, then a robot b performs a Look-Compute-Move complete cycle, then a computes and moves based on its previous observation (that does not correspond to the current configuration anymore). As a result, the set of executions that are possible in the CORDA model are a strict superset of those that are possible in the ATOM model. So, an impossibility result that holds in the ATOM model also holds in the CORDA model, while an algorithm that performs in the CORDA model is also correct in the ATOM model. Note that the converse is not necessarily true.

The RoboCast Problem. The RoboCast communication abstraction provides a set of robots located at arbitrary positions in a two-dimensional space the possibility to broadcast their local information to each other. The RoboCast abstraction offers robots two communication primitives: *RoboCast(M)* sends *Message* M to all other robots, and *Deliver(M)* delivers *Message* M to the local robot. The message may consists in the local coordinate system, the robot chirality, the unit of measure, or any binary coded information.

Consider a run at which each robot r_i in the system invokes *RoboCast*(m_i) at some time t_i for some message m_i. Let t be equal to $max\{t_1, \ldots, t_n\}$. Any protocol solving the RoboCast Problem has to satisfy the following two properties:

> *Validity*: For each message m_i, there exists a time $t'_i > t$ after which every robot in the system has performed *Deliver*(m_i).
> *Termination*: There exists a time $t_T \geq max\{t'_1, \ldots, t'_n\}$ after which no robot performs a movement that causally depends on the invocations of *RoboCast*(m_i).

3 Local Coordinate System RoboCast

In this section we present algorithms for robocasting the local coordinate system. For ease of presentation we first propose an algorithm for two-robots then the general version for systems with n robots.

The local coordinate system is defined by two axes (abscissa and ordinate), their positive directions and the unity of measure. In order to robocast this information we use a modular approach. That is, robots invoke first the robocast primitive (*LineRbcast*1 hereafter) to broadcast a line representing their abscissa. Then, using a parametrized module (*LineRbcast*2), they robocast three successive lines encoding respectively their ordinate, unit of measure and the positive direction of axes. This invocation chain is motivated by the dependence between the transmitted lines. When a node broadcasts a line, without any additional knowledge, two different points have to be sent in order to uniquely identify the line at the destination. However, in the case of a coordinate system, only for the first transmitted axis nodes need to identify the two points. The transmission of the subsequent axes needs the knowledge of a unique additional point.

3.1 Line RoboCast

In robot networks the broadcast of axes is not a new issue. Starting with their seminal paper [9], Suzuki and Yamashita presented an algorithm for broadcasting the axes via motion that works in the ATOM model. Their algorithm heavily relies on the atomicity of cycles and the observation focus on the different positions of the other robots during their Move phase.

This type of observation is totally useless in asynchronous CORDA model. In this model, when a robot r moves towards its destination, another robot r' can be activated $k > 1$ times with k arbitrarily large, and thus observe r in k different positions without having any clue on the number of complete cycles executed by r. In other words, the number of different positions observed for a given robot is not an indicator on the number of complete executed cycles since in CORDA cycles are completely uncorrelated.

Our solution uses a novel strategy. That is, the focus moves from observing robots in different positions to observing their change of direction: each robot changes its direction of movement when a particular stage of the algorithm is completed; this change allows the other robots to infer information about the observed robot.

Line RoboCast Detailed Description. Let r_0 and r_1 be the two robots in the system. In the sequel, when we refer to one of these robots without specifying which, we denote it by r_i and its peer by r_{1-i}. In this case, the operations on the indices of robots are performed modulo 2. For ease of presentation we assume that initially each robot r_i translates and rotates its local coordinate system such that its x-axis and origin coincide with the line to be broacast and its current location respectively. We assume also that each robot is initially located in the origin of its local coordinate system.

At the end of the execution, each robot must have broadcast its own line and have received the line of its peer. A robot "receives" the line broadcast by its peer when it knows at least two distinct positions of this line. Thus, to send its line, each robot must move along it (following a scheme that will be specified later) until it is sure that it has been observed by the other robot.

The algorithm idea is simple: each robot broadcasts its line by moving along it in a certain direction (considered to be positive). Simultaneously, it observes the different positions occupied by its peer r_{1-i}. Once r_i has observed r_{1-i} in two distinct positions, it informs it that it has received its line by changing its direction of movement, that is, by moving along its line in the reverse direction (the negative direction if the first movement have been performed in the positive direction of the line). This change of direction is an acknowledgement for the reception of the peer line. A robot finishes the algorithm once it changed its direction and observed that the other robot also changed its direction. This means that both robots have sent their line and received the other's line.

The algorithm is described in detail as Algorithm 1. Its proof can be found in [3]. Each robot performs four stages referred in Algorithm 1 as states:

- state S_1: This is the initial state of the algorithm. At this state, the robot r_i stores the position of its peer in the variable pos_1 and heads towards the position (1.0) of its local coordinate system. That is, it moves along its line in the positive direction. Note that r_i stays only one cycle in this state and then goes to state S_2.
- state S_2: A this point, r_i knows only one point of its peer line (recorded in pos_1). To be able to compute the whole peer line, r_i must observe r_{1-i} in another (distinct) position of this line. Hence, each time it is activated, r_i checks if r_{1-i} is still located in pos_1 or if it has already changed its position. In the first case (line 2.a of the code), it makes no movement by selecting its current position as its destination. Otherwise (line 2.b), it saves the new position of r_{1-i} in pos_2 and delivers the line formed by pos_1 and pos_2. Then, it initiates a change of direction by moving towards the point (-1.0) of its local coordinate system, and moves to state S_3.
- state S_3: at this point r_i knows the line of its peer locally derived from pos_1 and pos_2. Before finishing the algorithm, r_i must be sure that also r_{1-i} knows its line. Therefore, it observes r_{1-i} until it detects a change of direction (the condition of line 3.a). If this is not the case and if r_i is still in the positive part of its x-axis, then it goes to the position $(-1, 0)$ of its local coordinate system (line 3.b). Otherwise (if r_i is already in the negative part of its x-axis), it performs a null movement (line 3.c). When r_i is in state S_3 one is sure, as we shall show later, that r_{1-i} knows at least one position of l_i, say p. Recall that l_i corresponds to the x-axis of r_i. It turns out that p is located in the positive part of this axis. In moving towards the negative part of its x-axis, r_i is sure that it will eventually be observed by r_{1-i} in a position distinct from p which allows r_{1-i} to compute l_i.
- state S_4: At this stage, both r_i and r_{1-i} received the line sent by each others. That is, r_i has already changed its own direction of movement, and observed that r_{1-i} also changed its direction. But nothing guarantees that at this step r_{1-i} knows that r_i changed its direction of movement. If r_i stops now, r_{1-i} may remain stuck forever (in state S_3). To announce the end of the algorithm to its peer, r_i heads towards a position located outside l_i, That is, it will move on a line $nextl_i$ (distinct from l_i) which is given as parameter to the algorithm. During the move from l_i to $nextl_i$, r_i should avoid points outside these lines. To this end, r_i must first pass through $myIntersect$ - which is the intersection of l_i and $nextl_i$ - before moving to a point located in $nextl_i$ but not on l_i (refer to lines 3.a.2, 3.a.3 and 4.a of the code).

  Note that the robocast of a line is usually followed by the robocast of other information (e.g. other lines that encode the local coordinate system). To helps this process the end of the robocast of l_i should mark the beginning of the next line, $nextl_i$, robocast. Therefore, once r_i reaches $myIntersect$, r_i rotates its local coordinate system such that its x-axis matches now with $nextl_i$, and then it moves toward the point of (1,0) of its (new) local coordinate system. When r_{1-i} observes r_i in a position that is not on l_i, it learns that r_i knows that r_{1-i} learned l_{1-i}, and so it can go to state S_4 (lines 3.$a.*$) and finish the algorithm.

Algorithm 1. Line RoboCast **LineRbcast1 for two robots: Algorithm for robot** r_i

Variables:
$state$: initially S_1
pos_1, pos_2: initially $\perp$
$destination, myIntersect$: initially $\perp$

Actions:
1. State $[S_1]$: *%Robot r_i starts the algorithm%*

a. $pos_1 \leftarrow observe(1-i)$
b. $destination \leftarrow (1,0)_i$
c. $state \leftarrow S_2$
d. Move to destination

2. State $[S_2]$: *%r_i knows one position of l_{1-i}%*

a. **if** $(pos_1 = observe(1-i))$ **then** $destination \leftarrow observe(i)$
b. **else**
1. $pos_2 \leftarrow observe(1-i)$
2. $l_{1-i} \leftarrow line(pos_1, pos_2)$
3. Deliver (l_{1-i})
4. $destination \leftarrow (-1,0)_i$
5. $state \leftarrow S_3$ **endif**
c. Move to destination

3. State $[S_3]$: *%r_i knows the line robocast by robot r_{1-i}%*

a. **if** (pos_2 is not inside the line segment $[pos_1, observe(1-i)]$) **then**
1. $state \leftarrow S_4$
2. $myIntersect \leftarrow intersection(l_i, nextl_i)$
3. $destination \leftarrow myIntersect$
b. **else if** $(observe(i) \geq (0,0)_i)$ **then** $destination \leftarrow (0,-1)_i$
c. **else** $destination \leftarrow observe(i)$ **endif endif**
d. Move to destination

4. State $[S_4]$: *%r_i knows that robot r_{1-i} knows its line l_i%*

a. **if** $(observe(i) \neq myIntersect)$ **then** $destination \leftarrow myIntersect$
b. **else**
1. r_i rotates its coordinate system such that its x-axis and the origin match with $nextl_i$ and $myIntersect$ respectively.
2. $destination \leftarrow (1,0)_i$; return **endif**
c. Move to destination

3.2 Line RoboCast: A Composable Version

Line RoboCast primitive is usually used as a building block for achieving more complex tasks. For example, the RoboCast of the local coordinate system requires the transmission of four successive lines representing respectively the abscissa, the ordinate, the value of the unit measure and a forth line to determine the positive direction of axes. In stigmergic communication a robot has to transmit at least a line for each binary information it wants to send. In all these examples, the transmitted lines are dependent one of each other and therefore their successive transmission can be accelerated by directly exploiting this dependence. Indeed, the knowledge of a unique point (instead of two) is sufficient for the receiver to infer the sent line. In the following we propose modifications of the Line RoboCast primitive in order to exploit contextual information that are encoded in a set of predicates that will be detailed in the sequel.

In the case of the local coordinate system, the additional information the transmission can exploit is the fact that the abscissa is perpendicular to the ordinate. Once the abscissa is transmitted, it suffices for a robot to simply send a single position of its ordinate, say $pos1$. The other robots can then calculate the ordinate by finding the line that passes through $pos1$ and which is perpendicular

to the previously received abscissa. In the modified version of the Line RoboCast algorithm the predicate *isPerpendicular* encodes this condition.

For the case of stigmergy, a robot transmits a binary information by robocasting a line whose angle to the abscissa encodes this information. The lines transmitted successively by a single robot are not perpendicular to each others. However, all these lines pass through the origin of the coordinate system of the sending robot. In this case, it suffices to transmit only one position located on this line as long as it is distinct from the origin. We say in this case that the line satisfies the predicate *passThrOrigin*.

A second change we propose relates to the asynchrony of the algorithm. In fact, even if robots execute in unison, they are not guaranteed to finish the execution of *LineRbcast*1 at the same time (by reaching S_4). A robot r_i can begin transmitting its k-th line l_i when its peer r_{1-i} is still located in its $(k-1)$-th line $ancientl_{1-i}$ that r_i has already received. r_i should ignore the positions transmitted by r_{1-i} until it leaves $ancientl_{1-i}$ for a new line. It follows that to make the module composable, the old line that the peer has already received from its peer should be supplied as an argument ($ancientl_{1-i}$) to the function. Thus, it will not consider the positions occupied by r_{1-i} until the latter leaves $ancientl_{1-i}$.

In the following, we present the code of the new Line RoboCast function that we denote by *LineRbcast*2. Its description and its formal proof are omitted since they follow the same lines as those of *LineRbcast*1.

3.3 RoboCast of the Local Coordinate System

To robocast their two axes (abscissa and ordinate), robots call LineRbcast1 to robocast the abscissa, then LineRbcast2 to robocast the ordinate. The parameter $\neq myOrdinate$ of *LineRbcast*2 stands for the next line to be robocast and it can be set to any line different from *myOrdinate*. The next line to robocast (*unitLine*) is a line whose angle with the x-axis encodes the unit of measure. This angle will be determined during the execution *LineRbcast*2.

1. $peerAbscissa \leftarrow LineRbcast1(myAbscissa, myOrdinate)$
2. $peerOrdinate \leftarrow$
$LineRbcast2(myOrdinate, \neq myOrdinate, peerAbscissa, isPerpendicular)$

After executing the above code, each robot knows the two axes of its peer coordinate system but not their positive directions neither their unit of measure. To robocast the unit of measure we use a technique similar to that used by [9]. The idea is simple: each robot measures the distance d_i between its origin and the peer's origin in terms of its local coordinate system. To announce the value of d_i to its peer, each robot robocast via LineRbcast2 a line, *unitLine*, which passes through its origin and whose angle with its abscissa is equal to $f(d_i)$ where for $x > 0, f(x) = (1/2x) \times 90°$ is a monotonically increasing function with range $(0°, 90°)$. The receiving robot r_{1-i} can then infer d_i from $f(d_i)$ and compute the unit measure of r_i which is equal to d_{1-i}/d_i. The choice of $(0°, 90°)$ as a range for $f(x)$ (instead of $(0°, 360°)$) is motivated by the fact that the positive

Algorithm 2. Line RoboCast **LineRbcast2 for two robots: Algorithm for robot** r_i

Inputs:
l_i : the line to robocast
$nextl_i$: the next line to robocast after l_i
$precedentl_{1-i}$: the line robocast precedently by r_{1-i}
$predicate$: a predicate on the output l_{1-i}, for example $isPerpendicular$ and $passThrOrigin$.

Outputs:
l_{1-i} : the line robocast by r_{1-i}

Variables:
$state$: initially S_1
pos_1: initially $\perp$
$destination, myIntersect, peerIntersect$: initially $\perp$

Actions:
1. State $[S_2]$: *%r_i starts robocasting its line l_i%*

a. **if** $(observe(1-i) \in precedentl_{1-i})$ **then** $destination \leftarrow observe(i)$
b. **else**
1. $pos3 \leftarrow observe(1-i)$
2. $l_{1-i} \leftarrow$ the line that passes through $pos3$ and satisfies $predicate$.
3. Deliver (l_{1-i})
4. $peerIntersect \leftarrow$ intersection between l_{1-i} and $precedentl_{1-i}$
5. $destination \leftarrow (0,-1)_i$
6. $state \leftarrow S_3$ **endif**
c. Move to destination

2. State $[S_3]$: *%r_i knows the line robocast by robot r_{1-i}%*

a. **if** ($pos3$ is not inside the line segment $[peerIntersect, observe(1-i)]$) **then**
1. $state \leftarrow S_4$
2. $myIntersect \leftarrow intersection(l_i, nextl_i)$
3. $destination \leftarrow myIntersect$
b. **else if** $(observe(i) \geq (0,0)_i)$ **then** $destination \leftarrow (0,-1)_i$
c. **else** $destination \leftarrow observe(i)$ **endif endif**
d. Move to destination

3. State $[S_4]$: similar to state S_4 of the $lineRbcast1$ function.

directions of the two axes are not yet known to the robots. It is thus impossible to distinguish between an angle α with $\alpha \in (0°, 90°)$ and the angles $\Pi - \alpha$, $-\alpha$, and $\Pi + \alpha$. To overcome the ambiguity and to make $f(x)$ injective, we restrict the range to $(0°, 90°)$. In contrast, Suzuki and Yamashita [9] use a function $f'(x)$ slightly different from ours: $(1/2x) \times 360°$. That is, its range is equal to $(0°, 360°)$. This is because in ATOM, robots can robocast at the same time the two axis and their positive directions, for example by restricting the movement of robots to only the positive part of their axes. Since the positive directions of the two axes are known, *unitLine* can be an oriented line whose angle $f'(x)$ can take any value in $(0°, 360°)$ without any possible ambiguity.

Positive directions of axes. Once the two axes are known, determining their positive directions amounts to selecting the upper right quarter of the coordinate system that is positive for both x and y. Since the line used to robocast the unit of distance passes through two quarters (the upper right and the lower left), it remains to choose among these two travelled quarters which one corresponds to the upper right one. To do this, each robot robocast just after the line encoding the unit distance another line which is perpendicular to it such that their intersection lays inside the upper right quarter.

Generalization to n robots. The generalization of the solution to the case of $n > 2$ robots has to use an additional mechanism to allow robots to "recognize" other robots and distinguish them from each others despite anonymity.

Let us consider the case of three robots r_1, r_2, r_3. When r_1 looks the second time, r_2 and r_3 could have moved (or be moving), each according to its local coordinate system and unit measure. At this point, even with memory of past observations, r_1 may be not able to distinguish between r_2 and r_3 in their new positions given the fact that robots are anonymous. Moreover, r_2 and r_3 could even switch places and appear not to have moved. Hence, the implementation of the primitive $observe(i)$ is not trivial. For this, we use the collision avoidance techniques presented in the next section to instruct each robot to move only in the vicinity of its initial position. This way, other robots are able to recognize it by using its past positions. The technical details of this mechanism are given at the end of the next section.

Apart from this, the generalization of the protocol with n robots is trivial. We present its detailed description in [3].

3.4 Motion Complexity Analysis

Now we show that the total number of robot moves in the coordinate system RoboCast is upper bounded. For the sake of presentation, we assume for now that the scheduler does not interrupt robots execution before they reach their planned destination. Each robot is initially located at the origin of its local coordinate system. To robocast each axis, a robot must visit two distinct positions: one located in the positive part of this axis and the other one located in its negative part. For example, to robocast its x-axis, a robot has first to move from its origin to the position $(1.0)_i$, then from $(1.0)_i$ to the $(-1, 0)_i$. Then, before initiating a robocast for the other axis, the robot must first return back to its origin. Hence, at most 3 movements are needed to robocast each axis. This implies that to robocast the whole local coordinate system, at most 12 movements have to be performed by a particular robot.

In the general CORDA model, the scheduler is allowed to stop robots before they reach their destination, as long as a minimal distance of δ_i has been traversed. In this case, the number of necessary movements is equal to at most $8 * (1 + 1/\delta_i)$. This worst case is obtained when a robot is *not* stopped by the scheduler when moving from its origin towards another position (thus letting it go the farthest possible), but stopped whenever possible when returning back from this (far) position to the origin.

This contrasts with [9] and [6] where the number of positions visited by each robot to robocast a line is unbounded (but finite). This is due to the fact that in both approaches, robots are required to make a non null movement whenever activated until they know that their line has been received. Managing an arbitrary large number of movements in a restricted space to prevent collisions yields severe requirements in [6]: either robots are allowed to perform infinitely small movements (and such movements can be seen by other robots with infinite precision), or the scheduler is restricted in its choices for activating robots (no robot can be activated more than k times, for a given k, between any two activations of another robot) and yields to a setting that is not fully asynchronous. Our solution does not require any such hypothesis.

4 Collision-Free RoboCast

In this section we enhance the algorithms proposed in Section 3 with the collision-free feature. In this section we propose novel techniques for collision avoidance that cope with the system asynchrony.

Our solution is based on the same principle of locality as the Voronoi Diagram based schemes. However, acceptable moves for a robot use a different geometric area. This area is defined for each robot r_i as a local *zone of movement* and is denoted by ZoM_i. We require that each robot r_i moves only inside ZoM_i. The intersection of different ZoM_i must remain empty at all times to ensure collision avoidance. We now present three possible definitions for the zone of movement: ZoM_i^1, ZoM_i^2 and ZoM_i^3. All three ensure collision avoidance in CORDA, but only the third one can be computed in a model where robots do not know the initial position of their peers.

Let $P(t) = \{p_1(t), p_2(t) \ldots, p_n(t)\}$ be the configuration of the network at time t, such that $p_i(t)$ denotes the position of robot r_i at time t expressed in a global coordinate system. This global coordinate system is unknown to individual robots and is only used to ease the presentation and the proofs. Note that $P(t_0)$ describes the initial configuration of the network.

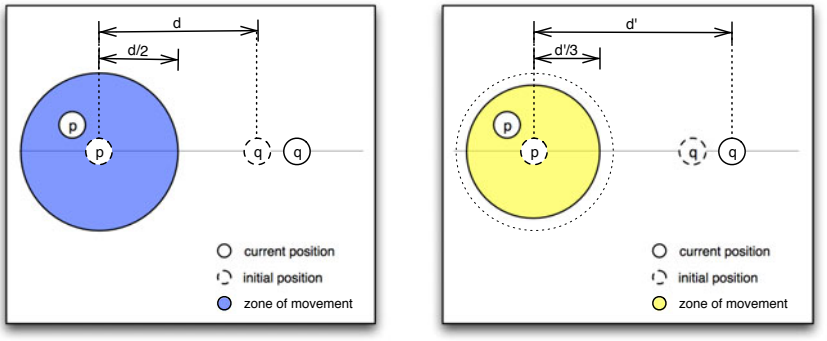
Definition 1. *(**Voronoi Diagram**) [2] The Voronoi diagram of a set of points $P = \{p_1, p_2, \ldots, p_n\}$ is a subdivision of the plane into n cells, one for each point in P. The cells have the property that a point q belongs to the Voronoi cell of point p_i iff for any other point $p_j \in P$, $dist(q, p_i) < dist(q, p_j)$ where $dist(p, q)$ is the Euclidean distance between p and q. In particular, the strict inequality means that points located on the boundary of the Voronoi diagram do not belong to any Voronoi cell.*

Definition 2. *(ZoM_i^1) Let $DV(t_0)$ be the Voronoi diagram of the initial configuration $P(t_0)$. For each robot r_i, the zone of movement of r_i at time t, $ZoM_i^1(t)$, is the Voronoi cell of point $p_i(t_0)$ in $DV(t_0)$.*

Definition 3. *(ZoM_i^2) For each robot r_i, define the distance $d_i = min\{dist(p_i(t_0), p_j(t_0))$ with $r_j \neq r_i\}$. The zone of movement of r_i at time t, $ZoM_i^2(t)$, is the circle centered in $p_i(0)$ and whose diameter is equal to $d_i/2$. A point q belongs to $ZoM_i^2(t)$ iff $dist(q, p_i(t_0)) < d_i/2$.*

Definition 4. *(ZoM_i^3) For each robot r_i, define the distance $d_i(t) = min\{dist(p_i(t_0), p_j(t))$ with $r_j \neq r_i\}$ at time t. The zone of r_i at time t, $ZoM_i^3(t)$, is the circle centered in $p_i(t_0)$ and whose diameter is equal to $d_i(t)/3$. A point q belongs to $ZoM_i^3(t)$ iff $dist(q, p_i(t_0)) < d_i(t)/3$.*

Note that ZoM^1 and ZoM^2 are defined using information about the initial configuration $P(t_0)$, and thus cannot be used with the hypotheses of Algorithm 2. In contrast, robot r_i only needs to know its *own* initial position and the *current* positions of other robots to compute ZoM_i^3. As there is no need for r_i to know the *initial* positions of other robots, ZoM_i^3 can be used with Algorithm 2. It



(a) ZoM_p^2 (b) ZoM_p^3

Fig. 1. Example zones of movement: The network is formed of two robots: p and q. d is the distance between the initial positions of p and q (dashed circles), d' is the distance between the initial position of p and the current position of q. The diameter of ZoM_p^2 (blue) is $d/2$ and that of ZoM_p^3 (yellow) is $d'/3$.

remains to prove that ZoM_i^3 guarantees collision avoidance. We first prove that ZoM_i^1 does, which is almost trivial because its definition does not depend on time. Then, it suffices to prove that $ZoM_i^3 \subseteq ZoM_i^2 \subseteq ZoM_i^1$. Besides helping us in the proof, ZoM_i^2 can be interesting in its own as a cheap collision avoidance scheme in the ATOM model, as computing a cycle of radius half the distance to the nearest neighbor is much easier that computing a full blown Voronoi diagram.

Lemma 4.1. *If $\forall t$, for each robot r_i, the destination point computed by r_i at t remains inside $ZoM_i^1(t)$, then collisions are avoided.*

Proof. By definition of Voronoi diagram, different Voronoi cells do not overlap. Moreover, for a given i, ZoM_i^1 is static and does not change over time. Hence, $\forall i, j \in \Pi$, $\forall t, t'$, $ZoM_i^1(t) \cap ZoM_j^1(t') = \emptyset$.

Clearly, $ZoM_i^2 \subseteq ZoM_i^1$ which means that ZoM_i^2 ensures also collision avoidance.

Lemma 4.2. *If $\forall t$, for each robot r_i, the destination point computed by r_i at t always remains inside $ZoM_i^2(t)$, then collisions are avoided.*

The proof of the above lemma follows directly from the fact that $\forall t ZoM_i^2(t) \subseteq ZoM_i^1(t)$ and Lemma 4.1.

Lemma 4.3. $\forall t, ZoM_i^3(t) \subseteq ZoM_i^2(t)$.

Ensuring Collision-freedom in Line Robocast Algorithms. To make LineRbcast1 and LineRbcast2 collision-free, it is expected that any destination computed by a robot r_i at t be located within its $ZoM_i^3(t)$. The computation of destinations is modified as follows: Let $dest_i(t)$ be the destination computed by a robot r_i at time t. Based on $dest_i(t)$, r_i computes a new destination $dest'_i(t)$

that ensures collision avoidance. $dest'_i(t)$ can be set to any point located in $[p_i(t_0), dest_i(t)] \cap ZoM_i^3(t)$. For example, we can take $dest'_i(t)$ to be equal to the point located in the line segment $[p_i(t_0), dest_i(t)]$ and distant from $p_i(t_0)$ by a distance of $d_i(t)/2$ with $d_i(t)$ computed as explained in Definition 4.

This modification of the destination computation method does not impact algorithms correctness since it does not depend on the exact value of computed destinations, but on the relationship between the successive positions occupied by each robot. The algorithms remain correct as long as robots keep the capability to freely change their direction of movement and to move in both the positive and the negative part of each such direction. This capability is not altered by the collision avoidance scheme since the origin of the coordinate system of each robot - corresponding to its original position - is strictly included in its zone of movement, be it defined by ZoM^1, ZoM^2 or ZoM^3.

Generalisation of the Protocols to n Robots. As explained at the end of Section 3, the generalisation of our algorithms to the case of n robots has to deal with the issue of distinguishing robots from each others despite their anonymity. The solution we use is to instruct each robot to move in the close neighbourhood of its original position. Thus, other robots can recognize it by comparing its current position with past ones. For this solution to work, it is necessary that each robot always remains the closest one to all the positions it has previously occupied. Formally speaking, we define the zone of movement ZoM^4 in a similar way as ZoM^3 except that the diameter is this time equal to $d_i(t)/6$ (vs. $d_i(t)/3$). We now show that ZoM^4 provides the required properties. Let r_i and r_j be an arbitrary pair of robots and Let d_{ij} denotes the distance between their initial positions. It can easily shown, using the same arguments as the proof of Lemma 4.3, that:

1. Neither of the two robots moves away from its initial position by a distance greater than $d_{ij}/4$. This implies that each robot remains always at a distance strictly smaller than $d_{ij}/2$ from all the positions it has previously held.
2. The distance between r_i (resp. r_j) and all the positions held by r_j (r_i) is strictly greater than $d_{ij}/2$.

Hence, r_i can never be closer than r_j to a position that was occupied by r_j, and vice versa. This implies that it is always possible to recognize a robot by associating it with the position which is closest to it in some previously observed configuration.

5 RoboCast Applications

5.1 Asynchronous Deterministic 2-Gathering

Given a set of n robots with arbitrary initial locations and no agreement on a global coordinate system, n-Gathering requires that all robots eventually reach the same unknown beforehand location. n-Gathering was already solved when

$n > 2$ in both ATOM [9] and CORDA [4] oblivious models. The problem is impossible to solve for $n = 2$ even in ATOM, except if robots are endowed with persistent memory [9]. In this section we present an algorithm that uses our RoboCast primitive to solve 2-Gathering in the non-oblivious CORDA model.

A first "naive" solution is for each robot to robocast its abscissa and ordinate axes and to meet the other robot at the midpoint m of their initial positions. RoboCasting the two axes is done using our Line RoboCast function described above in conjunction with the ZoM^3-based collision avoidance scheme.

A second possible solution is to refine Algorithm $\psi_{f-point(2)}$ of [9,10] by using our Line RoboCast function to "send" lines instead of the one used by the authors. The idea of this algorithm is that each robot which is activated for the first time translates and rotates its coordinate system such that the other robot is on its positive y-axis, and then it robocasts its (new) x-axis to the other robot using our Line Robocast function. In [9], the authors give a method that allows each robot to compute the initial position of one's peer by comparing their two robocast x-axes defined above. Then each robot moves toward the midpoint of their initial positions. Our Line RoboCast routine combined with the above idea achieves gathering in asynchronous systems within a bounded (*vs.* finite in [9]) number of movements of robots and using only two (*vs.* four) variables in their persistent memory.

Theorem 5.1. *There is an algorithm for solving deterministic gathering for two robots in non-oblivious asynchronous networks (CORDA).*

5.2 Asynchronous Stigmergy

Stigmergy [6] is the ability of a group of robots that communicate only through vision to exchange binary information. Stigmergy comes to encode bits in the movements of robots. Solving this problem becomes trivial when using our RoboCast primitive. First, robots exchange their local coordinate system as explained in Section 3. Then, each robot that has a binary packet to transmit robocasts a line to its peers whose angle with respect to its abscissa encodes the binary information. Theoretically, as the precision of visual sensors is assumed to be infinite, robots are able to observe the exact angle of this transmitted line, hence the size of exchanged messages may be infinite also. However, in a more realistic environment in which sensor accuracy and calculations have a margin of error, it is wiser to discretize the measuring space. For this, we divide the space around the robot in several sectors such that all the points located in the same sector encode the same binary information (to tolerate errors of coding). For instance, to send binary packets of 8 bits, each sector should have an angle equal to $u = 360°/2^8$. Hence, when a robot moves through a line whose angle with respect to the abscissa is equal to α, the corresponding binary information is equal to $\lfloor \alpha/n \rfloor$. Thus, our solution works in asynchronous networks, uses a bounded number of movements and also allows robots to send binary packets and not only single bits as in [6].

6 Conclusion and Perspectives

We presented a new communication primitive for robot networks, that can be used in fully asynchronous CORDA networks. Our scheme has the additional properties of being motion, memory, and computation efficient. We would like to raise some open questions:

1. The solution we presented for collision avoidance in CORDA can be used for protocols where robots remain in their initial vicinity during the whole protocol execution. A collision-avoidance scheme that could be used with all classes of protocol is a challenging issue.
2. Our protocol assumes that a constant number of positions is stored by each robot. Investigating the minimal number of stored positions for solving a particular problem would lead to interesting new insights about the computing power that can be gained by adding memory to robots.

References

1. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation 15(5), 818–828 (1999)
2. Aurenhammer, F.: Voronoi diagrams a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR) 23(3), 405 (1991)
3. Bouzid, Z., Dolev, S., Potop-Butucaru, M., Tixeuil, S.: RoboCast: Asynchronous Communication in Robot Networks. Technical report (2010)
4. Cieliebak, M., Flocchini, P., Prencipe, G., Santoro, N.: Solving the robots gathering problem. In: Automata, Languages and Programming, pp. 192–192 (2003)
5. Défago, X., Konagaya, A.: Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In: POMC, pp. 97–104 (2002)
6. Dieudonné, Y., Dolev, S., Petit, F., Segal, M.: Deaf, dumb, and chatting asynchronous robots. In: OPODIS, pp. 71–85 (2009)
7. Flocchini, P., Prencipe, G., Santoro, N., Widmayer, P.: Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In: Aggarwal, A.K., Pandu Rangan, C. (eds.) ISAAC 1999. LNCS, vol. 1741, pp. 93–102. Springer, Heidelberg (1999)
8. Prencipe, G.: On the feasibility of gathering by autonomous mobile robots. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 246–261. Springer, Heidelberg (2005)
9. Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM J. of Computing 28(4), 1347–1363 (1999)
10. Suzuki, I., Yamashita, M.: Erratum: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM J. of Computing 36(1), 279–280 (2006)