






# Mutual Visibility by Asynchronous Robots on Infinite Grid

Ranendu Adhikary<sup>(✉)</sup>, Kaustav Bose, Manash Kumar Kundu,  
and Buddhadeb Sau

Department of Mathematics, Jadavpur University, Kolkata, India  
{ranenduadhikary.rs,kaustavbose.rs,  
manashkrkundu.rs}@jadavpuruniversity.in, bsau@math.jdvu.ac.in

**Abstract.** Consider a set of autonomous, identical, opaque point robots in the Euclidean plane. The Mutual Visibility problem asks the robots to reposition themselves, without colliding, to a configuration where they all see each other, i.e., no three of them are collinear. In this paper, we consider the problem in a grid based terrain where the movements of the robots are restricted only along grid lines and only by a unit distance in each step. We consider the luminous robots model, in which each robot is equipped with an externally visible light which can assume a constant number of predefined colors. These colors serve both as internal memory and as a form of communication. The robots operate in Look-Compute-Move cycles under a fully asynchronous scheduler. The robots do not have any common global coordinate system or chirality and do not have the knowledge of the total number of robots. Our proposed distributed algorithm solves the problem for any arbitrary initial configuration and guarantees collision-free movements.

**Keywords:** Distributed computing · Autonomous robots · Mutual visibility · Robots with lights · Asynchronous · Look-Compute-Move cycle · Grid

## 1 Introduction

Robot swarms are a distributed system of autonomous mobile robots that collaboratively execute some complex tasks. Swarms of low-cost, weak, simple robots are emerging as a viable alternative to using a single powerful and expensive robot. In the traditional model of robot swarms, the mobile robots are assumed to be *autonomous* (there is no central control), *homogeneous* (they execute the same distributed algorithm), *anonymous* (they have no unique identifiers), *identical* (they are indistinguishable by their appearance) and *disoriented* (they do not agree on any global coordinate system). The robots do not have any direct means of communication. Each robot is equipped with sensor capabilities (i.e., vision) to perceive the positions of the other robots. The robots operate in Look-Compute-Move (LCM) cycles: when a robot becomes active it takes a snapshot of the positions of the other robots, then computes a destination based on the

snapshot using a deterministic algorithm (Compute), and then moves towards the destination along a straight line (Move).

The *opaque robots* or *obstructed visibility* model assumes that visibility can be obstructed by the presence of other robots: that is, two robots can see each other if and only if no other robot lies on the line segment joining them. The fundamental problem in this model is the MUTUAL VISIBILITY problem: starting from arbitrary distinct positions in the plane, the robots have to reposition themselves, within finite time and without colliding, to a configuration in which they are in distinct locations and no three of them are collinear. The problem is important as it provides a basis for any subsequent task requiring complete visibility. We consider this problem in the *robots with lights* or *luminous robots* model [8, 10]. In this model, each robot is equipped with an externally visible light which can assume a constant number of predefined colors.

## 1.1 Our Contribution

In this paper, we have considered the MUTUAL VISIBILITY problem in a grid based terrain. The infinite grid is a natural discretization of the Euclidean plane. Traditional spatial representation methods in robot navigation commonly represent the world as a two dimensional grid around the robot. Grid type floor layouts are also commonly implemented in real life robot navigation systems, e.g., industrial Automated Guided Vehicles (AGV), using magnets or optical guidances on the floor [3]. The 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. Our proposed distributed algorithm solves the MUTUAL VISIBILITY problem on infinite grid for any arbitrary initial configuration. We have solved the problem in the luminous robots model using 11 colors.

## 1.2 Earlier Works

While fundamental problems in autonomous mobile robots like GATHERING have been studied in grid environments [5–7, 11, 17], the MUTUAL VISIBILITY problem has only been studied in continuous Euclidean plane. The first distributed algorithm for the MUTUAL VISIBILITY problem was presented by Di Luna et al. [9] for oblivious and semi-synchronous robots. Later, Sharma et al. [13] analyzed and modified the round complexities of the algorithm in the fully synchronous model. The MUTUAL VISIBILITY problem under the luminous robots model was first studied by Di Luna in [8]. They solved the problem with semi-synchronous scheduler using 3 colors and with asynchronous schedulers using 3 colors under one axis agreement. Later Sharma et al. [14] attained this result using only 2 colors both for semi-synchronous and for asynchronous robots. Then a series

of papers [15,16] appeared aiming towards reducing the runtime of the algorithm. Recently Bhagat and Mukhopadhyaya [4] have solved the problem for asynchronous robots without any agreement on coordinate axes or chirality. The problem has also been considered for fat robots [12] and faulty robots [2].

## 2 Model and Definitions

In this section, we present the model and some basic definitions.

**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. All the robots are initially positioned on distinct grid points. The robots are assumed to be dimensionless and modeled as points on the plane. The robots do not have access to any global coordinate system. The total number of robots  $N$  is not known to them.

**Movement:** The movement of the robots are restricted only along grid lines from one grid point to one of its four neighboring grid points. Traditionally in discrete domains, robot movements are assumed to be instantaneous. For simplicity of analysis, we also assume the movements to be instantaneous. This implies that the robots are always seen on grid points, not on edges. However, our strategy will also work without this assumption.

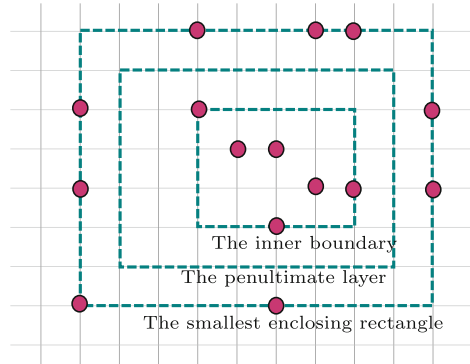
**Lights:** Each robot is equipped with an externally visible light which can assume a constant number of predefined colors. The robots explicitly communicate with each other using these lights. The lights are persistent (i.e., the color is not erased at the end of a cycle), but otherwise the robots are oblivious. The colors used in our algorithm are  $\mathcal{C} = \{Off, Boundary, RequestExpansion, Expanding, Moving1, Rectangle, Square, NextToCorner, Moving2, Moving3, Done\}$ .

**Visibility:** The visibility range of the robots is unlimited, but can be obstructed by the presence of other robots. A robot  $r_i$  can see another robot  $r_j$  if and only if there are no robots on the straight line segment  $\overline{r_i r_j}$ . The set of positions of all robots visible by a robot  $r$  at time  $t$ , expressed in its local coordinate system, is denoted by  $\mathcal{V}_r(t)$ , or simply  $\mathcal{V}_r$  when there is no ambiguity.

**Look-Compute-Move Cycles:** The robots, when active, operate according to the so-called LOOK-COMPUTE-MOVE cycle. In each cycle, a previously idle or inactive robot wakes up and executes the following steps. In the LOOK phase a robot takes the snapshot of the positions of the robots visible to it represented in its own coordinate system. Based on the perceived configuration, the robot performs computations according to a deterministic algorithm to decide a destination point  $p \in \mathbb{Z}^2$  (either the grid point on which it currently resides or one of the four neighboring grid points) and a color  $c \in \mathcal{C}$ . Finally based on the outcome of the algorithm, the robot changes its light to the computed color, and either remains stationary or makes an instantaneous move to an adjacent grid point.

**Scheduler:** We assume that the robots are controlled by a fully asynchronous adversarial scheduler. This implies that the amount of time spent in 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 perceived by a robot during the LOOK phase may significantly change before it actually makes a move.

**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 the robots. The boundary of the largest rectangle contained inside the smallest enclosing rectangle is called the *penultimate layer*, and each side of the rectangle is called *penultimate line segment*. The robots on the boundary of the smallest enclosing rectangle will be called *boundary robots*, and otherwise *interior robots*. The boundary of the smallest enclosing rectangle of the interior robots is called the *inner boundary*, and each side of the rectangle is called *inner boundary side*. A configuration will be called an *empty rectangle* if there are only boundary robots. 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  $\mathcal{L}$ , and otherwise it will be called *terminal on  $\mathcal{L}$*  (Fig. 1).



**Fig. 1.** Illustrations for the geometric definitions given in Sect. 2.

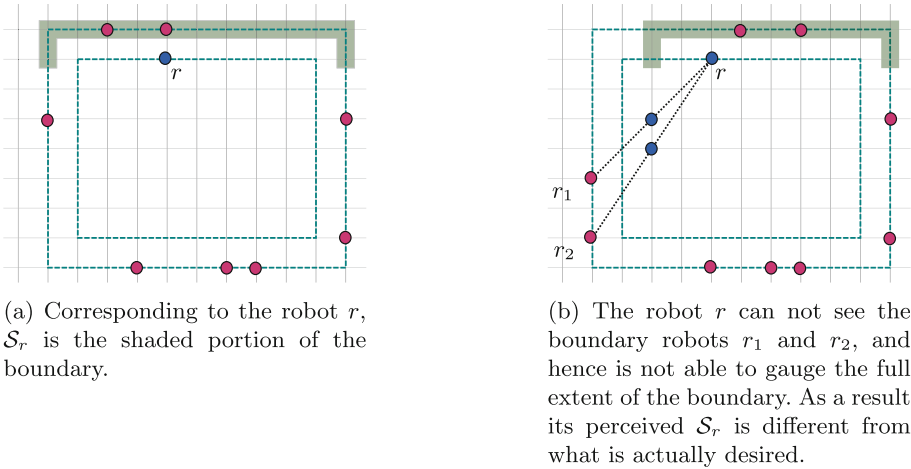
### 3 The Algorithm

The main difficulty of the problem arises from the restrictions imposed on the movements of the robots. If the four neighboring grid points of a robot are occupied, then any move made by it will lead to a collision. Our plan is to first create an empty rectangle configuration where all the robots are positioned on the boundary of the smallest enclosing rectangle. This phase is called the *Interior Depletion* phase. Our main idea is to exploit the symmetry of the empty rectangle to our advantage. In the *Symmetric Movements* phase, the robots will sequentially leave the empty rectangle and form a mutually visible configuration. During the movements, the robots may not be able to perceive the positions of other robots due to obstructed visibility, but can predict their movements from the symmetries of the empty rectangle. The robots at the corners of the empty rectangle will not move in the symmetric movements phase, but, however, will play an important role in the process. The specified lights of the corner robots will guide the movements of the other robots. From their positions relative to these corners, the robots will deduce their destination. However, the empty rectangle created in the interior depletion phase may not have robots at the corner points.

So in an intermediate phase, called *Corner Creation*, the robots terminal on the sides of the empty rectangle will reposition themselves at the corners. These phases are described in more detail in the following subsections.

### 3.1 Interior Depletion

The main idea of the algorithm is to sequentially move the interior robots to the boundary. In order to avoid collisions, only those robots that are on the inner boundary will move. However, there may not be an empty grid point on the boundary for the robot to position itself. In that case, the robot, using its lights, will ask the boundary robots to expand the boundary. A pseudo-code description of the procedure is presented in Algorithm 1. The geometric functions used in the algorithm are explained briefly in the following. The lights used in this phase are  $\{Off, Boundary, RequestExpansion, Expanding, Moving1\}$ .



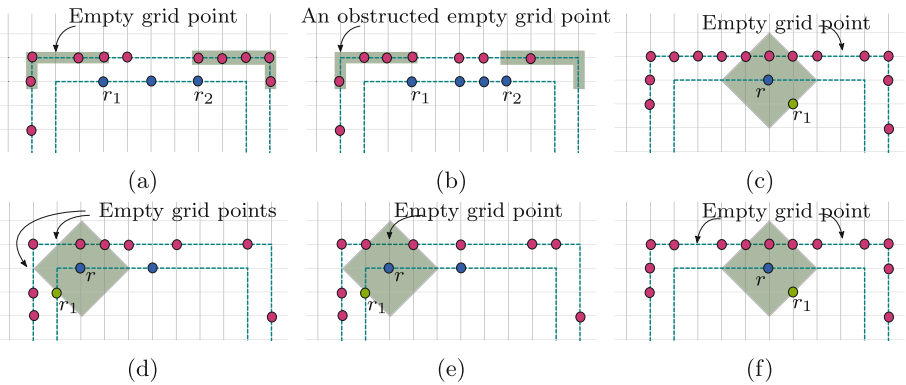
**Fig. 2.** Illustrations for the function  $FINDSPACE()$ .

The lights of all the robots are initially set to *Off*. Upon waking up, a robot  $r$  calls the function  $ONBOUNDARY()$  to decide if it is on the boundary. If it finds that there is an open-half plane, delimited by one of the grid lines passing through itself, containing no robots, then it concludes that it is a boundary robot and sets its light to *Boundary*. If  $r$  finds itself in the interior, then it calls the function  $ONINNERBOUNDARY()$ . If there is an open-half plane, delimited by one of the grid lines passing through itself, containing only robots with lights set to *Boundary*, then it is on the inner boundary. Only the robots on the inner boundary are allowed to move towards the boundary. Now, there are two cases to consider: the robot on the inner boundary is either on the penultimate layer or not.

If  $r$  is not on the penultimate layer and is terminal on an inner boundary side, then it has to move towards the boundary. But it will not move immediately. First, it will set its light to *Moving1*. Then in the next round, it will redo the same computations. An interior robot will move only if its light is already set to *Moving1* (in some previous round).

On the other hand, if  $r$  finds itself on a penultimate line segment  $\mathcal{L}$  and is terminal on  $\mathcal{L}$ , then it has to decide whether it is possible to move to the boundary. It does so using the function  $\text{FINDSPACE}()$ . The function  $\text{FINDSPACE}()$  works in the following way.

For the robot  $r$ , let  $\mathcal{S}_r$  denote the portion of the boundary as shown in the Fig. 2a. If  $\mathcal{L}$  contains more than one robot, then define  $\mathcal{H}_r$  as the closed-half plane, delimited by the grid line perpendicular to  $\mathcal{L}$  and passing through  $r$ , such that  $\mathcal{L} \cap \mathcal{H}_r$  contains no robot other than  $r$ . If there is no other robot on  $\mathcal{L}$  except  $r$ , then  $\mathcal{H}_r$  is the entire plane. The robot  $r$  will scan  $\mathcal{H}_r \cap \mathcal{S}_r$  for an empty grid point. If it finds an empty point, it makes sure that the shortest path to that point is not blocked by some other robot. Even if it finds an unobstructed empty point on  $\mathcal{H}_r \cap \mathcal{S}_r$ , a move towards 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. See Fig. 3.



**Fig. 3.** Illustrations for the function  $\text{FINDSPACE}()$ . (a) The robot  $r_1$  finds an empty grid point, while  $r_2$  does not find any empty grid point. (b) The robot  $r_1$  finds an empty grid point, but the shortest paths leading to it are blocked by other robots. (c)–(d) The robot  $r$  finds an empty grid point but there is a robot with light *Moving1* within Manhattan distance 2 in the direction it should move to get there. (e) There is a robot  $r_1$  with light *Moving1* within Manhattan distance 2, but not in the direction towards the empty point. Hence,  $\text{FINDSPACE}()$  will return True for  $r$ . (f)  $r$  is the only terminal robot on the penultimate line segment, and hence  $\mathcal{H}_r \cap \mathcal{S}_r = \mathcal{S}_r$ . It has found two empty grid points in  $\mathcal{S}_r$ . Here, it will choose the empty grid point on the left, because on the right side there is a robot with light *Moving1* within Manhattan distance 2.

If the function  $\text{FINDSPACE}()$  returns True, then  $r$  will move towards the empty grid point on the boundary. Again, it will move only if its light is already

set to *Moving1*, otherwise it will only change its light to *Moving1*. If `FINDSPACE()` returns `False`,  $r$  will set its light to *RequestExpansion*, requesting the boundary robots to expand the smallest enclosing rectangle so that an empty space is created on the boundary.

---

### Algorithm 1. Interior Depletion

---

```

1 Procedure INTERIORDEPLETION()
2    $r \leftarrow \text{myself}$ 
3   while EMPTYRECTANGLE() = FALSE do
4     if  $r.\text{color} = \text{Off}$  then
5       if ONBOUNDARY() = True then
6          $r.\text{color} \leftarrow \text{Boundary}$ 
7       else if ONINNERBOUNDARY() = True then
8         if ONPENULTIMATE() = True then
9           if TERMINALONPENULTIMATE() = True then
10            if FINDSPACE() = True then
11               $r.\text{color} \leftarrow \text{Moving1}$ 
12            else
13               $r.\text{color} \leftarrow \text{RequestExpansion}$ 
14          else if TERMINALONINNERBOUNDARY() = True then
15             $r.\text{color} \leftarrow \text{Moving1}$ 
16        else if  $r.\text{color} = \text{RequestExpansion}$  then
17          if ONPENULTIMATE() = False then
18             $r.\text{color} \leftarrow \text{Off}$ 
19        else if  $r.\text{color} = \text{Moving1}$  then
20          if ONBOUNDARY() = True then
21             $r.\text{color} \leftarrow \text{Boundary}$ 
22          else if ONINNERBOUNDARY() = True then
23            if ONPENULTIMATE() = True then
24              if TERMINALONPENULTIMATE() = True then
25                if FINDSPACE() = True then
26                  Move towards the empty boundary point
27                else
28                   $r.\text{color} \leftarrow \text{RequestExpansion}$ 
29            else if ONPENULTIMATE() = False then
30              Move towards boundary
31        else if  $r.\text{color} = \text{Expanding}$  then
32          if EXPANSIONCOMPLETED() = True then
33             $r.\text{color} \leftarrow \text{Boundary}$ 
34        else if  $r.\text{color} = \text{Boundary}$  then
35          if ONBOUNDARY() = False then
36            Move towards the boundary
37          else if All terminal robots on the penultimate grid line segment next to it
38             have lights set to RequestExpansion then
39             $r.\text{color} \leftarrow \text{Expanding}$ 
40            Move outward

```

---

Now consider a robot  $r'$  with light *Boundary*. It will first recheck if it is still on the boundary. It may happen that some of the boundary robots on its grid line had started the expansion earlier, leaving  $r'$  inside the smallest enclosing rectangle. However, these robots will only move at most one hop from the previous boundary. Thus, if  $r'$  finds itself in the interior, it can move to the new boundary in a single step. If  $r'$  is on the boundary, and it observes that all terminal robots on the penultimate line segment next to it have set their lights to *RequestExpansion*, then it will change its light to *Expanding* and will start

moving outwards. If a robot finds its light set to *Expanding*, it checks whether all its fellow boundary robots from the previous round have completed their moves by checking the penultimate line next to it. If it finds that the expansion is completed, it will change its light to *Boundary*.

A rigorous proof of correctness of the algorithm is omitted due to space constraints. The following two lemmas address the two main issues regarding the correctness of the algorithm. The proofs of the lemmas in this subsection are briefly presented in Appendix A.

**Lemma 1.** *The interior depletion phase is collision free.*

**Lemma 2.** *If a robot  $r$  at time  $t_0$  on the penultimate layer sets its light to *RequestExpansion*, then there exist a time  $t$  ( $> t_0$ ) when the robot  $r$  reaches the boundary.*

The interior depletion phase is completed when all the robots are on the boundary of the smallest enclosing rectangle, and the lights of all the robots are set to *Boundary*. However, it may not be possible for the robots to locally detect this. This is because, if the first condition is attained, a robot  $r$  can not determine whether the second condition is satisfied, as it may not be able to see all the robots on boundary line on which it resides. We say that a robot detects the partial completion of the interior depletion phase if it can determine if the first condition is satisfied. It does so using the function `EMPTYRECTANGLE()`, which returns True if the following conditions are satisfied:

1. all robots in  $\mathcal{V}_r$  are on the boundary of their smallest enclosing rectangle,
2. the lights of all the robots in  $\mathcal{V}_r$  are set to *Boundary*.

**Lemma 3.** *The function `EMPTYRECTANGLE()` can detect the partial completion of the interior depletion phase.*

**Theorem 1.** *The algorithm `INTERIOR DEPLETION` creates an empty rectangle configuration starting from any arbitrary configuration.*

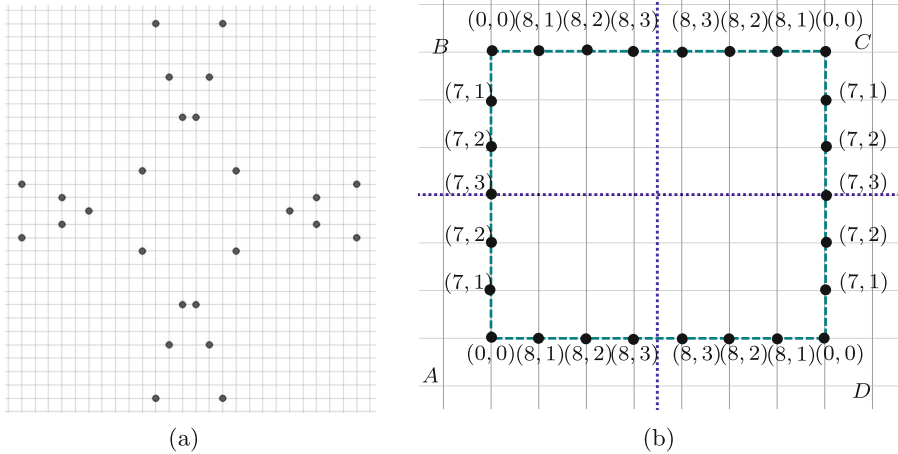
### 3.2 Symmetric Movements

Due to space constraints, we will not describe the corner creation phase. A brief discussion on this phase is given in Appendix C. This phase will require four different lights, namely, *Rectangle*, *Square*, *NextToCorner* and *Moving2*. The objective of the corner creation phase is to create an empty rectangle configuration with its four corners occupied by robots with specified lights. However, this may not be achievable if the empty rectangle is a square. Due to space restrictions, we will describe the symmetric movements phase assuming that the starting configuration is the generic configuration of a non-square rectangle having four corner robots with lights set to *Rectangle*. The algorithms for other configurations, like squares with possibly some missing corners or a straight line configuration, are based on the same movement strategy subject to some minor



modifications. The lights that will be used in the symmetric movements phase are  $\{Moving3, Done\}$ . See Appendix B, for the proofs of the claims in this subsection.

In this phase, the non-terminal robots on the sides of the empty rectangle will leave the boundary and move outwards along the grid line passing through its starting position. The extent of their movement will depend on (1) the length of the sides of the empty rectangle, and (2) the starting position of the robot on the boundary.



**Fig. 4.** (a) The final mutually visible configuration for a  $7 \times 8$  empty rectangle. (b) The coordinate system of the boundary of the rectangle.

The grid points on the boundary of the rectangle will be given coordinates  $(p, k)$ , where  $p =$  the size of the side of the rectangle it belongs to, and  $k =$  its distance from the closest corner. The coordinates of the four corners will be  $(0, 0)$ . This coordinate scheme is illustrated in Fig. 4b. The group of symmetries of the rectangle is generated by reflections with respect to perpendicular bisectors of its sides. The group of symmetries induces an equivalence relation on the grid points on the rectangle:  $P \sim Q$  if and only if  $Q$  can be obtained from  $P$  by some reflection operations. This partitions the set of grid points on the rectangle into equivalence classes. The distance, that two robots on starting points belonging to the same equivalence class should move, have to be equal. Two points are equivalent if and only if their coordinates are equal (See Fig. 4b). We shall exploit these symmetries to design a recursive function called `DESTINATION` that computes the destination points of the robots. The distance that a robot starting from  $(p, k)$  should move is `DESTINATION` $(m, n, p, k)$ , where  $m, n$  ( $m \geq n$ ) are size of the sides.

The pseudocode of the function `DESTINATION` is omitted. We shall briefly sketch out the recursive computation of the destination points corresponding to

all the points on the rectangle. At each step, the algorithm computes destinations corresponding to all the points belonging to an equivalence class. In other words, the iteration runs over the set of equivalence classes of the grid points on the rectangle. The set of all the destinations computed up to the  $i$ th step of the procedure will be denoted by  $\mathcal{C}_i$ .

**Step 0:** Robots at the corners ( $k = 0$ ) will not move. Hence,  $\mathcal{C}_0$  consists of the starting positions of the corner robots.

**Step 1 and 2:** At step 1 and 2, the destinations corresponding to the middle points of the sides, i.e.,  $k = \lceil \frac{m}{2} \rceil - 1$  and  $\lceil \frac{n}{2} \rceil - 1$ , are computed. Suppose we are computing the destinations corresponding to the middle points of a side  $AB$ . Draw two straight lines through  $A$  and  $B$ , parallel to the two diagonals of the rectangle. Let  $\mathcal{H}_A$  and  $\mathcal{H}_B$  be the open half-planes delimited by these straight lines that do not contain the rectangle. Then the destination of the robot(s) at the middle of  $AB$  will be the nearest grid points belonging to  $\mathcal{H}_A \cap \mathcal{H}_B$  (the shaded region in Fig. 8 in the Appendix).

**Step 3 to  $\lceil \frac{m}{2} \rceil$ :** In these steps, the destinations corresponding to the remaining grid points on the larger side are computed. This is done in a recursive manner. Suppose that at the  $i$ th step, we are to compute the destinations for grid points  $\{x_1, x_2, x_3, x_4\}$ . We shall denote the computed destination corresponding to  $x_j$  as  $y_j$ . The destinations are computed according to the following rules.

1. The destinations computed in step  $i$  are strictly farther from the rectangle than the ones computed in step  $i - 1$ .
2. Choose any one of  $\{x_1, x_2, x_3, x_4\}$ , say  $x_1$ . Then the corresponding destination  $y_1$  is the grid point (on the grid line passing through  $x_1$ ) closest to the rectangle (respecting condition 1) such that no three points in  $\mathcal{C}_{i-1} \cup \{y_1\}$  are collinear. The destinations  $y_2, y_3, y_4$  are obtained from  $y_1$  from the reflectional symmetries. Since no two points in  $\mathcal{C}_{i-1}$  are collinear with  $y_1$ , from the reflectional symmetries we can say that the same is true for  $y_2, y_3$  and  $y_4$ . But it is still not apparent that no three points in  $\mathcal{C}_i = \mathcal{C}_{i-1} \cup \{y_1, y_2, y_3, y_4\}$  are collinear. We prove this in Lemma 4.

**Step  $\lceil \frac{m}{2} \rceil + 1$  to  $\lceil \frac{m}{2} \rceil + \lceil \frac{n}{2} \rceil - 2$ :** In these steps, the destinations corresponding to the grid points on the smaller side are computed. The procedure is the same as before.

**Lemma 4.** *If no three points in  $\mathcal{C}_{i-1}$  are collinear, then the same is true for  $\mathcal{C}_i$ .*

**Theorem 2.** *No three points of the destinations computed by the function DESTINATION are collinear.*

*Proof.* Since no three points of  $\mathcal{C}_0$  are collinear, the result immediately follows from the Lemma 4.

We shall now describe the movement strategy. A pseudocode description of the procedure is given in Algorithm 2. As mentioned earlier, the algorithm is

described for only non-square empty rectangle configurations with four occupied corners. In the function `DESTINATION`, the destinations corresponding to the middle points of the sides were computed first in the recursive process. But the movements will occur in the exactly opposite order. The robots will sequentially leave the boundary with the ones closest to the corner moving first. A boundary robot will call the function `ELIGIBLETOMOVE()` to determine whether it should start moving. It checks if the following conditions are satisfied:

1. It can see at least one corner robot on its boundary side, and two corner robots on the opposite side. In Fig. 5,  $r_1$  can see  $c_1, c_3$  and  $c_4$ .  $r_3$  can also see  $c_2, c_3$  and  $c_4$ . But  $r_5$  cannot see  $c_1$  or  $c_2$ , and so it is not eligible to move yet.
2. If there were robots initially on its boundary side between it and the corner(s) it can see, they have already completed their movements and changed their lights to *Done*. The robot  $r_1$  checks this by scanning the shaded region *A*, which is empty. Hence  $r_1$  is eligible to move. But when  $r_3$  scans the region *B*, it finds  $r_2$  with its light set to *Moving3*, and hence it will not move.

If `ELIGIBLETOMOVE()` returns True, the robot will change its light to *Moving3* and leave the boundary. Note that a robot can leave the boundary even before the corner creation phase is completed. This is because the robot leaves the boundary when it sees at least three corners. The fourth corner is probably yet not created. We call this a premature move. But it can determine if it has made a premature move just after moving one hop from the boundary. This is because if the other corner is created, it will be able to see from the grid line one hop away from the boundary. If it can't, it will wait for the completion of the corner creation phase. Note that at most one robot on a boundary line can make a premature move. Also note that `PREMATURE()` will always return false if the robot is more than one hop away from the boundary.

When the robot is moving, after each one hop move it has to compute `DESTINATION( $m, n, p, k$ )` to find whether it has reached its destination. But that requires the knowledge of  $m$  and  $n$ . Consider the robot  $r_2$  in Fig. 5. Since the robots closer to the corners move farther,  $r_2$  will always be able to see  $c_2$  and  $c_3$ . But to know the size of both the boundary sides, it also has to see another corner. If it cannot see  $c_1$ , then its view must be obstructed by some robot in the shaded region in Fig. 5. So  $r_2$  now has to decide if its view is obstructed by a moving robot or a robot that has reached its destination. If  $r_2$  scans the shaded region and it finds a robot with light *Moving3* below it, then `BLOCKEDBYMOVINGROBOT()` will return True. Notice that if  $r_2$  can't see  $c_1$ , it can not identify

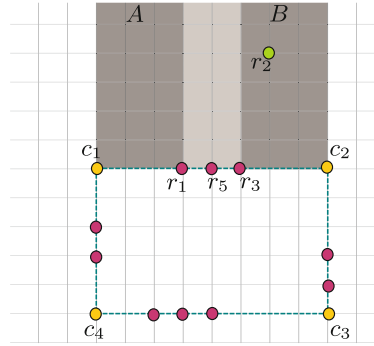


Fig. 5. Illustrations supporting the proof of Theorem 3

the actual extent (on the left side) of the shaded region. But this is not a major problem as there can not be any robots (from any branch) moving in the area beyond the left boundary of the shaded region. The robots, upon reaching their destination points, will change their lights to *Done*.

---

### Algorithm 2. SYMMETRIC MOVEMENTS

---

```

1 Procedure SYMMMOVEMENT()
2   r ← myself
3   if I am on the empty rectangle and ELIGIBLETOMOVE() = True then
4     r.color ← Moving3
5     Move outwards
6   else if r.color = Moving3 and PREMATURE() = False then
7     if I can see at least three robots with light Rectangle then
8       d ← DESTINATION(m, n, p, k)
9       if r.position = d then
10        r.color = Done
11      else
12        Move
13    else if BLOCKEDBYMOVINGROBOT() = False then
14      Move

```

---

**Theorem 3.** *Algorithm SYMMETRIC MOVEMENTS correctly leads all the robots to the destinations computed by the DESTINATION function.*

From Theorem 2 and 3, we can conclude the following.

**Theorem 4.** *The MUTUAL VISIBILITY problem on infinite grid can be solved using 11 colors.*

Note that the robots terminate the execution once their lights are set to *Rectangle* or *Done*. We say that the MUTUAL VISIBILITY problem is solved with detection if we additionally require that a robot terminates only after it detects that the mutual visibility is attained. This can be easily achieved, but will require one extra color. Each corner robot can determine if the symmetric movements have been completed in its quadrant, and then changes its light to the extra color. When all four corner robots change their colors, it implies that a mutually visible configuration is attained and all the robots in the configuration can detect this.

**Theorem 5.** *The MUTUAL VISIBILITY problem on infinite grid can be solved with detection using 12 colors.*

## 4 Conclusion

Our proposed distributed algorithm solves the MUTUAL VISIBILITY problem on infinite grid for any arbitrary initial configuration under the luminous robots model using 11 colors. We considered the robots as dimensionless points. A more realistic model would be to consider robots with physical extent, i.e., fat robots. The MUTUAL VISIBILITY problem for fat robots is solved as a subroutine of the gathering algorithm presented in [1]. But it is assumed that each robot knows

the size of the team. Recently, Sharma et al. have solved the problem in a fully synchronous setting [12]. It would be interesting to see if our strategy can be extended to solve the problem for fat robots in asynchronous setting with less assumptions. Another direction would be to investigate if the number of colors used or the number of moves by the robots can be reduced.

**Acknowledgements.** The first three authors are supported by CSIR, Govt. of India, NBHM, DAE, Govt. of India and UGC, Govt. of India respectively. We would like to thank the anonymous reviewers for their valuable comments which helped us improve the quality and presentation of this paper.

## Appendix A Correctness of Interior Depletion

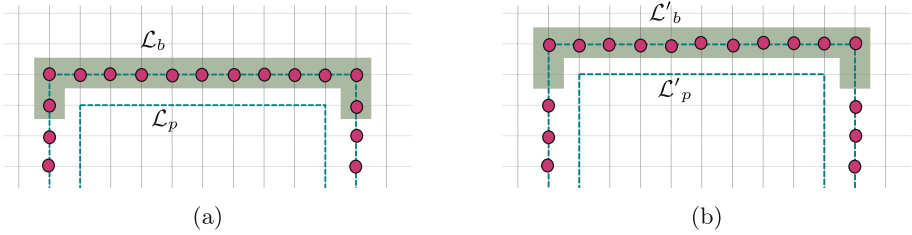
### A.1 Proof of Lemma 1

Collision can only occur when a robot  $r$  is moving along the penultimate layer towards an empty space in the boundary in the situations shown in Fig. 3c and d. Suppose that two robots  $r$  and  $r_1$ , at Manhattan distance 2 from each other, computes the same destination point. Initially none of them had their lights set to *Moving1*. This is because if one of them had its light set to *Moving1* when the other one takes the snapshot, it would not have computed a destination point. So they will first change their lights to *Moving1*, say at time  $t$  and  $t_1$  respectively. Let  $t \leq t_1$ . Now in its next LOOK phase at time  $t_2 (> t_1 \geq t)$ ,  $r_1$  perceives that  $r$  has either already made its move or is yet to move but has set its light to *Moving1*. In either case,  $r_1$  will not move according to our algorithm. Hence, there will be no collision.  $\square$

### A.2 Proof of Lemma 2

Assume that the robot  $r$  on a penultimate line segment  $\mathcal{L}$  at time  $t_0$  has set its light to *RequestExpansion*. If the other terminal robot on  $\mathcal{L}$ , say  $r'$ , also sets its light to *RequestExpansion*, then the corresponding boundary robots will execute the expansion. Otherwise,  $r'$  and subsequently the other robots that will become terminal on  $\mathcal{L}$  will move to the boundary. Eventually, either we have another terminal robot requesting expansion, or  $r$  is the only robot remaining on  $\mathcal{L}$  still with light *RequestExpansion*. Therefore, the corresponding boundary will eventually execute expansion.

After the expansion,  $r$  is now not on the penultimate line. Then there is a time  $t'$  when it will again move to the new penultimate line  $\mathcal{L}'$ . Now there could be at most two robots on  $\mathcal{L}'$ , since only the terminal robots on the inner boundary line move. An expansion always creates at least two empty points on the boundary (See Fig. 6), and one of them is in  $\mathcal{H}_r \cap \mathcal{S}_r$ . If  $r$  moves to the empty point, then we are done. If not, then it implies that there is a robot  $r_1$  with light *Moving1* within Manhattan distance two in the direction it intends to move.

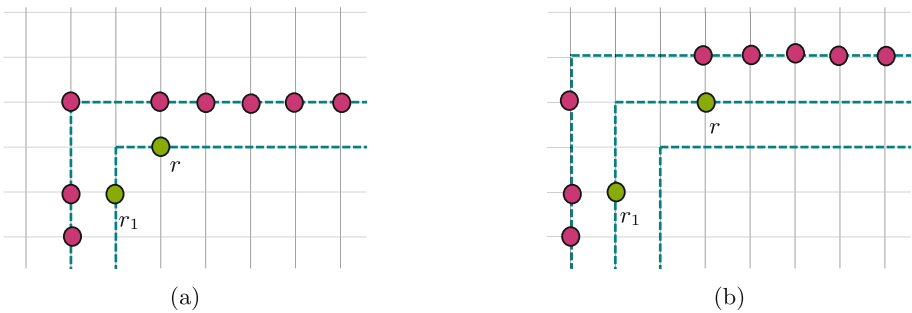


**Fig. 6.** (a) There is no empty grid point in the shaded region. (b) Two new empty points are created after the expansion

**Case 1:** Assume that  $r_1$  is also on  $\mathcal{L}'$ . But since  $\mathcal{H}_r \cap \mathcal{S}_r$  and  $\mathcal{H}_{r_1} \cap \mathcal{S}_{r_1}$  are in opposite directions,  $r_1$  is not in the direction in which  $r$  wants to move (and vice versa).

**Case 2:** Let  $r_1$  be on the grid line below  $\mathcal{L}'$ , as shown in Fig. 3c and f. But since  $r$  is the only robot on  $\mathcal{L}'$ , it has at least two empty points available in two directions. Then it will choose the empty point which is not towards  $r_1$ .

**Case 3:** Now consider the situation shown in Fig. 3d, where  $r_1$  is a robot moving on an adjacent penultimate line segment. Then  $r$  and  $r_1$  will request another expansion. It can be seen from Fig. 7, that in the new configuration, both robots will be able to move to an empty boundary point.  $\square$



**Fig. 7.** (a) The situation is similar to the example shown in Fig. 3d. (b) The subsequent configuration after both boundary sides expand and both  $r$  and  $r_1$  move to the new penultimate layer.

### A.3 Proof of Lemma 3

Suppose that (1) the robots in  $\mathcal{V}_r$  form an empty rectangle, (2) the lights of all the robots in  $\mathcal{V}_r$  are set to *Boundary*. It may happen that some robots in  $\mathcal{V}_r$  (with lights set to *Boundary*) are actually not on the boundary, but on the penultimate layer. Then these robots are in the middle of an expansion, but are

yet to change their lights to *Expanding*, and are obstructing some robots on the boundary also having light *Expanding*. We argue that this is not possible.

First of all, if  $r$  itself was executing an expansion in a previous round, then all the fellow robots, with which it had previously shared a boundary side, must have also completed the expansion. This is because  $r$  has its light set to *Boundary*. Now for the remaining three boundary sides, if the robots are executing an expansion, then they must be instructed to do so by some robot in the interior with light *RequestExpansion*. But  $\mathcal{V}_r$  has only robots with light *Boundary*. Hence, all the robots in  $\mathcal{V}_r$  are indeed boundary robots. Hence, the empty rectangle configuration is achieved.  $\square$

## Appendix B Correctness of Symmetric Movements

### B.1 Proof of Lemma 4

Suppose that there are three points in  $\mathcal{C}_i$ , say  $\{u, v, w\}$ , that are collinear. No three points in  $\mathcal{C}_{i-1}$  are collinear. So at least one of the three points is in  $\mathcal{C}_i \setminus \mathcal{C}_{i-1}$ , say  $u$ . But  $u$  is computed in such a way that it is collinear with no two points in  $\mathcal{C}_{i-1}$ . So another one among the three points must be in  $\mathcal{C}_i$ , say  $v$ . From the symmetries, we can say that  $v$  is one of the three possible points shown in Fig. 8 as  $\{v_1, v_2, v_3\}$ . Clearly the straight lines through  $u$  and  $v_1$  or  $u$  and  $v_3$  do not pass through any other point in  $\mathcal{C}_{i-1}$ . If the straight line through  $u$  and  $v_2$  passes through a point  $z \in \mathcal{C}_{i-1}$ , from symmetry it will also pass through another point  $z' \in \mathcal{C}_{i-1}$  (See Fig. 8). This implies that the straight line passes through two points in  $\mathcal{C}_{i-1}$ , which is not possible. Hence, no three points in  $\mathcal{C}_i$  are collinear.  $\square$

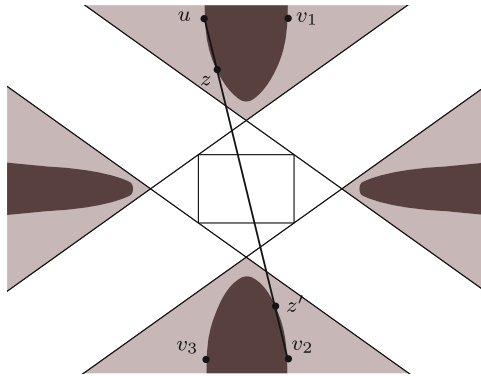


Fig. 8. Illustration supporting the proof of Lemma 4.

## B.2 Proof of Theorem 3

Since the robots closer to the corners move farther, a robot  $r$  will always be able to see at least two corners. Due to obstructions, it may not see another corner. We show that this will not create any lock cases.

**Case 1: ( $\text{BlockedByMovingRobot}() = \text{False}$ ):** If it sees no robot with light  $\overline{\text{Moving3}}$ , its view must be obstructed by a robot with light  $\text{Done}$  that has already reached its destination. Hence, this is clearly not the destination point of  $r$ . So  $r$  will move.

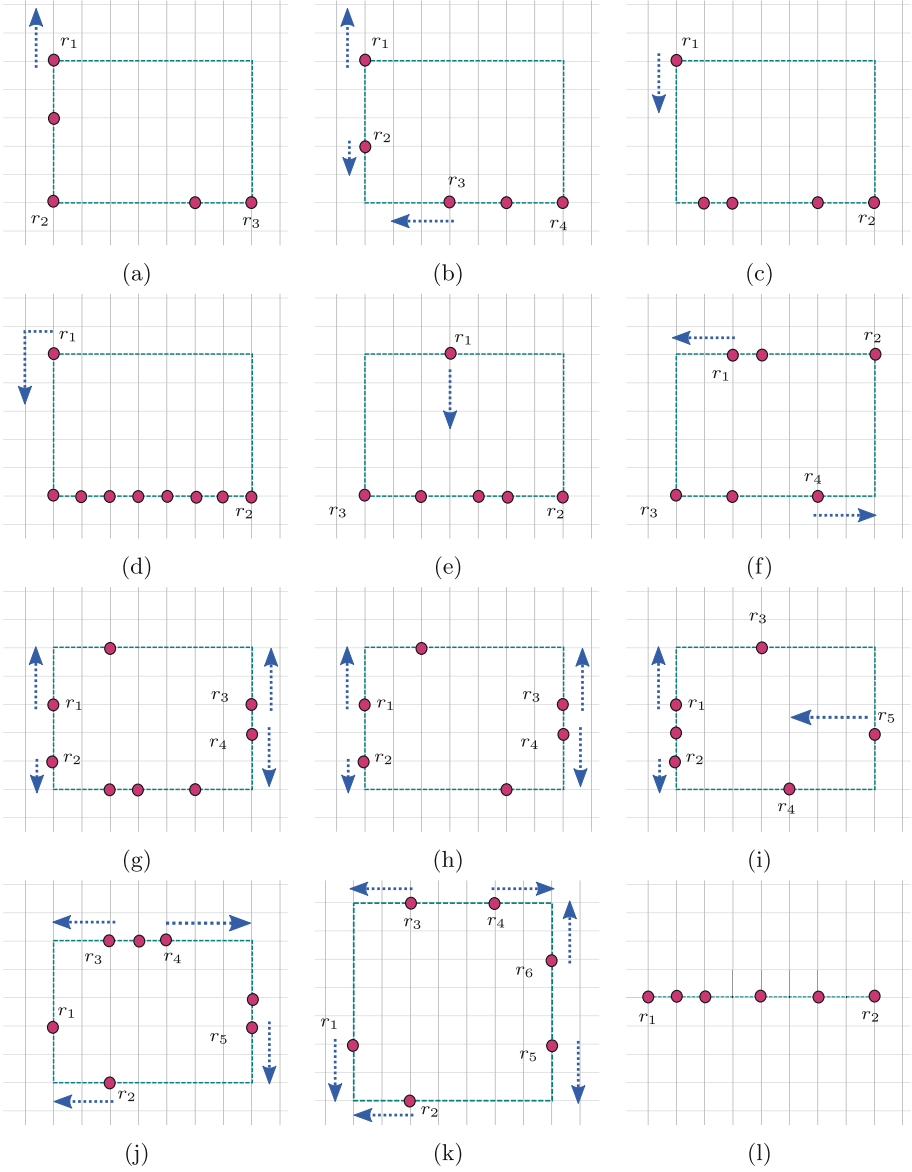
**Case 2 ( $\text{BlockedByMovingRobot}() = \text{True}$ ):** If it sees a robot with light  $\overline{\text{Moving3}}$ , it simply waits. In our movement strategy, at any time at most two robots can be moving in the shaded region (see Fig. 5). Hence, it sees exactly one moving robot below it, say  $r'$ . Clearly no moving robot is obstructing  $r'$ 's view. Hence,  $r'$  will eventually move or turn its light to  $\text{Done}$ .  $\square$

## Appendix C Corner Creation

The lights that will be used in this phase are  $\{\text{Rectangle}, \text{Square}, \text{NextToCorner}, \text{Moving2}\}$ . The objective of this phase is to occupy the corners of the empty rectangle by robots with specified lights. For simplicity, we shall first assume that there are at least two robots on each of the four sides. Only the robots terminal on the boundary sides will move in this phase. If the configuration is a non-square rectangle, then the terminal robots on the larger side will move to the corner and set its light to  $\text{Rectangle}$ . Note that the robots can determine the length of the sides of the rectangle. However, if it is a square, it may not be always possible to break tie. If two terminal robots on adjacent sides of the square move to the corner, there will be a collision. If it is possible to break tie, then one of them will go to the corner and set its light to  $\text{Square}$ . Otherwise, the robots will move to the point adjacent to the corner and then set its light to  $\text{NextToCorner}$ . While moving, the terminal robots will set their lights to  $\text{Moving2}$ .

However, this simple scheme will not be always applicable. The initial empty rectangle configuration may have different anomalies. For example, some sides may have only a single robot, or all the robots could lie on a single straight line, or form an L-shape, etc. (See Fig. 9). While designing algorithms for these configurations, the following issues should be properly addressed. A robot may not be always able to distinguish between two configurations from their local views. In these cases, the movement specified for the robot in both configurations should not contradict each other. During the movements, the configuration may change from one case to another. Due to the asynchronous scheduler, the adversary may delay the move of a robot, which will have a pending move based on an out-dated view of the configuration. Such pending moves should not cause any inconsistencies in the algorithm. The algorithms for these different configurations are pictorially presented in Fig. 9. Proofs and other details of the algorithms are omitted due to space constraints.





**Fig. 9.** Movements in the corner creation phase for atypical empty rectangle configurations.

## References

1. Agathangelou, C., Georgiou, C., Mavronicolas, M.: A distributed algorithm for gathering many fat mobile robots in the plane. In: ACM Symposium on Principles of Distributed Computing, PODC 2013, 22–24 July 2013, Montreal, QC, Canada, pp. 250–259 (2013). <https://doi.org/10.1145/2484239.2484266>
2. Aljohani, A., Sharma, G.: Complete visibility for mobile robots with lights tolerating faults. *Int. J. Netw. Comput.* **8**(1), 32–52 (2018). <http://www.ijnc.org/index.php/ijnc/article/view/166>
3. Barberá, H.M., Quiñero, J.P.C., Zamora-Izquierdo, M.A., Gómez-Skarmeta, A.F.: i-fork: a flexible AGV system using topological and grid maps. In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation, ICRA 2003, 14–19 September 2003, Taipei, Taiwan, pp. 2147–2152 (2003). <https://doi.org/10.1109/ROBOT.2003.1241911>
4. Bhagat, S., Mukhopadhyaya, K.: Optimum algorithm for mutual visibility among asynchronous robots with lights. In: Proceedings of 19th International Symposium Stabilization, Safety, and Security of Distributed Systems, SSS 2017, 5–8 November 2017, Boston, MA, USA, pp. 341–355 (2017). [https://doi.org/10.1007/978-3-319-69084-1\\_24](https://doi.org/10.1007/978-3-319-69084-1_24)
5. Bose, K., Adhikary, R., Chaudhuri, S.G., Sau, B.: Crash tolerant gathering on grid by asynchronous oblivious robots. *CoRR* abs/1709.00877 (2017). <http://arxiv.org/abs/1709.00877>
6. D’Angelo, G., Stefano, G.D., Klasing, R., Navarra, A.: Gathering of robots on anonymous grids and trees without multiplicity detection. *Theor. Comput. Sci.* **610**, 158–168 (2016). <https://doi.org/10.1016/j.tcs.2014.06.045>
7. Fischer, M., Jung, D., Meyer auf der Heide, F.: Gathering anonymous, oblivious robots on a grid. In: Fernández Anta, A., Jurdzinski, T., Mosteiro, M.A., Zhang, Y. (eds.) *ALGOSENSORS 2017*. LNCS, vol. 10718, pp. 168–181. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-72751-6\\_13](https://doi.org/10.1007/978-3-319-72751-6_13)
8. Luna, G.A.D., Flocchini, P., Chaudhuri, S.G., Poloni, F., Santoro, N., Viglietta, G.: Mutual visibility by luminous robots without collisions. *Inf. Comput.* **254**, 392–418 (2017). <https://doi.org/10.1016/j.ic.2016.09.005>
9. Luna, G.A.D., Flocchini, P., Poloni, F., Santoro, N., Viglietta, G.: The mutual visibility problem for oblivious robots. In: Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada (2014). <http://www.cccg.ca/proceedings/2014/papers/paper51.pdf>
10. Peleg, D.: Distributed coordination algorithms for mobile robot swarms: new directions and challenges. In: Pal, A., Kshemkalyani, A.D., Kumar, R., Gupta, A. (eds.) *IWDC 2005*. LNCS, vol. 3741, pp. 1–12. Springer, Heidelberg (2005). [https://doi.org/10.1007/11603771\\_1](https://doi.org/10.1007/11603771_1)
11. Poudel, P., Sharma, G.: Universally optimal gathering under limited visibility. In: Spirakis, P., Tsigas, P. (eds.) *SSS 2017*. LNCS, vol. 10616, pp. 323–340. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-69084-1\\_23](https://doi.org/10.1007/978-3-319-69084-1_23)
12. Sharma, G., Alsaedi, R., Busch, C., Mukhopadhyay, S.: The complete visibility problem for fat robots with lights. In: Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN 2018, 4–7 January 2018, Varanasi, India, pp. 21:1–21:4 (2018). <https://doi.org/10.1145/3154273.3154319>
13. Sharma, G., Busch, C., Mukhopadhyay, S.: Bounds on mutual visibility algorithms. In: Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015, 10–12 August 2015, Kingston, Ontario, Canada (2015). <http://research.cs.queensu.ca/cccg2015/CCCG15-papers/43.pdf>

14. Sharma, G., Busch, C., Mukhopadhyay, S.: Mutual visibility with an optimal number of colors. In: Bose, P., Gasieniec, L.A., Römer, K., Wattenhofer, R. (eds.) ALGOSENSORS 2015. LNCS, vol. 9536, pp. 196–210. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-28472-9\\_15](https://doi.org/10.1007/978-3-319-28472-9_15)
15. Sharma, G., Vaidyanathan, R., Trahan, J.L., Busch, C., Rai, S.: Complete visibility for robots with lights in  $O(1)$  time. In: Bonakdarpour, B., Petit, F. (eds.) SSS 2016. LNCS, vol. 10083, pp. 327–345. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-49259-9\\_26](https://doi.org/10.1007/978-3-319-49259-9_26)
16. Sharma, G., Vaidyanathan, R., Trahan, J.L., Busch, C., Rai, S.:  $O(\log n)$ -time complete visibility for asynchronous robots with lights. In: 2017 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2017, 29 May–2 June 2017, Orlando, FL, USA, pp. 513–522 (2017). <https://doi.org/10.1109/IPDPS.2017.51>
17. Stefano, G.D., Navarra, A.: Gathering of oblivious robots on infinite grids with minimum traveled distance. *Inf. Comput.* **254**, 377–391 (2017). <https://doi.org/10.1016/j.ic.2016.09.004>