Intermittent Connectivity Maintenance with Heterogeneous Robots

Rosario Aragues[1,3], Dimos V. Dimarogonas[2], Pablo Guallar[1] and Carlos Sagues[1]

***Abstract*—We consider a scenario of cooperative task servicing, with a team of heterogeneous robots with different maximum speeds and communication radii, in charge of keeping the network intermittently connected. We abstract the task locations into a $1D$ cycle graph that is traversed by the communicating robots, and we discuss intermittent communication strategies so that each task location is periodically visited, with a worst–case revisiting time. Robots move forward and backward along the cycle graph, exchanging data with their previous and next neighbors when they meet, and updating their region boundaries. Asymptotically, each robot is in charge of a region of the cycle graph, depending on its capabilities. The method is distributed, and robots only exchange data when they meet.**

***Index Terms*—Distributed robot systems, multi–robot systems, connectivity maintenance, heterogeneous robots.**

I. Introduction

Servicing tasks is a core multi–robot application [1]. We consider a cooperative task servicing scenario, with a team of task–robots, visiting different task locations to service them, and a team of communicating–robots in charge of keeping the task locations intermittently connected. Considering heterogeneous teams of robots with different roles and aims has a long history, e.g., [2]. Here, when a task–robot wants to propagate and get data updates, it waits at the current task place for a communicating–robot to show up, it exchanges data, and then moves to the next task location. The problem of visiting tasks located on the plane can be abstracted into a $1D$ scenario by building a cycle graph connecting the task locations [3], [4]. We focus on the coordination of the communicating–robots on this cycle graph, which are heterogeneous and have different maximum speeds and communication radii.

The problem of connectivity control has received a lot of attention during the last years. A review of several methods can be found at [5]. A first approach consists of keeping the network connected at all times. This can be achieved by keeping the initial set of links, with possible link additions [1], [6], or by keeping pairs of links according to some underlying topology which is updated as robots move, for undirected [7]–[9], or directed graphs [10]. Several works use global connectivity approaches [11], [12], that rely on global parameters like the algebraic connectivity and Fiedler eigenvector. These works usually encode additional terms on the model, like obstacle of inter–robot collision avoidance, and they often study the performance degradation of the high–level task due to the effect of the connectivity maintenance action. Depending on the environment size, and the amount of robots and their capabilities, it may not be possible to accomplish a high–level task using a strategy based on keeping the network connected at all times.

An alternative are intermittent connectivity scenarios [13]–[15]. The network may be disconnected at every time instant, but it is jointly connected *over time* and infinitely often. The key idea is to design the robot motions to ensure this behavior. These methods are more flexible, since they are always guaranteed to work, even if the environment size is larger, at the cost of performance degradation. One of the notable approaches on intermittent connectivity is [13], where the goal is to ensure the connectivity on an environmental graph, by making robots move forward and backward on the links of this graph. For the method to work properly, the number of robots must equal the number of links in the graph. In addition, since each robot is trapped on its associated edge, the method cannot take any advantage from heterogeneous robots with larger maximum speeds and communication radii, which have to move slower depending on the worst–case robot motion (the slowest robot and / or the one assigned to the largest link). In [14], [15] robots are not restricted to the links of a fixed graph. Robots are organized in teams, and they meet at a point in the environment chosen by the team members [14], or form connected sub-networks in the space [15] to exchange data. Some robots belong to more than one team, and the team graph must be connected. Although there is more flexibility, [14], [15] require the robot teams to be selected by the user and they also require the offline schedule of the communication events. Thus, [14], [15] do not take advantage of the improved capabilities of individual robots. Moreover, [13]–[15] do not self adapt to communicating robots entering and leaving the communicating team or varying their communication radii and maximum speeds. In [16] another example of application of intermittent connectivity ideas is presented. There, there are robots in charge of gathering data, with limited buffer capabilities, that meet with some relay robots to upload the data. Compared to our work, in [16] the cooperation between relay robots in charge of the communication is weaker, since it only happens due to spontaneous (unplanned) meetings.

We propose an intermittent connectivity strategy where the robots move forward and backward on the $1D$ cycle graph of the environment. Each robot has two neighbors in the cycle graph, and robots exchange data when they meet at the boundaries of their assigned regions. Robots which are faster or with larger communicating radius, are in charge of larger regions in the cycle graph. These regions are updated online in a distributed way, using local data on the involved robots.

*Supported by COMMANDIA SOE2/P1/F0638 (Interreg Sudoe Programme, ERDF), CAS18/00082, Ministerio de Ciencia, Innovación y Universidades, PGC2018-098719-B-I00 (MCIU/AEI/FEDER, UE), DGA T45-17R (Gobierno de Aragón), and the Knut och Alice Wallenberg Foundation (KAW)

[1]R. Aragues, P. Guallar and C. Sagues are with DIIS Universidad de Zaragoza and Instituto de Investigación en Ingeniería de Aragón I3A, Spain raragues@unizar.es, pabloguallar@gmail.com, csagues@unizar.es

[2]D. V. Dimarogonas is with the School of Electrical Engineering and Computer Science, KTH, Stockholm, Sweden dimos@kth.se

[3]Corresponding author.

The proposed method is similar to a *beads–on–a–ring* strategy [17]–[19], where each robot moves forward and backward on a specific region of the ring, impacting with its previous and next neighbors, and exchanging data only during the impacts. However, in beads–on–a–ring methods, the aim is that the robots synchronize to move at the same speed, which may be pre–established [17], or may depend on the average of the initial robot speeds [18], [19], and end up covering regions of equal length. Here instead, the aim is that robots are in charge of larger regions if they are faster or have larger communication radii. In addition, [17]–[19] let robots to speed up without restrictions. In the proposed method robots cannot move faster than their maximum speed, so the strategy and approach differs to accommodate for this restriction. Our work is also related to works on coverage over a ring [20], although there the aim is to make robots converge to fixed points with associated coverage regions, instead of making them move forward and backward. The assumptions on the data exchange and on the communication capabilities are different in both scenarios, and so are the methods used.

The contributions of this paper are: (i) a distributed method that does not depend on a specific number of robots, that takes advantage of the heterogeneous nature of the robots, and that only requires data exchange during robot meetings; (ii) the proof that, asymptotically, each robot is in charge of a region with a length depending on its maximum speed and communication radius; (iii) the proof of convergence to configurations with performance guaranties; and (iv) the validation of the method in a realistic simulation environment using ROS/Gazebo. A preliminary version of this work appears in [21]. Compared to [21], here we make a thorough study of the performance of the method in terms of the revisiting times of the locations on the cycle graph (Theorems 4.2 and 4.3). In order to prove these theorems, we build on several theoretical results, that are developed along Sections VI and VII, and that are also novel compared to [21].

II. Notation and Problem Description

Assume a team of so called task–robots is in charge of servicing some tasks. Task–robots travel to the different locations of the l tasks placed in an environment as in Fig. 1. To provide task–robots with data exchange capabilities, we place in the area a dedicated team of communicating–robots, that communicate among them and arrive at the task locations periodically, as it is required by classical distributed algorithms such as distributed averaging, $\max$/$\min$ consensus, or flooding. When a task–robot wants to propagate or get data updates, it just waits at its current task location for a communicating–robot to show up, and then, it exchanges data and moves to its next task location. In this paper, we focus on the behavior of the communicating robots, called from now on *robots*.

A cycle connecting the l task locations is pre–computed or obtained in a centralized fashion, and is available to the communicating–robots. The cycle graph can be built using, e.g., a Minimum–distance Spanning Tree (MST) with duplicated edges [3], or computing an approximate or exact solution

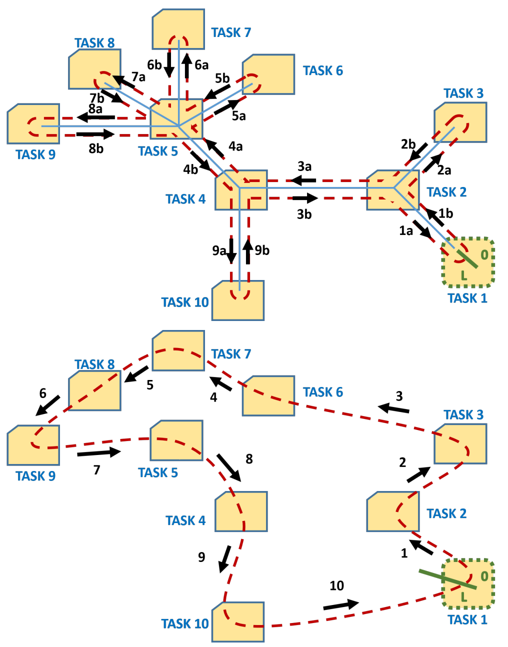


Fig. 1. **(Top)** Example of cycle graph connecting the locations of 10 tasks (orange regions), obtained from a Tree (in blue) with duplicated edges (red dashed). We take, e.g., positions 0 and L of the cycle graph at the location of Task 1, between $1a$ and $1b$ (in green). Then, the cycle graph involves edges $1b, 2a, 2b, \ldots$ and finally, $1a$, getting back to the initial position at Task 1. Robots move forward and backward on the cycle graph. **(Bottom)** Other example of cycle graph (approximate TSP), connecting the task locations. Positions 0 and L of the cycle graph are at the location of Task 1. The edges are $1, 2, \ldots, 10$ (red dashed), getting back to the initial position at Task 1. We do not make any assumptions on the relation between the number of robots n and the number of task locations l and we do not restrict the robots to remain within one specific edge.

of the Traveling Salesman Problem (TSP) [4]. A location on this cycle graph is established as position 0. There are n communicating–robots (*robots*), with different maximum motion speeds and communication radii, that move forward and backward through the edges of the graph, meeting and exchanging data with their neighbors. We do not make any assumptions on the relation between n and l, and also we do not restrict the robots to remain within one specific edge.

Since every scenario with tasks located on a plane can be transformed into a cycle graph [3], [4], from now on, we will no longer consider the underlying structure. We will focus instead on the behavior of the method on the associated $1D$ cycle graph (the mapping from this $1D$ cycle graph and the $2D/3D$ scene is commented later in Sec. VIII). We let L be the total length of the cycle graph, i.e., the sum of the lengths of the edges that connect the l tasks in the cycle graph. Note that different cycle graphs will give rise to different values of this total length L.

We consider n robots moving along the cycle graph. Each robot $i = 1, \ldots, n$ has a communication radius $r_i \in \mathbb{R}_{\geq 0}$ and a maximum motion speed $v_i \in \mathbb{R}_{>0}$, and it is assigned a *scalar* $p_i(t) \in \mathbb{R}$, which represents its position in the cycle graph, $p_i(t) \in [0, L]$, for $i = 1, \ldots, n$. In the simulations we will represent with a line the robot positions between 0 and L. Due to the cyclic structure of the cycle graph, the position L is then equivalent to position 0. Robots cannot move faster than their maximum speed. At every time instant, each robot i can move forward, backward, or be stopped. This information is represented with the activity $a_i(t)$ and orientation $o_i(t)$ variables. The activity $a_i(t) \in \{0, 1\}$ represents that robot i is respectively stopped or moving, whereas the orientation $o_i(t) \in \{-1, +1\}$ represents that robot i moves respectively backward or forward. Note that we use two variables $a_i(t)$ and $o_i(t)$ since we want stopped robots to have an orientation $o_i(t)$ associated to them. This property will be used later in the paper. Robots either move at their maximum speeds or remain stopped, so that

$$\dot{p}_i(t) = v_i a_i(t) o_i(t). \tag{1}$$

***Problem** 2.1:* We assume that the cycle graph cannot be covered by the robots at static positions using their radii r_i. Thus, for some periods of time, a task location will remain unconnected (it will have no communicating robot nearby). The aim is to design a strategy for the communicating robots moving and exchanging data on the cycle graph that ensures the task locations receive the visit of a robot periodically, and that provides theoretical guarantees on the time elapsed between visits of the robots to the task locations. Robots must meet each other so that the information can travel along the cycle graph (i.e., between all the task locations). The strategy must make use of the capabilities of the robots, which are heterogeneous and have different speeds and communication radii. Moreover, we are interested in providing a solution where robots exchange data only when they meet in the cycle graph, and that is robust to changes in the capabilities of the robots, i.e., a distributed asynchronous method.

In the remaining of this section, we explain the notation and definitions used. The proposed distributed asynchronous method is presented in Sec. III, and its properties and performance guarantees are given in Sec. IV.

Consider the robots on the cycle graph. Each robot $i \in 1, \ldots, n$ has two neighbors, its left $(i-1)$ and right $(i+1)$ neighbor. For the clarity of the presentation, we assume the robot identifiers are sorted according to their position on the cycle graph, from left to right. From now on, $i+1 = 1$ for $i = n$, and $i-1 = n$ for $i = 1$. Between robots i and $i+1$, for $i = 1, \ldots, n$, there is a *boundary* $y_i(t)$ (its computation is explained in Sec. III). Each robot i is responsible of the region in the cycle graph within its boundaries $y_{i-1}(t)$, $y_i(t)$. Robot i moves forward and backward within its region, until its communication zone reaches its boundaries (Fig. 2). When robot i meets its neighbor $i+1$ at the boundary $y_i(t)$ (or neighbor $i-1$ at boundary $y_{i-1}(t)$) at a time t_e, they can exchange data. We let $d_i(t)$ be the *length* of the region associated to a robot i, that depends on its boundaries $y_{i-1}(t)$, $y_i(t)$,

$$d_1(t) = y_1(t),\ d_i(t) = y_i(t) - y_{i-1}(t),\ i = 2, \ldots, n. \tag{2}$$

Note from Fig. 2 that when robot i reaches the boundary $y_i(t)$, its position is $p_i(t) = y_i(t) - r_i$, and when it reaches the boundary $y_{i-1}(t)$, its position is $p_i(t) = y_{i-1}(t) + r_i$. Thus, the *traversing time* $e_i(t)$ that robot i needs to move between

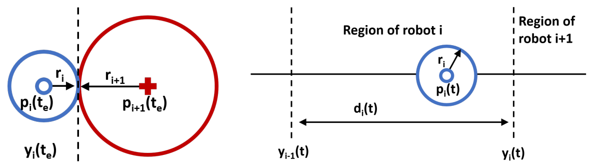


Fig. 2. **Left:** Events like arriving to a boundary and meeting, catching, or discovering a neighbor (described in Sec. III), take place when the communication regions of robots get in touch, or touch the boundary. **Right:** Region associated to robot i, length $d_i(t)$ of the region, and position of the boundaries $y_{i-1}(t), y_i(t)$.

its boundaries (from position $y_i(t) - r_i$ to $y_{i-1}(t) + r_i$) at maximum speed v_i, for $i = 1, \ldots, n$, is

$$e_i(t) = \frac{y_i(t) - r_i - (y_{i-1}(t) + r_i)}{v_i} = \frac{d_i(t) - 2r_i}{v_i}, \tag{3}$$

where $d_i(t)$ is given by (2) and, for $i = 1$, $y_{i-1}(t) = y_n(t)$.

A. Regions with Common Traversing Times

In the following discussion, we add a $\star$ symbol to all the variables (region boundaries $y_i^\star$, region lengths $d_i^\star$, and traversing times $e_i^\star$, for $i = 1, \ldots, n$) to refer to the goal values we want the method to achieve. Later, in Sec. III, we will present algorithms to compute these variables in a distributed and asynchronous way. Similar to eqs. (2) and (3), $y_i^\star$, $d_i^\star$ and $e_i^\star$ are related by:

$$\begin{aligned} &d_1^\star = y_1^\star, \quad d_i^\star = y_i^\star - y_{i-1}^\star, \text{ for } i = 2, \ldots, n, \\ &e_i^\star = (d_i^\star - 2r_i)/v_i, \text{ for } i = 1, \ldots, n. \end{aligned} \tag{4}$$

Given the cycle graph with length L, represented with a line between 0 and L, we want to partition it into n regions and to assign each of these to a robot i. The region associated to each robot i, for $i = 1, \ldots, n$, is defined by its boundaries $y_{i-1}^\star, y_i^\star$, and it has associated the length $d_i^\star$. We want each point in the cycle graph to be periodically revisited by the robot i in charge of the associated region. Thus, we are interested in regions that are disjoint, with the only common point being the boundary, and whose union is the cycle graph $(0 \ldots L)$,

$$d_1^\star + d_2^\star + \cdots + d_n^\star = L. \tag{5}$$

An example can be found at Figure 3.

In order to take advantage of the maximum speeds v_i and communication radii $r_i, i = 1, \ldots, n$ of each robot i, we want the traversing times $e_i^\star$ employed by each robot i for moving between its boundaries, to be the same, i.e.,

$$t_\star = e_1^\star = \ldots e_n^\star, \tag{6}$$

and we let $t_\star$ be the *common traversing time*. From (3), (5), (6), the common traversing time $t_\star$ is given by:

$$\begin{aligned} &t_\star = \frac{d_1^\star - 2r_1}{v_1} = \frac{d_2^\star - 2r_2}{v_2} = \cdots = \frac{d_n^\star - 2r_n}{v_n}, \\ &(v_1 + \cdots + v_n)t_\star = d_1^\star + \cdots + d_n^\star - 2r_1 - \cdots - 2r_n, \\ &t_\star = (L - 2\sum_{i=1}^{n} r_i)/(v_1 + v_2 + \cdots + v_n). \end{aligned} \tag{7}$$

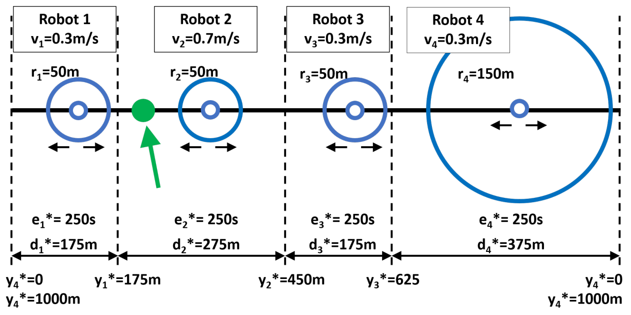


Fig. 3. Regions with common traversing times. Four heterogeneous robots (blue circles) move forward and backward on a cycle graph with length $L = 1000m$. Robots 1 and 3 have maximum speeds and communication radius $v_i = 0.3m/s$, $r_i = 50m$, $i = 1, 3$. Robot 2 can move faster ($v_2 = 0.7m/s, r_2 = 50m$), and robot 4 has a larger communication radius ($v_4 = 0.3m/s, r_4 = 150m$). If we assign regions to them with common traversing times (7), here $t_\star = e_i = 250s$, $i = 1, \ldots, n$, and they always move at their maximum speed, then we can ensure that each particular point in the cycle graph (e.g., the green dot), is revisited (Def. 2.1) every $2t_\star = 500s$.

Having smaller environments, more robots, faster, or with larger communication radii, produce lower common traversing times (Fig. 3). Before continuing, we point out some facts about $t_\star$ (7). Suppose that, instead of n robots, there was a single entity traversing the cycle graph, comprising the capabilities of all the robots: with maximum speed equal to $v_1 + \cdots + v_n$ and communication radius equal to $r_1 + \cdots + r_n$ (and diameter twice this quantity). In order to traverse a cycle graph with total length L, this single entity would employ a time equal to (7). In order to achieve this performance with a multi–robot team, robots must be correctly organized, as we propose in this paper. Note that the traversing times of all the robots must equal $t_\star$. Otherwise, if a robot has a shorter traversing time, other robots will necessarily have longer traversing times, since they have to cover longer regions. Thus, we would not take full advantage of their capabilities.

From (4), (7), the length $d_i^\star$ of the region associated to robot i, for $i = 1, \ldots, n$, is:

$$d_i^\star = v_i t_\star + 2r_i = \frac{v_i(L - 2\sum_{j=1}^{n} r_j)}{v_1 + v_2 + \cdots + v_n} + 2r_i, \tag{8}$$

and the boundary position $y_i^\star = d_1^\star + \cdots + d_i^\star$, for $i = 1, \ldots, n$, obtained by summing up the region lengths $d_j^\star$, is:

$$y_i^\star = \sum_{j=1}^{i} \frac{v_j(L - 2\sum_{j'=1}^{n} r_{j'})}{v_1 + v_2 + \cdots + v_n} + 2r_j, \tag{9}$$

where $y_n^\star$ gives L as expected.

***Definition** 2.1 (Revisiting times):* : We define the *revisiting time* as the time required for a robot to visit a particular point in the cycle graph, arriving back at the point with the same orientation. ■

Note that the *revisiting time* t_{rev} includes:

- $2e_i(t)$ to traverse the robot region in both directions at maximum speed, getting back to the original point with the same orientation, plus
- additional times where the robot is e.g., stopped.

In the definition of the *revisiting time* t_{rev} we impose that the orientation of the robot must be the same, in order to represent that robot i has completed traversing its associated region (move in one direction until it reaches its boundary, reverse and move to the opposite boundary, reverse and move back until the starting position). Thus, the condition on the orientation and position of the robot being the same represents that robot i carries out fresh data from both neighbors $i-1$ and $i+1$, that now is available to the revisited point.

Note that the optimal boundaries for each robot (9) could be pre–computed in an off–line fashion (centralized alternative). In this paper, we consider that the task locations are static or slowly changing, whereas the team of communicating–robots is more dynamic. Thus, off–line centralized computation is reasonable for the cycle graph, since it only depends on the task locations. On the other hand, we are interested in solutions that do not require knowing and keeping track of the characteristics of all the involved communicating–robots (the total amount n of robots, their communication radii r_i, their speeds v_i, and their ordering in the cycle graph, with the associated cost linear in n), and that can self adapt to changes in the team (e.g., robots increasing / decreasing their communication radius [22], [23] or their maximum speeds). Thus, we propose a distributed solution, where each robot computes asymptotically its boundaries (9) using only local information. This distributed method is presented in Sec. III.

Note also that, if robots were assigned regions with common traversing times $t_\star$ (7) and they never stopped, then each point would have a revisiting time equal to $t_{\text{rev}} = 2t_\star$. However, robots can only exchange data when they meet, and this requires some additional coordination that may make the performance degrade. In this paper, we propose a method where, under some conditions, robots achieve $t_{\text{rev}} = 2t_\star$. We provide a thorough analysis of the performance degradation when these conditions are not satisfied.

III. Intermittent Connectivity Maintenance with Heterogeneous Robots

Robots run the distributed asynchronous algorithm presented in this section for meeting intermittently, and for computing their boundaries $y_i(t)$. Later, in Sections IV to VII, we discuss the properties of the method, and the relation between the boundaries $y_i(t)$ computed in a distributed way and the boundaries $y_i^\star$ that could be obtained if all the information from all the robots was known by a central unit.

The method roughly consists of each robot i moving until it reaches or defines a boundary, waiting at this boundary until it meets with its neighbor, updating their data, and then moving to its other boundary and repeating the process. We distinguish between the following behaviors for the robots:

- Participating in an *event*: events are associated with an event time $t_{e_1}, t_{e_2}, t_{e_3}, \ldots$, or generically t_e, they affect at most two neighbors, and they modify their values of the activity $a_i(t_e)$, orientation $o_i(t_e)$, and boundary $y_i(t_e)$.
- *Between events*: robot positions $p_i(t)$ evolve according to (1). Robots may be active ($a_i(t) = 1$), moving from their current position until they arrive to or define a boundary ($a_i(t) = 1$), or inactive ($a_i(t) = 0$), waiting at a boundary.

Along the section, we define the types of events, and how robots react to them. We make the following assumptions:

***Assumption** 3.1 (A1, A2, A3):* $(A1)$ Robots n and 1 have a fixed boundary $y_n(t) = L$ for all $t \geq 0$, placed at position L (equivalently, due to the cycle structure, at position 0). $(A2)$ $o_i(0) \neq o_j(0)$ for at least a pair of robots $i, j, i \neq j$. $(A3)$ Robots $i = 1, \ldots, n$ start placed at positions $p_i(0) \in [0, L]$ so that their communication zones do not overlap, with initial orientations $o_i(0)$ satisfying $(A2)$, and all are active $a_i(0) = 1$, for all $i = 1, \ldots, n$.

The proposed algorithm works as follows.

***Algorithm** 3.1 (Discovery and Catch):* Initially, robots do not know their boundaries $y_i(0), i = 1, \ldots, n-1$ but $y_n(0) = L$ from Assumption $(A1)$. Robots move to discover their neighbors and set an initial value for their boundaries. There are two events associated:

Discovery event: Robots i and $i+1$ are moving, ($a_i(t) = 1$ and $a_{i+1}(t) = 1$), with robot i moving forward and robot $i+1$ moving backward, $o_i(t) > 0, o_{i+1}(t) < 0$, and they do not know $y_i(t)$. The discovery happens when their communication regions get in touch at $t_e \geq t$ (Fig. 2),

$$p_i(t_e) + r_i = p_{i+1}(t_e) - r_{i+1}. \tag{10}$$

At the *discovery*, the involved robots i and $i+1$ initialize their common boundary $y_i(t)$ with this discovery position, they reverse their orientations and move in opposite directions, as follows:

$$y_i(t_e^+) = p_i(t_e) + r_i, \quad o_i(t_e^+) = -1, \; o_{i+1}(t_e^+) = +1. \tag{11}$$

Catch event: Robots i and $i+1$ do not know $y_i(t)$. The catcher is robot i when it is active $a_i(t) = 1$, both i and $i+1$ are oriented forward $o_i(t) > 0, o_{i+1}(t) > 0$, and robot $i+1$ is stopped $a_{i+1}(t) = 0$ or moving slower $a_{i+1}(t) = 1, v_{i+1} < v_i$. (Equivalently, the catcher is robot $i+1$ when $a_{i+1}(t) = 1, o_i(t) < 0, o_{i+1}(t) < 0$, and robot i is stopped $a_i(t) = 0$ or moving slower $a_i(t) = 1, v_i < v_{i+1}$). The catch happens when their communication regions get in touch at $t_e \geq t$ as in eq. (10) and Fig. 2. At the *catch*, robots i and $i+1$ initialize their common boundary $y_i(t)$, the catcher robot (e.g., robot i) remains waiting at this boundary, and the caught robot $i+1$ moves or keeps on moving towards the opposite boundary,

$$y_i(t_e^+) = p_i(t_e) + r_i, \quad a_i(t_e^+) = 0, \quad a_{i+1}(t_e^+) = 1. \tag{12}$$

Note that the updates due to the events (11), (12) are designed so that the number of positive and negative robot orientations is kept. For the *catch* event, this requires the catcher to remain stopped at the boundary (instead of reversing its orientation, as it happens for the *discovery*).

During the first time instants, some robots may be discovering and catching neighbors (Algorithm 3.1), and others may have already discovered them. Once robot i has discovered both its neighbors, it knows $y_{i-1}(t), y_i(t)$, and it moves within these boundaries, updating them, and acting from then on according to the following algorithm.

***Algorithm** 3.2 (Arrivals and Meetings):* Robot i executes this algorithm only when it already knows its boundaries $y_{i-1}(t), y_i(t)$ (otherwise, it moves to discover or catch its neighbor as per Algorithm 3.1). In this phase, the events that can take place are the following:

Arrival event: Robot i moving backward $a_i(t) = 1$, $o_i(t) < 0$ (or moving forward $a_i(t) = 1$, $o_i(t) > 0$) arrives at its $y_{i-1}(t)$ boundary (or at $y_i(t)$ respectively) at a time $t_e \geq t$ when its communication region touches the boundary (Fig. 2), i.e.,

$$y_{i-1}(t_e) = p_i(t_e) - r_i, \; (\text{resp. } y_i(t_e) = p_i(t_e) + r_i) \tag{13}$$

After an *arrival* to a boundary, robot i waits at the boundary,

$$a_i(t_e^+) = 0. \tag{14}$$

The event is *arrival* if robot i arrives to the boundary and there is no neighbor waiting there. Otherwise, it is a *meeting* event.

Meeting event: Robots i and $i+1$ meet at time $t_e \geq t$ when both of them arrive at their common boundary $y_i(t)$,

$$y_i(t_e) = p_i(t_e) + r_i = p_{i+1}(t_e) - r_{i+1}. \tag{15}$$

At the *meeting*, robots i, $i+1$ update their common boundary:

$$y_i(t_e^+) = \frac{v_{i+1}(y_{i-1}(t_e) + 2r_i) + v_i(y_{i+1}(t_e) - 2r_{i+1})}{v_i + v_{i+1}}. \tag{16}$$

Then, robots reverse their orientations and get active,

$$o_i(t_e^+) = -1, o_{i+1}(t_e^+) = +1, a_i(t_e^+) = 1, a_{i+1}(t_e^+) = 1. \tag{17}$$

In fact, only i or $i+1$ (the first one that arrived to the boundary) should be inactive. If both arrivals take place at the same time, we establish an order, e.g., to start with the left robot. After the *meeting* and the updates (16), (17), robots i, $i+1$ move away from each other, towards their opposite boundary. E.g., robot i moves towards the common boundary $y_{i-1}(t)$ with its neighbor $i-1$. Note that robots i and $i-1$ must have the same value for $y_{i-1}(t)$, since it can only be updated when both i and $i-1$ meet, and thus it cannot have been changed in the meantime. When robot i gets to the $y_{i-1}(t)$ boundary, a new *arrival* or *meeting* takes place.

***Remark** 3.1 (Execution using information of times):* In the above algorithm, the update of the robot regions is performed by taking into account the positions of the boundaries $y_i(t_e)$ (16). As we will see later, in the Proof of Prop. 5.1 (Appendix A), the previous algorithm can be equivalently run by considering traversing times instead, i.e., $e_i(t_e)$:

$$\begin{aligned} e_i(t_e^+) &= e_i(t_e) + \frac{\epsilon_i}{v_i}(e_{i+1}(t_e) - e_i(t_e)), \; \epsilon_i = \frac{v_i v_{i+1}}{v_i + v_{i+1}}, \\ e_{i+1}(t_e^+) &= e_{i+1}(t_e) - \frac{\epsilon_i}{v_{i+1}}(e_{i+1}(t_e) - e_i(t_e)). \end{aligned} \tag{18}$$

Robots, instead of moving until they touch the boundary, would move at their maximum speed v_i for $e_i(t_e^+)$ time.

***Remark** 3.2 (On Assumption 3.1):* In the next sections, we will analyze the performance of the method under Assumption 3.1. Note that, in fact, Assumption 3.1(A1) can always be ensured: robots 1 and n do not need to know they are the first and the last ones. During the discovery and catch (Algorithm 3.1), if a robot moving backwards gets to the position $p_i(t) = 0 + r_i$ (i.e., it arrives to a boundary placed at the position 0 in the cycle graph), then it records this boundary and remains waiting at this boundary (equivalently, a robot moving forward and arriving to the position L in the cycle graph). Assumption 3.1(A3) is required for simplifying the analysis, although in practical setups it can be relaxed so that robots navigate to reach the cycle graph.

Algorithm 1 *Discovery and Catch (Alg. 3.1)* - Robot i $(v_i, r_i, p_i(0), o_i(0))$

```
1: Initialize empty boundaries ($y_{i-1}(0) :=<>$, $y_i(0) :=<>$)
2: Get active ($a_i(0) := 1$)
3: while either $y_{i-1}(t)$ or $y_i(t)$ are empty do
4:     Move according to (1) until an event occurs:
5:     switch event do
6:         ▷ Events for forward behavior ($o_i(t) = 1$)
7:         case discovery of neighbor $i + 1$:
8:             Initialize boundary ($y_i(t) := p_i(t) + r_i$)
9:             Reverse orientation ($o_i(t) := -1$)
10:        case catch of neighbor $i+1$ (or arrival to $y_n = L$):
11:            Initialize boundary ($y_i(t) := p_i(t) + r_i$)
12:            Wait at boundary $y_i(t)$ ($a_i(t) := 0$)
13:            Resume when neighbor $i + 1$ arrives to $y_i(t)$
14:            Get active ($a_i(t) := 1$)
15:            Reverse orientation ($o_i(t) := -1$)
16:        case catch: robot $i$ is caught by neighbor $i - 1$:
17:            Initialize boundary ($y_{i-1}(t) := p_i(t) - r_i$)
18:            Keep orientation ($o_i(t) = 1$)
19:        ▷ Events for backward behavior ($o_i(t) = -1$)
20:        case discovery of neighbor $i - 1$:
21:            Initialize boundary ($y_{i-1}(t) := p_i(t) - r_i$)
22:            Reverse orientation ($o_i(t) := +1$)
23:        case catch of neighbor $i-1$ (or arrival to $y_n = L$):
24:            Initialize boundary ($y_{i-1}(t) := p_i(t) - r_i$)
25:            Wait at boundary $y_{i-1}(t)$ ($a_i(t) := 0$)
26:            Resume when neighbor $i-1$ arrives to $y_{i-1}(t)$
27:            Get active ($a_i(t) := 1$)
28:            Reverse orientation ($o_i(t) := +1$)
29:        case catch: robot $i$ is caught by neighbor $i + 1$:
30:            Initialize boundary ($y_{i+1}(t) := p_i(t) + r_i$)
31:            Keep orientation ($o_i(t) = -1$)
32: end while
33: Run Arrivals and Meetings algorithm (Alg. 2)
```

For clarity, we include in Algorithms 1 and 2 the pseudo–code instructions that are run by each robot i participating in Alg. 3.1 (Discovery and Catch) and Alg. 3.2 (Arrivals and Meetings). First, robot i runs Alg. 1 (Discovery and Catch), using its maximum speed v_i and radius r_i, its initial position on the cycle graph $p_i(0)$ and its initial orientation $o_i(0)$. Once it has discovered its two boundaries, it proceeds with the main algorithm (Arrivals and Meetings, Alg. 2), using its maximum speed v_i, radius r_i, current position on the cycle graph $p_i(t)$, orientation $o_i(t)$, activity $a_i(t)$ and the current values for the boundaries $(y_{i-1}(t), y_i(t))$.

Observe the low complexity of the proposed method (Algs. 1 and 2), which is constant in memory, computation,and data exchange at each robot encounter.

IV. Main Results

Here we state the main properties of the method. The proofs of Theorems 4.1, 4.2 and 4.3 are given in Appendix J, and they depend on several properties presented in Sections V, VI and VII.

Algorithm 2 *Arrivals and Meetings (Alg. 3.2)* - Robot i $(v_i, r_i, p_i(t), o_i(t), a_i(t), y_{i-1}(t), y_i(t))$

```
1: while true (run for ever) do
2:     Move according to (1) until an event occurs:
3:     switch event do
4:         ▷ Events for forward behavior ($o_i(t) = 1$)
5:         case arrival to boundary $y_i(t)$
6:             if neighbor $i + 1$ is not yet at $y_i(t)$ then
7:                 Wait at boundary $y_i(t)$ ($a_i(t) := 0$)
8:                 Resume when neighbor $i+1$ arrives to $y_i(t)$
9:             end if
10:            ▷ Meeting event
11:            if $i < n$ ($y_n = L$ fixed, Asm. 3.1) then
12:                Send $v_i, r_i, y_{i-1}(t)$ to neighbor $i + 1$
13:                Receive $v_{i+1}, r_{i+1}, y_{i+1}(t)$ from $i + 1$
14:                Update $y_i(t)$ with eq. (15)
15:            end if
16:            Get active ($a_i(t) := 1$)
17:            Reverse orientation ($o_i(t) := -1$)
18:        ▷ Events for backward behavior ($o_i(t) = -1$)
19:        case arrival to boundary $y_{i-1}(t)$
20:            if neighbor $i - 1$ is not yet at $y_{i-1}(t)$ then
21:                Wait at boundary $y_{i-1}(t)$ ($a_i(t) := 0$)
22:                Resume when neig. $i-1$ arrives to $y_{i-1}(t)$
23:            end if
24:            ▷ Meeting event
25:            if $i > 1$ ($y_n = L$ fixed, Asm. 3.1) then
26:                Send $v_i, r_i, y_i(t)$ to neighbor $i - 1$
27:                Receive $v_{i-1}, r_{i-1}, y_{i-2}(t)$ from $i - 1$
28:                Update $y_{i-1}(t)$ with eq. (15)
29:            end if
30:            Get active ($a_i(t) := 1$)
31:            Reverse orientation ($o_i(t) := +1$)
32: end while
```

***Theorem** 4.1 (Convergence to common traversing times):* Consider that robots run Algorithms 3.1, 3.2 in Section III under Assumptions 3.1. Then, the traversing times $e_i(t)$, region lengths $d_i(t)$, and boundaries $y_i(t)$ (eqs. (3), (2), (16)) asymptotically converge to the goal values $t_\star$, $d_i^\star$, $y_i^\star$ in eqs. (7), (8), (9), for $i = 1, \dots, n$.

Proof: See Appendix J. ■

The performance achieved depends on the amount of robots moving with positive and negative orientations.

***Definition** 4.1 (Balanced and Unbalanced orientations):* Let n_+ and n_- be the number of robots initially moving with positive ($o_i(0) > 0$) and negative orientations ($o_i(0) < 0$), with $n = n_+ + n_-$, and let n_{bal} be $n_{bal} = \min\{n+, n_-\}$. Robot orientations are *balanced* when $n_{bal} = n_+ = n_- = n/2$ (note that n must be even in this case), and they are *unbalanced* otherwise. Without loss of generality, in the paper we consider that there are more robots with positive orientations so that $n_+ \geq n_-$ (all the discussions apply equivalently to the opposite case). ■

***Theorem** 4.2 (Performance for Balanced Orientations):* A robot team with balanced orientations (Def. 4.1) running Algorithms 3.1, 3.2 in Section III under Assumptions 3.1 and using the common traversing times $e_i(t) = t_\star$ for $i = 1, \ldots, n$ (7), converges to a configuration where the n robots perform all their $n/2$ meetings simultaneously, and with *revisiting time* $t_{\rm rev}$ (Def. 2.1) given by

$$t_{\rm rev} = 2t_\star. \tag{19}$$

Proof: See Appendix J. ■

***Theorem** 4.3 (Performance for Unbalanced Orientations):* A robot team with unbalanced orientations (Def. 4.1) running Algorithms 3.1, 3.2 in Section III under Assumptions 3.1 and using the common traversing times $e_i(t) = t_\star$ for $i = 1, \ldots, n$ (7), converges to a configuration where, at every round, there are n_{bal} meetings involving $2n_{bal}$ robots. Each pair of robots meet at their common boundary n_{bal} times every n rounds. The *revisiting time* $t_{\rm rev}$ (Def. 2.1), averaged along n_{bal} meetings, is given by

$$t_{\rm rev} = t_\star n/n_{bal}. \tag{20}$$

Proof: See Appendix J. ■

In the proposed algorithm, the task locations are not connected at all times. Instead, they are disconnected most of the time, and they are visited from time to time by communicating robots. The interest of Theorems 4.2 and 4.3 is that they provide theoretical guarantees on the elapsed time that a particular location on the cycle graph (for instance, a task location) will remain disconnected in the configuration asymptotically achieved by the robot team. The task location will receive in average two visits of a communicating robot every $t_{\rm rev}$ time (the value is exact for balanced orientations, Th. 4.2). Note also that these performance metrics only depend on the number of robots with balanced orientations (Def. 4.1) and on the common traversing time $t_\star$ (7), which depends on the total length of the cycle graph L and on the maximum speeds and communication radii of all the involved robots.

The performance metrics given in Theorems 4.2 and 4.3 will be clearer in Sec. VI.

***Remark** 4.1:* Observe from Theorems 4.1, 4.2 and 4.3 that one of the main strengths of the solution we propose, is that it does not depend on the robot IDs or their initial positions, i.e., the same common traversing times and revisiting times are obtained regardless of the robot initial ordering or initial positions in the cycle graph.

In Section VIII we present simulations carried out on the $1D$ cycle graph, we explain the mapping between the positions on this $1D$ cycle graph and the positions on $2D$ or $3D$ environments, and we present additional simulations on $2D$ environments using differential–drive ground robots. In Sec. V, VI, VII and in the appendices, we present several theoretical results which are required in order to prove Theorems 4.1, 4.2, 4.3. For clarity, the analysis in these sections is performed considering the $1D$ cycle graph.

V. Convergence to Common Traversing Times

Here, we discuss the proof of Theorem 4.1. It relies on auxiliary results from [21], which are included here to make the manuscript self–contained. Some of these results are also used later in Sections VI and VII. To prove Theorem 4.1, first, we rewrite eq. (16) in terms of the traversing times $e_i(t)$ (3) and show that it is an asynchronous weighted consensus method [24], [25]. We prove its convergence in Proposition 5.1, assuming that the set of communication graphs that *occur infinitely often* [26] [27] are jointly connected. An event occurs infinitely often [14]–[16], [26], [27] if, considering an infinite sequence of events, the particular event in the sequence holds true for an infinite number of indices. Then, in Proposition 5.2, we prove that the set of communication graphs that occur infinitely often are indeed jointly connected.

***Proposition** 5.1: (Weighted consensus on traversing times [21, Prop. 5.1]):* Assume that algorithm (3.2) gives rise to a network in which the set of communication graphs that occur infinitely often are jointly connected. Then, the traversing times $e_i(t)$, region lengths $d_i(t)$, and boundaries $y_i(t)$ (eqs. (3), (2), (16)) asymptotically converge to the goal values $t_\star$, $d_i^\star$, $y_i^\star$ in eqs. (7), (8), (9), for $i = 1, \ldots, n$.

Proof: See Appendix A. ■

We give some intermediary results to prove that, under our algorithm, the set of communication graphs that occur infinitely often are jointly connected. In our discussion, we focus on Algorithm 3.2 and consider that each robot i has run the Discovery and Catch phase (Algorithm 3.1) and thus has set an initial value for both boundaries $y_i(t), y_{i+1}(t)$. Note that Algorithm 3.1 is only run during the first time instants and, after that, robots always run Algorithm 3.2.

***Lemma** 5.1 ((Active) [21, Lemma 5.1]):* Robots are active ($a_i(t) = 1$, Section III) during a bounded time and, after that, an *arrival* or a *meeting* event always occurs.

Proof: See Appendix B. ■

This observation allows us to focus on the behavior of the discrete asynchronous version of the method.

***Definition** 5.1 (Discrete asynchronous behavior):* The discrete asynchronous version of the method (Algorithm 3.2), includes only the event times $t_{e_1}, t_{e_2}, t_{e_3}, \ldots$. Each robot i is always placed at one of its boundaries (13), $p_i(t_e) \in \{y_{i-1}(t_e) + r_i, y_i(t_e) - r_i\}$. The states $y_i(t_e)$, $o_i(t_e)$, $a_i(t_e)$, change due to *meeting* events (16), (17) (equivalently, $d_i(t_e)$, $e_i(t_e)$ (2), (3)). After a meeting between robots i, $i+1$ at time t_e, two arrival events take place in the future:

$$\begin{aligned} t_{e'} &= t_e + e_i(t_e^+), & p_i(t_{e'}^+) &= y_{i-1}(t_e) + r_i, \text{ and} \\ t_{e''} &= t_e + e_{i+1}(t_e^+), & p_{i+1}(t_{e''}^+) &= y_{i+1}(t_e) - r_{i+1}. \end{aligned} \tag{21}$$

■

***Lemma** 5.2 (Discrete Asynchronous Behavior):* Assume all robots have finished the Discovery and Catch phase (Algorithm 3.1). Then the discrete asynchronous behavior (Def. 5.1) and Algorithm 3.2 have the same: (i) Events (arrivals and meetings) and event times $t_{e_1}, t_{e_2}, t_{e_3}, \ldots$; ($ii$) States $p_i(t_e)$, $o_i(t_e)$, $a_i(t_e)$ at the event time t_e, for the robot or robots involved in the event.

Proof: See Appendix B. ■

Note that at a time event t_e, the states of the robots *not involved* in the meeting will differ between Def. 5.1 and Algorithm 3.2, since in the first one it is as if they were still on the boundary, whereas in the second one they are currently moving. However, the event only depends on the robots involved and not on the remaining ones. Thus, Lemma 5.2 holds, and we can use the representation in Def. 5.1 to study the method in a simpler way.

***Lemma** 5.3 (Properties [21, Lemma 5.2]):* Consider n robots executing algorithm 3.2. The method satisfies the following facts:

- (i) $\sum_{i=1}^{n} o_i(t)$ remains constant for all t.
- (ii) The regions associated to each robot are disjoint, with the only common point being the boundary.
- (iii) In the discrete asynchronous behavior (Def. 5.1) the order of the robots is preserved.

Proof: See Appendix B. ■

Depending on the relative speeds of robot i and $i+1$, it may be the case that, between events involving i, $i+1$ they exchange positions. E.g., if $y_i(t_e^+) > y_i(t_e)$, and $v_i >> v_{i+1}$, robot i may get to $y_{i-1}(t_e)$ and get back to $y_i(t_e^+)$ before robot $i+1$ has reached $y_i(t_e^+)$. This is temporary: robot i will stop at $y_i(t_e^+)$, but robot $i+1$ will continue to $y_{i+1}(t_e) \geq y_i(t_e^+)$. Thus, in the discrete asynchronous behavior (Def. 5.1), the order of the robots is preserved, and robots do not need to e.g., exchange identifiers.

Now, we discuss the joint connectivity of the network. We prove that each robot $i = 1, \ldots, n$ meets its neighbors $i-1$ and $i+1$ after some bounded amount of time.

***Proposition** 5.2 (Joint connectivity [21, Prop. 5.2]):* Algorithm (3.2) under Assumptions $(A1)$, $(A2)$, gives rise to a network in which the set of communication graphs that occur infinitely often are jointly connected.

Proof: See Appendix C. ■

The proof of Prop. 5.2 uses Assumption 3.1$(A2)$ and Lemma 5.3(ii) in order to ensure that, during all the executions of the algorithm, at least two robots will have different orientations. This property is key in order to prove that the algorithm does not exhibit blocking (robots never get blocked waiting at different boundaries).

We are ready to prove Theorem 4.1 (see Appendix J).

The analysis of the Discovery and Catch phase (Algorithm 3.1) that takes place during the first time instants is omitted for clarity. Note that it could be done in a similar way to the previous analysis of Algorithm 3.2.

VI. Performance for Interlaced Orientations

In this section and the next one, we present the tools to prove Th. 4.2 and 4.3 regarding the performance of the method. All along the section, we will assume robots have already run enough iterations of the algorithm (Algs. 3.1, 3.2 in Sec. III) and they already work with $y_i^\star$, $t_\star$, $d_i^\star$ (Th. 4.1) for all i. We first present a tool for analyzing the method using an equivalent discrete synchronous version based on the concept of *rounds*. Then, we introduce the concept of balanced and unbalanced *interlaced* configurations, and we discuss the implications regarding the performance of the algorithm (Theorems 4.2 and 4.3). Later, in Sec. VII, we prove the convergence of the method to these interlaced configurations.

A. Round–based method

***Assumption** 6.1:* We assume robots have run the method in Section III (Algs. 3.1, 3.2) for a time t_0 that we will call *initial time*. We assume t_0 is large enough, so that the traversing times have already converged to the common traversing time, $e_i(t_0) = e_i^\star = t_\star$, with $t_\star$ as in eq. (7), and equivalently $y_i(t_0) = y_i^\star$, $d_i(t_0) = d_i^\star$. We impose that the initial time t_0 is not an event time.

***Remark** 6.1:* Note that the convergence to the common traversing time $t_\star$ as in eq. (7) is asymptotic (Theorem 4.1) instead of finite time. The difference between $e_i(t)$ and $t_\star$ can be anyway quite small by considering the time t_0 large enough. Thus, the performance discussed in this section and the next one will be in practice affected by some small perturbations, although in all our simulations we have observed almost no differences for t_0 large enough.

***Definition** 6.1 (Initial states and arrival times):* Under Assumption 6.1, at t_0 each robot $i \in \{1, \ldots, n\}$ has an *initial state* given by its position $p_i(t_0)$, orientation $o_i(t_0)$, and activity $a_i(t_0)$. We let t_i^e be the time at which each robot i arrives from its *initial* position $p_i(t_0)$ to one of its boundaries ($t_i^e = t_0$ if robot i is waiting at a boundary $a_i(t_0) = 0$ at the initial time t_0, and otherwise it is the time to arrive to boundary $y_{i-1}^\star$ if $o_i(t_0) < 0$ and $a_i(t_0) = 1$, or to boundary $y_i^\star$ if $o_i(t_0) > 0$ and $a_i(t_0) = 1$), i.e.,

$$t_i^e = \begin{cases} t_0 + \frac{y_i^\star - r_i - p_i(t_0)}{v_i} & \text{if } o_i(t_0) > 0 \text{ and } a_i(t_0) = 1, \\ t_0 + \frac{p_i(t_0) - y_{i-1}^\star + r_i}{v_i} & \text{if } o_i(t_0) < 0 \text{ and } a_i(t_0) = 1, \\ t_0 & \text{if } a_i(t_0) = 0. \end{cases}$$

Note that $t_i^e \in [t_0, t_0 + t_\star)$. ■

Now we define the concept of *round* and present the equivalent discrete–time synchronous version of the method.

***Definition** 6.2 (Round k):* The $k-$th round, with $k = 0, 1, 2, \ldots$ is the following time interval with duration $t_\star$:

$$[t_0 + kt_\star, t_0 + (k+1)t_\star), \qquad (22)$$

with the initial time t_0 as in Assumption 6.1. ■

***Definition** 6.3 (Discrete Synchronous Behavior):* The discrete synchronous version of the algorithm in Section III (Alg. 3.2) under the conditions in Assumption 6.1 and the initial states in Def. 6.1, updates states and schedules events at each *round* k (Def. 6.2). Instead of the time t or the event time t_e, in the discrete synchronous behavior, variables include the round k associated to the time interval (22). For each robot i, the algorithm keeps track of its position $p_i(k) \in \{y_{i-1}^* + r_i, y_i^* - r_i\}$, orientation $o_i(k) \in \{-1, +1\}$, and the event times $t_i^e(k)$ at which robot i arrives to its boundaries (it is not necessary to keep track of $a_i(k)$). At round $k = 0$, these states are initialized for $i = 1, \ldots, n$ with

$$t_i^e(0) = t_i^e, \qquad p_i(0) = p_i(t_i^e), \qquad o_i(0) = o_i(t_0), \qquad (23)$$

where t_i^e is given in Def. 6.1. Variable $t_i^e(k)$ represents the latest time of arrival to a boundary. If an arrival takes place during the current round k (eq. (22)), then $t_i^e(k) \in [t_0 + kt_\star, t_0 + (k+1)t_\star)$. Otherwise, $t_i^e(k)$ represents a time that belongs to a previous round, $t_i^e(k) < t_0 + kt_\star$. Variable $o_i(k)$ represents the orientation of robot i at the beginning of round

k. Variable $p_i(k)$ represents the position of robot i at the time $t_i^e(k)$ of its latest arrival to a boundary, i.e., $p_i(k) = p_i(t_i^e(k))$. If the arrival takes place during the current round, then the robot position $p_i(t)$ takes the value $p_i(k)$ during the current round, at time $t = t_i^e(k)$. If the arrival took place in a previous round, then the robot position equals $p_i(k)$ at the beginning of the round. At round k, with $k = 0, 1, 2, \dots$, there is a meeting between each pair $(i, i+1)$ of robots satisfying

$$\begin{aligned} p_i(k) - r_i = p_{i+1}(k) + r_i, \text{ equivalently,} \\ o_i(k) = +1, \text{ and } o_{i+1}(k) = -1. \end{aligned} \tag{24}$$

The time at which the meeting event takes place within round k is $\max\{t_i^e(k), t_{i+1}^e(k)\}$. Note that when (24) is satisfied, the meeting takes place during the round k, in particular at time $\max\{t_i^e(k), t_{i+1}^e(k)\}$. Note also that within a round k, there may be several different meeting events.

The states of robots i not involved in meetings remain unchanged during the next round $k+1$, i.e., $p_i(k+1) = p_i(k)$, $o_i(k+1) = o_i(k)$, $t_i^e(k+1) = t_i^e(k)$. For all the $(i, i+1)$ robots involved in meetings, the states are updated to show their arrivals to the boundary during the next round $k+1$:

$$\begin{aligned} p_i(k+1) = y_{i-1}^\star + r_i, \quad & p_{i+1}(k+1) = y_{i+1}^\star - r_{i+1}, \\ o_i(k+1) = -1, \quad & o_{i+1}(k+1) = +1, \\ t_{i+1}^e(k+1) = t_i^e(k+1) \quad & = \max\{t_i^e(k), t_{i+1}^e(k)\} + t_\star. \end{aligned} \tag{25}$$

Note that $t_i^e(k+1) = \max\{t_i^e(k), t_{i+1}^e(k)\} + t_\star$ is the event time at which robots will arrive to the opposite boundary, since $\max\{t_i^e(k), t_{i+1}^e(k)\}$ is the time at which the meeting took place during round k, and $t_\star$ is the common traversing time needed by robots to arrive to the opposite boundary after the meeting. ■

***Lemma** 6.1 (Rounds):* Consider the behavior of the discrete asynchronous version (Def. 5.1) of the method (Algorithm 3.2) under Assumptions 3.1 and 6.1, and the discrete synchronous behavior (Def. 6.3). The following properties hold: (i) If at round k there is a meeting between robots $(i, i+1)$, with event time t_e and updates given by eq. (21) then, during round $k+1$ there are exactly two arrivals to boundaries at times $t_i^e(k+1), t_{i+1}^e(k+1)$ given by (25). During round k, there are no additional events (arrivals or meetings) involving i or $i+1$. (ii) Equation (25) encodes the same robot positions and orientations as the discrete asynchronous version (Def. 5.1), for the event times $t_i^e(k)$ associated to arrivals to boundaries, i.e., $p_i(k) = p_i((t_i^e(k))^+)$, $o_i(k) = o_i(t_i^e(k))$, regardless of the order in which events take place at round k. (iii) Condition $o_i(k) = +1$ and $o_{i+1}(k) = -1$ and condition $p_i(k) - r_i = p_{i+1}(k) + r_i$ associated to round k represent the same information.

Proof: See Appendix D. ■

An example can be observed in Figure 6. Rounds are separated by vertical gray solid lines and have duration $t_\star = 127.8$. During round $k = 1$, robot $i = 6$ has states $p_i(k) = y_6^\star - r_6$, $o_i(k) = +1$, $t_i^e(k) = 175.9 = k{*}t_\star{+}48.1$. For all the duration $t_\star = 127.8$ of the round, robot $i = 6$ reaches at most one of its boundaries.

B. Interlaced Orientations

Now, we use the concept of *rounds* (Definition 6.2) and the equivalent representation of the algorithm based on the discrete synchronous version (Definition 6.3), to study the performance of the method for balanced and unbalanced orientations (Def. 4.1). In this section, we will consider orientations which are *interlaced*, which is a concept we explain next. After this, in section VII, we will prove that in fact, the method interlaces the orientations.

***Definition** 6.4 (Interlaced Orientations):* Let n_+, n_-, and n_{bal} be as in Def. 4.1. The robot orientations are *interlaced* at round k when there exist distinct robot indexes

$$i_1, i_2, \dots, i_{n_{bal}}$$

such that, for all $j = 1, \dots, n_{bal}$:

$$o_{i_j}(k) = +1 \text{ and } o_{i_j+1}(k) = -1. \tag{26}$$

Note that in case $n_+ > n_-$, the orientations of the remaining robots are positive, and if the orientations are balanced ($n_{bal} = n/2$), there are no remaining robots. ■

An example of interlaced orientations can be seen in Fig. 4 **Left**: $n = 8$, $n_{bal} = 2$, and $i_1 = 1, i_2 = 4$, so that $o_1(k_0) = +1, o_2(k_0) = -1$, and $o_4(k_0) = +1, o_5(k_0) = -1$, and the other robots have positive orientation. The orientations in Fig. 4 **Right**, are interlaced as well, and balanced in this case: $n = 8$, $n_{bal} = n/2 = 4$, and $i_1 = 1, i_2 = 3, i_3 = 5, i_4 = 7$, so that $o_1(k_0) = o_3(k_0) = o_5(k_0) = o_7(k_0) = +1$, and $o_2(k_0) = o_4(k_0) = o_6(k_0) = o_8(k_0) = -1$.

***Lemma** 6.2 (Interlaced Meetings):* Assume at round k the configuration is interlaced with the robot indexes in Definition 6.4 being $i_1, i_2, \dots, i_{n_{bal}}$. Without loss of generality, assume $n_+ \geq n_-$. Then: (i) At round $k+1$ the configuration is interlaced, with robot indexes $i_1{-}1, i_2{-}1, \dots, i_{n_{bal}}{-}1$. (ii) At all the successive rounds $k' \geq k$ the configuration remains interlaced. (iii) At all the successive rounds $k' \geq k$ there are exactly n_{bal} meetings. (iv) Each robot $i = 1, \dots, n$ arrives to $y_i^\star$ boundary n_{bal} times every n rounds.

Proof: See Appendix D. ■

***Proposition** 6.1 (Unbalanced Interlaced Performance):* Under assumption 6.1, a robot team with unbalanced interlaced orientations (Defs. 4.1 and 6.4) running the algorithm in Section III (Alg. 3.2), has a performance in terms of the revisiting times t_{rev} (Def. 2.1) averaged along n_{bal} meetings, given by

$$t_{rev} = t_\star n / n_{bal}. \tag{27}$$

Proof: See Appendix E. ■

Moreover, when the orientations are not only *interlaced* but also *balanced*, as the following result shows, robots synchronize to perform exactly their meetings simultaneously in the network.

***Proposition** 6.2 (Balanced Interlaced Synchronization):* Assume the conditions in Assumption 6.1, Def. 6.1 hold and there is a round k_0 at which the robots achieve a balanced interlaced configuration (Defs. 4.1, 6.4). Let t'_0 be the initial time of round k_0, i.e., $t'_0 = t_0 + k_0 {*} t_\star$, and $t_1^e(k_0), \dots, t_n^e(k_0)$ be the time of arrival to boundary of each robot i at round k_0,

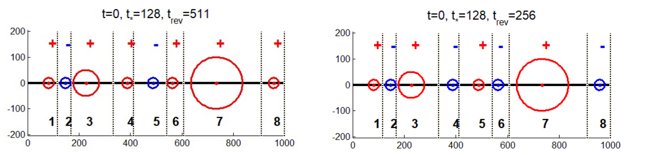


Fig. 4. Two examples of robots running the method in Section III (Algs. 3.1,3.2) with different maximum speeds and communication radii (circles around the robots), moving forward (red) and backward (blue) between their boundaries $y_i^\star$ (9) (gray solid, in vertical) as in Asm. 6.1, Def. 6.1. **Left**: Unbalanced interlaced orientations (Defs. 4.1 and 6.4), with $n_{bal} = 2$, and with robot indexes $i_1 = 1, i_2 = 4$ at round $k = 0$ ($o_1(0) = +1, o_2(0) = -1$, $o_4(0) = +1, o_5(0) = -1$, and the remaining orientations are positive). **Right**: Balanced interlaced orientations (Defs. 4.1, and 6.4), with $n_{bal} = n/2$.

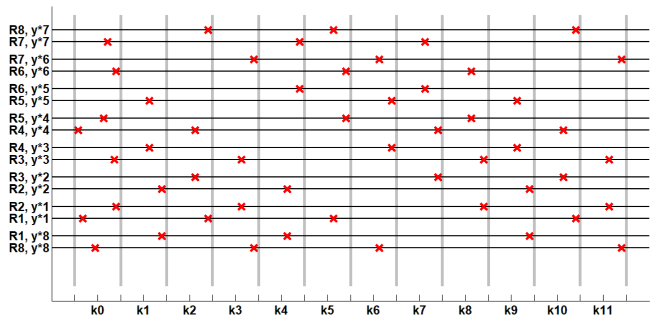


Fig. 5. Time of arrival (red crosses) of each robot i to boundaries $y_{i-1}^\star$ and $y_i^\star$ (y–axis) along time and rounds (x–axis), for the scenario in Fig. 4 **Left** (unbalanced interlaced orientations). Gray solid vertical lines represent the separation between rounds $k = 0, 1, 2, \ldots$ (Def. 6.2), where we take $k_0 = 0$. Each robot i arrives at the $y_i^\star$ boundary (red crosses on lines $R_i, y_i^\star$) but for robots $i = 2$ and $i = 5$, which arrive at the $y_{i-1}^\star$ boundary, (red crosses on lines $R_2, y_1^\star$ and $R_5, y_4^\star$). At round $k = 1$, orientations are unbalanced interlaced, with indexes $i_1 - 1 = 1, i_2 - 1 = 4$ (red crosses on lines $R_1, y_n^\star$ and $R_4, y_3^\star$), and so on. At each round there are $n_{bal} = 2$ meetings involving $2n_{bal} = 4$ robots (4 red crosses at each round k) as in Lemma 6.2. Every n rounds, e.g., $1 \leq k \leq 7$, or $2 \leq k \leq 8$, each robot i arrives n_{bal} times to boundary $y_i^\star$ (red crosses).

defined as t_i^e in Def. 6.1 and (23) using the new t_0' instead of t_0. Then, after $n/2$ rounds, all robots $i = 1, \ldots, n$ synchronize their event times, meaning that, for all rounds $k \geq k_0 + n/2$, all the $n/2$ meeting events associated to the round k take place simultaneously at time

$$t_i^e(k) = (k - k_0)t_\star + \max_j\{t_j^e(k_0)\}. \tag{28}$$

After these $n/2$ rounds, from then on, the revisiting times t_{rev} are exactly

$$t_{rev} = 2t_\star. \tag{29}$$

Proof: See Appendix E. ■

Figures 4, 5, and 6 show an example of the previous properties.

VII. Convergence to Interlaced Orientations

In the previous section, we have characterized the system performance, assuming that the robot orientations were interlaced (Def. 6.4). In this section we prove that in fact, the method in Sec. III (Algs. 3.1, 3.2) makes the orientations become interlaced in a finite number of rounds. Then, as stated by Lemma 6.2, the interlaced configuration will be kept for all subsequent rounds, giving the performance in Prop. 6.1, 6.2. As in the previous section, here we assume the conditions in Assumption 6.1 and Def. 6.1 hold, and we analyze the algorithm using the Discrete Synchronous version of the method (Def. 6.3) and the concept of round k (Def. 6.2).

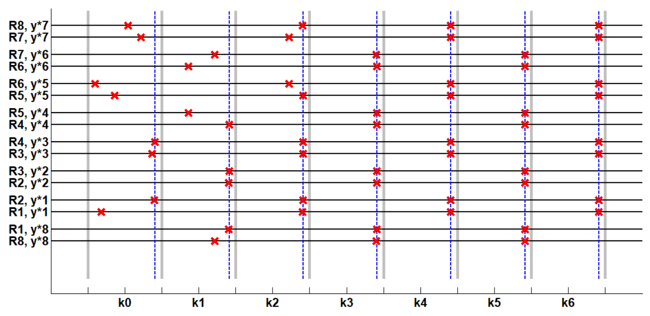


Fig. 6. Time of arrival (red crosses) of each robot i to boundaries $y_{i-1}^\star$ and $y_i^\star$ (y–axis) along time and rounds (x–axis), for the scenario in Fig. 4 **Right** (balanced interlaced orientations). Gray solid vertical lines represent the separation between rounds $k = 0, 1, 2, \ldots$ (Def. 6.2), where we take $k_0 = 0$. At every round k there are $n_{bal} = n/2$ meetings involving $2n_{bal} = n$ robots (red crosses). After $n/2 = 4$ rounds, robots synchronize their arrival times (red crosses) according to eq. (28) (blue dashed vertical lines) as in Prop. 6.2.

Since the interlaced property (Def. 6.4) depends exclusively on the robot orientations, in this section we focus on the evolution of the orientations along different rounds $o_i(k)$. We represent with $+$ and $-$ positive and negative orientations ($o_i(k) > 0$, $o_i(k) < 0$), and study how they evolve under the meeting events (25).

***Definition** 7.1 (Orientation words and sequences):* Given a specific round k, we represent the robot orientations with a word $w(k)$ of length n consisting of $+$ and $-$ characters. We let $w(k)_{j1}$ be the element placed at position $j1$ in $w(k)$, and $w(k)_{j1:j2}$ be the elements in the orientation word $w(k)$ between positions $j1$ and $j2$. We use $[seq, l]$ (*sequence*) to refer to l consecutive elements, which start with a $+$ element, which has the same number of $+$ and $-$ elements, and which is interlaced:

$$[seq, l] = + - \cdots + -. \tag{30}$$

Note that the length l associated to the sequence is an even number ($l = 2, 4, \ldots$). We use the term *letter* to refer to the $+$, $-$ characters in the word $w(k)$ which do not belong to any sequence. There may be several sequences $[seq, l], [seq', l'], [seq'', l''], \ldots$ in $w(k)$ with different lengths, and sequences are separated by one or more letters.

$$w(k) = - \cdots - + \cdots + [seq, l] - \cdots - + \cdots + [seq', l'] \ldots$$

■

Some examples of orientation words and sequences are given in Figure 7.

We first discuss the evolution of the sequences after each round. We show that this evolution depends on the letters around it, being represented by different rules. After this, we discuss the evolution of sequences under these rules, and the effect on the word. We finally discuss the convergence of $w(k)$ to interlaced configurations.

***Lemma** 7.1 (Sequence evolution):* Consider the orientation word $w(k)$ at round k. Every sequence $[seq, l]$ placed in $w(k)$ in positions $w(k)_{j1:j1+l-1}$ evolves between rounds k and $k{+}1$ depending on the letters around it at positions $w(k)_{j1-1}$ and $w(k)_{j1+l}$, according to the following rules:

- *Move+* rule: The sequence moves one position to the left, consuming the $+$ letter to its left, and providing a $+$ letter to its right:

$$\begin{aligned} \text{if } w(k)_{j1-1:j1+l} = &\quad +[seq, l]+ \\ \text{then } w(k+1)_{j1-1:j1+l} = &\quad [seq, l]+(+) \end{aligned}$$

- *Move-* rule: The sequence moves one position to the right, consuming the $-$ letter to its right, and providing a $-$ letter to its left:

$$\begin{aligned} \text{if } w(k)_{j1-1:j1+l} = &\quad -\ [seq, l]- \\ \text{then } w(k+1)_{j1-1:j1+l} = &\quad (-)-[seq, l] \end{aligned}$$

- *Expand* rule: The sequence *Expands*, increasing its length, and consuming both the $+$ letter to its left and the $-$ letter to its right:

$$\begin{aligned} \text{if } w(k)_{j1-1:j1+l} = &\quad +[seq, l]- \\ \text{then } w(k+1)_{j1-1:j1+l} = &\quad [seq, l+2] \end{aligned}$$

- *Reduce* rule: The sequence *Reduces*, decreasing its length, and providing both a $+$ letter to its right and a $-$ letter to its left:

$$\begin{aligned} \text{if } w(k)_{j1-1:j1+l} = &\quad -\ [\ seq, l+2\ ]\ + \\ \text{then } w(k+1)_{j1-1:j1+l} = &\quad (-)-[seq, l]+(+) \end{aligned}$$

  The remaining elements in $w(k)$ not affected by these rules (letters not surrounding sequences), remain the same in $w(k{+}1)$. The elements that appear between parenthesis $(+)$, $(-)$, take in fact these values in $w(k{+}1)$ but, due to possible interactions between sequences (other sequences may consume these elements), they may not be letters but be part of a sequence.
- *Merge* rule: In addition, after *Move+*, *Move-*, or *Expand* rules, the sequence may *Merge* with sequences in the left, the right, or both. The *Merge* rule, if any, takes place at round $k+1$, and it is applied as many times as necessary to ensure sequences are correctly organized.

$$\begin{aligned} \text{if } w(k+1)_{j1:j1+l+l'-l} = &\quad [seq, l][seq', l'] \\ \text{then } w(k+1)_{j1:j1+l+l'-l} = &\quad [\ seq,\ l+l'\ ] \end{aligned}$$

Proof: See Appendix F. ■

***Lemma** 7.2 (Sequence properties):* Consider the orientation word $w(k)$ and the sequences and letters acting according to Lemma 7.1. The following properties hold:

- (i): Sequences move at most one position to the left and/or to the right per round.
- (ii): No new sequences can be created.

Proof: See Appendix F. ■

Lemma 7.1 states the sequence evolution rules between consecutive rounds k and $k+1$. Now, we study how the sequences $[seq, l]$ (Def. 7.1) evolve *along several rounds.* Recall that, without loss of generality, we assume $n_+ \geq n_-$. As we will show, several sequences act in a collaborative way: they move all the $+$ letters they find to their right, or the $-$ letters to their left, to aid the remaining sequences. In addition, several sequences disintegrate, placing their own $+$ and $-$ letters respectively to their right and their left, so that they can be used by the other sequences.

***Lemma** 7.3: (Sequences that Reduce):* As long as a sequence $[seq, l]$ experiences the *Reduce* rule for the first time, it keeps on running the *Reduce* rule until it disappears after $l/2$ rounds. At each of these rounds, this sequence provides a $+$ letter through its right, and a $-$ letter through its left.

Proof: See Appendix F. ■

***Lemma** 7.4: (Sequences that Move-):* As long as a sequence $[seq, l]$ experiences the *Move-* rule for the first time, it keeps on running the *Move-* rule while there are $-$ letters to the right. It eventually disappears (*Reduce*), placing at every round one $+$ letter to its right, and one $-$ letter to its left. If during this process, it *Merges* with another sequence $[seq', l']$, both sequences eventually disappear (*Reduce*).

Proof: See Appendix G. ■

***Lemma** 7.5: (Sequences that Move+):* As long as a sequence $[seq, l]$ experiences the *Move+* rule for the first time, it keeps on running the *Move+* rule while there are $+$ letters to the left. (i) When orientations are balanced as in Def. 4.1 ($n_{bal} = n/2$, i.e., $n_+ = n_-$), the sequence eventually disappears (*Reduce*, Lemma 7.3) providing $+$ letters to its right, and $-$ letters to its left. If during this process, it *Merges* with other sequence $[seq', l']$, both sequences eventually disappear (*Reduce*). (ii) When orientations are unbalanced ($n_{bal} < n/2$) the sequence may either behave as in (i), or it may keep on running the *Move+* rule for ever. If during this process, it *Merges* with other sequence $[seq', l']$, both sequences run the *Move+* rule for ever. This represents interlaced orientations (Def. 6.4) and meetings that take place according to Lemma 6.2 and its proof.

Proof: See Appendix H. ■

Now, we prove that the orientation word $w(k)$ evolves until an interlaced configuration (Def. 6.4) is reached. This is achieved when $w(k)$ is exclusively composed of $+$ letters, and sequences with lengths summing up to $2n_{bal}$ (recall we assume $n_+ \geq n_-$). In our analysis, we will consider the evolution of the sequences (Lemmas 7.3, 7.4, 7.5) and we will show that, from the initial sequences, at least one of them has the property of being expansive, meaning that at every round, it consumes a $+$ and a $-$ letter, and its length is increased by 2, until there are no $-$ letters in the word $w(k)$, i.e., the lengths of the existing sequences sum up to $2n_{bal}$.

***Proposition** 7.1 (Always Expand sequence):* Assume the conditions in Assumption 6.1, Def. 6.1 hold for some round k_0 that we take here as $k_0 = 0$. Consider the initial orientation word $w(0)$ associated to the robot orientations (Def. 7.1). Then: (i) From the initial set of sequences $[seq, l]$ in $w(0)$, at least one sequence experiences the *Expand* rule during all the rounds, until all sequences in $w(k)$ have lengths summing up to $2n_{bal}$. (ii) The number of rounds required to achieve this interlaced configuration is smaller than n_{bal}. (iii) When the orientations are balanced ($n = 2n_{bal}$), after

(i) is achieved, there is a single sequence with length n. (iv) When the orientations are unbalanced with $n_+ > n_-$, after (i) is achieved, then all the sequences run the *Move+* rule for ever. (v) The orientations become *interlaced* (Def. 6.4) in a finite number of rounds.

Proof: See Appendix I. ■

Figure 7 shows some examples of the properties in Lemmas 7.3, 7.4, 7.5, and Prop. 7.1. In the **Left column**, robot orientations are balanced (Def. 4.1). Initially ($k = 0$), there are 7 sequences (in red) and several letters, so that the orientations are not interlaced (Def. 6.4). From $k = 0$ to $k = 1$, four sequences *Expand* (red), and their length increases by two. Also, six sequences *Merge* into three longer sequences (equivalently, three sequences disappear (in black)). During rounds $k = 0, 1, ...$, some sequences *Move+* or *Move-*, keeping their lengths unchanged (in blue), and eventually *Reducing* (in black), making their lengths drop to zero. Only one sequence always *Expands* (always in red), and it reaches finally a length equal to n. The process takes less than $n/2$ rounds, giving rise to interlaced orientations (Def. 6.4). In the **Right column**, orientations are unbalanced, with $n_+ = 20$, $n_- = 12$, $n_{bal} = 12$ (Def. 4.1). Initially ($k = 0$), there are 6 sequences (in red), with lengths summing up to $16 < 2n_{bal} = 24$, so that the orientations are not interlaced (Def. 6.4). Sequences s_1 and s_5 *Expand* (red), and their length increases by two at these rounds. The remaining Sequences experience the *Move+* rule (in blue), keeping their lengths unchanged. After few rounds (from $k = 2$ to $k = 3$), only sequence s_5 *Expands*. Also, sequence s_5 *Merges* with s_6, making s_6 disappear (its length falls to 0). The sequence s_5 that always runs the *Expand* rule, has now length $8+4 = 12$, and the sum of its own length and of the other sequences (2 + 2 + 2 + 12 + 6=24) becomes equal to $2n_{bal} = 24$. The process takes less than n_{bal} rounds. From then on, the five sequences always experience the *Move+* rule, with the orientations interlaced (Def. 6.4).

We are finally ready to present the proofs of the Theorems 4.2 and 4.3 (see Appendix J).

VIII. Simulations

A. Simulations in the $1D$ cycle graph

Figure 8 shows a simulation with $n = 8$ robots traversing a $1D$ cycle graph with length $L = 1000$ meters. The black line between position 0 and L represents the position of robots in the cycle graph (Fig. 1). Robots have maximum speeds $\{v_1, \ldots, v_n\} = \{0.6, 0.1, 0.5, 0.3, 0.7, 0.2, 0.8, 0.4\}m/s$, and communication radii equal to $20m$ apart from $r_3 = 50m$ and $r_7 = 100m$. They start randomly placed in the cycle graph, with their communication regions non overlapping. **Left**: the initial orientations are balanced ($o_1(0), \ldots, o_4(0) = -1$, and $o_5(0), \ldots, o_8(0) = +1$). Asymptotically, robots reach a configuration where their regions have common traversing times $t_\star$ which, in this balance configuration, equals $t_{\text{rev}}/2$ (Theorems 4.1, 4.2). Recall from Sec. II that $t_\star$ and $t_{\text{rev}} = 2t_\star$ is the performance that would be achieved by a centralized system. Thus, the proposed method achieves the same performance as the centralized approach, with lighter memory costs. **Right**: the initial orientations are unbalanced, with $n_{bal} = 3$. The times $e_i(t)$ required to traverse the regions associated to the robots, converge to have common traversing times $t_\star$ (Theorem 4.1). As a metric for the *revisiting time* (Def. 2.1), in our simulations we use the *inter–meeting time* $f_i(t)$, which is the time elapsed between consecutive meetings of robots i and $i+1$, and which is in fact the revisiting time of the $y_i(t)$ boundary. Due to the unbalanced orientations, the inter–meeting times $f_i(t)$ (green dashed), averaged along n_{bal} meetings (blue solid), converge to $t_{\text{rev}} = nt_\star/n_{bal}$ (Theorem 4.3).

We have run a set of simulations, using an increasing number of robots $n = 2, \ldots, 20$ (Fig. 9 (**Left**)). All robots have maximum speed $v_i = 2m/s$ and radius $r_i = 50m$, for $i = 1, \ldots, n$. The number of balanced orientations equals $floor(n/2)$. The length of the cycle graph equals $10Km$. Observe that, as the number of robots increases, the revising time t_{rev} (Th. 4.2, Th. 4.3) decreases. Fig. 9 (**Right**) shows three additional sets of simulations with 6 robots with balanced orientations collaborating to keep intermittently connected a graph cycle with length $10Km$. Robots have maximum speed $v_i = 2m/s$ and radius $r_i = 50m$. In the first set of simulations (in red), we consider different radii for robots $i = 1, 2$, given by $r_1 = r_2 = 50m * factor$. In the second set of simulations (in blue), we keep the radius constant, but we consider different speeds for robots $i = 1, 2$, given by $v_1 = v_2 = 2m/s * factor$. Finally, in the third set of simulations (in black), we let both the radii r_1, r_2 and speeds v_1, v_2 of robots $i = 1, 2$ to change in each case according to the associated factor. The $factor$ used is shown in the $x-$axis (examples $\times\frac{1}{5}$, $\times 3$, and $\times 15$). Observe that, as the robots have improved capabilities, the associated revisiting time t_{rev} decreases, i.e., the performance is improved. Thus, the proposed method (Algs. 3.1, 3.2 in Sec. III) makes a proper use of the available resources.

B. Simulations in Gazebo/ROS

We have conducted experiments using ground robots Turtlebot 3, with differential drive motion, in a planar environment, simulated in Gabezo 7.0, under Ubuntu 16 LTS and ROS Kinetic (Fig. 10).

We briefly explain how the $1D$ position $p_i(t) \in [0, L]$ discussed in the paper is transformed to the $2D$ position $p_i^{xy}(t) \in \mathbb{R}^2$ in the working plane. Each robot i knows the ordered set of $2D$ positions $q_s^{xy} \in \mathbb{R}^2$ that constitute the *cycle graph* (Sec. II), where $q_1^{xy} \in \mathbb{R}^2$ is associated to position 0 in $1D$ and L in $1D$ equals the sum of the lengths of edges connecting the $2D$ positions ($\|q_2^{xy} - q_1^{xy}\| + \|q_3^{xy} - q_2^{xy}\| + \ldots$). Given a $1D$ position $p_i(t) \in [0, L]$, its associated $2D$ position is obtained in two steps. First the particular edge associated to $p_i(t)$ is found, i.e., to find s such that

$$L_s <= p_i(t) <= L_{s+1}, \tag{31}$$

where L_s is the sum of the length of the edges from q_1^{xy}, up to the $2D$ $s-$th position q_s^{xy},

$$L_s = \|q_2^{xy} - q_1^{xy}\| + \cdots + \|q_s^{xy} - q_{s-1}^{xy}\|. \tag{32}$$

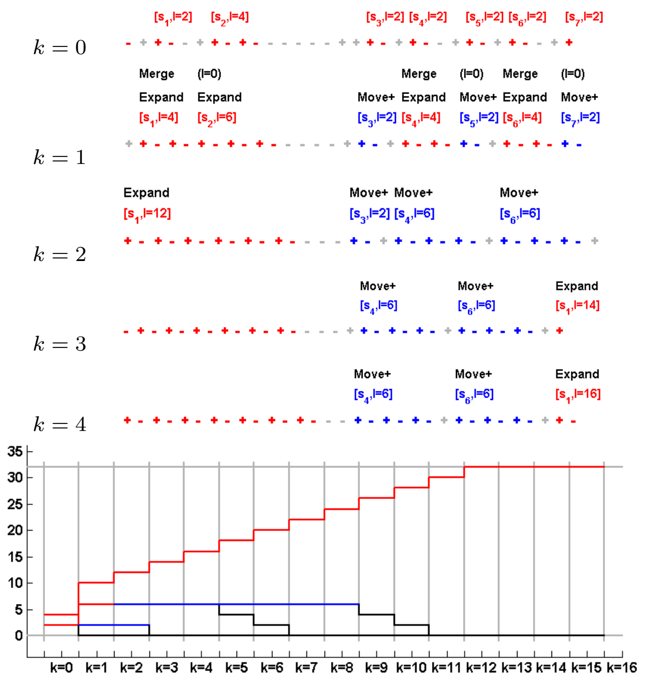
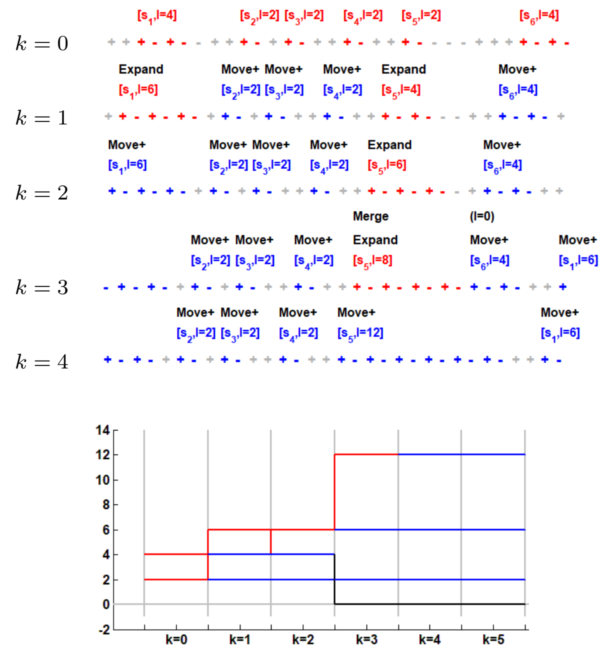



Fig. 7. Interlacing for two examples. 32 robots run the method in Section III (Alg. 3.2) with different maximum speeds and communication radii, under the conditions in Assumption 6.1 and Def. 6.1. In one example (**Left column**), robot orientations are balanced (Def. 4.1), and in the other example (**Right column**) orientations are unbalanced, with $n_+ = 20$, $n_- = 12$, $n_{bal} = 12$. **Top** We show the evolution of the sequences and letters associated to the robot orientations along some rounds, where $+$ means $o_i(k) > 0$ and $-$ means $o_i(k) < 0$. At each round k, we display the rule that was applied to transition from $k-1$ to k. **Bottom**: Evolution along the rounds $k = 0, 1, 2, \ldots$ of the length of every sequence. Note that, initially ($k = 0$), orientations are not interlaced (Def. 6.4).

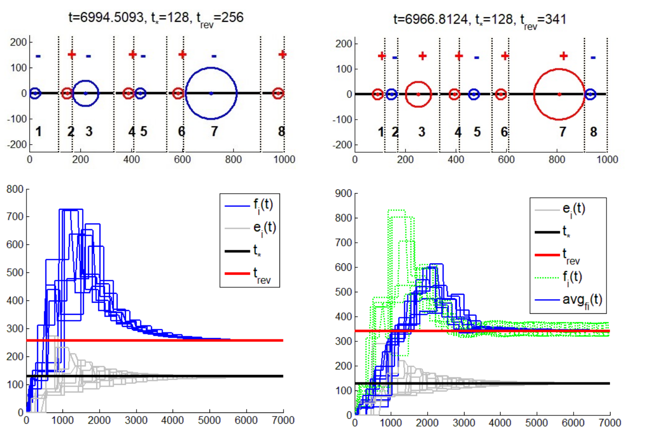


Fig. 8. Robots running Algorithms 3.1, 3.2 in Section III with different maximum speeds and communication radii (circles around the robots). They move forward (red) and backward (blue) at their maximum speeds between their boundaries $y_i(t)$ (black dashed, in vertical), which converge to the boundaries $y_i^\star$ (9)(gray solid, in vertical) associated to the common traversing time $t_\star$ (7). **Left Top:** Final configuration for balanced orientations (Def. 4.1). **Left Bottom:** Evolution of $e_i(t)$ (3) (gray) and $f_i(t)$ (Def. 2.1) (blue) compared to $t_\star$ (black) and $t_{\rm rev}$ (red) for balanced orientations. **Right Top:** Final configuration for unbalanced orientations, with $n_{bal} = 3$. **Right Bottom:** Evolution of $e_i(t)$ (gray), $f_i(t)$ (green dashed), and $f_i(t)$ averaged along n_{bal} meetings (blue solid), compared to $t_\star$ (black) and $t_{\rm rev}$ (red).

Second, the position $p_i^{xy}(t)$ on this edge between the $2D$ positions q_{s+1}^{xy}, q_s^{xy}, is found:

$$p_i^{xy}(t) = q_s^{xy} + (q_{s+1}^{xy} - q_s^{xy})\frac{(p_i(t) - L_s)}{(L_{s+1} - L_s)}. \qquad (33)$$

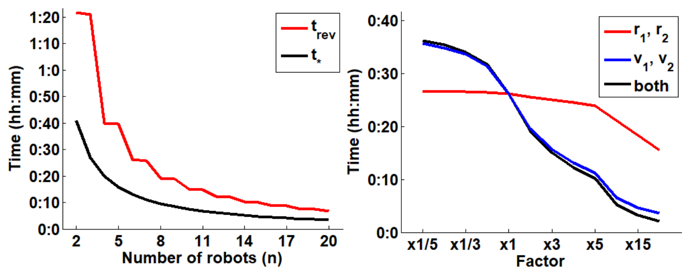


Fig. 9. Results of running several instances of Algorithms 3.1, 3.2 in Section III, using in each case different values of the number of robots n, the maximum speed v_i and radius r_i, in order to analyze the effects of these parameters on the revisiting time $t_{\rm rev}$ (Th. 4.2, Th. 4.3). Robots run Algorithms 3.1, 3.2 in Section III. **Left:** Using a different number of robots n. **Right:** Using a different maximum speed and radius for robots $i = 1, 2$.

Note that the Euclidean distance between any pair of $2D$ robot positions $\|p_{i+1}^{xy}(t) - p_i^{xy}(t)\|$ on the cycle graph is always smaller than or equal to their distance on the $1D$ representation ($|p_i(t) - p_{i+1}(t)|$). Thus, if robots i and $i+1$ are close enough in the $1D$ representation, this immediately means that they are also close enough in the Euclidean sense in what refers to their $2D$ positions. Thus, they can safely exchange data, as required by the method.

Fig. 10 shows the original scenario. The task locations are numbered 1 to 8. The cycle graph used is the TSP associated to the task locations (in yellow, dashed). Robots travel the cycle graph until they reach a boundary, where they perform the needed calculations for the system to converge (Alg. 1,

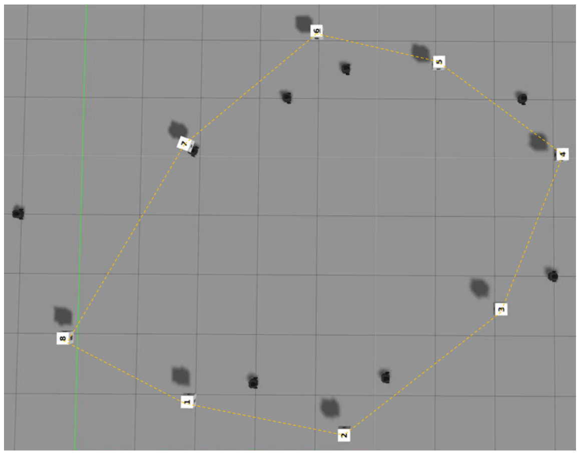


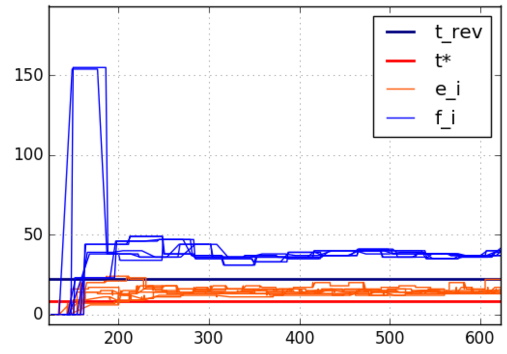
Fig. 10. Gazebo simulation. Example of scenario with 8 robots (*turtlebots*) with balanced orientations (Def.4.1), and with 8 tasks (white dice) on the plane that must be intermittently connected.

2 in Sec. III). This experiment is meant to check that no major problems arise when changing from a non-physical environment like Matlab into a more realistic one, and makes it one step closer to the possibility of real implementation. In the example in Fig. 10, the team is composed of 8 robots, with balanced orientations. The results associated to this case are in Fig. 11(a).

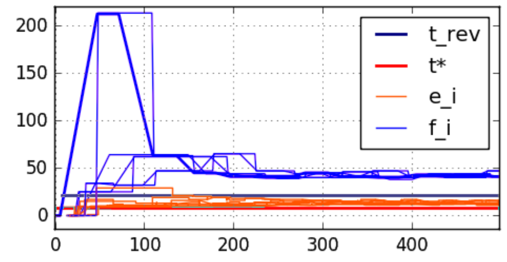
Robots run the proposed method (Sec. III, Algs. 3.1, 3.2). Figure 11 shows the results for a balanced configuration with $n = 8$ robots (a), and for a team of $n = 8$ robots with unbalanced orientations with $n_{bal} = 3$ (b). Note that robots have physical restrictions (they cannot immediately stop or reach their maximum speeds, and their motions are no longer straight lines). Thus, as shown in Figure 11, the traversing $e_i(t)$ (3) and revisiting times $f_i(t)$ (Def. 2.1) converge to values which are close to the theoretical $t_\star$ (7) and t_{rev} (Th. 4.2 and 4.3). These values are slightly larger than the theoretical ones, due to the additional time employed to initiate and stop the motions, to turn around, and to exchange data.

We also include an attached video with a simulation in Gazebo with $n = 8$ robots and 9 tasks. Note that robots are not in charge of any fixed number of tasks. Instead, they are assigned to larger or smaller regions depending on their capabilities. In this example, after some time, one of the robots decreases its maximum speed. As a result, traversing $e_i(t)$ (3) and revisiting times $f_i(t)$ (Def. 2.1) temporarily increase. The other team members self adapt to be in charge of larger regions, succeeding to decrease $e_i(t)$ and $f_i(t)$ after some iterations (Fig. 11 (c)). Thus, the proposed method succeeds to take advantage of the capabilities of all the robots in the team.

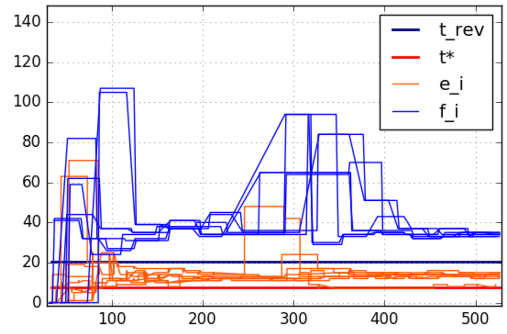
Additional videos with simulations in Gazebo (differential drive robots) and in Matlab (ideal robots) can be found at https://www.youtube.com/playlist?list=PLmyvo-kjDwz30b_i8vW6gW0NeC3NHBvo6 and at https://www.youtube.com/playlist?list=PL2pZRSxEnFj4AfrEbY1bjRUplmaSpZrBc. These videos also include cases like the one in Fig. 10, but with $n = 6$ robots instead of the $n = 8$.



(a) $n = 8$, $n_{bal} = 4$



(b) $n = 8$, $n_{bal} = 3$



(c) $n = 8$, $n_{bal} = 4$. A robot decreases its speed (attached video).

Fig. 11. Evolution of $e_i(t)$ (3) and $f_i(t)$ (Def. 2.1) compared to $t_\star$ (7) and t_{rev} (Th. 4.2 and 4.3). (a): Balanced orientations with $n = 8$. (b): Unbalanced case, with $n = 8$ and $n_{bal} = 3$. (c): Balanced orientations with $n = 8$ robots, and 9 tasks. After some time, one robot decreases its speed and the robot team self–adapts to this new situation (attached video).

IX. Conclusions

We presented a method to ensure a robotic network is kept intermittently connected. Robots move forward and backward on their regions, meeting intermittently with their previous and next neighbors. Simultaneously, they run a weighted consensus method to update their boundaries, so that the final regions associated to each robot can be traversed by them in a common time, that depends on the robots maximum speeds and communication radii. For balanced situations with the same number of robots moving forward and backward, robots converge to a configuration where the meetings take place simultaneously in the network and robots never wait at the region boundaries. For unbalanced situations, the performance degrades depending on how unbalanced the situation is. For this reason, future extensions of this work include the design of distributed methods where robots change their orientations in order to get as close as possible to balanced configurations.

Appendix A: Proof of Proposition 5.1

In this section, we let matrix V, scalar ϵ_i, and matrices P_i, $\tilde{P}_i$, $\mathcal{L}_i$, $\tilde{\mathcal{L}}_i$ associated to the link $(i, i+1)$, be

$$\begin{aligned} &V = \operatorname{diag}(v_1, \ldots, v_n), \quad \epsilon_i = (v_i v_{i+1})/(v_i + v_{i+1}), \\ &P_i = \mathbf{I} - (\operatorname{diag}(v_1, \ldots, v_n))^{-1} \epsilon_i \mathcal{L}_i, \\ &\tilde{P}_i = V^{1/2} P_i V^{-1/2} = \mathbf{I} - \tilde{\mathcal{L}}_i, \quad \tilde{\mathcal{L}}_i = V^{-1/2} \epsilon_i \mathcal{L}_i V^{-1/2}, \end{aligned} \tag{34}$$

where the entries in P_i, $\tilde{P}_i$, $\mathcal{L}_i$, and $\tilde{\mathcal{L}}_i$ are given by

$$\begin{aligned} &[P_i]_{j,j'} = 0, \text{ except for} \\ &[P_i]_{i,i} = 1 - (\epsilon_i / v_i), \quad [P_i]_{i+1,i+1} = 1 - (\epsilon_i / v_{i+1}), \\ &[P_i]_{i,i+1} = \epsilon_i / v_i, \quad [P_i]_{i+1,i} = \epsilon_i / v_{i+1}, \\ &[P_i]_{j,j} = 1, \text{ for all } j \neq i, j \neq i+1, \end{aligned} \tag{35}$$

$$\begin{aligned} &[\mathcal{L}_i]_{j,j'} = 0, \text{ except for} \\ &[\mathcal{L}_i]_{i,i} = 1, [\mathcal{L}_i]_{i+1,i+1} = 1, [\mathcal{L}_i]_{i,i+1} = -1, [\mathcal{L}_i]_{i+1,i} = -1, \end{aligned} \tag{36}$$

$$\begin{aligned} &[\tilde{\mathcal{L}}_i]_{j,j'} = 0, \text{ except for} \\ &[\tilde{\mathcal{L}}_i]_{i,i} = v_{i+1}/(v_i + v_{i+1}), \quad [\tilde{\mathcal{L}}_i]_{i+1,i+1} = v_i/(v_i + v_{i+1}), \\ &[\tilde{\mathcal{L}}_i]_{i,i+1} = [\tilde{\mathcal{L}}_i]_{i+1,i} = -\sqrt{v_i}\sqrt{v_{i+1}}/(v_i + v_{i+1}), \end{aligned} \tag{37}$$

$$\begin{aligned} &[\tilde{P}_i]_{j,j'} = 0, \text{ except for} \\ &[\tilde{P}_i]_{i,i} = 1 - \frac{v_{i+1}}{v_i + v_{i+1}}, \quad [\tilde{P}_i]_{i+1,i+1} = 1 - \frac{v_i}{v_i + v_{i+1}}, \\ &[\tilde{P}_i]_{i,i+1} = [\tilde{P}_i]_{i+1,i} = \sqrt{v_i}\sqrt{v_{i+1}}/(v_i + v_{i+1}), \\ &[\tilde{P}_i]_{j,j} = 1, \text{ for all } j \neq i, j \neq i+1. \end{aligned} \tag{38}$$

***Lemma** A.1:* [21] The eigenvalues of the matrices P_i, $\tilde{P}_i$ defined in (34), for all $(i, i+1)$ links, with $i = 1, \ldots, n-1$, satisfy:

$$\lambda(P_i) \in (-1, 1] \qquad \lambda(\tilde{P}_i) \in (-1, 1]. \tag{39}$$

Proof: The eigenvalues of $\tilde{P}_i$ are

$$\lambda(\tilde{P}_i) = 1 - \lambda(V^{-1/2} \epsilon_i \mathcal{L}_i V^{-1/2}) = 1 - \lambda(\tilde{\mathcal{L}}_i). \tag{40}$$

Note that $\mathcal{L}_i$ (34), (36) is the unweighted symmetric Laplacian matrix associated to the link $(i, i+1)$, and thus it is positive semidefinite [24]. Since $\epsilon_i > 0$, and matrix $V^{-1/2}$ is positive definite and symmetric, then $\tilde{\mathcal{L}}_i = \lambda(V^{-1/2} \epsilon_i \mathcal{L}_i V^{-1/2})$ (34) (37) is positive semidefinite [28, Chapter 7.1], with eigenvalues larger than or equal to 0, and with its largest eigenvalue begin smaller than or equal to the infinite matrix norm $\|\tilde{\mathcal{L}}_i\|_\infty = \max_j(|[\tilde{\mathcal{L}}_i]_{j1}| + \cdots + |[\tilde{\mathcal{L}}_i]_{jn}|)$,

$$\begin{aligned} \lambda_{\max}(\tilde{\mathcal{L}}_i) \leq \|\tilde{\mathcal{L}}_i\|_\infty &= \frac{\max(v_i, v_{i+1}) + \sqrt{v_i}\sqrt{v_{i+1}}}{v_i + v_{i+1}} \\ &\leq (\max(v_i, v_{i+1}) + \max(v_i, v_{i+1}))/(v_i + v_{i+1}) < 2. \end{aligned} \tag{41}$$

From eqs. (40),(41), for all $i = 1, \ldots, n-1$,

$$-1 = (1 - 2) < \lambda(\tilde{P}_i) \leq (1 - 0) = 1, \tag{42}$$

and since P_i is *similar* to $\tilde{P}_i$ (34), then P_i and $\tilde{P}_i$ have the same eigenvalues, and thus we conclude (39). ■

***Proposition** A.1:* [21] Let matrices $P_{i(t)}$, $\tilde{P}_{i(t)}$ be as in (34). If the set of matrices that appear infinitely often are jointly connected, then, for all $z(0)$, $e(0)$, the iterations $z(t^+) = \tilde{P}_{i(t)} z(t)$, $e(t^+) = P_{i(t)} e(t)$, with

$$z(t) = V^{1/2} e(t), \qquad e(t) = V^{-1/2} z(t), \tag{43}$$

and V as in (34), converge respectively to

$$\begin{aligned} z_\star &= (V^{1/2} \mathbf{1}\mathbf{1}^T V^{1/2} / \mathbf{1}^T V \mathbf{1}) z(0), \\ e_\star &= (\mathbf{1}\mathbf{1}^T / \mathbf{1}^T V \mathbf{1}) V e(0). \end{aligned} \tag{44}$$

Proof: Consider the matrix $\tilde{P}_{i_1:i_j}$ associated to a particular jointly connected sequence $i_j \ldots i_1$ (the sequence takes place in the opposite order to matrix multiplication),

$$\tilde{P}_{i_1:i_j} = \tilde{P}_{i_1} \tilde{P}_{i_2} \ldots \tilde{P}_{i_j}. \tag{45}$$

For this matrix,

$$\begin{aligned} \rho(\tilde{P}_{i_1:i_j}) &\leq \|\tilde{P}_{i_1:i_j}\|_2 \leq \|\tilde{P}_{i_1}\|_2 \|\tilde{P}_{i_2}\|_2 \ldots \|\tilde{P}_{i_j}\|_2 \\ &= \rho(\tilde{P}_{i_1}) \rho(\tilde{P}_{i_2}) \ldots \rho(\tilde{P}_{i_j}) = 1, \end{aligned} \tag{46}$$

where we have used Lemma A.1 ($\lambda(\tilde{P}_i) \in (-1, 1]$ for all $i = 1, \ldots, n-1$), and the fact that $\tilde{P}_{i_1}, \tilde{P}_{i_2}, \ldots, \tilde{P}_{i_j}$ are symmetric, and their spectral norms equal their spectral radius, $\|\tilde{P}_i\|_2 = \rho(\tilde{P}_i) = \max(|\lambda(\tilde{P}_i)|)$. Thus, all the eigenvalues of matrix $\tilde{P}_{i_1:i_j}$ are between $[-1, +1]$.

Now we pay attention to the structure of matrix $\tilde{P}_{i_1:i_j}$. Every matrix $\tilde{P}_i$ (38) has all the entries equal to zero, apart from the diagonal terms $(1,1) \ldots (n,n)$, and the entries $(i, i+1)$, $(i+1, i)$, which are strictly positive. After multiplying matrices $\tilde{P}_{i_1}, \ldots, \tilde{P}_{i_j}$, we get a nonnegative matrix $\tilde{P}_{i_1:i_j}$ that has *at least* the following elements strictly positive (the remaining entries may be zero or positive): the diagonal terms $(1,1) \ldots (n,n)$, and all the $(i, i+1)$ and $(i+1, i)$ entries associated to the $i, i+1$ links that appear in each associated matrix P_i, for $i = i_1 \ldots i_j$. Matrix $\tilde{P}_{i_1:i_j}$ contains at least all matrices associated to the $n-1$ different links $(i, i+1)$, for all $i = 1, \ldots, n-1$. Then, the structure of $\tilde{P}_{i_1:i_j}$ contains *at least* positive elements in all the entries (i, i) for $i = 1, \ldots, n$, and $(i, i+1)$, $(i+1, i)$ for $i = 1, \ldots, n-1$. Thus, matrix $\tilde{P}_{i_1:i_j}$ is *primitive* and [29], [28] among its n eigenvalues, there is exactly one with the largest magnitude, and this eigenvalue is the only one possessing an eigenvector with all positive entries, and the remaining $n-1$ eigenvalues are all strictly smaller in magnitude than the largest one. From (46), this eigenvalue has modulus smaller than or equal to 1.

Now note that for each matrix P_i (34),

$$\begin{aligned} &P_i \mathbf{1} = \mathbf{1}, \qquad \text{and that} \\ &\tilde{P}_{i_1:i_j} = \tilde{P}_{i_1} \tilde{P}_{i_2} \ldots \tilde{P}_{i_j} = V^{1/2} P_{i_1} P_{i_2} \ldots P_{i_j} V^{-1/2}. \end{aligned} \tag{47}$$

From (47), we conclude that $V^{1/2}\mathbf{1}$ is the eigenvector of $\tilde{P}_{i_1:i_j}$ associated to the eigenvalue 1,

$$\tilde{P}_{i_1:i_j} V^{1/2} \mathbf{1} = V^{1/2} \mathbf{1}. \tag{48}$$

This eigenvector has all its entries positive, and it is associated to the largest modulus eigenvalue, which has to be 1 and not -1. Matrix $\tilde{P}_{i_1:i_j}$ is also paracontractive (e.g., [30, Corollary 2], using $\operatorname{span}(V^{1/2}\mathbf{1})$ instead of $\operatorname{span}(\mathbf{1})$).

From [26, Theorem 1] [27, Theorem 2]: suppose that a finite set of square matrices $\{W_1,\dots,W_j\}$ are paracontractive, and denote $\mathcal{J}$ the set of integers that appear infinitely often in the sequence. Then, for all $\tilde{z}(0)$, the sequence of vectors $\tilde{z}(t_e^+) = W_{i(t_e)}\tilde{z}(t_e)$ has a limit $z_\star \in \cap_{i\in\mathcal{J}}\mathcal{H}(W_i)$, with $\mathcal{H}(W_i) = \{z | W_i z = z\}$. In our case, we use the fact that all our possible jointly connected matrices have the common eigenvector $V^{1/2}\mathbf{1}$ associated to the eigenvalue 1 and it is the only one. Thus, $\tilde{z}(t_e^+) = \tilde{P}_{i_1:i_j(t_e)}\tilde{z}(t_e)$, and thus $z(t_e^+) = \tilde{P}_{i(t_e)}z(t_e)$, converge to $z_\star$ in (44). Due to (43), $e(t_e^+) = P_{i(t_e)}e(t_e)$ converges to $e_\star$ (44). ■

Using the previous intermediary results, we are ready to prove Proposition 5.1.

Proof of Proposition 5.1: We first consider how the region lengths $d_i(t)$, $d_{i+1}(t)$ (eq. (2)) evolve when robots i, $i+1$, $i \in 1,\dots,n-1$, update their boundary $y_i(t_e^+)$ with eq. (16) (the other region lengths are not affected by (16)). If several simultaneous events take place, we will consider any ordering, e.g., first the ones with lower identifiers. Note that (49) will be the same in the presence of simultaneous updates, since it depends on boundaries which require actions from robots i and $i+1$ and, since they are currently involved in their meeting at the common boundary $y_i(t)$, they cannot be simultaneously involved in other meetings at the other boundaries (i.e., robot i is not at boundary $y_{i-1}(t)$, and robot $i+1$ is not at $y_{i+1}(t)$),

$$d_i(t_e^+) = y_i(t_e^+) - y_{i-1}(t_e^+) = y_i(t_e^+) - y_{i-1}(t_e), \quad (49)$$
$$d_{i+1}(t_e^+) = y_{i+1}(t_e^+) - y_i(t_e^+) = y_{i+1}(t_e) - y_i(t_e^+).$$

After some manipulation, (49) is equivalent to:

$$d_i(t_e^+) = \frac{v_i(d_{i+1}(t_e)+d_i(t_e))}{v_i+v_{i+1}} + 2\frac{v_{i+1}r_i - v_i r_{i+1}}{v_i+v_{i+1}}, \quad (50)$$
$$d_{i+1}(t_e^+) = \frac{v_{i+1}(d_{i+1}(t_e)+d_i(t_e))}{v_i+v_{i+1}} - 2\frac{v_{i+1}r_i - v_i r_{i+1}}{v_i+v_{i+1}}.$$

The traversing times $e_i(t)$, $e_{i+1}(t)$ (eq. (3)) of robots i and $i+1$ are also affected by (16), due to (50) (the other traversing times are not affected by (16)):

$$e_i(t_e^+) = e_i(t_e) + \frac{\epsilon_i}{v_i}(e_{i+1}(t_e) - e_i(t_e)), \; \epsilon_i = \frac{v_i v_{i+1}}{v_i+v_{i+1}},$$
$$e_{i+1}(t_e^+) = e_{i+1}(t_e) - \frac{\epsilon_i}{v_{i+1}}(e_{i+1}(t_e) - e_i(t_e)). \quad (51)$$

In matrix form, eq. (51) is a discrete–time switching weighted consensus, with Perron matrix [24] P_i as in (34) found in Appendix A,

$$e(t_e^+) = P_{i(t_e)}e(t_e), \quad e(t) = [e_1(t),\dots,e_{n-1}(t)]^T. \quad (52)$$

From Proposition A.1 in Appendix A, if the set of communication graphs that occur infinitely often are jointly connected, then $e(t)$ in (52) converges to $e_\star$ in (44). Thus, for all $i = 1,\dots,n$, $e_i(t)$ defined by eq. (3) and evolving as in (51), converge to the weighted average of $e_j(0)$, with weighting vector given by $[v_1,\dots,v_n]^T$ (Proposition A.1 in the Appendix and eq. (34)),

$$\lim_{t\to\infty} e_i(t) = \frac{\sum_{j=1}^n v_j e_j(0)}{v_1+\cdots+v_n} = \frac{\sum_{j=1}^n \frac{v_j(d_j(0)-2r_j)}{v_j}}{v_1+\cdots+v_n} = t_\star, \quad (53)$$

since $d_1(0)+\cdots+d_n(0) = y_n(t) = L$ (eq. (2)), with $t_\star$ as in (7), and thus, the region lengths $d_i(t)$, and the boundaries $y_i(t)$ converge to the values in (8), (9). ■

Appendix B: Proofs of Lemmas 5.1, 5.2 and 5.3

Proof of Lemma 5.1 [21]: Since L is fixed, and $y_n(t) = L$ is fixed then, for all $i = 1,\dots,n$, $d_i(t) \leq L$ and thus $e_i(t) \leq L/v_i$ (3). Active robots move from a position inside their region to one of their boundaries, employing thus a time less or equal to $e_i(t)$, which as we saw, is bounded. After that, the event is an *arrival* if the boundary is empty, and a *meeting* if the neighbor is already waiting at the boundary. ■

Proof of Lemma 5.2 [21]: The proof follows by inspection of Def. 5.1 and Algorithm 3.2. ■

Proof of Lemma 5.3 [21]: (i) *About the orientations*: Orientations $o_i(t)$ only change during *discovery* and *meeting* events and, in both cases (eqs. (11), (17)), the orientations of the two involved agents i and $i+1$ are simultaneously changed, and thus the sum of all orientations remains unchanged.

(ii) *About the regions being disjoint, with the only common point being the boundary*: This is true during the *discovery* and *catch*, where robots i, $i+1$ define their common boundary at the same time. After, at meetings (eq. (16)) robots i, $i+1$ change simultaneously their common boundary $y_i(t_e^+)$, and the update rule makes $y_i(t_e^+)$ remain strictly between $y_{i-1}(t_e^+) = y_{i-1}(t_e)$ and $y_{i+1}(t_e^+) = y_{i+1}(t_e)$.

(iii) *About robots not exchanging positions*: The region associated to each robot i is defined by the boundaries $y_{i-1}(t)$ and $y_i(t)$. After meeting with robot $i+1$, the position of $y_i(t_e^+)$ changes (eq. (16)), with $y_i(t_e^+) \in [y_{i-1}(t_e), y_{i+1}(t_e)]$. Then, robot i goes to its other boundary $y_{i-1}(t) \leq y_i(t)$, and when later it comes back to boundary $y_i(t)$, it holds $y_i(t) \leq y_{i+1}(t)$, regardless of the fact that robot $i+1$ may have updated or not $y_{i+1}(t)$. Thus, robot i will reach $y_i(t)$ and will never reach $y_{i+1}(t)$. Thus, in the discrete asynchronous behavior (Def. 5.1), robots do not exchange the positions. ■

Appendix C: Proof of Proposition 5.2

Proof of Proposition 5.2: Considering the discrete asynchronous behavior (Def. 5.1) of the algorithm, we represent the system as n boundaries and n robots placed at the boundaries. As we show next, the system cannot experience blocking, and the meetings propagate through the network.

Blocking: The only possible blocking situation is one with each robot waiting at a different boundary since, as long as two robots fall in a common boundary, a *meeting* event takes place, making the system evolve. At the blocking, robots with $o_i(t) > 0$ would be at their right boundary, and robots with $o_i(t) < 0$ at their left boundary, with these boundaries being the unique common points between the disjoint regions of each robot (Lemma 5.3(ii)). From Assumption $(A2)$, at least one robot has a different initial orientation than the others, and by Lemma 5.3 (i) this continues during the whole execution of the method. Thus, at least two robots will reach the same boundary, giving rise to a *meeting* event.

Propagation: After robots i, $i+1$ meet, they move to their opposite boundaries, thus propagating the process to the preceding and following robots $i-1$ and $i+2$, since the order of the robots is preserved (Lemma 5.3 (iii)). Repeating the same reasoning with robots $i-1$, $i+2$, we conclude that meetings are propagated through the network. The only reasons not to

propagate would be either a blocking (we proved this is not possible), or that the same subset of robots would be meeting each other, without involving the others. But in order for $i, i+1$ to meet again, there must have been a meeting between $i-1, i$ and $i+1, i+2$, so this case is discarded as well.
Bounded times: The discrete asynchronous behavior (Def. 5.1) does not exhibit blocking and ensures propagation of the meetings, and the time between events is bounded (Lemma 5.1). Therefore, the time interval required for the network to be jointly connected is bounded. ■

Appendix D: Proofs of Lemmas 6.1 and 6.2

Proof of Lemma 6.1: (i) Let t_e be the time at which the meeting between robots i, $i+1$ takes place within the current round k, with

$$t_0 + kt_\star \leq t_e < t_0 + (k+1)t_\star. \tag{54}$$

Note that the time t_e of the meeting of robots i, $i+1$ is associated to the last robot that arrived to the boundary, so that $t_e = \max\{t_i^e(k), t_{i+1}^e(k)\}$ in (25). From (21), with $e_i(t_e^+) = t_\star$ (Asm. 6.1), the arrivals of robots i, $i+1$ to the opposite boundaries will take place at time

$$t_{e'} = t_e + t_\star = \max\{t_i^e(k), t_{i+1}^e(k)\} + t_\star, \tag{55}$$

giving $t_{e'} = t_i^e(k+1) = t_{i+1}^e(k+1)$ in (25). From eq. (54), these arrival times $t_{e'} = t_i^e(k+1) = t_{i+1}^e(k+1)$ take place during the round $k+1$:

$$t_0 + (k+1)t_\star \leq t_e + t_\star < t_0 + (k+2)t_\star. \tag{56}$$

Since between the meeting time t_e at round k and the arrival time $t_{e'}$ at round $k+1$ robots i and $i+1$ are traveling between boundaries, they cannot be involved in any additional event (arrivals or meetings). This concludes the proof of (i).
(ii) Due to the previous property, a robot i cannot be involved in more than a meeting during the current round. Since meetings affect at most the two robots involved, their state updates can be performed independently of the state updates of the other robots involved in meetings during the current round. The proof is completed by considering this observation, together with the equivalence of the event times discussed in the proof of (i), and the definition of the Discrete Synchronous Behavior (Def. 6.3 and eq. (25)).
(iii) Both states are simultaneously updated (25), and both represent the fact that, during round k, both robots will be at or arrive to the common boundary $y_i^\star$ and a meeting will take place. ■

Proof of Lemma 6.2: At the beginning of round k, orientations are interlaced as in Definition 6.4 with $n_+ \geq n_-$. Then, during the current round k, all robots with indexes i_j, for all $j = 1, \ldots, n_{bal}$, as in (26), will arrive to the boundary $y_{i_j}^\star$, and all robots with indexes $i_j + 1$ will arrive to the boundary $y_{i_j}^\star$. Thus, during the current round k there will be n_{bal} meetings at these boundaries, involving $2n_{bal}$ robots. Due to these meetings (eq. (25)), the involved robots will arrive to the opposite boundaries at round $k+1$ and will change their orientations (Lemma 6.1), i.e., for all $j = 1, \ldots, n_{bal}$:

$$o_{i_j}(k+1) = -1 \text{ and } o_{i_j+1}(k+1) = +1. \tag{57}$$

Since $n_+ \geq n_-$ and, from Lemma 5.3, n_+, n_-, n_{bal} remain unchanged for all rounds and times, then the orientations of the remaining robots at round $k+1$ are positive. In particular, $o_{i_1-1} = +1$, and from (57), during round $k+1$, the orientations are interlaced with robot indexes $i_1 - 1, i_2 - 1, \ldots, i_{n_{bal}} - 1$, i.e., for all $j = 1, \ldots, n_{bal}$:

$$o_{i_j-1}(k+1) = +1 \text{ and } o_{i_j}(k+1) = -1. \tag{58}$$

The proof is completed by repeating the reasoning for the successive rounds. ■

Appendix E: Proofs of Propositions 6.1 and 6.2

Proof of Prop. 6.1: It is a consequence of Lemma 6.2: each robot i arrives to the $y_i^\star$ boundary n_{bal} times every n rounds. And from Assumption 6.1 and Def. 6.2, the duration of each round equals $t_\star$. ■

Proof of Prop. 6.2: We use Lemma 6.2 with $n_{bal} = n/2$ since orientations are balanced. Since at round k_0 the orientations are balanced interlaced, from Lemma 6.2 they remain balanced interlaced for all the successive rounds. At each round k there are exactly $n/2$ meetings involving n robots. Thus, if at round k the robot indexes are $i_1, i_2, \ldots, i_{n/2} = \{1, 3, \ldots, n-1\}$, at round $k+1$ the robot indexes are $i_1 - 1, i_2 - 1, \ldots, i_{n/2} - 1 = \{0, 2, \ldots, n-2\}$, i.e., due to the cycle structure they equal to $\{2, \ldots, n-2, n\}$. Thus, every robot i meets at every round with one of its neighbors, and with the opposite neighbor during the next round.

Now consider the arrival of robots i and $i+1$ at the common boundary $y_i^\star$ at round k, which takes place at times $t_i^e(k)$ and $t_{i+1}^e(k)$. This will give rise to a meeting during the current round k that will take place at time $\max\{t_i^e(k), t_{i+1}^e(k)\}$. Equivalently, this will give rise to two arrivals to boundaries (the opposite ones) that will take place at times (25)

$$t_i^e(k+1) = t_{i+1}^e(k+1) = \max\{t_i^e(k), t_{i+1}^e(k)\} + t_\star. \tag{59}$$

Now note that, at every round, every robot i meets exactly once, so that eq. (59) adds $t_\star$ to every $t_i^e(k)$ at every round. Note also that during round k, robot i met with robot $i-1$, and robot $i+1$ met with robot $i+2$. Thus, (59) gives

$$t_i^e(k+1) = 2t_\star + \max_{j=i-1}^{i+2}\{t_j^e(k-1)\}. \tag{60}$$

During round $k-1$, the states of the robots involved in (60) were updated due to meetings between robots $(i-2, i-1)$, $(i, i+1)$, $(i+2, i+3)$, so that

$$t_i^e(k+1) = 3t_\star + \max_{j=i-2}^{i+3}\{t_j^e(k-2)\},$$

and, for k large enough, the reasoning can be repeated until it is propagated to the set of initial values $t_j^e(k_0)$. Thus, we can see that (59) is equivalent to a $\max$ consensus [31] on the initial values $t_1^e(k_0), \ldots, t_n^e(k_0)$, plus the term $(k - k_0)t_\star$ which places the event within the current round. Due to the graph structure of the network, its diameter equals $n/2$, and thus, after $n/2$ iterations the $\max$ consensus converges to the maximal initial value, and equivalently, all the time events associated to the arrivals take place at:

$$t_i^e(k) = (k - k_0)t_\star + \max_j\{t_j^e(k_0)\}, \text{ for all } i = 1, \dots, n. \tag{61}$$

Thus, from each pair of simultaneous arrivals, the meeting takes place immediately. From this observations, we get the revisiting times $2t_{rev} = 2t_\star$. ■

Appendix F: Proofs of Lemmas 7.1, 7.2 and 7.3

Proof of Lemma 7.1: From (25), at every round $k+1$, all the letters in $w(k)$ remain the same in $w(k+1)$, and all the sequences in $w(k)$ reverse their values ($+$ become $-$ and $-$ become $+$) due to the meetings that take place at round k. The previous rules follow from direct application of this fact. ■

Proof of Lemma 7.2: (i): The proof is immediate, by looking at the rules. (ii): The creation of a new sequence requires $+-$ to appear at round $k+1$. By inspection of the rules, this cannot happen, neither by the individual application of the rules, or by the interactions between the rules of different sequences. ■

Proof of Lemma 7.3: From Lemmas 7.1, 7.2, sequences move at most one position per round and, a sequence that experiences a *Reduce* rule, generates letters $-$ and $+$ around it which cannot be consumed in a single round by other sequences. Thus, in round $k+1$ it will *Reduce* again, and again at $k+2, \dots$ until it disappears. ■

Appendix G: Proof of Lemma 7.4

Proof of Lemma 7.4: Consider sequence $[seq, l]$ experiences the *Move-* rule and there are additional $-$ letters to its right. This sequence always introduces a $-$ letter to the left. Since every other sequence $[seq', l']$ to the left of $[seq, l]$ cannot move faster than 1 position per round (Lemma 7.2) then, even if this other sequence consumes this recent letter, it cannot consume it faster than sequence $[seq, l]$ creates it. Thus, as long as a sequence $[seq_j, l]$ experiences the *Move-* rule for the first time, it keeps on running the *Move-* rule while there are $-$ letters to the right. We consider sequence $[seq, l]$ has already consumed all the extra $-$ letters to its right, placing them to its left. The different situations that it can find are described and analyzed next:

$$\begin{aligned}(a) \text{ if } w(k)_{j1:j1+l+l'+2} &= \; -[seq,l]-[seq',l']- \\ \text{then } w(k+1)_{j1:j1+l+l'+2} &= \; (-)-[seq,l]-[seq',l']\end{aligned}$$

Both sequences *Move-* to the right, until something happens to $[seq', l']$ that makes $[seq, l]$ act accordingly. Since $n_+ \geq n_-$, at some point the sequence $[seq', l']$ will meet a $+$ and will act as described in case (b).

$$\begin{aligned}(b) \text{ if } w(k)_{j1:j1+l+l'+2} &= \; -[seq,l]-[seq',l']+ \text{ then} \\ w(k+1)_{j1:j1+l+l'+2} &= \; (-)-[seq,l]-[seq',l'-2]+(+)\end{aligned}$$

Sequence $[seq', l']$ *Reduces*, feeding with additional $-$ to $[seq, l]$ until $[seq', l']$ disappears (Lemma 7.3) and $[seq, l]$ finds the first $+$. This case (e) is explained later.

$$\begin{aligned}(c) \text{ if } w(k)_{j1:j1+l+l'+3} &= \; -[seq,l]-+[seq',l']- \\ \text{then } w(k+1)_{j1:j1+l+l'+3} &= \; (-)-[\; seq', l'+l+2 \;]\end{aligned}$$

Sequence $[seq', l']$ *Expands*, $[seq, l]$ *Move-*, and then both *Merge* in a larger sequence. Note this merged sequence has a $-$ to the left. Thus (Lemma 7.1), it will either *Reduce* (discussed in Lemma 7.3) or start performing *Move-* (behaving according to the different cases discussed in this proof).

$$\begin{aligned}(d) \text{ if } w(k)_{j1:j1+l+l'+3} &= \; -[seq,l]-+[seq',l']+ \\ \text{then } w(k+1)_{j1:j1+l+l'+3} &= \; (-)-[seq',l'+l]+(+)\end{aligned}$$

Sequence $[seq, l]$ *Move-*, $[seq', l']$ *Move+*, and then both *Merge* in a larger sequence. Since this sequence has a letter $-$ to the left and $+$ to the right, then it will *Reduce*, behaving as discussed in Lemma 7.3.

$$\begin{aligned}(e) \text{ if } w(k)_{j1:j1+l+3} &= \; -\;[seq,l]\;-++ \\ \text{then } w(k+1)_{j1:j1+l+3} &= \; (-)-[seq,l]+(+)\end{aligned}$$

Finally, we consider that sequence $[seq, l]$ has some $+$ letters to the right that do not belong to sequences (thus, there are at least two consecutive $+$ letters). It makes a *Move-*, and finishes with a $-$ to the left and $+$ to the right. Then, it will *Reduce*, behaving as discussed in Lemma 7.3. ■

Appendix H: Proof of Lemma 7.5

Proof of Lemma 7.5: The proof is organized by considering each case separately. Consider sequence $[seq, l]$ experiences the *Move+* rule and there are additional $+$ letters to its left. This sequence always introduces a $+$ letter to the right. Since every other sequence $[seq', l']$ to the right of $[seq, l]$ cannot move faster than 1 position per round (Lemma 7.2) then, even if this other sequence consumes this recent letter, it cannot consume it faster than sequence $[seq, l]$ creates it. Thus, as long as a sequence $[seq_j, l]$ experiences the *Move+* rule for the first time, it keeps on running the *Move+* rule while there are $+$ letters to its left, placing them to its right.

When the orientations are unbalanced, with $n_+ > n_-$, it may be the case that there are always $+$ letters to the left of the sequence, so that it always runs the *Move+* rule for ever. Otherwise, the sequence consumes $+$ letters until some of the following five situations take place:

$$\begin{aligned}(a) \text{ if } w(k)_{j1:j1+l+l'+2} &= \; +[seq',l']+[seq,l]+ \\ \text{then } w(k+1)_{j1:j1+l+l'+2} &= \; [seq',l']+[seq,l]+(+)\end{aligned}$$

Both sequences *Move+* to the left, until something happens to $[seq', l']$ that makes $[seq, l]$ act accordingly.

When the orientations are balanced, $n_+ = n_-$, i.e., there are as many $+$ as $-$. Thus, at some point the sequence $[seq', l']$ will meet a $-$ and will act as described next in (b). When the orientations are unbalanced, $n_+ > n_-$, i.e., there are more $+$ letters than $-$ letters. Sequence $[seq', l']$ may meet a $-$, behaving both sequences as described next in (b). It may be also the case that sequence $[seq', l']$ only meets $+$ letters; then, both sequences will behave for ever running the *Move+* rule.

$$(b) \text{ if } w(k)_{j1:j1+l+l'+2} = \ -[seq', l'] + [seq, l] + \text{ then } w(k+1)_{j1:j1+l+l'+2} = (-) - [seq', l'-2] + [seq, l] + (+)$$

Sequence $[seq', l']$ *Reduces* (Lemma 7.3), feeding with additional $+$ to $[seq, l]$ until $[seq', l']$ disappears and $[seq, l]$ finds the first $-$. This case (e) is explained later.

$$(c) \text{ if } w(k)_{j1:j1+l+l'+3} = \ +[seq', l'] - +[seq, l]+ \text{ then } w(k+1)_{j1:j1+l+l'+3} = \ [\ seq',\ l'+l+2] + (+)$$

Sequence $[seq', l']$ *Expands*, $[seq, l]$ *Move+*, and then both *Merge* in a larger sequence. Note this merged sequence has a $+$ to the right. Thus (Lemma 7.1), it will either *Reduce* (Lemma 7.3) or start performing *Move+* (behaving according to the different cases discussed in this proof).

$$(d) \text{ if } w(k)_{j1:j1+l+l'+3} = \ -[seq', l'] - +[seq, l]+ \text{ then } w(k+1)_{j1:j1+l+l'+3} = \ (-) - [seq', l'+l] + (+)$$

Sequence $[seq', l']$ *Move-*, $[seq, l]$ *Move+*, and then both *Merge* in a larger sequence $[seq', l'+l]$. Since this sequence has a letter $-$ to the left and $+$ to the right, then it will *Reduce*, behaving as discussed in Lemma 7.3.

$$(e) \text{ if } w(k)_{j1:j1+l+3} = \ - \ - +[seq, l\ ] + \text{ then } w(k+1)_{j1:j1+l+3} = \ (-) - [seq, l] + (+)$$

Sequence $[seq, l]$ has some $-$ letters to its left that do not belong to sequences (thus, there are at least two consecutive $-$ letters). It makes a *Move+*, and finishes with a $-$ to the left and $+$ to the right. Then, it will *Reduce*, behaving as discussed in Lemma 7.3. ■

Appendix I: Proof of Proposition 7.1

Proof of Proposition 7.1: Sequences evolve according to the rules *Move+*, *Move-*, *Expand*, *Reduce*, and *Merge* in Lemma 7.1. As stated by Lemmas 7.3, 7.4, if at some point a sequence experiences the *Reduce*, or *Move-* rule, then it will eventually disappear. Moreover, this includes the merging between sequences where one of them has experienced the *Reduce*, or *Move-* rule. Thus, in order for a sequence to survive, it must either *Expand* at every round, or *Move+* during all the rounds, or be the result of merging a *Expand* and a *Move+* sequences. On the other hand, sequences that *Move+* keep their length unchanged, and only sequences that always *Expand* increase the length of the sequence (Lemma 7.1).

As proved next, as long as there are $-$ letters not belonging to *Move+* or to *Expand* sequences, one *Expand* sequence survives. Note from Lemmas 7.3, 7.4 that sequences that *Reduce*, or *Move-* collaborate to partially sort the letters, placing the $-$ to the left and the $+$ to the right. Note also that if a *Move+* sequence finds a $-$, it *Reduces* (Lemma 7.5 and its proof). Thus, if all the *Expand* sequences expire, when the last collaborative sequence (*Reduce*, or *Move-*) expires, it leaves all the letters sorted: n_- $-$ letters followed by n_+ $+$ letters, e.g., $++-----++$ (recall the cycle graph structure). Thus, a new sequence starting with $+-$ would appear (e.g., between positions 2 and 3 in the example). However, as stated by Lemma 7.2, no new sequences are created. Thus, a *Expand* sequence must remain alive in order to consume these $+$ and $-$ letters. This concludes (i).

To prove (ii), it is noted that at least one sequence $[seq, l]$ *Expands* at every round, increasing its length by 2 at every round, until its length together with the lengths of the *Move+* sequences, sum up to $2n_{bal}$. Thus, this takes $(2n_{bal} - l)/2$ rounds, with $l \geq 2$, so that the number of rounds required to achieve the balanced configuration is smaller than n_{bal}.

The proof of (iii) is immediate from (i): the sequences lengths sum up to $2n_{bal}$ which equals n in the balanced case. Thus, there are no letters in $w(k)$ and all the sequences must be together forming a single sequence.

The proof of (iv) follows from (i): since $w(k)$ only contains sequences and $+$ letters, then sequences behave according to rule *Move+* (Lemma 7.1).

Finally (v) follows from the previous properties and from the definition of interlaced orientations (Def. 6.4). ■

Appendix J: Proof of Theorems 4.1, 4.2 and 4.3

Proof of Theorem 4.1: It follows from Propositions 5.1 and 5.2. ■

Proof of Theorem 4.2: As stated by Proposition 7.1, when robot regions have common traversing times $t_\star$ and they have balanced orientations, then the robot orientations acquire an interlaced configuration (Def. 6.4) in less than $n_{bal} = n/2$ rounds (Defs. 4.1, 6.2). And from Proposition 6.2, robots running the method with balanced interlaced configurations and common traversing times associated to their regions, synchronize in less than $n/2$ rounds (Def.6.2) to a configuration where they perform their meetings simultaneously in the network. ■

Proof of Theorem 4.3: As stated by Proposition 7.1, when robot regions have common traversing times $t_\star$, then the robot orientations acquire an interlaced configuration (Def. 6.4) in less than n_{bal} *rounds* (Defs. 4.1, 6.2). And from Proposition 6.1, a robot team running the method with unbalanced interlaced configurations and common traversing times associated to their regions, has a performance in terms of the revisiting times t_{rev} (Def. 2.1) averaged along n_{bal} meetings given by (20). ■

References

[1] M. Guo, J. Tumova, and D. V. Dimarogonas, "Communication-free multi-agent control under local temporal tasks and relative-distance constraints," *IEEE Transactions on Automatic Control*, vol. 61, no. 12, pp. 3948–3962, 2016.

[2] M. Guo and M. M. Zavlanos, "Distributed data gathering with buffer constraints and intermittent communication," in *IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 279–284.

[3] Y. Gabriely and E. Rimon, "Spanning-tree based coverage of continuous areas by a mobile robot," *Annals of mathematics and artificial intelligence*, vol. 31, no. 1-4, pp. 77–98, 2001.

[4] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[5] M. M. Zavlanos, M. B. Egerstedt, and G. J. Pappas, "Graph-theoretic connectivity control of mobile robot networks," *Proceedings of the IEEE*, vol. 99, no. 9, pp. 1525–1540, 2011.

[6] D. Boskos and D. V. Dimarogonas, “Robustness and invariance of connectivity maintenance control for multiagent systems,” *SIAM Journal on Control and Optimization*, vol. 55, no. 3, pp. 1887–1914, 2017.

[7] T. Soleymani, E. Garone, and M. Dorigo, “Distributed constrained connectivity control for proximity networks based on a receding horizon scheme,” in *American Control Conference*, Chicago, IL, USA, Jul. 2015, pp. 1369–1374.

[8] M. Schuresko and J. Cortes, “Distributed tree rearrangements for reachability and robust connectivity,” *SIAM Journal on Control and Optimization*, vol. 50, no. 5, pp. 2588 – 2620, 2012.

[9] M. Aranda, R. Aragues, G. Lopez-Nicolas, and C. Sagues, “Connectivity-preserving formation stabilization of unicycles in local coordinates using minimum spanning tree,” in *American Control Conference*, Boston, USA, Jun. 2016, pp. 1968–1974.

[10] H. Poonawala and M. W. Spong, “Preserving strong connectivity in directed proximity graphs,” *IEEE Transactions on Automatic Control*, vol. 62, no. 9, pp. 4392–4404, 2017.

[11] A. Gasparri, L. Sabattini, and G. Ulivi, “Bounded control law for global connectivity maintenance in cooperative multirobot systems,” *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 700–717, 2017.

[12] T. Nestmeyer, P. R. Giordano, H. H. Bulthoff, and A. Franchi, “Decentralized simultaneous multi-target exploration using a connected network of multiple robots,” *Autonomous Robots*, vol. 41, no. 1, pp. 989–1011, 2017.

[13] Y. Kantaros and M. Zavlanos, “Distributed intermittent connectivity control of mobile robot networks,” *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, 2017.

[14] Y. Kantaros, M. Guo, and M. M. Zavlanos, “Temporal logic task planning and intermittent connectivity control of mobile robot networks,” *IEEE Transactions on Automatic Control*, vol. 64, no. 10, pp. 4105–4120, 2019.

[15] R. Khodayi-mehr, Y. Kantaros, and M. M. Zavlanos, “Distributed state estimation using intermittently connected robot networks,” *IEEE Transactions on Robotics*, vol. 35, no. 3, pp. 709–724, 2019.

[16] M. Guo and M. M. Zavlanos, “Multirobot data gathering under buffer constraints and intermittent communication,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1082–1097, 2018.

[17] H. Wang and Y. Guo, “Synchronization on a segment without localization: Algorithm, applications, and robot experiments,” *International Journal of Intelligent Control and Systems*, vol. 15, no. 1, pp. 9–17, 2010.

[18] S. Susca and F. Bullo, “Synchronization of beads on a ring,” in *IEEE Conf. on Decision and Control*. IEEE, 2007, pp. 4845–4850.

[19] S. Susca, P. Agharkar, S. Martínez, and F. Bullo, “Synchronization of beads on a ring by feedback control,” *SIAM Journal on Control and Optimization*, vol. 52, no. 2, pp. 914–938, 2014.

[20] C. Song, L. Liu, G. Feng, and S. Xu, “Coverage control for heterogeneous mobile sensor networks on a circle,” *Automatica*, vol. 63, pp. 349–358, 2016.

[21] R. Aragues and D. V. Dimarogonas, “Intermittent connectivity maintenance with heterogeneous robots using a beads-on-a-ring strategy,” in *American Control Conference*, Philadelphia, PA, USA, Jul. 2019, pp. 120–126.

[22] J. Hong, J. Votion, Y. Cao, and Y. Jin, “Adaptive communication and control co–design for multi–agent coordination with second–order dynamics,” in *American Control Conference*, Philadelphia, PA, USA, Jul. 2019, pp. 5322–5327.

[23] L. Sabattini, C. Secchi, and N. Chopra, “Decentralized estimation and control for preserving the strong connectivity of directed graphs,” *IEEE Transactions on Cybernetics*, vol. 45, no. 10, pp. 2273–2286, 2015.

[24] R. Olfati-Saber, J. A. Fax, and R. M. Murray, “Consensus and cooperation in networked multi-agent systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[25] W. Zhang, Z. Wang, Y. Guo, H. Liu, Y. Chen, and J. Mitola III, “Distributed cooperative spectrum sensing based on weighted average consensus,” in *IEEE Global Telecommunications Conference*, 2011, pp. 1–6.

[26] L. Elsner, I. Koltracht, and M. Neumann, “On the convergence of asynchronous paracontractions with application to tomographic reconstruction from incomplete data,” *Linear Algebra and its Applications*, vol. 130, pp. 65–82, 1990.

[27] L. Xiao, S. Boyd, and S. Lall, “A scheme for robust distributed sensor fusion based on average consensus,” in *Int. Symposium on Information Processing in Sensor Networks*, 2005, pp. 63–70.

[28] R. A. Horn, R. A. Horn, and C. R. Johnson, *Matrix analysis*. Cambridge university press, 1990.

[29] A. Jadbabaie, J. Lin, and A. S. Morse, “Coordination of groups of mobile autonomous agents using nearest neighbor rules,” *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 988–1001, 2003.

[30] G. C. Calafiore, “Distributed randomized algorithms for probabilistic performance analysis,” *Systems & Control Letters*, vol. 58, no. 3, pp. 202–212, 2009.

[31] A. Tahbaz-Salehi and A. Jadbabaie, “A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times,” in *IEEE Conf. on Decision and Control*, San Diego, CA, Dec. 2006, pp. 4664–4669.

Rosario Aragues received the M.S. and Ph.D. degrees in System Engineering and Computer Science from Univ. Zaragoza, Spain, in 2008 and 2012. She was a postdoctoral researcher at the Institut Pascal, CNRS, UMR 6602, Clermont-Ferrand, France, from April 2012 to October 2013. Currently she works as an Assistant Professor with the Department of Computer Science and Systems Engineering at the University of Zaragoza. Her research interests include multi-robot perception and control using distributed and decentralized strategies.

Dimos V. Dimarogonas Prof. Dimos V. Dimarogonas was born in Athens, Greece, in 1978. He received the Diploma in Electrical and Computer Engineering in 2001 and the PhD in Mechanical Engineering in 2007, both from National Technical University of Athens (NTUA), Greece. He has held postdoctoral positions at KTH, Sweden and MIT, USA. He is currently Professor at the Division of Decision and Control, KTH Royal Institute of Technology, Stockholm, Sweden. His research interests include Multi-Agent Systems, Hybrid Systems and Control, Robot Navigation and Networked Control. He serves in the Editorial Board of Automatica, the IEEE Transactions on Control of Network Systems and is a Senior member of the IEEE.

Pablo Guallar is a Msc Industrial Engineer, graduated in 2019 at the Universidad de Zaragoza, Spain. During his master, and after graduating, he has worked with the Robotics, Real-Time and Perception research group. His main working areas are multi-robot systems, navigation, and cooperative control. In 2020, he received a grant from the Instituto de Investigación en Ingeniería de Aragón to carry on his research in the robotics field.

Carlos Sagues received the M.S. degree in Computer Science and Systems Engineering and the Ph.D. degree in Industrial Engineering from the Universidad de Zaragoza, Spain. During the course of his Ph.D. he worked on force and infrared sensors for robotics. In 1994, he became an Associate Professor with the Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, where he has also occupied the Head Teacher position. Since 2009, he has been a Full Professor with this department. He is a member of the Instituto de Investigación en Ingeniería de Aragón. His current research interests include control systems, computer vision, visual robot navigation, and multivehicle cooperative control.