

RANENDU ADHIKARY 
MANASH KUMAR KUNDU 
BUDDHADEB SAU

CIRCLE FORMATION BY ASYNCHRONOUS OPAQUE ROBOTS ON INFINITE GRID

Abstract *This paper presents a distributed algorithm for the CIRCLE FORMATION problem under the infinite grid environment by asynchronous mobile opaque robots. Initially, all of the robots acquire distinct positions, and they must form a circle over the grid. The movements of the robots are only restricted along the grid lines; they do not share any global coordinate system. The robots are controlled by an asynchronous adversarial scheduler that operates in LOOK-COMPUTE-MOVE cycles. The robots are indistinguishable by their nature, and they do not have any memory of their past configurations nor previous actions. We consider the problem under a luminous model, where robots communicate via lights; other than that, they do not have any external communication systems. Our protocol solves the CIRCLE FORMATION problem using seven colors. A subroutine of our algorithm also solves the LINE FORMATION problem using three colors.*

Keywords distributed computing, autonomous robots, circle formation, line formation, robots with lights, asynchronous, Look-Compute-Move cycle, grid

Citation Computer Science 22(1) 2021: 81–100

Copyright © 2021 Author(s). This is an open access publication, which can be used, distributed and reproduced in any medium according to the Creative Commons CC-BY 4.0 License.

1. Introduction

Executing a task by a team of people is always better than merely by a single person. This is certainly true for any team, whether it consists of people or robots. Completing a task by a single robot is cumbersome and sometimes not even possible. A team of autonomous mobile robots can distributively and collaboratively execute complex tasks. These simple low-cost robots are emerging as an alternative for single powerful and expensive robots. One of the research and development activities performed on these robots is distributed motion coordination. By controlling these robots' movements, we can form specific patterns and move in formation toward cooperating for the achievement of certain tasks. Motion-planning algorithms for robotic systems that are made up of robots that change their positions to form a given pattern are very important and may become challenging in the case of severe limitations, such as in the communication between robots, hardware constraints, natural calamities, etc. The significance of positioning the robots based on some given patterns may be useful for various tasks, such as bridge building, forming adjustable buttresses to support collapsing buildings, rescue operations on disaster sites, satellite recoveries in inhospitable areas, or tumor excision [24]. Also, distributed motion-planning algorithms for robotic systems are potentially useful in environments that are inhospitable to humans or are hard to control and observe (e.g., outer space, under water, forests, and battlefields).

1.1. Background and problem definition

In the CIRCLE FORMATION problem [8, 9, 15, 18–20, 28–30], the robots must form a circle over a region. The robots are not given any knowledge about the circle through any coordinates. To form a circle, we need two things: 1) a center, and 2) a radius. Nevertheless, neither of these are indicated to the robots beforehand. Initially, the robots are at distinct locations; they each have a different coordinate system. These robots are assumed to be *autonomous* (i.e., there is no central authority to control them), *homogeneous* (i.e., they carry out the same protocol), *anonymous* (i.e., they have no unique identifiers), or *identical* (i.e., they are indistinguishable by their appearance) [16, 17]. In the continuous setting, the robots are assumed to be able to execute accurate movements in arbitrary directions and by arbitrarily small amounts. Hence, the robots can maneuver to avoid collisions (even in densely crowded situations). Certain models also permit the robots to move along curved trajectories; in particular, along the circumference of a circle [18]. For robots with weak mechanical capabilities, it may not be possible to precisely execute such intricate movements. This motivates us to consider the problem in a grid-based terrain where the robots' movements are restricted to following grid lines and only by a particular unit distance with each step. Grid-type floor layouts can be easily implemented in real-life robot navigation systems by using magnets or optical guidance [2]. Clearly, the shape of a circle will not be the same as in the continuous domain in a grid-based domain. If the circle moves over a grid point, we will place a robot there; otherwise,

we place it at the nearest grid point. As we mentioned before, these robots act without a central control; they work by a sequence of LOOK-COMPUTE-MOVE cycles. In the LOOK phase, a robot registers all of the other robots' positions in its local coordinate system. After this, each robot computes the next position according to a deterministic algorithm (i.e., the COMPUTE phase). In the MOVE phase, it will either move to the desired location along a straight line or make a null move. At each cycle, a robot can move to one of its four adjacent grid points. Now, depending on the activation schedule and timing assumptions of the LOOK-COMPUTE-MOVE cycles, there are three models that have mainly been studied in the literature: an asynchronous model (*Async*), a fully synchronous model (*Fsync*), and a semi-synchronous model (*Ssync*). In the *Fsync* model, there is a global clock on which every robot agrees. The robots operate during a synchronous atomic round; all of the robots that are active in a round terminate their cycles by the subsequent round. The *Ssync* model is the same as the *Fsync* model; however, a subset of the robots activates during each round here. In the *Async* model, there is no notion of a global clock; each robot has its own clock and operates accordingly.

The robots operate under the *luminous model* or '*robots with lights*' model introduced by Peleg [25]. In this model, the robots communicate with each other through lights and can assume a set number of predefined colors. These lights serve as both a form of persistent memory and as a form of communication. Other than this, the robots have *oblivious memory*; i.e., they do not remember any past computations nor have any other communication systems. The robots have *obstructed visibility*; i.e., a robot's view can be obstructed by other robots. If three robots are collinear, then the outside robots can only see a single robot (whereas the middle one can see two robots).

1.2. Earlier works

While fundamental problems in autonomous mobile robots like GATHERING [4, 7, 11, 14, 26], ARBITRARY PATTERN FORMATION [5, 6, 21], and MUTUAL VISIBILITY [1, 27] have been studied in grid environments, the CIRCLE FORMATION problem has only been studied on a continuous Euclidean plane. In the CIRCLE FORMATION problem, robots must place themselves over a circumference of a circle. If the robots are placed in such a way on the circumference of the circle where they are equidistant from each other, then the problem is defined as UNIFORM CIRCLE FORMATION. The CIRCLE FORMATION problem was first discussed by Sugihara and Suzuki [28]; they proposed a simple heuristic distributed algorithm. However, their algorithm formed an approximation of a circle (mainly, a Reuleaux triangle, which is a hybrid shape between a triangle and a circle). Tanaka [30] later improved this algorithm by generating a better approximation of a circle. Later on, Suzuki and Yamasihita [29] achieved a uniform circle formation for *non-oblivious* robots (those that remember all of their past actions). Défago and Konogaya [8] provided an *Ssync* algorithm by which a group of oblivious robots eventually converge toward a uniform circle. In [9], Défago and

Souissi presented an *Ssync* non-uniform self-organized circle formation algorithm. However, they assumed that the robots agree on the chirality of the system and do not have obstructed visibilities. Until now, the results have assumed the scheduler to be *Ssync*. In the *Async* model, several results have assumed implicit agreements among the robots. Flocchini et al. [15] provided a simple algorithm for UNIFORM CIRCLE FORMATION where the robots have the same orientation or chirality. Later on, the results were improved by a general result from Fujinaga et al. [20]. They assumed that the local coordinate system of all of the robots is right-handed. Imposing no assumptions on the local coordinates of the robots, the UNIFORM CIRCLE FORMATION algorithm was devised in [18]. However, they assumed that the robots could move along circular arcs as well as straight-line segments. Finally, without any extra assumptions, a solution was provided for $n \neq 4$ robots in [19], and a solution for the special case of $n = 4$ appeared in [22]. Using one bit of internal memory in [3], the authors studied the CONSTRAINED CIRCLE FORMATION problem. The CONSTRAINED CIRCLE FORMATION problem demands that, in addition, the maximum distance that a robot moves to solve a problem should be minimized. In [12], the authors solved the problem for *fat* robots (those with a finite extent) under limited visibility but considering an agreement over a global coordinate system. An *Ssync* algorithm for transparent robots that perform rigid movements and agree on one axis was provided in [23]. In [10], the authors also considered the problem of circle formation, but their model of computation is different than ours. They considered a circle formation around an unknown target with limited sensing and communication capabilities using holonomic robots (a holonomic system is when the number of controllable degrees of freedom is equal to the total degrees of freedom). They assumed that there is a global coordinate system on which all of the robots agree. Also, they assumed that an inactive robot can sense its vicinity. The closest solution to our problem was studied in [13]; however, they considered the problem under an *Fsync* scheduler.

1.3. Our contribution

The CIRCLE FORMATION problem has been quite extensively studied in the continuous domain. In a continuous domain, a robot can move in any direction (sometimes in a curve) with arbitrary precision. The problem has never been studied in a discrete setting. We have devised a deterministic distributed algorithm that can solve the CIRCLE FORMATION problem over a grid-based environment. The infinite grid is the most realistic model for a practical purpose. Here, the robots can only move in four directions with unit lengths. Also, the infinite grid is a natural discretization of a Euclidean plane. This simple model of movement along grid lines from one grid point to another can be easier to implement for robots with weak mechanical capabilities, as they may not be able to execute accurate movements in arbitrary directions or by arbitrarily small amounts. Although the simple model of movement may be easier to physically execute, the restrictions imposed on the movements of the robots pose the main difficulty of the algorithmic problem. We have assumed that all

of the robots agree on the positive y -axis. Other than this, the robots do not have any other agreement among their local coordinate systems. The robots are controlled by an asynchronous adversarial scheduler. Although robots have unlimited visibility, they are opaque (i.e., they can block each others' views). They are not given any coordinates nor any knowledge about the total number of robots. Robots are equipped with lights by which they can communicate with other robots. These lights can also be used as memory (by which, a robot can only recall its previous state). Other than this, robots have oblivious memory. We have solved two problems in this grid-based environment: **LINE FORMATION** and **CIRCLE FORMATION**. From any arbitrary initial configuration, our algorithm solves **LINE FORMATION** using three colors and **CIRCLE FORMATION** using seven colors.

2. Model and definitions

Two robots are always on a circle, as we can take the line between them as a diameter. Now, three robots may or may not be on the circle. Suppose that all of them are collinear. If the middle robot makes a movement in this situation, then there is a circle that passes through all of them. So, we consider the number of robots to be at least 3. Now, we present the model and some basic definitions that will help us understand the protocols.

Robots. We consider a set of $n \geq 3$ homogeneous, autonomous, anonymous, and identical robots $\mathcal{R} = \{r_1, r_2, \dots, r_n\}$ deployed on a two-dimensional infinite grid where each is initially positioned on distinct grid points. The robots are assumed to be dimensionless and modeled as points on the plane. The robots have an agreement over the positive direction of the y -axis; i.e., all of the robots have an agreement over up and down. They do not have a common origin nor any agreement over the x -axis. Actually, the robots do not have access to any global coordinate system other than the agreement over the positive direction of the y -axis. Total number of robots n is not known to them.

Look-Compute-Move cycles. An active robot r operates according to the so-called **LOOK-COMPUTE-MOVE** cycle (LCM for short). In the **LOOK** phase, a robot registers the positions of all of the other robots in its own local coordinate system. After this, the robot computes the next position and a color according to a deterministic algorithm; i.e., the **COMPUTE** phase. In the **MOVE** phase, it will either move a unit length to the desired location along a straight line or make a null move.

Scheduler. We assume that the robots are controlled by an asynchronous adversarial scheduler. This implies that the amount of time spent in the **LOOK**, **COMPUTE**, **MOVE**, and inactive states by different robots is finite but unbounded and unpredictable. As a result, the robots do not have a common notion of time, and the configuration that is perceived by a robot during the **LOOK** phase may significantly change before it actually makes a move.

Movement. As the robots are deployed on an infinite grid, a robot can only move to its four adjacent grid points. The movements are not *rigid*; i.e., in any

LOOK-COMPUTE-MOVE cycle, a robot can only move from one grid point to another. In discrete domains, a robot's movements are assumed to be atomic. This implies that the robots are always seen on grid points, not on edges.

Visibility. The visibility range of the robots is unlimited but can be obstructed by the presence of other robots. Robot r_i can see another robot r_j if and only if there are no robots on straight line segment $\overline{r_i r_j}$.

Lights. Each robot is equipped with an externally visible light, which can assume a $\mathcal{O}(1)$ number of predefined colors. The robots explicitly communicate with each other using these lights. The lights are not erased at the end of a cycle; otherwise, the robots are oblivious. The colors used in our algorithm are $\{Off, Line, Moving1, Corner, Moving2, Moving3, Done\}$.

Geometric definitions. Given a configuration of robots at time t , the *smallest enclosing rectangle* is defined as the smallest axis-aligned rectangle that contains all of the robots. We denote the smallest enclosing rectangle by $ABCD$, where AB is the lowest boundary and CD the top boundary. We define the length of a line segment to be the number of grid points between the terminal robots (including the grid points where the terminal robots reside). Similarly, we define the length of a line. In our case, the length of a line segment and the length of a line that contains that line segment is the same. H denotes the height of the configuration, and \mathcal{L}_1 and \mathcal{L}_H are the horizontal lines that contain AB and CD , respectively. Note that we can think of configuration $ABCD$ as a union of horizontal line segments. We define $\mathcal{L}_1, \mathcal{L}_2, \dots$, and \mathcal{L}_H to be the horizontal lines that contain the robots of the configuration in increasing order (Fig. 1).

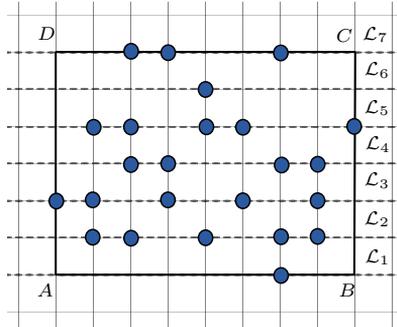


Figure 1. Illustrations for geometric definitions given in Section 2

So, \mathcal{L}_1 and \mathcal{L}_H are the horizontal lines that contain the robots of AB and CD , respectively. We define \mathcal{L}_{least} ($\neq \mathcal{L}_1$) as the horizontal line such that there are no horizontal lines that contain robots between \mathcal{L}_1 and \mathcal{L}_{least} . Also, we define $\mathcal{L}(r)$ to be the horizontal line that contains r . A robot r on a grid line segment \mathcal{L} will be called *non-terminal on \mathcal{L}* if it lies between two robots on \mathcal{L} ; otherwise, it will be called *terminal on \mathcal{L}* . In an infinite grid, the distance between two points is defined by the ℓ_1 norm. The distance in the ℓ_1 norm is sometimes called the Manhattan distance.

3. Algorithm

A circle is defined with a center and a radius. The easiest way to get this information is to take a line segment as a diameter and define the middle point of the line segment as the center. So, we first need to arrange the robots in such a way as to form a line. This phase is called **LINE FORMATION**. After the completion of this phase, the main challenge will be to place the robots on the circumference of the circle. This phase is defined as **CIRCLE FORMATION**. Our algorithm works in these two phases. In **LINE FORMATION**, the robots from any arbitrary initial configuration will create a line segment, and in the last phase (**CIRCLE FORMATION**), the robots will set up a circle from the line segment. The difficulty arises from the restriction on the movements and the opaqueness of the robots. Note that, in the grid type environment, a robot can only move one step at a time (according to each robot's own clock) to its four adjacent grid points. Apparently, if there are no adjacent grid points, a robot cannot move. Additionally, a robot's perceived view can be significantly different from the actual global picture due to its obstructed visibility.

3.1. Line formation

In this phase, we will put down all of the robots from the initial configuration to a line segment. We have assumed that the robots have an agreement over the positive y-axis; i.e., they have an accord over up and down. The main idea behind this phase is that the robots from the upper portion of the configuration will sequentially move to the lowest portion of the configuration. Finally, the configuration will transform into a line segment. A pseudo-code description of the procedure is presented in Algorithm 1. The lights used in this phase are $\{Off, Line, Moving1\}$.

Now, we characterize the initial configurations depending on the presence of the robots on the boundary of the smallest enclosing rectangle. Note that \mathcal{L}_1 always has at least one robot on it (Fig. 2). The robots on line \mathcal{L}_1 can identify that they are at the lowest level of height.

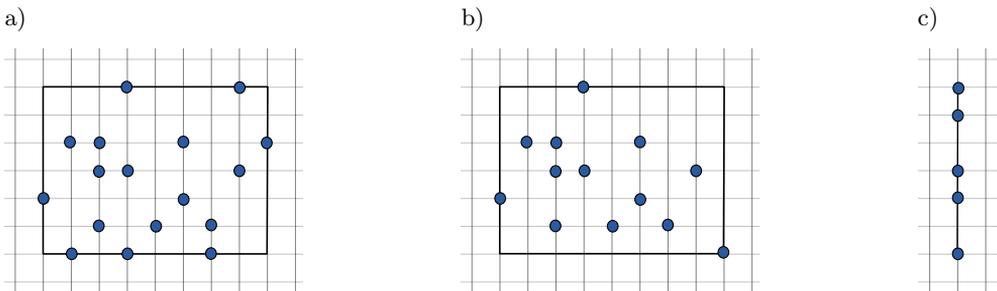


Figure 2. Initial configuration always has at least one robot on \mathcal{L}_1

The crucial part of this phase is to define the movements of the robots. Note that \mathcal{L}_1 contains the lower boundary line segment of the smallest enclosing rectangle.

Any robot on \mathcal{L}_1 can identify that it is on the lower portion of the smallest enclosing rectangle. Identifying this, the robots on \mathcal{L}_1 will alter their lights to *Line*. If a robot cannot see a robot with color *Line*, it cannot gauge its height relative to \mathcal{L}_1 . These robots will remain inactive. Upon waking up, if a robot perceives a robot with color *Line*, then it will measure its height relative to \mathcal{L}_1 . Clearly, \mathcal{L}_2 is the adjacent of \mathcal{L}_1 .

Algorithm 1: Line Formation

```

1 Procedure LINEFORMATION()
2    $r \leftarrow$  myself
3   while LINEFORMATION() = FALSE do
4     if  $r.color = Off$  then
5       if  $r$  is on  $\mathcal{L}_1$  then
6          $r.color \leftarrow Line$ 
7       else if  $r$  is on  $\mathcal{L}_2$  then
8         if  $r$  is a terminal robot then
9           if  $FINDEMPTYPOINT() = True$  then
10             $r.color \leftarrow Moving1$ 
11        else if  $r$  is on  $\mathcal{L}_{least}$  and  $\mathcal{L}_2$  is empty then
12          if  $r$  is a terminal robot then
13            if  $FINDEMPTYPOINT() = True$  then
14               $r.color \leftarrow Moving1$ 
15      else if  $r.color = Moving1$  then
16        if  $r$  is on  $\mathcal{L}_1$  then
17           $r.color \leftarrow Line$ 
18        else if  $r$  is on  $\mathcal{L}_2$  then
19          if  $r$  is a terminal robot then
20            if  $FINDEMPTYPOINT() = True$  then
21               $r.color \leftarrow Moving1$ 
22              Move toward the empty point on  $\mathcal{L}_1$ 
23            else
24               $r.color \leftarrow Off$ 
25        else if  $r$  is on  $\mathcal{L}_{least}$  and  $\mathcal{L}_2$  is empty then
26          if  $r$  is a terminal robot then
27            if  $FINDEMPTYPOINT() = True$  then
28               $r.color \leftarrow Moving1$ 
29              Move toward  $\mathcal{L}_2$ 
30            else
31               $r.color \leftarrow Off$ 
32      else if  $r$  is on  $\mathcal{L}_{i+1}$  and  $\mathcal{L}_i$  is non-empty with  $i \geq 2$  then
33         $r.color \leftarrow Off$ 

```

For a terminal robot r on \mathcal{L}_2 , we define \mathcal{V}_r to be the visible portion of the lower boundary of the smallest enclosing rectangle that r can see and \mathcal{S}_r to be the portion of \mathcal{L}_1 where r will move. If \mathcal{L}_2 contains a single robot, then \mathcal{V}_r with two extra grid points (one on each side of \mathcal{V}_r) will be the region \mathcal{S}_r where r will move (Figs. 3a, 3b). Now, suppose that \mathcal{L}_2 contains more than one robot. Then, for a terminal robot r , we

define $\mathcal{L}(r)_\perp$ to be the perpendicular line of \mathcal{L}_2 at r . Let $r_{\mathcal{L}_1}$ be the intersection point $\mathcal{L}(r)_\perp \cap \mathcal{V}_r$ on \mathcal{L}_1 . Suppose that r and r' are the terminal robots on \mathcal{L}_2 ; then, \mathcal{S}_r is the region that starts at $r_{\mathcal{L}_1}$ and ends at the opposite direction of r' , with one extra grid point at the end of \mathcal{V}_r (Fig. 3c). Now, if \mathcal{L}_2 contains no robots, then the terminal robots of \mathcal{L}_{least} will move to \mathcal{L}_2 .

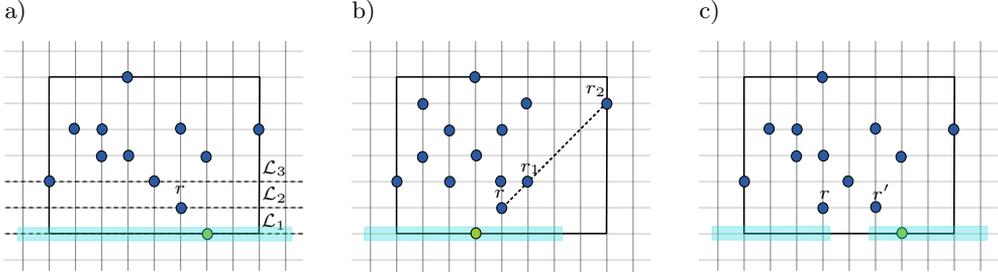


Figure 3. Illustrations of desired empty points on \mathcal{L}_1 : a) shaded region \mathcal{S}_r is place where r will place itself; b) robot r 's view is blocked by r_1 , and its \mathcal{S}_r is different than global; c) \mathcal{S}_r and $\mathcal{S}_{r'}$ are corresponding regions in respect to robots r and r'

Now, if a robot r on \mathcal{L}_2 finds an empty point on \mathcal{L}_1 , then the movements of r can cause collisions. So, we will now state some conditions under which a robot will move. The conditions are described as function $\text{FINDEMPTYPOINT}()$, which is defined below.

FindEmptyPoint(). First of all, the robots of \mathcal{L}_1 will not execute this function. The function takes a terminal robot r of a horizontal line that can perceive a robot with color *Line* as an input and returns “True” if the following conditions are satisfied.

Condition 1. If r is on \mathcal{L}_2 , then:

1. First of all, it checks line \mathcal{L}_1 such that all of the robots on it have color *Line*.
2. Now, it scans \mathcal{S}_r as shown in Figure 3 to find an empty grid point. If r finds more than one grid point, then the empty point that has the least distance from r will be chosen as the destination. Here, the distance is being measured under the ℓ_1 norm (i.e., Manhattan distance).
3. Even if it finds an empty grid point on \mathcal{L}_1 , a move toward it can lead to a collision. To avoid this, it must make sure that there are no robots with light *Moving1* within Manhattan distance 2 in the direction in which it intends to move (Fig. 4).

Condition 2. If r is not on \mathcal{L}_2 , then:

1. $\mathcal{L}(r) = \mathcal{L}_{least}$.

Brief discussion. First of all, the robots on \mathcal{L}_1 will not move in this phase. Initially, all of the robots have lights that are set to *Off*. Note that, if a robot r in the configuration can perceive a robot with color *Line* after waking up, then it can distinguish the

height of $\mathcal{L}(r)$ from \mathcal{L}_1 . Otherwise, these robots will remain inactive. The robots on \mathcal{L}_1 will alter their lights to *Line*. Now, the terminal robots on $\mathcal{L}(r)$ summon function $\text{FINDEMPYPOINT}()$; if the result is “True,” then those robots are qualified for movement. Note that function $\text{FINDEMPYPOINT}()$ treats the robots on \mathcal{L}_2 and the robots on \mathcal{L}_{least} differently. If \mathcal{L}_2 contains no robots, a terminal robot r on \mathcal{L}_{least} will move toward \mathcal{L}_2 . However, it will not move immediately. First, it will alter its light to *Moving1*; then, it will redo the same computations in the next LCM cycle. If the function returns “False,” the robot will set its light to *Off*. If a terminal robot r is on \mathcal{L}_2 , r will first check whether there is a robot with color *Moving1* in the direction that it wants to move. However, it will not move immediately. First, it will alter its light to *Moving1*; then, it will redo the same computations in the next LCM cycle. If the function returns “False,” the robot will set its light to *Off*. Now, the crucial thing of this phase is to make the movements collision-less. To tackle this situation, we used two steps: (1) only those robots on line \mathcal{L}_2 are allowed to move toward \mathcal{L}_1 , and (2) a terminal robot on $\mathcal{L}(r)$ will move only if \mathcal{L}_2 is empty and $\mathcal{L}(r) = \mathcal{L}_{least}$. Note that we always have two empty points on \mathcal{L}_1 . As we can have two terminal robots on a line at most, we will never have a problem with finding empty points.

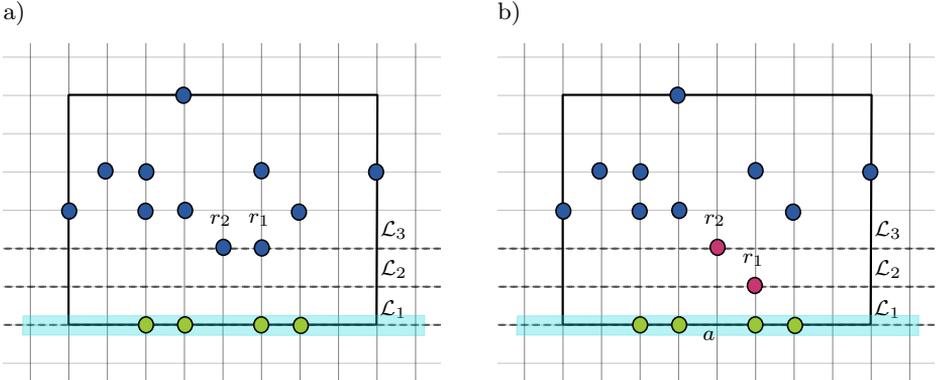


Figure 4. Illustration for function $\text{FINDEMPYPOINT}()$: a) r_1 and r_2 find that \mathcal{L}_2 is empty. So, they will change their color and move to \mathcal{L}_2 . Suppose that r_1 moves first and r_2 has a pending move due to asynchrony; b) r_1 finds an empty point a . But, r_2 is within Manhattan distance 2 in which direction r_1 wants to move. So, r_1 will not move

Now, if \mathcal{L}_2 is empty, then the terminal robots on \mathcal{L}_{least} will move toward \mathcal{L}_2 . Now, if \mathcal{L}_2 is non-empty, then the terminal robots on \mathcal{L}_2 find the desired empty point and move to \mathcal{L}_1 . However, we can still have a collision due to asynchrony. In Figure 4, we have depicted this scenario. Robots r_1 and r_2 are terminal robots on \mathcal{L}_3 . As \mathcal{L}_2 is empty, $\text{FINDEMPYPOINT}()$ is true for both r_1 and r_2 . Suppose that r_1 moves first and r_2 has a pending move due to asynchrony. Now, r_1 is a terminal robot on \mathcal{L}_2 , and $\text{FINDEMPYPOINT}()$ is true for r_1 . Let the desired empty point be a , as a is closest to the position of r_1 . Now, the asynchronous scheduler can make the movements of

r_1 and r_2 simultaneously. So, r_1 and r_2 will collide with each other. However, this will never happen, as robot r_2 has color *Moving1*, it is within Manhattan distance 2 from r_1 , and it is in the direction in which r_1 intends to move. So, r_1 will not move.

Note that \mathcal{L}_1 has at least two empty points on it. So, a terminal robot r on \mathcal{L}_2 will always find empty points on \mathcal{L}_1 . Therefore, our main concern is about collision. We must make sure that there will be no collisions during the movements.

Lemma 1. *There exists a time t such that a robot r on $\mathcal{L}(r)$ will reach line \mathcal{L}_1 .*

Proof. Depending on the horizontal position of a robot r , we can characterize the behavior of the robot. We can primarily have three cases:

- **Case 1.** $\mathcal{L}(r) = \mathcal{L}_1$ (i.e., r is on lowest horizontal line);
- **Case 2.** $\mathcal{L}(r) = \mathcal{L}_2$ (i.e., r is on line adjacent to \mathcal{L}_1);
- **Case 3.** $\mathcal{L}(r) = \mathcal{L}_{least}$ (i.e., r is on least top line that has robot on it).

We will give correctness for each of the cases below:

- **Case 1.** Nothing to prove, as r is already at the desired location.
- **Case 2.** Only the terminal robots of \mathcal{L}_2 will move. Suppose that r is the only robot on \mathcal{L}_2 . Now, if `FINDEMPTYPOINT()` returns “True,” then r will move to the desired point in \mathcal{S}_r . For robot r , `FINDEMPTYPOINT()` can return “False” for two reasons. One where there is a robot – say, r' – on \mathcal{L}_3 within Manhattan distance 2 with color *Moving1*, and the other when one of the robots on \mathcal{L}_1 has not altered its color to *Line*. For the second case, there will clearly be a time t' when `FINDEMPTYPOINT()` will return “True” for r . For the first case, if r' is in the direction that r wants to move, then `FINDEMPTYPOINT()` will return “False” for r and r will alter its light to *Off*. Now, we can have two issues regarding this situation. If r' has no pending move, then it will alter its light to *Off* in the next LCM cycle. Note that r' will remain inactive, as \mathcal{L}_2 is non-empty. Eventually, there will be a time when r will become active and `FINDEMPTYPOINT()` will return “True” for r . Now, if r' has a pending move, then it will move to \mathcal{L}_2 . Note that `FINDEMPTYPOINT()` can now return “False” for one reason for the robot r' ; this is when one of the robots on \mathcal{L}_1 has not altered its color to *Line*. So, r' will eventually move to \mathcal{L}_2 . Clearly, there will be a time t'' when robot r will move to \mathcal{L}_1 . Now, consider the case when there is more than one robot on \mathcal{L}_2 . Note that, in this case, there cannot be a robot with light *Moving1* on \mathcal{L}_3 . Now, if r is a terminal robot on \mathcal{L}_2 , then it will recall function `FINDEMPTYPOINT()`. If `FINDEMPTYPOINT()` returns “True,” then r will move to \mathcal{S}_r . Note that `FINDEMPTYPOINT()` returns “False” for only these two reasons.
- **Case 3.** If r is a lone robot on \mathcal{L}_{least} , then it will eventually reach \mathcal{L}_2 . Suppose that there is more than one robot on \mathcal{L}_{least} . Let the terminal robots be r and r' . If only one of them moves toward \mathcal{L}_2 , then the other robot will stay put. Suppose that both of them execute the LOOK-COMPUTE-MOVE cycle synchronously. Then, both r and r' will alter their colors to *Moving1* if `FINDEMPTYPOINT()`

returns “True.” If they execute all of the future LOOK-COMPUTE-MOVE cycles synchronously, then both will eventually move to \mathcal{L}_2 . Suppose that, due to asynchrony, one of the robots makes a move first and the other has a pending move toward \mathcal{L}_2 (a similar scenario is depicted in Figure 4). In this figure, robots r_1 and r_2 are terminal robots on \mathcal{L}_{least} ($= \mathcal{L}_3$). `FINDEMPTYPOINT()` returns “True” for both of them. So, they alter their colors to *Moving1*. In the next LCM cycle, they will again call function `FINDEMPTYPOINT()`. Now, we can have two possible future events due to asynchrony: one in which `FINDEMPTYPOINT()` returns “True” for r_1 and r_2 has not recalled the function yet, and another in which `FINDEMPTYPOINT()` returns “True” for both r_1 and r_2 simultaneously. In the former case, r_2 will alter its light to *Off*, as \mathcal{L}_2 is now non-empty (r_1 is now on \mathcal{L}_2). In the latter case, both will move to \mathcal{L}_2 . So, in the first case, r_2 will remain inactive until r_1 moves to \mathcal{L}_1 .

□

It is easy to see that a robot can always check whether a line formation has completed or not.

Lemma 2. *A robot r can always detect whether a line formation has completed or not.*

Theorem 1. *Algorithm 1 LINE FORMATION will transform any initial configuration into a line.*

Proof. From Lemma 1 and 2, we can conclude that there will be a time t when we have all of the robots on a line. □

3.2. Circle formation

When Algorithm 1 terminates for all of the robots in the configuration, we have a configuration where all of the robots are on line \mathcal{L}_1 and have color *Line*. For our circle to form, we have a line whose middle point that we will consider to be the center and whose length will be treated as the diameter. Remember that we have defined the length $|\mathcal{L}|$ of a line \mathcal{L} to be the number of grid points between the terminal robots (including the grid points where the terminal robots reside). In Figure 5a, the length of the line is 11. Furthermore, the distance between two robots defined to be the number of grid points between them, including the grid points on which they reside. So, the distance between a robot and itself is one. A pseudo-code description of the procedure is presented in Algorithm 2. The lights used in this phase are $\{Corner, Moving2, Moving3, Done\}$.

The robots that are at the corners of \mathcal{L}_1 can identify themselves. After the identification, the corner robots will alter their lights to *Corner*. Note that the corner robots will not move in this phase. Now, we will define a coordinate system on this line (Fig. 5). All of the robots have a given coordinate (l, d) , where l denotes length $|\mathcal{L}_1|$ and d denotes the distance from the nearest corner. We define r_d to be robot r at distance d from the nearest corner. Clearly, a robot r on \mathcal{L}_1 does not have any information about l or d . To obtain this information, a robot has to move upward or

downward to see the line. Now, if robot r sees a corner robot on \mathcal{L}_1 , then it will move upward (i.e., on \mathcal{L}_2). Eventually, all of the other robots will move upward. Note that robot r can now see the corner robots of \mathcal{L}_1 and have full knowledge about l and d . Now, circle \mathcal{C} will have diameter l , and the center is at the middle point of l . Note that the center may not have coordinates with integer. We denote $\mathcal{L}(r)_\perp$ to be the line that is perpendicular to \mathcal{L}_1 at r . c'_d denotes the intersection between circle \mathcal{C} and line $\mathcal{L}(r)_\perp$. Now, c'_d may not be a grid point. We define $c_d = \lceil c'_d \rceil$. Note that c_d is a grid point. We define *middle* to be the y-coordinates of the middle points of \mathcal{L}_1 . Note that we can express these by one coordinate whether or not $|\mathcal{L}_1|$ has one or two middle points. If l is odd, then the middle robot – say, $r_{(l+1)/2}$ (*middle* = $(l + 1)/2$) – is at the center of circle \mathcal{C} . If l is even, then we have two middle robots – say, $r_{l/2}$ and $r_{l/2+1}$ (*middle* = $l/2$) – and the center of circle \mathcal{C} is at the middle of $r_{l/2}$ and $r_{l/2+1}$. In Figure 5b, $l = 11$ is odd and $(l + 1)/2 = 6$. So, middle robot r_6 is at $(11, 6)$. Without a loss of generality, let us suppose that l is odd. All of the r robots on \mathcal{L}_2 will move to the horizontal line – say, $\mathcal{L}_{c_{(l+1)/2}}$ passing through point $c_{(l+1)/2}$. However, they will move sequentially. After reaching line $\mathcal{L}_{c_{(l+1)/2}}$, robot $r_{(l+1)/2}$ is now on the circle. Now, the robots at $(l, (l + 1)/2 - i)$ will move toward point $c_{(l+1)/2-i}$ if the robots at $(l, (l + 1)/2 - i + 1)$ have completed their movements where $1 \leq i \leq (l + 1)/2 - 2$ (Fig. 6). Hence, all of the robots will eventually be on circle \mathcal{C} .

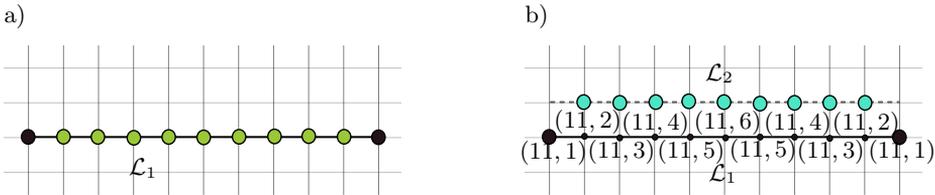


Figure 5. Prescribed coordinate system: a) corner robots alter their colors to *Corner*; b) after movements of non-corner robots, robots can have knowledge of their coordinates

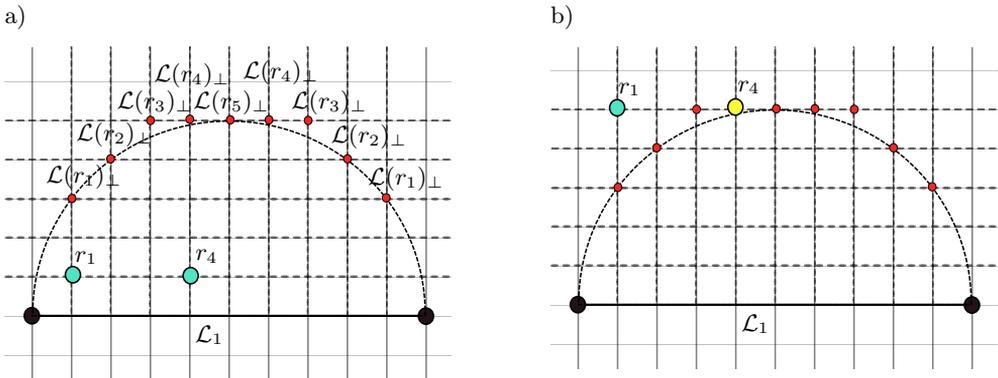


Figure 6. Illustration of movements of robots: a) red dots are points where non-corner robots will place them; b) scenario where r_4 has completed its movement; as there are no robots between r_1 and r_4 , r_1 will start its movement

Brief discussion. The terminal robots of \mathcal{L}_1 will alter their lights to *Corner*. If it sees a robot with color *Corner*, a robot r on \mathcal{L}_1 will move to line \mathcal{L}_2 with color *Moving2* (Fig. 6a). Upon reaching its destination, r can identify its position on line \mathcal{L}_1 . When all of the robots have reached \mathcal{L}_2 , the robots will move to horizontal line $\mathcal{L}_{c_{(l+1)/2}}$. A robot at line \mathcal{L}_j will move to line \mathcal{L}_{j+1} if there are no robots between \mathcal{L}_1 and \mathcal{L}_j . Eventually, the robots will arrive at horizontal line $\mathcal{L}_{c_{(l+1)/2}}$ (Fig. 6b). If there is a robot $r_{(l+1)/2}$ in the configuration, then it has clearly arrived at the desired location on the circle and it will alter its light to *Done*. The robots at $(l, (l+1)/2 - i)$ will move to point $c_{(l+1)/2-i}$ if the robots at $(l, (l+1)/2 - i + 1)$ have color *Done*. Now, position $(l, (l+1)/2 - i + 1)$ may not contain a robot; however, that will not be a problem. Then, the robot will check for the nearest position (l, ρ) (Fig. 6b) that contains a robot and can evaluate the task, where ρ is the least distance from the middle point of \mathcal{L}_1 such that it contains a robot. Eventually, all of the robots will place themselves on circle \mathcal{C} .

Algorithm 2: Circle Formation

```

1 Procedure CIRCLEFORMATION()
2    $r \leftarrow$  myself.
3    $c_{r_d}$  is the desired position of robot  $r_d$  of coordinate  $(l, d)$ .
4    $\mathcal{L}_{c_{middle}}$  is the horizontal line passing through point  $c_{middle}$ .
5   if  $r.color = Line$  then
6     if  $r$  is a corner robot then
7        $r.color \leftarrow Corner$ 
8     else if  $r$  is a non-corner robot of  $\mathcal{L}_1$  and sees a robot with color Corner then
9        $r.color \leftarrow Moving2$ 
10      Move upward along  $\mathcal{L}(r)_\perp$ 
11  else if  $r.color = Moving2$  then
12    if  $\mathcal{L}(r) \neq \mathcal{L}_1$  then
13      if  $\mathcal{L}(r)$  is adjacent to  $\mathcal{L}_1$  i.e.,  $r$  is on  $\mathcal{L}_2$  then
14        if There are only corner robots on  $\mathcal{L}_1$  then
15          Move upward along  $\mathcal{L}(r)_\perp$  toward the line  $\mathcal{L}_{c_{middle}}$ 
16        else
17          if There are no robots between  $\mathcal{L}(r)$  and  $\mathcal{L}_1$  then
18            if  $r$  is on line  $\mathcal{L}_{c_{middle}}$  then
19              if  $r$  is at  $(l, middle)$  then
20                 $r.color \leftarrow Done$ 
21              else if  $r$  sees a robot with color Done at  $(l, \rho)$  then
22                 $r.color \leftarrow Moving3$ 
23                Move to  $c_{r_d}$ 
24              else if There are no robots with color Done or Moving3 in
                the vicinity of  $r$  then
25                 $r.color \leftarrow Moving3$ 
26                Move to  $c_{r_d}$ 
27            else
28              Move toward line  $\mathcal{L}_{c_{middle}}$ 
29  else if  $r.color = Moving3$  then
30    if  $r$  is at  $c_{r_d}$  then
31       $r.color \leftarrow Done$ 
32    else
33      Move to  $c_{r_d}$ 

```

Theorem 2. *Algorithm 2 CIRCLE FORMATION will create a circle \mathcal{C} from line \mathcal{L}_1 .*

Proof. The algorithm works in two folds: (1) first, it will move all of the non-corner robots of \mathcal{L}_1 to the horizontal line $\mathcal{L}_{c_{middle}}$ passing through point c_{middle} ; and (2) it will place the robots of $\mathcal{L}_{c_{middle}}$ on circle \mathcal{C} one by one. Note that only the second phase starts when all of the robots are on horizontal line $\mathcal{L}_{c_{middle}}$. So, we can treat each case separately.

- **Case 1.** Suppose that all of the robots are on line \mathcal{L}_1 . Now, a robot r cannot gauge the position of itself nor the length of the line. If it is a corner robot, it will change its color to *Corner*. Suppose that r is a non-corner robot – if it sees a robot with color *Corner*, then it will move to line \mathcal{L}_2 with color *Moving2*. Now, it can gauge the length of \mathcal{L}_1 and the position of it on \mathcal{L}_1 . Now, robot r will only move if there are no non-corner robots on \mathcal{L}_1 . It will move toward horizontal line $\mathcal{L}_{c_{middle}}$. Note that the movements occur sequentially. The robots move from one horizontal line \mathcal{L}_j to horizontal line \mathcal{L}_{j+1} when there are no robots between \mathcal{L}_1 and \mathcal{L}_j . Eventually, all of the robots will reach line $\mathcal{L}_{c_{middle}}$. Certainly, there will be no collisions. Now, we must show that a robot r can always evaluate line $\mathcal{L}_{c_{middle}}$. Note that the robots are moving upward. So, if r has completed its move and there are robots that have not completed their moves, then r may not see the corner. However, this is not the case in our algorithm, as r will only move if there are no robots in between $\mathcal{L}(r)$ and \mathcal{L}_1 .
- **Case 2.** Suppose that all of the robots are on line $\mathcal{L}_{c_{middle}}$. Now, if r is a middle robot, then it will alter its light to *Done*. Suppose that r is not a middle robot and is of coordinates (l, d) . Now, if it sees a robot with color *Done* at position (l, ρ) , then it will alter its light to *Moving3* and will move toward point c_{r_d} . It may happen that there are no robots with color *Done* nor *Moving3*. This means that r is at position (l, ρ) . Then, it will alter its light to *Moving3* and will move toward $c_{r_{d'}}$, where $d' = \rho$. Hence, all of the robots will eventually have color *Done*. Clearly, there are no collisions. Now, we must show that a robot r of coordinates (l, d) can always evaluate point c_{r_d} . For this, r needs the length of line \mathcal{L}_1 . Now, position $c_{r_{middle-i}}$ is closer than position $c_{r_{middle-i+1}}$ to line \mathcal{L}_1 . So, when robot $r_{middle-i+1}$ has completed its move, robot $r_{middle-i}$ will move to point $c_{r_{middle-i+1}}$. Now, robot $r_{middle-i}$ can see the corner robots and will eventually move to point $c_{r_{middle-i}}$, where $1 \leq i \leq middle - 2$.

Circle formation. Note that Algorithm 2 creates a semi-circle configuration from any initial configuration. By merely modifying this algorithm, we can easily solve the circle formation problem. First, a non-corner robot on \mathcal{L}_1 will check whether it can see at least one robot with color *Corner*. Now, if it can see exactly one robot with color *Corner* and there are no robots in the configuration with color *Moving2*, then it will move upward (in Figure 7a, r_1 and r_2 move upward). Now, if it can see exactly one robot with color *Corner* and there are robots in the configuration with color *Moving2* (Fig. 7b), then it will move downward if the closest *Moving2* robot is in the

up direction and vice versa (in Figure 7a, r_3 and r_4 move downward). Now, if there are two of the closest *Moving2*-colored robots or a non-corner robot can see both robots with color *Corner*, then it will move in either the up or down direction arbitrarily. Note that \mathcal{L}_1 cuts the configuration into two halves – an upper half, and a lower half. The robots in both halves will execute Algorithm 2, and we will eventually have a circle (Fig. 7d).

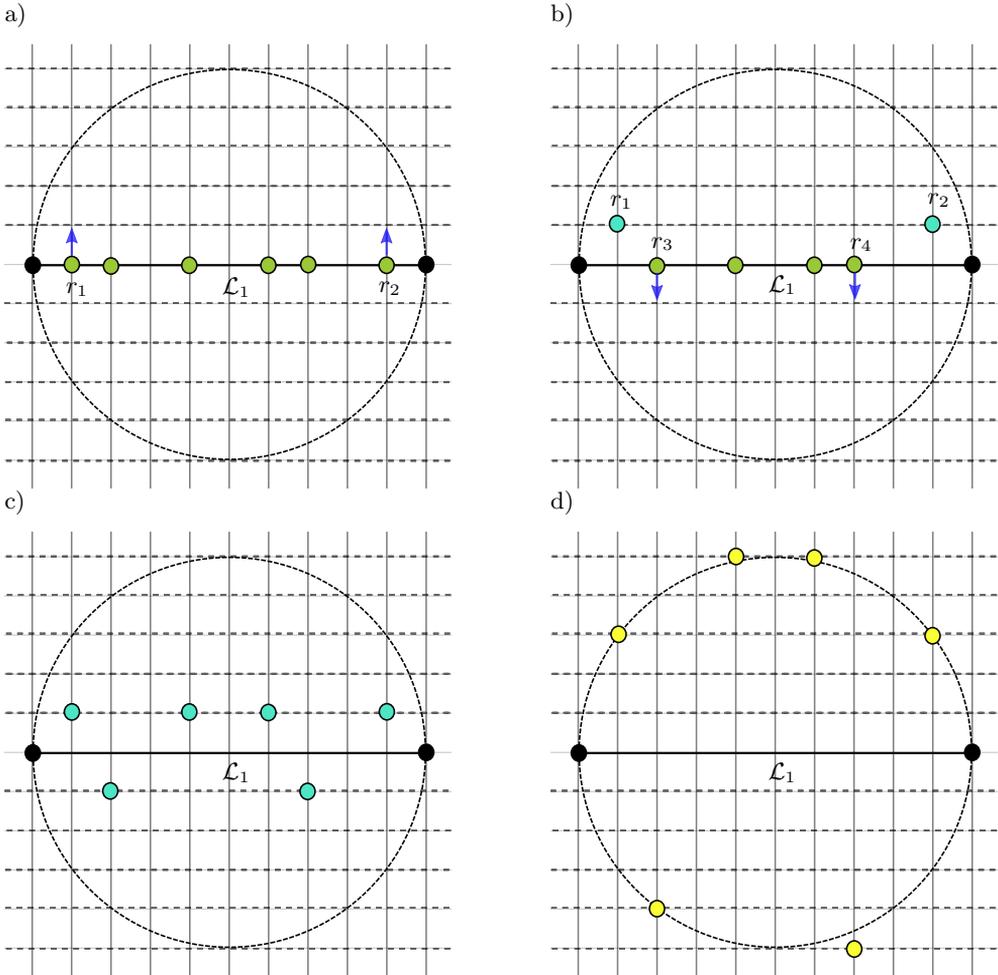


Figure 7. Illustration of movements of robots for circle formation: a) r_1 and r_2 see exactly one corner robot; as there are no robots with color *Moving2*, they move upward; b) r_3 and r_4 move downward; as closest *Moving2* colored robots are in up direction, they move downward; c) robots in both halves will execute Algorithm 2; d) completion of circle formation

□

4. Conclusion

Our algorithm solves the problem of line formation and circle formation deterministically from any initial configuration. This paper investigates the CIRCLE FORMATION problem for a set of oblivious asynchronous opaque robots on an infinite grid. Although the robots have unlimited visibility, the view of a robot can be blocked by another robot. The main difficulty of working in a grid environment is the movements of the robots. A robot can only move to one of its four adjacent grid points. Now, if these adjacent points are non-empty, then the robot cannot move, as moving to these points would create a collision. On the contrary, this situation will never appear in the continuous domain. Also, a robot moves one unit of length at each step in a straight line in a grid environment, whereas a robot can move in any direction by any amount with infinite precision in a continuous environment. These restrictions on movements increase the difficulty of the problem. In this discrete setting, we have shown that, from any given initial configuration, our algorithm solves the CIRCLE FORMATION problem deterministically using seven colors. Also, a subroutine of our algorithm deterministically solves the LINE FORMATION problem using three colors.

Several directions for future research may be explored. For instance, a more realistic model would be to consider *fat* robots (i.e., those with a finite extent). In a continuous environment, this problem has been studied. Another realistic model would be to work under limited visibility. The immediate next step would be to optimize the number of movements of the robots and try to use fewer colors. Another line of research could be to consider no agreements over the coordinate system.

Acknowledgements

The first author is supported by CSIR, Govt. of India. We thank the anonymous reviewers for their valuable comments, which helped us improve the quality and presentation of this paper.

References

- [1] Adhikary R., Bose K., Kundu M.K., Sau B.: Mutual Visibility by Asynchronous Robots on Infinite Grid. In: *Algorithms for Sensor Systems – 14th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGO-SENSORS 2018, Helsinki, Finland, August 23–24, 2018, Revised Selected Papers, Lecture Notes in Computer Science*, vol. 11410, pp. 83–101, Springer, 2018.
- [2] Barberá H.M., Quiñonero J.P.C., Zamora-Izquierdo M.A., Skarmeta A.G.: I-Fork: a flexible AGV system using topological and grid maps. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2, pp. 2147–2152, IEEE, 2003.
- [3] Bhagat S., Mukhopadhyaya K.: Optimum circle formation by autonomous robots. In: *Advanced Computing and Systems for Security*, pp. 153–165, Springer, 2018.

- [4] Bose K., Adhikary R., Chaudhuri S.G., Sau B.: Crash tolerant gathering on grid by asynchronous oblivious robots. In: *arXiv preprint arXiv:1709.00877*, 2017.
- [5] Bose K., Adhikary R., Kundu M.K., Sau B.: Arbitrary pattern formation on infinite grid by asynchronous oblivious robots. In: *WALCOM: Algorithms and Computation – 13th International Conference, WALCOM 2019, Guwahati, India, February 27 – March 2, 2019, Proceedings, Lecture Notes in Computer Science*, vol. 11355, pp. 354–366, Springer, 2019.
- [6] Bose K., Adhikary R., Kundu M.K., Sau B.: Arbitrary pattern formation on infinite grid by asynchronous oblivious robots, *Theoretical Computer Science*, vol. 815, pp. 213–227, 2020.
- [7] D’Angelo G., Di Stefano G., Klasing R., Navarra A.: Gathering of robots on anonymous grids and trees without multiplicity detection, *Theoretical Computer Science*, vol. 610, pp. 158–168, 2016.
- [8] Défago X., Konagaya A.: Circle formation for oblivious anonymous mobile robots with no common sense of orientation. In: *Proceedings of the second ACM international workshop on Principles of mobile computing*, pp. 97–104. 2002.
- [9] Défago X., Souissi S.: Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity, *Theoretical Computer Science*, vol. 396(1–3), pp. 97–112, 2008.
- [10] Deshpande A.M., Kumar R., Radmanesh M., Veerabhadrapa N., Kumar M., Minai A.A.: Self-Organized Circle Formation around an Unknown Target by a Multi-Robot Swarm using a Local Communication Strategy. In: *2018 Annual American Control Conference (ACC)*, pp. 4409–4413, IEEE, 2018.
- [11] Di Stefano G., Navarra A.: Gathering of oblivious robots on infinite grids with minimum traveled distance, *Information and Computation*, vol. 254, pp. 377–391, 2017.
- [12] Dutta A., Chaudhuri S.G., Datta S., Mukhopadhyaya K.: Circle Formation by Asynchronous Fat Robots with Limited Visibility. In: Ramanujam R., Ramaswamy S. (eds.), *Distributed Computing and Internet Technology – 8th International Conference, ICDCIT 2012, Bhubaneswar, India, February 2–4, 2012. Proceedings, Lecture Notes in Computer Science*, vol. 7154, pp. 83–93, Springer, 2012.
- [13] Feletti C., Mereghetti C., Palano B.: Uniform circle formation for swarms of opaque robots with lights. In: *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*, pp. 317–332, Springer, 2018.
- [14] Fischer M., Jung D., Meyer auf der Heide F.: Gathering Anonymous, Oblivious Robots on a Grid. In: *Algorithms for Sensor Systems – 13th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGO-SENSORS 2017, Vienna, Austria, September 7–8, 2017, Revised Selected Papers*, pp. 168–181. 2017.

- [15] Flocchini P., Prencipe G., Santoro N.: Self-deployment Algorithms for Mobile Sensors on a Ring. In: *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pp. 59–70, Springer, 2006.
- [16] Flocchini P., Prencipe G., Santoro N.: *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2012.
- [17] Flocchini P., Prencipe G., Santoro N.: Distributed Computing by Mobile Entities, *Current Research in Moving and Computing*, vol. 11340, 2019.
- [18] Flocchini P., Prencipe G., Santoro N., Viglietta G.: Distributed Computing by Mobile Robots: Solving the Uniform Circle Formation Problem. In: *International Conference on Principles of Distributed Systems*, pp. 217–232, Springer, 2014.
- [19] Flocchini P., Prencipe G., Santoro N., Viglietta G.: Distributed computing by mobile robots: uniform circle formation, *Distributed Computing*, vol. 30(6), pp. 413–457, 2017.
- [20] Fujinaga N., Yamauchi Y., Kijima S., Yamashita M.: Asynchronous pattern formation by anonymous oblivious mobile robots. In: *International Symposium on Distributed Computing*, pp. 312–325, Springer, 2012.
- [21] Lukovszki T., Meyer auf der Heide F.: Fast Collisionless Pattern Formation by Anonymous, Position-Aware Robots. In: *International Conference on Principles of Distributed Systems*, pp. 248–262, Springer, 2014.
- [22] Mamino M., Viglietta G.: Square Formation by Asynchronous Oblivious Robots. In: *arXiv preprint arXiv:1605.06093*, 2016.
- [23] Mondal M., Chaudhuri S.G.: Uniform circle formation by mobile robots. In: *Proceedings of the Workshop Program of the 19th International Conference on Distributed Computing and Networking*, pp. 1–2, 2018.
- [24] Pamecha A., Ebert-Uphoff I., Chirikjian G.S.: Useful metrics for modular robot motion planning, *IEEE Transactions on Robotics and Automation*, vol. 13(4), pp. 531–545, 1997.
- [25] Peleg D.: Distributed Coordination Algorithms for Mobile Robot Swarms: New Directions and Challenges. In: *International Workshop on Distributed Computing*, pp. 1–12, Springer, 2005.
- [26] Poudel P., Sharma G.: Universally Optimal Gathering Under Limited Visibility. In: *Stabilization, Safety, and Security of Distributed Systems – 19th International Symposium, SSS 2017, Boston, MA, USA, November 5–8, 2017, Proceedings*, pp. 323–340, 2017.
- [27] Poudel P., Sharma G., Aljohani A.: Sublinear-time mutual visibility for fat oblivious robots. In: *Proceedings of the 20th International Conference on Distributed Computing and Networking, ICDCN 2019, Bangalore, India, January 04–07, 2019*, pp. 238–247, ACM, 2019.

- [28] Sugihara K., Suzuki I.: Distributed motion coordination of multiple mobile robots. In: *Proceedings of 5th IEEE International Symposium on Intelligent Control 1990*, pp. 138–143, IEEE, 1990.
- [29] Suzuki I., Yamashita M.: Distributed anonymous mobile robots: Formation of geometric patterns, *SIAM Journal on Computing*, vol. 28(4), pp. 1347–1363, 1999.
- [30] Tanaka O.: *Forming a circle by distributed anonymous mobile robots*, Bachelor thesis, Department of Electrical Engineering, Hiroshima University, Japan, 1992.

Affiliations

Ranendu Adhikary

Jadavpur University, Department of Mathematics, Kolkata, West Bengal – 700032, India,
ranenduadhikary.rs@jadavpuruniversity.in, ORCID ID: <https://orcid.org/0000-0002-9473-2645>

Manash Kumar Kundu

Gayeshpur Government Polytechnic, Department of Science and Humanities, Kalyani, West Bengal – 741234, India, manashkrkundu.rs@jadavpuruniversity.in,
ORCID ID: <https://orcid.org/0000-0003-4179-8293>

Buddhadeb Sau

Jadavpur University, Department of Mathematics, Kolkata, West Bengal – 700032, India,
buddhadeb.sau@jadavpuruniversity.in

Received: 31.05.2020

Revised: 09.09.2020

Accepted: 22.09.2020