Visibility and Pattern Formation Problems in Robotics

RUSUL JABBAR ABBAS ALSAEDI

Supervisor: Dr. André van Renssen
Associate Supervisor: Professor Joachim Gudmundsson

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

22 January 2024

Abstract

In this thesis, we study four problems related to robotics and computational geometry.

In Chapter 3, we consider the Mutual Visibility problem for fat robots with lights: given a set of $n \geq 1$ unit disk robots in the Euclidean plane, the robots must reposition themselves to reach a configuration where they all see each other. We present an algorithm that requires only 2 colors and $O(n)$ rounds. The number of colors is optimal since at least two colors are required for point robots [28].

In Chapter 4 and Chapter 5, we consider the Pattern Formation problem for fat robots: given a set of $n \geq 1$ unit disk robots in the Euclidean plane, the robots must reposition themselves to form a given target pattern. In Chapter 4, we consider this problem for robots with lights and reduce the number of colors needed. In particular, we present an algorithm requiring 7 colors when scaling the target pattern is allowed and an 8-color algorithm if scaling is not allowed. Our algorithms run in $O(n)$ rounds plus the time needed for the robots to elect a leader. In Chapter 5, we consider robots with memory and present an algorithm that runs in $O(n) + O(q \log n)$ rounds, where $q > 0$ is related to Leader Election. We assume that the robots have a small $O(1)$-sized memory that they can use to store information, but that cannot be communicated to the other robots.

In Chapter 6, we consider the shortest paths of mutually visible robots: given a set of n point robots inside a simple polygon P, the task is to move the robots from their starting positions to their target positions along their shortest paths, while the mutual visibility of these robots is preserved. We

present an $O(mn)$ time algorithm, where m is the complexity of the polygon, when all the starting positions lie on a line segment S, all the target positions lie on a line segment T, and S and T do not intersect. We also argue that there is no polynomial-time algorithm, whose running time depends only on n and m, that uses a single strategy for the case where S and T intersect.

Statement of Attribution

This thesis consists of four papers. One has been published, while the other three are under submission.

Chapter 3: Rusul J. Alsaedi, Joachim Gudmundsson, André van Renssen, "The Mutual Visibility Problem for Fat Robots". In Proceedings of the 18th Algorithms and Data Structures Symposium (WADS 2023), pages 15 - 28, volume 14079 of Lecture Notes in Computer Science (LNCS) - Springer.

Chapter 4: Rusul J. Alsaedi, Joachim Gudmundsson, André van Renssen, "Pattern Formation for Fat Robots with Lights". The manuscript is under submission.

Chapter 5: Rusul J. Alsaedi, Joachim Gudmundsson, André van Renssen, "Pattern Formation for Fat Robots with Memory". The manuscript is under submission.

Chapter 6: Rusul J. Alsaedi, Joachim Gudmundsson, André van Renssen, "Shortest Paths of Mutually Visible Robots". The manuscript is under submission.

Rusul J. Alsaedi

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

André van Renssen

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Rusul J. Alsaedi

Acknowledgements

I would like to thank my god (Allah) for giving me this great chance to study my PhD at the University of Sydney.

I would like to thank my supervisors Dr. André van Renssen and Professor Joachim Gudmundsson for giving me their great advice always.

I would like to thank my parents (daddy and mummy), my husband (Qasim), my little daughter (Lamar), my sisters (Hiba and Jannat), and my brother (Mohammed-Mustafa) for their continuous support.

Finally, I would like to thank my best friends, and everyone who wishes me the best.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Computational Geometry

Computational geometry is a field of computer science that concerns the study of algorithms and geometric problems [27]. There are many vital applications of computational geometry, including robotics that consider the motion and visibility of robots, computer vision concerning 3D problems, and other important applications [46, 54].

The measure of efficiency of an algorithm in computational geometry is related to the computational complexity of that problem. Every algorithm has an input and performs a process to give the desired output. The efficiency of an algorithm is expressed as the number of rounds or running time in general needed to solve a problem. Examples of algorithms in computational geometry comprise convex hull algorithms, polygon triangulation algorithms, and others [27].

In this thesis, we focus on subjects of computational geometry related to robotics, including mutual visibility, pattern formation, and shortest paths in simple polygons. We consider different models, which we introduce in Chapter 2.

1.2 Background and Motivation of Robotics

Robots are machines that are provided with vision capabilities and/or locomotion capabilities depending on the types of robots. Robots can be divided into two types based on mobility: static and mobile robots. Static robots cannot move since they do not have locomotion capabilities, where mobile robots have locomotion capabilities in addition to vision capabilities, so they move from one place to another [53]. Mobile robots can be further classified into two types: robots with external control and robots without external control (autonomous mobile robots). In Chapters 3, 4, and 5, we consider autonomous mobile robots since this type of robot works in an unknown environment [35]. Autonomous mobile robots do not know any information about each other because they cannot communicate with each other. They work independently. They can only observe the environment using their vision capabilities and perform their actions.

Autonomous mobile robots have been used for inspection, surveillance, transportation, and cleaning, with applications in fields such as household maintenance, space exploration, delivery of goods and services, and wastewater treatment [12, 42]. The autonomous mobile robots have been widely used in solving many different problems in computational geometry, distributed computing, robotics community, and sensor network because of their flexibility when they are used in many different applications. In this thesis, we solve many fundamental problems of mutual visibility, pattern formation, and shortest paths in robotics.

1.3 Mutual Visibility

We consider a set of n unit disk robots in $\mathbb{R}^2$ and aim to position these robots in such a way that each pair of robots can see each other (see Figure 1.1 for

an example initial configuration where not all robots can see each other and an end configuration where they can). This problem is fundamental in that it is typically the first step in solving more complex problems. We consider the problem under the classical oblivious robots model [33], where robots are autonomous (no external control), anonymous (no unique identifiers), indistinguishable (no external markers), history-oblivious (no memory of activities done in the past), silent (no means of direct communication), and possibly disoriented (no agreement on their coordination systems). We consider this problem under the fully synchronous model, where in every *round* all robots are activated. All robots execute the same algorithm, following Look-Compute-Move (LCM) cycles [26] (i.e., when a robot becomes active, it uses its vision to get a snapshot of its surroundings (Look), computes a destination point based on the snapshot (Compute), and finally moves towards the computed destination (Move)).

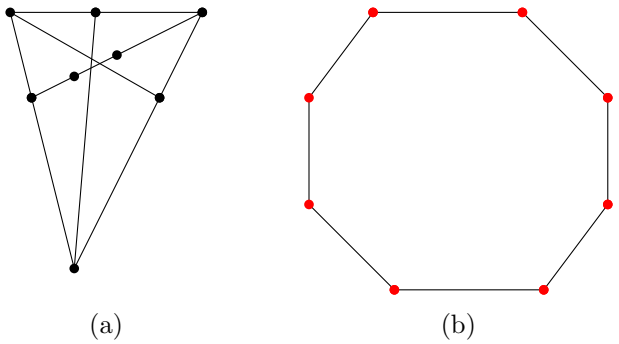

FIGURE 1.1: An example of an initial instance (a) and an end configuration (b).

This classical robot model has a long history and has many applications including coverage, exploration, intruder detection, data delivery, and symmetry breaking [22]. Unfortunately, most of the previous work considered the robots to be dimensionless point robots which do not occupy any space.

The classical model also makes the important assumption of unobstructed visibility, i.e., any three collinear robots are mutually visible to each other. This assumption, however, does not make sense for the unit disk robots

we consider. To remove this assumption, robots under obstructed visibility have been the subject of recent research [1, 13, 19, 23–25, 28, 29, 32, 40, 41, 48, 49, 51, 55]. Under obstructed visibility, robot r_i can see robot r_j if and only if there is no third robot on the straight line segment connecting r_i and r_j.

Additionally, a variation on this model received significant attention: the luminous robots model (or robots with lights model) [28, 40, 41, 44, 48, 49, 55]. In this model, robots are equipped with an externally visible light which can assume colors from a fixed set. The lights are persistent, i.e., the color of the light is not erased at the end of the LCM cycle. When the number of colors in the set is 1, this model corresponds to the classical oblivious robots model [28, 33].

Being the first step in a number of other problems, including the Gathering and Circle Formation problems [47], the Mutual Visibility problem received significant attention in this new robots with lights model. When robots are dimensionless points, the Mutual Visibility problem was solved in a series of papers [28, 40, 41, 48, 49, 55]. Unfortunately, the techniques developed for point robots do not apply directly to the unit disk robots, due to the lack of collision avoidance. For unit disk robots, much progress has been made in solving the Mutual Visibility problem [1, 13, 19, 24, 25, 29, 45, 47, 50], however these approaches either require additional assumptions such as chirality (the robots agree on the orientation of the axes, i.e., on the meaning of clockwise), knowledge of n, or without avoiding collisions. Additionally, some approaches require a large number of colors and not all approaches bound the number of rounds needed.

1.3.1 Related Work

Most of the existing work in the robots with lights model considers point robots [28, 40, 49, 55]. Di Luna *et al.* [28] solved the Mutual Visibility problem for those robots with obstructed visibility in the lights model, using 2 and 3 colors under semi-synchronous and asynchronous computation, respectively. Sharma *et al.* [49] provided a solution for point robots that requires only 2 colors, which is optimal since at least two colors are needed [28]. Unfortunately, the required number of rounds is not analyzed. Sharma *et al.* [52] also considered point robots in the robots with lights model. In the asynchronous setting, they provide an $O(1)$ time and $O(1)$ colors solution using their Beacon-Directed Curve Positioning technique to move the robots.

Mutual visibility has also been studied for fat robots. Agathangelou *et al.* [1] studied it in the fat robots model of Czyzowicz *et al.* [25], where robots are not equipped with lights. Their approach allows for collisions, assumes chirality, and the robots need to know n, making it unsuited for our setting. Sharma *et al.* [50] developed an algorithm that solves coordination problems for fat robots in $O(n)$ rounds in the classical oblivious model, assuming n is known to the robots.

Poudel *et al.* [45] studied the Mutual Visibility problem for fat robots on an infinite grid graph G and the robots have to reposition themselves on the vertices of the graph G. They provided two algorithms; the first one solves the Mutual Visibility problem in $O(\sqrt{n})$ time under a centralized scheduler. The second one solves the same problem in $\Theta(\sqrt{n})$ time under a distributed scheduler, but only for some special instances.

When considering both fat robots and the robots with lights model, the main result is by Sharma *et al.* [47]. Their solution uses 9 colors and solves the Mutual Visibility problem in $O(n)$ rounds.

1.3.2 Contributions

In Chapter 3, we consider $n \geq 1$ unit disk robots in the plane and study the problem of providing mutual visibility to them. We address this problem in the lights model. In particular, we present an algorithm that solves the problem in $O(n)$ rounds using only 2 colors while avoiding collisions. The number of colors is optimal since at least two colors are needed for point robots [28].

Our algorithm works under fully synchronous computation, where all robots are activated in each round and they perform their LCM cycles simultaneously in synchronized rounds. The moves of the robots are rigid, i.e., an adversary does not have the power to stop a moving robot before reaching its computed destination [33].

Our results improve on previous work in terms of the number of colors used [47] and by using fat robots and having a linear number of rounds [28, 49]. Additionally, we require no additional assumptions such as chirality or knowledge of n.

1.4 Pattern Formation

We consider a set of $n \geq 1$ unit disk robots in the two-dimensional Euclidean plane and aim to position these robots to form a given pattern. This problem is commonly referred to as the Pattern Formation problem. Similar to the Mutual Visibility problem, the robots work under the classical oblivious robots model [33] which means they are autonomous (no external control), anonymous (no unique identifiers), indistinguishable (no external markers), in some cases history-oblivious (no memory of activities done in the past), silent (no means of direct communication), and possibly disoriented (no agreement on their coordinate systems). All robots execute the

same algorithm and operate following Look-Compute-Move (LCM) cycles [26] (i.e., when a robot becomes active, it uses its vision to get a snapshot of its surroundings (Look), computes a destination point based on this snapshot (Compute), and finally moves towards the computed destination (Move)). In this problem, the robots are scheduled using the fully synchronous model, where there is a (discretized) notion of *rounds* and in every round all robots are activated.

A logical continuation of the Mutual Visibility Problem studied under obstructed visibility is the Pattern Formation problem: Starting from arbitrary distinct positions in the plane, determine a schedule to reposition the robots such that they form the given (target) pattern without collisions [15, 17, 56]. We say that two robots collide if at any time they share the same position. In previous work, the target pattern is allowed to be scaled, rotated, translated, and reflected.

We study this problem under two different models. The first is the robots with lights model described earlier. The other model, called the memory model, equips the robots with a small amount of memory. The aim of this model is to solve the problem while minimizing the size of the memory.

1.4.1 Related Work

There has been considerable work on the Pattern Formation problem for point robots [14, 16, 18, 36, 37, 39, 43] and some of these also considered the lights model while solving the problem [17, 56]. Other work in this area includes that by Cicerone *et al.* [21], who presented an algorithm to solve the Pattern Formation problem for point robots with chirality (the robots agree on the orientation of the axes, i.e., on the meaning of clockwise), and Flocchini *et al.* [34], who studied the problem for point robots in the

asynchronous setting, but they required that the robots agree on their environment and observe the positions of the other robots. While these works provided techniques to overcome various difficulties, they did not take the physical extents of the robots into account. Unfortunately, the techniques developed for point robots cannot be applied directly to solve the Pattern Formation for fat robots, due to the effect these extends have on collision avoidance.

The work most closely related to our result is due to Bose *et al.* [15]. They studied the Pattern Formation problem for fat robots in the robots with lights model and used 10 colors. Unfortunately, their solution assumes that all robots agree on an axis of the coordinate system. Our solutions remove this assumption and reduces the number of colors required. Our other solutions remove this assumption and use a constant memory while solving the problem without lights and without knowledge of the total number of robots.

Other related work for fat robots includes the work by Kundu *et al.* [39], who studied the Pattern Formation problem for fat robots with lights on an infinite grid with one axis agreement. They solved the problem using 9 colors. Unfortunately, they did not bound the running time of their algorithm.

To the best of our knowledge, there are no published results where the robots are allowed a small amount of memory only.

1.4.2 Contributions

In Chapter 4, we consider the robots with lights model. We first present two algorithms using at most 11 colors, which through careful analysis we improve to use only 7 colors when scaling of the pattern is allowed and 8 if this is not allowed. None of our algorithms require the pattern to be rotated, translated, or reflected. Our algorithms require $O(n) + O(q \log n)$ rounds.

Here $q > 0$ is related to leader election, which can be solved in $O(q \log n)$ rounds with probability at least $1 - n^{-q}$ [56].

Our algorithms work under the fully synchronous model and are collision-free. Interestingly, unlike previous work, our algorithms do not require any additional assumptions on the capabilities of the robots or any shared information or coordinate system. The moves of the robots are rigid, i.e., an adversary does not have the power to stop a moving robot before reaching its computed destination [33].

In Chapter 5, we consider robots with memory. We present an algorithm using a constant memory to solve the Pattern Formation when scaling of the pattern is allowed. Our algorithm does not require the pattern to be rotated, translated, or reflected. Our algorithm requires $O(n) + O(q \log n)$ rounds. Here $q > 0$ is related to leader election, which can be solved in $O(q \log n)$ rounds with probability at least $1 - n^{-q}$ [56].

Our algorithm works under the fully synchronous model and are collision-free. Our algorithm does not require any additional assumptions on the capabilities of the robots or any shared information or coordinate system except using a constant memory to let the robots store which phase they are in, and they can determine when a phase changes without communicating with each other.

1.5 Shortest Paths of Mutually Visible Robots

The problem of navigating robots in a polygonal domain has been studied extensively. In our problem, we assume that robots are independent point robots and execute the same centralized algorithm. The robots cannot communicate with each other, but they have full visibility of their environment. We study the following problem in this model:

PROBLEM 1.5.1. SHORTEST PATH MUTUAL VISIBILITY (SPMV) *Given a simple polygon P consisting of m vertices, a set $\mathcal{R}$ of n point robots, where each robot $r_i \in \mathcal{R}$ has a starting position s_i and a target position t_i, the problem is to move all robots from their starting positions to their target positions along their shortest paths while keeping all robots mutually visible.*

In Figure 1.2 we show two examples for the restricted case where the starting points lie on a segment S and the target points lie on a segment T.

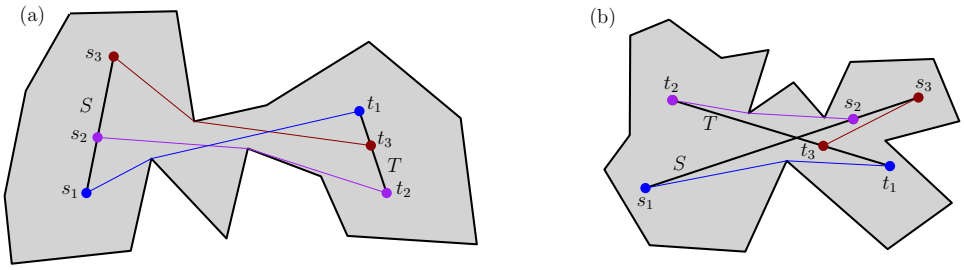

FIGURE 1.2: Two examples showing the SPMV problem instances considered in this thesis. (a) The case when S and T do not intersect, and (b) when S and T intersect.

Most of the existing work in the field focuses on the shortest path problem. Guibas *et al.* [38] presented algorithms to solve the shortest path problem between two points inside a simple polygon and showed the relationship between visibility and the shortest path problem. See also [2, 3, 5, 6, 8–11] for more recent work on shortest paths in simple polygons and polygonal domains.

The mutual visibility problem for robots in a polygonal domain has been widely studied; starting from any initial configuration, the aim is to move the robots into positions where every robot can see all other robots. The problem was solved under both obstructed visibility (i.e., three collinear robots are not mutually visible to each other) and unobstructed visibility (i.e., three collinear robots are mutually visible to each other) [4, 28, 40, 45, 49, 50, 55].

However, these works do not consider mutual visibility while the robots are moving. The most closely related work is by Fenwick *et al.* [30, 31]. They considered the SPMV problem for the special case where $n = 2$ (i.e., there are two moving robots the need to remain mutually visible while moving through the domain), whereas we consider the problem for n, i.e., an arbitrary number of robots.

1.5.1 Contributions

The results presented for this problem are twofold. In Section 6.2 we give an $O(nm)$ time algorithm for the SPMV problem when the start and target positions lie on two non-intersecting line segments S and T, respectively. In Section 6.3 we argue that when S and T intersect, any efficient algorithm must move the robots along a rotating line segment, however, we also show that there are instances where moving the robots along a rotating line segment does not give a running time that can be bounded using only n and m. Hence, this indicates that any polynomial-time algorithm for the SPMV problem cannot rely on a single strategy.

1.6 Thesis Outline

In this thesis, we proceed as follows. We present the models and preliminaries needed for Chapters 3, 4, and 5 in Chapter 2. In Chapter 3, we consider the Mutual Visibility problem for fat robots in the robots with lights model. We then shift our attention to the Pattern Formation problem for fat robots, first for robots with lights in Chapter 4, and then for robots with memory in Chapter 5. In Chapter 6, we consider the shortest paths problem for mutually visible robots in a simple polygon. Finally, we conclude in Chapter 7 with a consideration of future work.

CHAPTER 2

Models and Preliminaries

In this chapter, we present the model and the preliminaries used in Chapters 3, 4, and 5 of this thesis.

2.1 Mutual Visibility and Pattern Formation for Fat Robots with Lights

Consider a set of $n \geq 1$ anonymous robots $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ operating in the Euclidean plane. During the entire execution of the algorithm, we assume that n is *not* known to the robots. Each robot $r_i \in \mathcal{R}$ is a non-transparent disk with diameter 1, sometimes referred to as a fat robot. The center of the robot r_i is denoted by c_i and the position of c_i is also said to be the position of r_i. We denote by $\mathrm{dist}(r_i, r_j)$ the Euclidean distance between the two robots, i.e., the distance from c_i to c_j. To avoid collisions among robots, we have to ensure that $\mathrm{dist}(r_i, r_j) \geq 1$ between any two robots r_i and r_j at all times. Each robot r_i has its own coordinate system centered at itself, and it knows its position with respect to its coordinate system. Robots may not agree on the orientation of their coordinate systems, that is, there is no common notion of direction. Since all robots are of unit size, they implicitly agree on the unit of measure of other robots. The robots have a camera to take a snapshot, and the visibility of the camera is unlimited and has a 360° field of view provided that there are no obstacles (i.e., other robots) [1].

Following the fat robot model [1, 25], we assume that a robot r_i can see another robot r_j if there is at least one point on the bounding circle of r_j that is visible from r_i. Similarly, we say that a point p in the plane is visible to a robot r_i if there is a point p_i in the bounding circle of r_i such that the straight line segment $\overline{p_i p}$ does not intersect any other robot. We say that robot r_i fulfills the mutual visibility property if r_i can see all other robots in $\mathcal{R}$. Two robots r_i and r_j are said to *collide* at time t if the bounding circles of r_i and r_j share a common point at time t. For simplicity, we use r_i to denote both the robot r_i and the position of its center c_i.

Each robot r_i is equipped with an externally visible light that can assume any color from a fixed set $\mathcal{C}$ of colors. The set $\mathcal{C}$ is the same for all robots in $\mathcal{R}$. The color of the light of robot r at time t can be seen by all robots that are visible to r at time t.

A *configuration* $\mathbb{C}$ is a set of n tuples in $\mathcal{C} \times \mathbb{R}^2$ that define the colors and positions of the robots. Let $\mathbb{C}_t$ denote the configuration at time t. Let $\mathbb{C}_t(r_i)$ denote the configuration $\mathbb{C}_t$ for robot r_i, i.e., the set of tuples in $\mathcal{C} \times \mathbb{R}^2$ of the robots visible to r_i. A configuration $\mathbb{C}$ is *obstruction-free* if for all $r_i \in \mathcal{R}$, we have that $|\mathbb{C}(r_i)| = n$. In other words, it is obstruction free when all robots can see each other including themselves.

Let $\mathbb{H}_t$ denote the convex hull formed by $\mathbb{C}_t$. Let $\partial\mathbb{H}_t = \mathcal{V}_t \cup \mathcal{S}_t$ denote the set of robots on the boundary of $\mathbb{H}_t$, where $\mathcal{V}_t \subseteq \mathcal{R}$ is the set of corner robots lying on the corners of $\mathbb{H}_t$ and $\mathcal{S}_t \subseteq \mathcal{R}$ is the set of robots lying in the interior of the edges of $\mathbb{H}_t$. The robots in the set $\mathcal{V}_t$ are called *corner robots* and those in the set $\mathcal{S}_t$ are called *side robots*. The robots in the set $\mathcal{I}_t = \mathbb{H}_t \backslash \partial\mathbb{H}_t$ are called *interior robots*. Given a robot $r_i \in \mathcal{R}$, we denote by $\mathbb{H}_t(r_i)$ the convex hull of $\mathbb{C}_t(r_i)$. Note that $\mathbb{H}_t(r_i)$ can differ from $\mathbb{H}_t$ if r_i does not see all robots on the convex hull.

Given two points $a, b \in \mathbb{R}^2$, we denote by $|\overline{ab}|$ the length of the straight line segment $\overline{ab}$ connecting them. Given $a, b, d \in \mathbb{R}^2$, we use $\angle abd$ to denote the clockwise angle at point b between ab and bd.

At any time t, a robot $r_i \in \mathcal{R}$ is active or inactive. When active, r_i performs a sequence of *Look-Compute-Move* (LCM) operations:

- *Look:* a robot takes a snapshot of the positions of the robots visible to it in its own coordinate system;
- *Compute:* executes its algorithm using the snapshot. This returns a destination point $x \in \mathbb{R}^2$ and a color $c \in \mathcal{C}$; and
- *Move:* moves to the computed destination $x \in \mathbb{R}^2$ (if x is different than its current position) and sets its own light to color c.

We assume that the execution starts at time 0. Therefore, at time $t = 0$, the robots start in an arbitrary configuration $\mathbb{C}_0$ with $\mathrm{dist}(r_i, r_j) \geq 1$ for any two robots $r_i, r_j \in \mathbb{R}^2$, and the color of the light of each robot is set to *Off*.

Formally, the Mutual Visibility problem is defined as follows: Given any $\mathbb{C}_0$, in finite time, reach an obstruction-free configuration without having any collisions in the process. An algorithm is said to solve the Mutual Visibility problem if it always achieves an obstruction-free configuration from any arbitrary initial configuration. Each robot executes the same algorithm locally every time it is activated. We measure the quality of the algorithm both in terms of the number of colors and the number of rounds needed to solve the Mutual Visibility problem.

The Pattern Formation problem is defined as follows: Given any initial positions of the robots, the robots must reposition themselves to form a given target pattern without having any collisions in the process. The target pattern is given as input as a list of locations, one for each robot in the pattern and allowed to be scaled, rotated, translated, and reflected. An algorithm is

said to solve the Pattern Formation problem if it always achieves the target pattern from any initial configuration. The pattern should be constructed if this is indeed possible. We measure the quality of the algorithm using the number of distinct colors in the set C and the number of rounds needed to solve the Pattern Formation problem.

Finally, we need the following definitions to present the Mutual Visibility and Pattern Formation algorithms.

Let $e = \overline{v_1v_2}$ be a line segment connecting two corner robots v_1 and v_2 of $\mathbb{H}_t$. Following Di Luna *et al.* [40], we define the *safe zone* $S(e)$ as a portion of the plane outside $\mathbb{H}_t$ such that the corner robots v_1 and v_2 of $\mathbb{H}_t$ remain corner robots when a side robot is moved into this area: for all points $x \in S(e)$, we ensure that $\angle x v_1 v_2 \leq \frac{180^\circ - \angle v' v_1 v_2}{4}$ and $\angle v_1 v_2 x \leq \frac{180^\circ - \angle v_1 v_2 v_3}{4}$, where v', v_1, v_2, and v_3 are consecutive vertices of the convex hull of $\mathbb{H}_t$ (see Figure 2.1(a)). Side robots may not always be able to compute $S(e)$ exactly due to obstructions of visibility leading to different local views. However, if there is only one side robot on e, then it can compute $S(e)$ exactly. When there is more than one robot on e, $S'(e)$ is the safe region computed by a robot based on its local view. It is guaranteed that $S'(e) \subseteq S(e)$ (see Figure 2.1(b) for the safe zone of robot r_2).

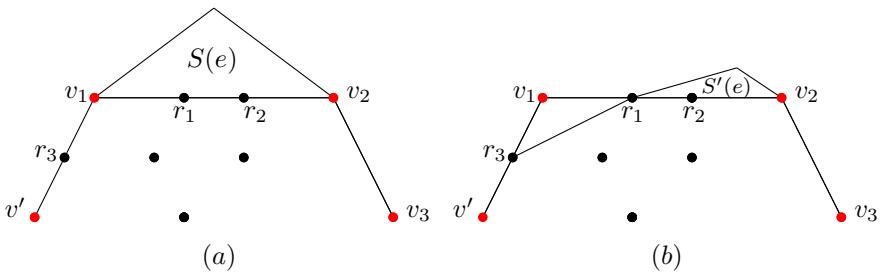

FIGURE 2.1: (a) The safe zone of $e = \overline{v_1v_2}$. (b) The safe zone of a side robot r_2 on e. Throughout the thesis the robots are shown as dots for simplicity.

2.2 Pattern Formation for Fat Robots with Memory

There are many similarities between this model and the lights model including the robot features, movements, visibility, and robots working according to Look-Compute-Move (LCM) cycles. These similarities are presented in Section 2.1, and thus we focus on the differences here.

We assume the robots have no lights. Instead, each robot is equipped with a small amount of memory. We assume that a single unit of memory suffices to store an integer of value at most n. In other words, a single unit of memory can store $O(\log n)$ bits. We also assume that the locations of the robots can be stored efficiently, using a constant number of units of memory, or a constant amount of memory for short. As all previous work assumes that the robots can perform computations based on the observed (locations of) robots, the only difference between this model and the existing model is that we allow a small number of things to be stored between rounds. Crucial is that the robots have no way to communicate with each other and thus they cannot combine what is stored in their respective memories.

The Pattern Formation problem in this setting is defined similarly as before: Given any initial positions of the robots, the robots must reposition themselves to form a given target pattern without having any collisions in the process. The target pattern is given as input as a list of locations, one for each robot in the pattern and allowed to be scaled, rotated, translated, and reflected. An algorithm is said to solve the Pattern Formation problem if it always achieves the target pattern from any initial configuration. The pattern should be constructed if this is indeed possible. We measure the quality of the algorithm using the amount of memory required to execute it and the number of rounds needed to solve the Pattern Formation problem.

CHAPTER 3

The Mutual Visibility for Fat Robots with Lights

In this chapter, we consider the Mutual Visibility problem in the robots with lights model.

3.1 The Mutual Visibility Algorithm

In this section, we present an algorithm that solves the Mutual Visibility problem for $n \geq 1$ unit disk robots under rigid movement in the robots with lights model. Our algorithm assumes the fully synchronous setting of robots. The algorithm needs two colors: $\mathcal{C} = \{Off, Red\}$. A red robot represents a corner robot. A robot whose light is off represents any other robot. See Figure 3.1 for an example. Initially, the lights of all robots are off.

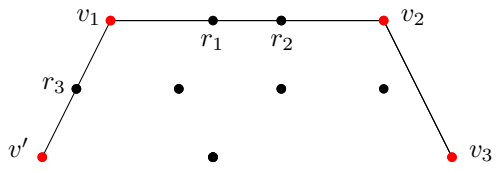

FIGURE 3.1: The different colors of the robots: corner robots (red), side robots (off), and interior robots (off).

It has been shown that positioning the robots in the corners (i.e., vertices) of an n-vertex convex hull provides a solution to the Mutual Visibility problem [28, 32, 40, 48, 49, 55]. Hence, our algorithm also ensures that the robots eventually position themselves in this way with the well-separation, so the convex hull provides full visibility if the robots are not all touching.

Conceptually, our general strategy consists of two phases, though the robots themselves do not explicitly discern between them. In the *Side Depletion* phase, some side robots move to become corner robots, ensuring that there are only corner and interior robots left. In the *Interior Depletion* phase, interior robots move and become corner robots of the convex hull. The move-algorithm checks if the robot's path shares any point with any other robots, ensuring that no collision occurs. Throughout both phases, corner robots slowly move to expand the convex hull to ensure that the interior robots can move through the edges of the convex hull when needed. This movement is deterministic and is taken into account when moving robots to become corners of the expanding hull.

Detailed pseudocode of the algorithm and its subroutines can be found in Section 3.3.

3.1.1 The Side Depletion Phase

The first phase of our algorithm is the Side Depletion (SD) phase. During this phase, every robot first determines if it is a corner, side, or interior robot and sets their light accordingly. Note that robots can make this distinction themselves, by checking what angle between consecutive robots it sees: if some angle is larger than 180° it is a corner robot, if the angle is exactly 180° it is a side robot, and otherwise it is an interior robot.

In every round, all corner robots move a distance of 1 along the angle bisector determined by its neighbors in the direction that does not intersect the interior of the convex polygon. In other words, in each round, the corner robots move to expand the size of the convex hull. We note that since all corner robots move this way, they all stay corner robots throughout this process.

Side robots that see at least one corner robot move to become new corner robots of $\mathbb{H}$ (using the safe zone described earlier and taking the above movement of corner robots into account) and change their light to red. Side robots that do not see a corner robot on their convex hull edge do not move and will become interior robots in the next round (due to the change to the convex hull), while keeping their light off.

More precisely, a side robot r on edge $e = \overline{v_1v_2}$ of $\mathbb{H}_k$ moves as follows: If at least one of its neighbors on $\overline{v_1v_2}$ is a corner robot, r moves to a point in the safe zone $S(e)$. There are at most two such robots r_1 and r_2 on each edge $\overline{v_1v_2}$ (see Figures 3.2 and 3.3). Sharma *et al.* [47] showed that these can move simultaneously to the safe zone outside the hull. Both r_1 and r_2 become new corners of $\mathbb{H}$ and change their lights to red (see Figure 3.3).

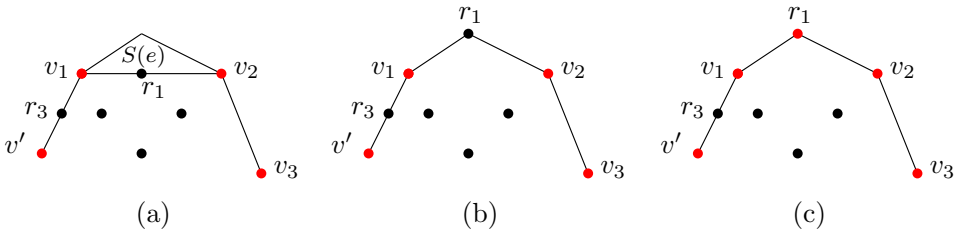

FIGURE 3.2: One side robot r_1 on an edge $e = \overline{v_1v_2}$ moves to become a corner robot.

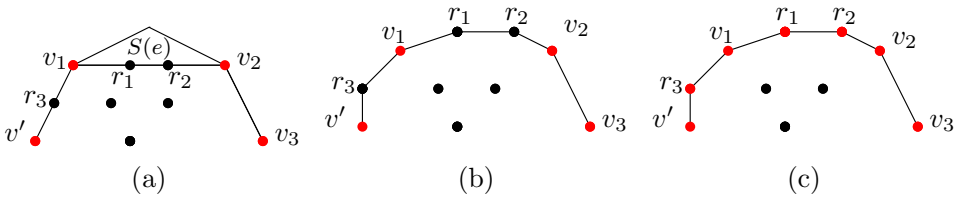

FIGURE 3.3: Two side robots r_1 and r_2 on an edge $e = \overline{v_1v_2}$ move to become corner robots.

If both of its neighbors on $\overline{v_1v_2}$ are not corners (see Figure 3.4), robot r does not move and stay in its place, and it will become an interior robot in the next round.

We only execute this phase once, at the start of our algorithm and only move each robot once.

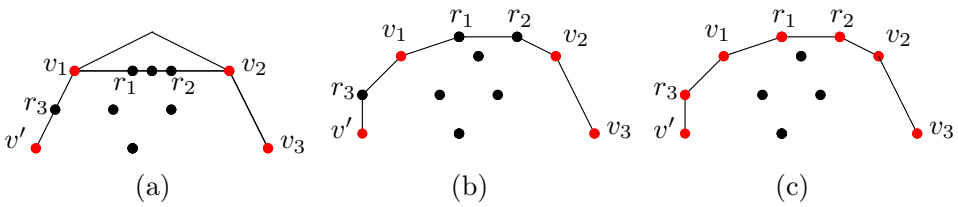

FIGURE 3.4: When there are more than two side robots on an edge of the convex hull, only two side robots on the edge move to become corner robots. These are the clockwise and the counterclockwise extreme side robots. In this case, robots r_1 and r_2 move to become corner robots.

3.1.2 The Interior Depletion Phase

Once the SD phase finishes, the Interior Depletion (ID) phase starts. During this phase the robots in the interior of the hull move such that they become new vertices of the hull.

In every round, all corner robots move as in the SD phase, expanding the convex hull. This ensures that the length of all edges increases and thus interior robots can move through these edges to in turn become corner robots themselves. All movement described in the remainder of this chapter takes the (predictably) expanding convex hull into account.

Next we describe how an interior robot moves. Given a robot r_i, we define its *eligible* edges as those edges of length at least 3 for which no other robot is closer to the edge.[1] The interior robots start by determining their eligible edges (see Figure 3.5(a)). In the figure, robot r_i finds edges $\overline{v_1v_2}$ and $\overline{v_2v_3}$ eligible, whereas r_j finds $\overline{v_2v_3}$, and r_l finds $\overline{v_3v_4}$ eligible. However, the robots between r_i, r_j find no edge eligible. Let Q denote the set of edges that are eligible to an interior robot r_i. Every interior robot that has an eligible edge moves perpendicular to some eligible edge towards this edge to become a corner robot by moving through this eligible edge (see

[1]The length of 3 is used to ensure that two robots can move through the same edge without colliding with each other (requiring a length of 2) while ensuring that they also do not collide with the corner robots on the edge (adding a length of 0.5 per corner robot).

Figure 3.5(b)), checking its path to avoid collisions with other robots (see Figure 3.5(c)). If the path is clear, it moves outside the hull into its safe zone to become a new corner as described earlier (see Figure 3.5(d)) and changes its color to red (see Figure 3.5(e)).

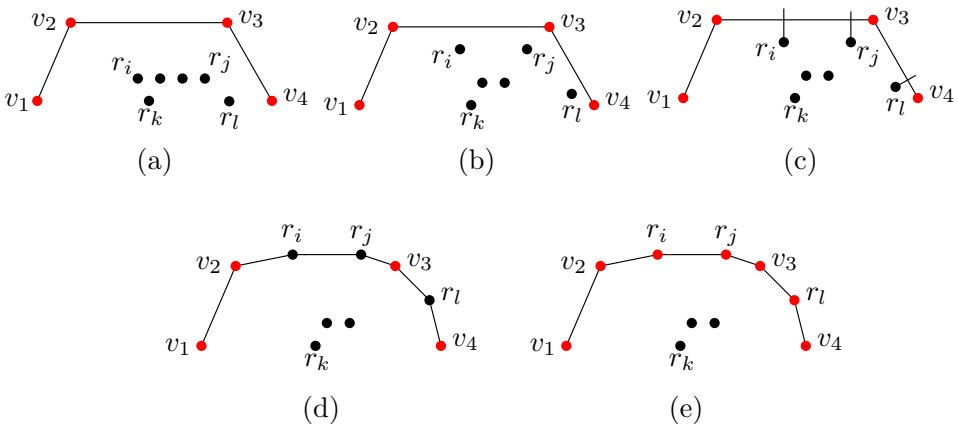

FIGURE 3.5: (a) The eligible edge computation. The robot r_i finds edges $\overline{v_1v_2}$ and $\overline{v_2v_3}$ eligible, whereas r_j finds $\overline{v_2v_3}$, and r_l finds $\overline{v_3v_4}$ eligible. The robots between r_i, r_j find no eligible edges. (b) The intermediate movements of the interior robots r_i, r_j and r_l. (c) The interior robots r_i, r_j and r_l check their paths to avoid collisions with other robots while moving outside the hull to become corners. (d) After the interior robots r_i, r_j and r_l move, they become corners. (e) Robots r_i, r_j and r_l change their lights to red.

When both phases are finished, the Mutual Visibility problem is solved, and all the robots are in the corners of the convex hull with red lights.

3.1.3 Special Cases

There are two special cases to consider: $n = 1$, and the case where the initial configuration is a line. The case $n = 1$ can be easily recognized by the only robot, since it does not see any other robot and thus it can terminate.

If in the initial configuration all robots lie on a single line, we differentiate between the robots that see only one other robot and the robots that see two other robots. If a robot r_i sees only one other robot r_j, when r_i is activated

for the very first time it sets its light to red and moves orthogonal to the line $\overline{r_i r_j}$ for some arbitrary positive distance. When r_i is activated in future rounds and $\mathbb{H}_k(r_i)$ is still a line segment, it can conclude that there are only two robots and it does nothing until it sees r_j set its light to red. Once r_j sets its light to red, r_i terminates.

If a robot r_i sees two other robots r_j and r_l, robot r_i will be able to tell if $\mathbb{H}_k(r_i)$ is a line segment as follows. Robot r_i will move orthogonal to line $\overline{r_j r_l}$ and set its light to red if and only if it sees that the lights of r_j and r_l are set to red, as this indicates that both other robots see only a single other robot, i.e., $n = 3$. Otherwise, moving the two extremal robots of the initial configuration as described above ensures that the configuration is no longer a line segment, allowing the SD and ID phase to solve the problem.

As these special cases add only a constant number of rounds to the running time and do not influence the number of colors, we focus on the general case in the remainder of this chapter.

3.2 Analysis

We proceed to prove that our algorithm solves the Mutual Visibility problem in a linear number of rounds, using only two colors and while avoiding collisions between the robots.

We start with some properties of the Side Depletion phase.

LEMMA 3.2.1. *Given a configuration $\mathbb{C}_k$ and an edge $e = \overline{v_1 v_2}$ of $\mathbb{H}_k$, if a robot $r_i \in e$ moves away from e, it will move into the safe zone $S(e)$.*

PROOF. We prove this lemma using proof techniques similar to those of Lemma 3 in [40]. Let v_1 and v_2 be the two corner robots that define e. If there is a single robot $r \in e$, r can compute $S(e)$ exactly and then move

into $S(e)$, proving the lemma. Consider the situation when there are at least two side robots on e. Let r_1 and r_2 be the two robots on e that are neighbors of v_1 and v_2, respectively. In the fully synchronous setting, both r_1 and r_2 move from e in the same round. Consider only the move of r_2 to $S(e)$ (the move of r_1 follows similarly).

Robot r_2 orders the robots it can see in clockwise order and let this ordering be $\{v', v, r_2, v_2, v_3\}$, where v' is the first robot non-collinear to r_2 in the clockwise direction with its light set to red, v is the robot that is collinear with r_2 in the clockwise direction, and v_2 is the collinear robot in the counterclockwise direction with its light set to red, and v_3 is the first non-collinear robot in the counterclockwise direction with its light set to red. Following the rules of Algorithm 5, r_2 computes $\alpha = 180^\circ - \angle v'vv_2$, $\beta = 180^\circ - \angle vv_2v_3$, and $\delta = \min\{\alpha/4, \beta/4\}$. We note that since we calculate α by subtracting $\angle v'vv_2$ from 180°, α is in fact a lower bound on the actual angle used to define $S(e)$. Therefore, any point x in the safe zone computed by r_2 is inside the safe zone of e, and thus, r_2 will move inside $S(e)$. The same holds for r_1. The other robots on e between r_1 and r_2 do not move. □

LEMMA 3.2.2. *Let r_i and r_j be the robots that are neighbors of endpoints v_1 and v_2 on edge e, respectively. When there are $p \leq 2$ side robots on e, r_i and r_j become corners and change their light to red in the next round. When there are $p > 2$ side robots on e, r_i and r_j become corners and change their light to red after which all the robots on e between r_i and r_j lie inside the convex hull and become interior robots.*

PROOF. If $p \leq 2$, r_i and r_j see only the corners and each other on e. Hence, both robots move and by Lemma 3.2.1 they move into $S(e)$. By moving r_i and r_j to $S(e)$, they become corner robots, as was also argued by Di Luna *et al.* [40].

When $p > 2$, a similar argument shows that both r_i and r_j become corners of $\mathbb{H}_k$ after they move once and change their light to red in the next round. The other side robots on e remain in their places and since $\mathbb{C}_k$ is not a line, moving r_i and r_j creates a hull that has more than three sides, implying that the robots between r_i and r_j lie strictly inside this hull. Thus, the other side robots become interior robots. □

LEMMA 3.2.3. *Given a configuration $\mathbb{C}_0$ with $q \geq 1$ side robots. After one round, all side robots become either corner robots or interior robots.*

PROOF. The movements of side robots on different edges of $\mathbb{H}$ do not interfere with each other. Therefore, we prove this lemma for a single edge e and the same argument applies for the side robots on other edges of $\mathbb{H}$.

When there is only one robot r on e, then r can compute $S(e)$ exactly and move to a point $x \in S(e)$ as soon as it is activated. When there are two or more robots on e, two side robots (the extreme ones on this edge) become corners in one round by Lemma 3.2.2. This causes the other robots on e to become interior robots.

Since the robots on different edges do not influence each other and the moves on any edge end in one round, this phase ends in one round. □

Now that there are no more side robots, we argue that the interior robots also eventually become corners. We first show that the interior robots can determine whether the SD phase has finished.

LEMMA 3.2.4. *Given a configuration $\mathbb{C}_k$ and an edge $e = \overline{v_1 v_2}$ of $\mathbb{H}_k$, no robot in the interior of $\mathbb{H}_k$ moves to $S(e)$ if there is a side robot on e.*

PROOF. If there are side robots in $\mathbb{H}_k$, it is easy to see that every corner robot of $\mathbb{H}_k$ on an edge that contains side robots sees at least one side robot. Similarly, when there are side robots, interior robots can easily infer that

the SD phase is not finished, and hence they do not move to their respective $S(e)$. □

Next, we argue in a series of lemmas that every interior robot will eventually become a corner robot and it does not collide with any robots in doing so. Let $\mathbb{C}_{SD}$ denote the configuration of robots after the SD phase is finished and let $\mathbb{H}_{SD}$ be the convex hull created by $\mathbb{C}_{SD}$.

LEMMA 3.2.5. *Let I_k be the set of interior robots in round k. In each round $k \in N^+$ until $I_k = \emptyset$, if there is an edge of length at least 3, there is at least one robot in I_k for which the set Q of line segments is not empty.*

PROOF. We note that every edge of the convex hull of the corner robots $\mathbb{H}_k$ is the closest to some interior robot(s). In particular, this holds for any edge of length at least 3. We note that this set of interior robots forms a line, as they all have the same closest distance to the edge. Out of these robots, by definition, the left and right extreme ones have the edge in their Q. □

LEMMA 3.2.6. *Let $\mathbb{C}_{SD}$ be the configuration after the SD phase ended and let $e = \overline{v_1 v_2}$ be the edge of $\mathbb{H}_{SD}$ closest to some interior robot r_i. If the robot $r_i \in I_k$ moves, it moves inside the safe zone $S(e)$.*

PROOF. We prove this lemma using the proof technique similar to the proof of Lemma 3 in [40]. When there is a single closest interior robot $r \in I_k$, r can compute the region $S(e)$ and move to it, proving the lemma. Consider now the situation when there are at least two closest interior robots. Let r_1 and r_2 be two of these robots. Since we work in the fully synchronous setting, both r_1 and r_2 move at the same time. Consider only the move of r_2 (the move of r_1 follows similarly). Robot r_2 orders the corner robots that are visible to it according to its local notion of clockwise direction and let this ordering be $\{v', v_1, v_2, v_3\}$, where v' is a corner robot preceding v_1 in the clockwise direction and v_3 is the corner robot following

v_2 in clockwise direction. Following the rules of our algorithm, r_2 computes $\alpha = 180° - \angle v'v_1v_2$, $\beta = 180° - \angle v_1v_2v_3$, and $\delta = \min\{\alpha/4, \beta/4\}$ (see Figure 2.1). We note that since we calculate α by subtracting $\angle v'v_1v_2$ from $180°$, α is in fact a lower bound on the actual angle that any robot in I_k at the same distance from edge e will compute. Let x' be the nearest to e in the safe zone outside the convex hull such that either $\angle x'v_1v_2 = \delta$ or $\angle x'v_2v_3 = \delta$ and define $x = x' + \overline{r_2m}$, where m is the intersection point of e. The same holds for r_1. Our algorithm guarantees that in every round at most two closest interior robots to an edge can move through this edge. □

LEMMA 3.2.7. *Given any initial configuration $\mathbb{C}_0$, no collisions of robots occur until $I_k = \emptyset$.*

PROOF. This lemma is proved by considering Algorithm 2. An interior robot r_i with light off does not collide with any other interior robot since the move of r_i is perpendicular to the closest edge $\overline{r_1r_2}$ and there is sufficient space on the edge for the robot to move through it. The robots moving through different edges of $\mathbb{H}_k$ do not collide since those robots are the closest robots to those edges because the $S(e)$ of different edges are disjoint. □

LEMMA 3.2.8. *There exists an integer $k \in N^+$ such that the robots in I_k closest to their eligible edge are able to move outside the convex hull $\mathbb{H}_k$ and become corner robots with their light set to red.*

PROOF. By Lemma 3.2.7, the robot r_i does not collide with other interior robots while it tries to move toward the edge $\overline{v_1v_2}$ of $\mathbb{H}_k$. Since there is no side robot after the first round by Lemma 3.2.3, those cannot block r_i's movement. By Lemma 3.2.7, there is no collision for robot r_i while it passes $e = \overline{v_1v_2}$ where v_1 and v_2 are the endpoints of the edge that r_i passes through to its computed point in $S(e)$. Since the movements are rigid, r_i

reaches its computed point in the safe zone once it moves and changes its color to red. □

LEMMA 3.2.9. *Given any initial configuration $\mathbb{C}_0$, there exists an integer $k \in N^+$ such that $I_k = \emptyset$ in $\mathbb{C}_k$ and the corner robots do not move in any round $k' > k$.*

PROOF. When $I_k \neq \emptyset$ each corner robot sees at least one robot with light off. Therefore, combining the results of Lemmas 3.2.5, 3.2.6, 3.2.7, and 3.2.8 with this observation, we have that, given any $\mathbb{C}_0$, there is some round $k \in N^+$ such that $I_k = \emptyset$.

Corner robots do not move after $I_k = \emptyset$, since they do not see robots with light off, thus terminating. □

THEOREM 3.2.10. *Given any initial configuration $\mathbb{C}_0$, there is some round $k \in N^+$ such that all robots lie on $\mathbb{H}_k$ and have their lights set to red.*

PROOF. Lemma 3.2.9 shows that there exists a round k such that there are no interior robots left. Interior robots that moved to become corner robots changed their lights to red as soon as they reached their corner positions. Furthermore, the interior robots move to the safe zone where they by definition become corners. Since Lemma 3.2.9 guarantees that there are no collisions, the robots occupy different positions of $\mathbb{H}_k$ and all their lights will be red. □

Next, we argue that the robots can determine when there are no interior robots left.

LEMMA 3.2.11. *If there exists a robot with light off, there is at least one interior robot that is visible to any corner robot r_i.*

PROOF. If there is at least one interior robot, every corner robot can see some interior robot (for example the one closest to it). By definition, every interior robot has its light off, proving the lemma. □

LEMMA 3.2.12. *Given a robot $r_i \in \mathcal{R}$ with its light set to red and a round $k \in N^+$, if all robots in $\mathbb{C}_k(r_i)$ have their light set to red, and no robot is in the interior of $\mathbb{H}_k(r_i)$, then $\mathbb{C}_k$ does not contain interior robots.*

PROOF. When all the robots in $\mathbb{C}_k(r_i)$ have their light set to red, this means that there is no robot with light off. Since any interior robot would have color off and by Lemma 3.2.11 at least one of these robots would be visible to r_i, this proves the lemma. □

We are now ready to prove that the Mutual Visibility problem is solvable using only two colors. Let $\mathbb{C}_{ID}$ denote the configuration of robots after the ID phase is finished and let $\mathbb{H}_{ID}$ be the convex hull created by $\mathbb{C}_{ID}$.

THEOREM 3.2.13. *The Mutual Visibility problem is solvable without collisions for unit disk robots in the fully synchronous setting using two colors in the robots with lights model.*

PROOF. We have from Lemma 3.2.3 that from any initial non-collinear $\mathbb{C}_0$, we reach a configuration $\mathbb{C}_{SD}$ without side robots after one round, some becoming corner robots and some becoming interior robots. Once the SD phase is over, Theorem 3.2.10 shows that the ID phase moves all interior robots to become corner robots. We have from Lemma 3.2.12 that robots can locally detect whether the ID phase is over and configuration $\mathbb{C}_{ID}$ is reached. By Lemma 3.2.7, no collisions occur in the SD and ID phases.

Therefore, starting from any non-collinear configuration $\mathbb{C}_0$, all robots eventually become corners of the convex hull, solving the Mutual Visibility problem without collisions.

It remains to show that starting from any initial collinear configuration $\mathbb{C}_0$ the robots correctly evolve into some non-collinear configuration from which we can apply the above analysis. If $n \leq 3$, this can be shown through a simple case analysis: For $n = 1$, when the only robot becomes active, it sees no other robot, changes its color to red and immediately terminates. For $n = 2$, robot r_i changes its color to red when it becomes active for the first time and moves orthogonal to line $\overline{r_i r_j}$ that connects it to the only other robot r_j it sees in $\mathbb{C}(r_i)$. When r_i later realizes that $|\mathbb{C}(r_i)|$ is still 2 and $r_j.light = red$, it simply terminates. For $n = 3$, when r_i realizes that both of its neighbors in $\mathbb{C}(r_i)$ have light set to red and are collinear with it, it moves orthogonal to that line and sets its light to red. The next time it becomes active, it finds itself at one of the corners and simply terminates as it sees all the other robots in the corners of the hull with light set to red.

For $n \geq 4$, let a and b be the two robots that occupy the corners of the line segment $\mathbb{H}_0$ (i.e. the endpoint robots of $\mathbb{H}_0$). Nothing happens until a or b is activated, setting its light to red, and moving orthogonal to $\mathbb{H}_0$. After a or b moves, when another robot becomes active, it realizes that the configuration is not a line anymore and enters the normal execution of our algorithm. It is easy to see that after the line segment $\mathbb{H}_0$ evolves into a polygonal shape, it never reverts to being a line.

Finally, since our algorithm uses only two colors, the theorem follows. □

It remains to analyze the number of rounds needed by our algorithm.

LEMMA 3.2.14. *After $O(n)$ rounds, the convex hull has grown enough in size to allow all n robots to become corners.*

PROOF. Since in every round all corner robots move a distance of 1 along the bisector of their exterior angle, the length of the convex hull grows by at least 1 in every round. Note that when a robot becomes a corner,

it moves outside the current convex hull and thus, by triangle inequality, extends the hull that way as well.

Hence, after at most $4n$ rounds the convex hull is long enough to ensure that there is space for all interior robots: there are at most n edges of the convex hull and for each of them to *not* be long enough, their total length is strictly less than $3n$. Hence, by expanding the convex hull by a total of $4n$, we ensure that there is enough space for each of the less than n interior robots of diameter 1. Expanding the convex hull a total of $4n$ takes $O(n)$ rounds, completing the proof. □

We note that for the above lemma the corner robots do not need to know n, as they can simply keep moving until the algorithm finishes.

LEMMA 3.2.15. *The Interior Depletion phase of the mutual visibility algorithm finishes in $O(n)$ rounds.*

PROOF. When an interior robot can move outside the convex hull to become a corner robot, it needs at most a constant rounds to do so. During those rounds the robot becomes active, checks its path while moving to the safe zone to become a corner robot, and changes its light to red. There are fewer than n interior robots and by Lemma 3.2.5 at least one robot can move when there is an edge of length at least 3. By Lemma 3.2.14 in $O(n)$ rounds there are sufficient long edges to allow the less than n interior robots to move through them. Therefore, the Interior Depletion phase of the mutual visibility algorithm finishes in $O(n)$ rounds. □

We now have the following theorem bounding the running time of our algorithm using Lemmas 3.2.3 and 3.2.15 and Theorem 3.2.13.

THEOREM 3.2.16. *Our algorithm solves the Mutual Visibility problem for unit disk robots in $O(n)$ rounds without collisions in the fully synchronous setting using two colors.*

3.3 Pseudocode

Algorithm 1: Mutual Visibility algorithm

// Look-Compute-Move cycle for robot r_i of unit disk size
$\mathbb{C}_k(r_i) \leftarrow$ configuration $\mathbb{C}_k$ for robot r_i (including r_i);
$\mathbb{H}_k(r_i) \leftarrow$ convex hull of the positions of the robots in $\mathbb{C}_k(r_i)$;
if $|\mathbb{C}_k(r_i)| = 1$ **then** Terminate;
else if $\mathbb{H}_k(r_i)$ *is a line segment* **then**
    if $|\mathbb{C}_k(r_i)| = 2$ **then**
        Let $r_j \in \mathbb{C}_k(r_i)$;
        if $r_i.light =$ *Off* **then**
            Move orthogonal to line $\overleftrightarrow{r_i r_j}$ by any non-zero distance;
            $r_i.light \leftarrow$ *Red*;
        else if $r_j.light =$ *Red* **then**
            $r_i.light \leftarrow$ *Red*;
            Terminate;
    else if $|\mathbb{C}_k(r_i)| = 3$ **then**
        Let $r_j, r_l \in \mathbb{C}_k(r_i)$;
        if $r_i.light =$ *Off* $\wedge\ r_j.light =$ *Red* $\wedge\ r_l.light =$ *Red* **then**
            Move orthogonal to line $\overleftrightarrow{r_j r_l}$ by any non-zero distance;
            $r_i.light \leftarrow$ *Red*;
else if r_i *is a corner robot of* $\mathbb{H}_k(r_i)$ **then** $Corner(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$;
else if r_i *is an interior robot of* $\mathbb{H}_k(r_i)$ **then** $Interior(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$;
else if r_i *is a side robot of* $\mathbb{H}_k(r_i)$ **then** $Side(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$;

Algorithm 2: $Interior(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$

if $r_i.light = Off$ **then**
  Order the robots in $\mathbb{H}_k(r_i)$ starting from any arbitrary robot v_1 in the clockwise order so that $\mathcal{T} = \{v_1, \ldots, v_{last}, v_1\}$, where v_1 is the first robot and v_{last} is the last robot;
  Let c, d be any pair of two consecutive robots in $\mathcal{T}$ with $c.light = Red$ and $d.light = Red$;
  Let HP_{cd} be the half-plane defined by line parallel to $\overleftrightarrow{cd}$ that passes through r_i such that c, d are in HP_{cd};
  $Q \leftarrow$ set of line segments $\overline{cd}$ such that:
    (a) the triangle r_i, c, d does not contain (neither inside nor on its edges) any other robot of $\mathbb{C}_k(r_i)$, and
    (b) there is no robot in edge $\overline{cd}$, and
    (c) there is no robot in $\mathbb{C}_k(r_i) \backslash \mathbb{H}_k(r_i)$ closer to edge $\overline{cd}$ than r_i, and
    (d) there are no two robots with equal distance to $\overline{cd}$ appearing counterclockwise and clockwise of r_i with respect to the local coordinate system of r_i, and
    (e) the length of $\overline{cd}$ is at least 3;
  if Q *is not empty* **then**
    $\overline{u_1u_2} \leftarrow$ the line segment in Q between two robots u_1, u_2 that is closest to r_i;
    if there is no other robot with light *Off* that is at equal distance to $\overline{u_1u_2}$ **then**
      $m \leftarrow$ midpoint of $\overline{u_1u_2}$;
      $L \leftarrow$ line perpendicular to $\overline{u_1u_2}$ passing through its midpoint m;
      Order the robots in the counterclockwise order of r_i (with respect to the local coordinate system of r_i) such that the order is $\mathcal{T}_i = \{v_1, v_2, v_3, v_4\}$;
      Compute angles $\alpha = 180^\circ - \angle v_4v_3v_2$ and $\beta = 180^\circ - \angle v_1v_2v_3$, and set $\delta = \min\{\alpha/4, \beta/4\}$;
      Compute a point x' such that $\angle x'v_3v_2 = \delta$ and a point x'' such that $\angle x''v_2v_3 = \delta$;
      $L(r_im) \leftarrow$ line segment connecting r_i and m;
      $x = x' + L(r_im)$, where x' is the nearest point to e in the safe zone outside the convex hull;
      $Move(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i), u_1, u_2, x)$;
    else if there exists a robot in the clockwise direction of r_i (with respect to the local coordinate system of r_i) with light *Off* that is at equal distance to $\overline{u_1u_2}$ **then**
      $m \leftarrow$ point in $\overline{u_1u_2}$ at $\frac{\text{length}(\overline{u_1u_2})}{3}$ from endpoint u_1;
      $L \leftarrow$ line perpendicular to $\overline{u_1u_2}$ passing through the point m;
      Order the robots in the counterclockwise order of r_i (with respect to the local coordinate system of r_i) such that the order is $\mathcal{T}_i = \{v_1, v_2, v_3, v_4\}$;
      Compute angles $\alpha = 180^\circ - \angle v_4v_3v_2$ and $\beta = 180^\circ - \angle v_1v_2v_3$, and set $\delta = \min\{\alpha/4, \beta/4\}$;
      Compute a point x' such that $\angle x'v_3v_2 = \delta$ and a point x'' such that $\angle x''v_2v_3 = \delta$;
      $L(r_im) \leftarrow$ line segment connecting r_i and m;
      $x = x' + L(r_im)$, where x' is the nearest point to e in the safe zone outside the convex hull;
      $Move(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i), u_1, u_2, x)$;
    else if there exists a robot in the counterclockwise direction of r_i (with respect to the local coordinate system of r_i) with light *Off* that is at equal distance to $\overline{u_1u_2}$ **then**
      $m \leftarrow$ point in $\overline{u_1u_2}$ at $\frac{\text{length}(\overline{u_1u_2})}{3}$ from endpoint u_2;
      $L \leftarrow$ line perpendicular to $\overline{u_1u_2}$ passing through the point m;
      Order the robots in the counterclockwise order of r_i (with respect to the local coordinate system of r_i) such that the order is $\mathcal{T}_i = \{v_1, v_2, v_3, v_4\}$;
      Compute angles $\alpha = 180^\circ - \angle v_4v_3v_2$ and $\beta = 180^\circ - \angle v_1v_2v_3$, and set $\delta = \min\{\alpha/4, \beta/4\}$;
      Compute a point x' such that $\angle x'v_3v_2 = \delta$ and a point x'' such that $\angle x''v_2v_3 = \delta$;
      $L(r_im) \leftarrow$ line segment connecting r_i and m;
      $x = x' + L(r_im)$, where x' is the nearest point to e in the safe zone outside the convex hull;
      $Move(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i), u_1, u_2, x)$;

Algorithm 3: $Corner(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$

Move r_i distance 1 along the angle bisector of its neighbors on $\mathbb{C}_k(r_i)$ in the direction that does not intersect the interior of $\mathbb{C}_k(r_i)$;
if $r_i.light = Off$ **then** $r_i.light \leftarrow Red$;
else if $\forall r \in \mathbb{C}_k(r_i)$, $r.light = Red$ **then** Terminate;

Algorithm 4: $Move(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i), u_1, u_2, x)$

$L_{r_i x} \leftarrow$ line segment connecting r_i and x;
$L'_{r_i x}, L''_{r_i x} \leftarrow$ lines parallel to $L_{r_i x}$ at distance $1/2$ on either side of $L_{r_i x}$ towards $\overline{u_1 u_2}$;
if $L'_{r_i x}$ and $L''_{r_i x}$ share no point occupied by any other robot **then**
    Move to point x in the safe zone;
    $r_i.light \leftarrow Red$;

Algorithm 5: $Side(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$

if *at least one neighbor of* r_i *in the edge* e *it belongs to has light Red* **then**
    Order the robots in the counterclockwise order of r_i (with respect to the local coordinate system of r_i) such that the order is $\mathcal{T}_i = \{v_3, v_2, r_i, r, v_0\}$, where v_3 is the first robot non-collinear to r_i in the clockwise direction of r_i with $v_3.light = Red$, v_2 is the robot that is collinear with r_i in the clockwise direction of r_i, and r is the collinear robot in the counterclockwise direction of r_i, and v_0 is the first non-collinear robot to r_i in the counterclockwise direction of r_i with $v_0.light = Red$;
    Compute angles $\alpha = 180° - \angle v_0 r r_i$ and $\beta = 180° - \angle r_i v_2 v_3$, and set $\delta = \min\{\alpha/4, \beta/4\}$;
    Compute points x' and x'' such that $\angle x' v_2 r_i = \delta$ and $\angle x'' r r_i = \delta$ and $r_i x'$ and $r_i x''$ are perpendicular to e;
    $x \leftarrow x'$ or x'' whichever is nearest to e;
    Move perpendicular to e with destination x;
    $r_i.light \leftarrow Red$;

CHAPTER 4

The Pattern Formation for Fat Robots with Lights

In this chapter, we consider the Pattern Formation problem in the robots with lights model.

4.1 Algorithm

In this section, we present an algorithm that solves the Pattern Formation problem for $n \geq 1$ robots of unit-size disk in the robots with lights model. Our algorithm assumes the fully synchronous setting. We first present two algorithms that use at most 11 colors: one algorithm allows the target pattern to be scaled, while the other does not. Neither algorithm allows the target pattern to be rotated or reflected. Our algorithms execute four phases: Mutual Visibility, Leader Election, Line Formation, and Pattern Formation.

4.1.1 Mutual Visibility

Starting from any initial configuration, the goal of this phase is to move the robots to positions where every robot can see all other $n - 1$ robots. Chapter 3 achieves this by positioning all robots on a convex hull and the presented algorithm runs in $O(n)$ rounds and uses only two colors (*off* for robots that are not yet a corner of the convex hull, and *corner* for robots that are). This algorithm runs in the fully synchronous setting with obstructed visibility in the robots with lights model and avoids collisions. While we use this algorithm as a black box, the one important thing to note is that

to ensure that there is always enough space for all robots on the boundary of the convex hull, the corner robots expand the convex hull in each round. Hence, until this phase is completed, the corner robots move each round. Figure 4.1 shows an initial configuration as well as the final situation of the Mutual Visibility phase.

FIGURE 4.1: An example of the Mutual Visibility phase: (a) an initial configuration and (b) the end configuration.

4.1.2 Leader Election

The goal of the Leader Election phase is for a single robot to be elected as a leader. After the Mutual Visibility phase, every robot sees all other $n-1$ robots, so the robots know the value of n (while they remain visible to each other, as they have no memory). It is known that electing a leader is not possible using a deterministic algorithm in an anonymous distributed system [7]. Hence, we use the randomized algorithm by Vaidyanathan *et al.* [56].

Initially, all robots are competing and use a color, say *competing*, to indicate this. The algorithm proceeds in iterations until it finishes with a single leader. Each iteration has a constant number of rounds. In an iteration with n competing robots, each robot flips a coin whose probability of success is $1/n$. If a robot is successful, then the robot leaves its color as *competing*. Otherwise, it changes its color to *non-competing*. If there is exactly one competing robot left, the robots have successfully elected a leader and this robot changes its color to the *leader* color. Otherwise, this iteration was

unsuccessful and all robots change their color to back *competing* and try again. Figure 4.2 shows an example of the Leader Election phase.

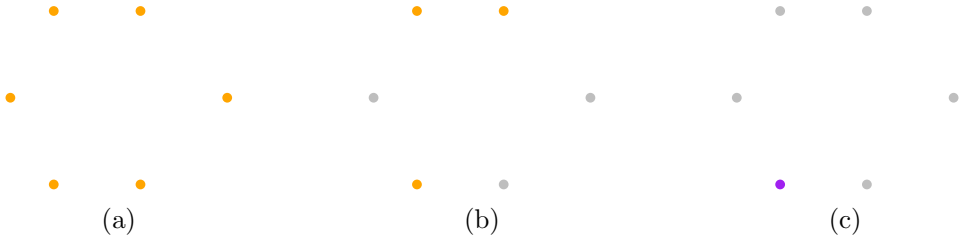

FIGURE 4.2: An example of the Leader Election phase: (a) initially all robots are competing (orange), (b) an unsuccessful iteration with competing and non-competing (gray) robots, and (c) a successful iteration, where a single robot is elected leader (purple).

4.1.3 Line Formation

The goal of the third phase is to move the robots from their convex hull positions to a line away from both their old positions and away from where the leader will build the target pattern in the next phase. In this phase, the leader will consider its own coordinate system and move the other robots one by one to achieve this goal. Where the leader forms the line of robots depends on the current location of the convex hull of robots and the area required to build the pattern (we assume without loss of generality that the leader will use its origin $(0, 0)$ as the topright corner of the bounding box of the target pattern).

Since the leader is a robot on the convex hull, at least one of the quadrants originating from the leader's position points away from the convex hull and thus does not contain any robots. We assume without loss of generality that this is the lowerleft quadrant in the leader's coordinate system. The leader will use this quadrant to build the line. Without loss of generality, we will explain the Line Formation and Pattern Formation phases using this quadrant. This assumption can easily be removed by mirroring the approach along the x- or y-axis.

The leader computes the topmost position of the line as follows. Let x_{hull} and y_{hull} denote the minimum x- and y-coordinate of any robot on the convex hull according to the leader's coordinate system. Similarly, let $x_{pattern}$ and $y_{pattern}$ denote the minimum x- and y-coordinate of the target pattern according to the leader's coordinate system. The leader now computes the topmost position (x_{line}, y_{line}) on the line as $x_{line} = \min(x_{hull}, x_{pattern}) - M$ and $y_{line} = \min(y_{hull}, y_{pattern}) - M$, where M is some large constant. Picking this coordinate guarantees that the line will be built below and to the left of both the convex hull and the target pattern. We note that while the leader can compute this coordinate, it cannot store it for use in future rounds, so it needs to move directly from its current position to where it wants to build the line.

If (x_{line}, y_{line}) has only a single closest robot on the convex hull, the leader now moves to this position (x_{line}, y_{line}) and changes its color to *follow the leader*. As there are no robots on the line yet, this signals that the robot closest to the leader's position should move to this location. In the next round, the leader moves down one unit (and sets its color to *do not follow*) to avoid colliding with the approaching robot. Once this robot reaches its position, it sets its color to *on the line*. Figure 4.3 shows these movements as well as the remaining ones in this phase. If (x_{line}, y_{line}) has more than one closest robot on the convex hull, the leader moves to ensure that the leftmost bottommost robot is the unique closest one in the next round. It does this by computing an x_{temp} such that this is the case and then setting $x'_{line} = \min(x_{temp}, x_{line})$ to get new coordinates (x'_{line}, y_{line}) ensuring that the moves described earlier activate a single robot to move to the leader's position. Once the first robot is in place, this robot will not move for the remainder of the phase.

Now that one robot is in place, the leader can observe the location of this robot to "remember" where the line should be built. The leader proceeds to

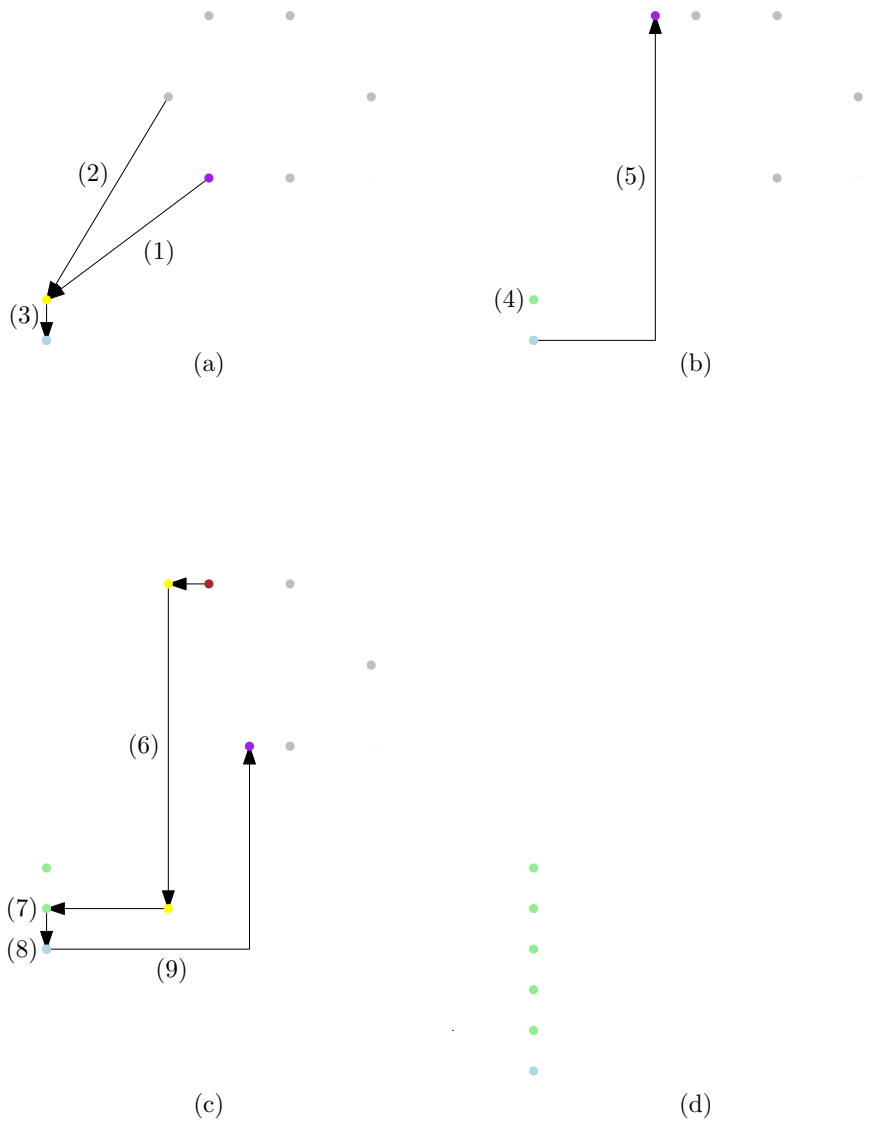

FIGURE 4.3: An example of the Line Formation phase (numbers indicate order of operations): (a) the leader (purple) moves to the first position on the line and signals (yellow) the closest robot to move there in the next round before moving out of the way and setting its color to *do not follow* (lightblue), (b) the first robot changes its color to *on the line* (green) and the leader moves next to the next robot to be moved, (c) the leader moves the next robot to its position on the line while the following robot has its color set to *following the leader* (brown), after which the leader moves to guide the next robot, and (d) the result of the Line Formation phase.

move the remaining robots to the line one at a time. To move a robot, the leader moves one unit to the left of the leftmost bottommost robot r_i remaining on the convex hull and changes its color to *follow the leader*. Robot r_i changes its color to *following the leader*, but does not move yet. The leader then repeatedly moves while robot r_i follows the leader by moving to the leader's previous observed position until it reaches its position on the line. The movements of each robot are: (1) r_i moves one unit to the left while the leader moves down until it reaches the y-coordinate of robot r_i's intended position on the line, (2) r_i moves down where the leader stopped, while the leader moves left to r_i's intended position on the line, and (3) the leader moves two units down and changes its color to *do not follow* while r_i moves to its position on the line and changes its color to *on the line*. The leader now moves right to observe the remaining robots to find the new leftmost bottommost remaining robot. This process is repeated until every robot is positioned on the line. To avoid collisions, the leader ensures that there are two units of vertical space between consecutive robots on the line. We note that a robot can determine whether the phase has ended by seeing whether it can see any colors other than *on the line* and any of the leader's colors.

4.1.4 Pattern Formation

The goal of the Pattern Formation phase is to relocate the robots to form the given target pattern. The leader can determine where the robots should be placed by placing the first robot at $(0, 0)$ with respect to its own coordinate system and building the pattern from there. The leader will build the pattern from this first robot towards the line, so given our assumed positioning this will be done from right to left, top to bottom. As the placement of the robots can be uniquely determined based on the robots on the line and the origin of the leader's coordinate system, the leader can determine what the last placed robot is and thus which part of the pattern has already been

completed. During this phase, unless instructed otherwise by the leader, the robots stay on the line and do not move.

We present two versions of the Pattern Formation phase: one that allows scaling of the pattern (requiring one new color) and one that does not (requiring two new colors).

4.1.4.1 The Algorithm with Scaling of the Target Pattern

In this version, we assume that the target pattern can be scaled. The first position of the first robot is at $(0, 0)$ in the leader's coordinate system. Again, we explain our algorithm in the setting where the leader builds the pattern in the lowerleft quadrant of the origin, but the algorithm can be mirrored to obtain the other quadrants. This phase uses one new color to allow robots to remember that they have reached their final position in the pattern.

After the Line Formation phase, all the robots are on a line with their light set to *on the line*, except for the leader. Similar to the previous phase, the idea is that the leader moves a single robot to the pattern by moving next to it and guiding the robot to the intended position in the pattern. The leader moves the robots from top to bottom as ordered along the line. The following process is illustrated in Figure 4.4. To activate a robot r_i on the line, the leader moves next to it and sets its color to *follow the leader*. The leader then moves such that its y-coordinate is equal to that of the intended position in the pattern and in the next round the leader moves to the intended position in the pattern. Robot r_i has set its color to indicate that it is following the leader and repeatedly moves to the leader's last observed position. To avoid collisions in the last step, the leader moves two units down and sets its color to *do not follow* to indicate that the robot reached its final position. At this point, r_i sets its color to *at final position*. The leader now proceeds to pick up the next robot and repeats this process until the pattern

is formed. If needed, the leader moves to fill the final position itself, if the pattern required exactly n robots.[1]

In order to ensure that after guiding a robot to its intended position the leader can indeed move down two units, we scale the pattern by a fixed constant factor, say 10. Since in the given target pattern robots can at most be touching each other (otherwise the pattern cannot be constructed, which can be detected by all robots before the algorithm even starts), scaling the pattern this way ensures that there is ample space between the robots for the leader to move as described.

4.1.4.2 The Algorithm without Scaling of the Target Pattern

In this version, we assume that the target pattern is not allowed to be scaled. As in the previous case, the pattern is built with respect to the origin $(0, 0)$ in the leader's coordinate system and the pattern will be built in the lowerleft quadrant. This version of the phase needs two new colors: one to "push" robots, and one to allow robots to remember when they have reached their final position in the pattern.

The high-level idea in this version of the Pattern Formation phase is that the leader first "pulls" a robot behind it to position it in such a way that there is a straight line path from the robot to its intended position in the target pattern. Once the robot reaches this position, the leader moves away to indicate to the robot how far it needs to move in the direction *opposite* to the direction the leader moved in, effectively "pushing" the robot to its final position.

In more detail (see also Figure 4.5), in order to move a robot r_i, the leader starts by moving to the position that has the same x-coordinate as the line and same y-coordinate of the final position of r_i in the target pattern. Here it changes its color to *follow the leader*, signaling to the topmost robot r_i of the

[1] We note that if the pattern requires more than n robots, it cannot be built and all robots can detect this situation in the Leader Election phase and terminate at that point.

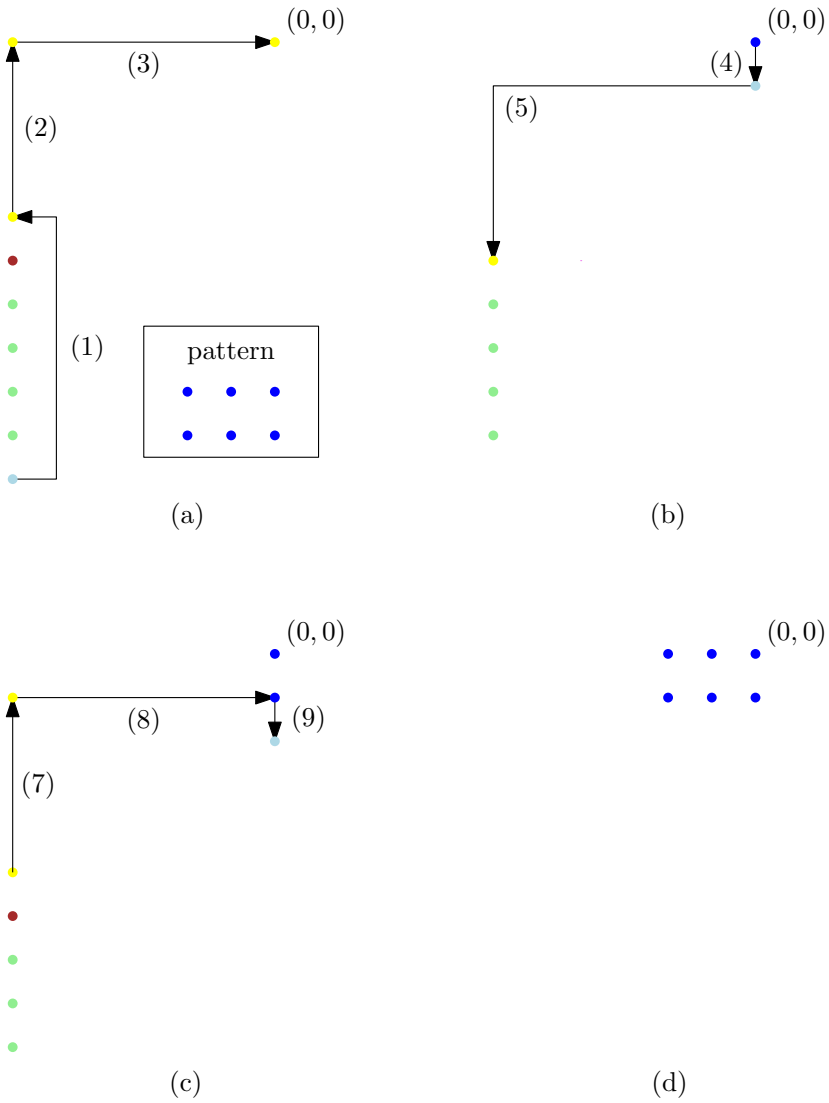

FIGURE 4.4: An example of the Pattern Formation phase when scaling is allowed (numbers indicate order of operations): (a) the leader moves to the position above the topmost robot r_1 on the line and signals that it should follow (yellow), which causes r_1 to change its color to *following the leader* (brown) and move accordingly, (b) the leader moves out of the way so robot r_1 can reach its final position, which the leader indicates by changing its color to *do not follow* (light-blue) and r_1 sets its color to *at final position* (blue), after which the leader moves to be immediately above the next robot on the line, (c) the leader guides the next robot to its final position, and (d) all robots have reached their final position.

line that it should move to the leader's current position. Robot r_i observes this and changes its color to *following the leader* to remember what it is supposed to do. Let d be the distance between the leader's current position and the position in the pattern where it wants to place r_i. Note that since the leader is already at the y-coordinate of this intended location, this distance is just the difference in x-coordinate. After determining d, the leader moves d to the left and changes its color to *push*. This indicates to robot r_i that in the next round it should move from its current position (where the leader used to be) to the position a distance of d away from its current position in the direction opposite to where it sees the leader. Once robot r_i reaches its final position, it changes its color to *at final position* to remember this. Since every robot is moved using exactly one "pull" and one "push", they know that after the push they have arrived at their final position without the leader having to indicate this in any way.

The leader now moves to fetch the next robot and this process is repeated until the pattern is formed. As before, if needed, the leader moves to fill the final position itself.

4.2 Analysis

We proceed to prove that our algorithms solve the Pattern Formation problem for fat robots and that there are no collisions among the robots. Analogous to our algorithm description, throughout this analysis we will assume without loss of generality that the leader uses the lowerleft quadrant in the Line Formation and Pattern Formation phases. We start by stating the result of Chapter 3 on the Mutual Visibility phase.

THEOREM 4.2.1. *Our algorithm solves the Mutual Visibility problem for unit disk robots in $O(n)$ rounds without collisions in the fully synchronous setting using two colors.*

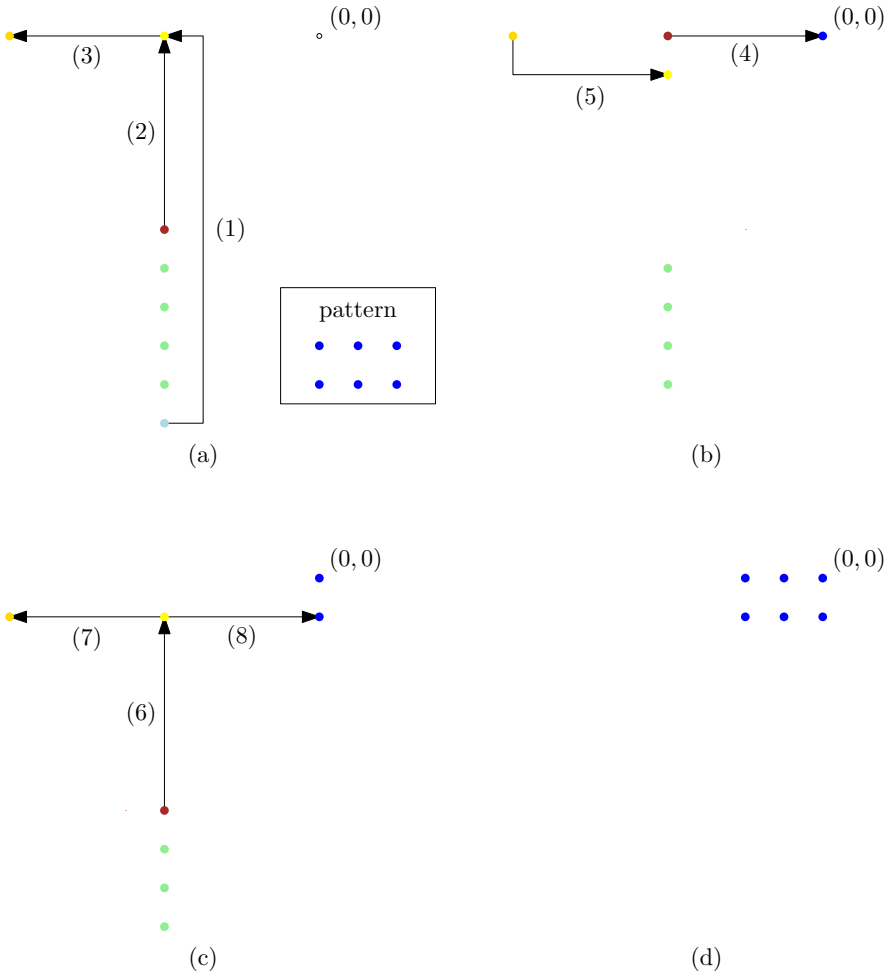

FIGURE 4.5: An example of the Pattern Formation phase when scaling is not allowed (numbers indicate order of operations): (a) the leader moves to the position on the line with y-coordinate equal to where the first robot r_1 needs to be placed and signals that r_1 should follow (yellow), which causes r_i to change its color to *following the leader* (brown) and move accordingly while the leader moves away preparing to push r_1, (b) the leader changes its color to *push* (gold) and r_1 moves distance equal to its current distance to the leader away from the leader to reach its final position and sets its color to *at final position* (blue), after which the leader moves to pull the next robot on the line, (c) the leader first pulls and then pushes the next robot to its final position, and (d) all robots have reached their final position.

Next, we consider the Leader Election phase. This phase builds on the algorithm described and analyzed by Vaidyanathan *et al.* [56], who bounded

the expected number of rounds needed for this phase as well as the associated probability. The number of colors follows directly from needing three new colors to discern competing, non-competing, and leader robots.

THEOREM 4.2.2. *For any $q > 0$, Leader Election can be solved in the fully synchronous setting for n robots in $O(q \log n)$ rounds with probability at least $1 - n^{-q}$ using three colors.*

Next, we analyze the Line Formation phase.

LEMMA 4.2.3. *In every round of the Line Formation phase, only one robot that is not the leader moves from the convex hull to be positioned on the line, avoiding collisions.*

PROOF. Robots only move when the leader activates them to do so. If there are no robots on the line, all robots can see each other and thus they can determine whether they are the robot closest to the leader, resulting in only a single robot being activated. Once there are robots on the line, the leader moves next to a robot and sets its color to *follow the leader* to activate it. As the leader activates the robots one at a time and completes moving a robot to the line before activating the next robot, only one robot moves from the convex hull to be positioned on the line. The "no collisions" part of the lemma then follows from the fact that we have only one moving robot, and that the robots are picked from left to right and start with a horizontal movement to move them away from the convex hull before moving vertically, thus ensuring there are no collisions. □

LEMMA 4.2.4. *The Line Formation phase uses four new colors.*

PROOF. The algorithm uses one color for the leader to indicate that a robot should follow it, one for a robot to store that it is following the leader, one for the leader to signal that the robot should stop following it, and one final color for the robot to store it is done for this phase. □

LEMMA 4.2.5. *The Line Formation phase takes $O(n)$ rounds.*

PROOF. In the Line Formation phase, the leader moves each robot from the convex hull to a line. The leader requires at most three rounds to move a robot to its position on the line. After each robot is moved, the leader uses at most three rounds to move next to the next robot. Therefore, the Line Formation phase takes $O(n)$ rounds to move all the robots from the convex hull to the line. □

Lemmas 4.2.3, 4.2.4, and 4.2.5 imply the following theorem.

THEOREM 4.2.6. *The Line Formation phase takes $O(n)$ rounds, avoids collisions, and uses four new colors.*

Next, we analyze the Pattern Formation phase. We start with the version where we can scale the pattern.

LEMMA 4.2.7. *In every round of the Pattern Formation phase with scaling, only one robot that is not the leader moves from the line to be positioned in the target pattern, avoiding collisions.*

PROOF. By construction, a robot only moves after it is activated by the leader. Since the leader moves the robots one at a time, only one robot will move. By moving the topmost robot of the line and starting by moving this robot vertically up (away from the other robots on the line), there are no collisions with other robots on the line. Furthermore, scaling the pattern by a large enough factor (such as 10), we ensure that there are also no collisions with robots in the pattern either: the pattern is built from right to left, meaning that we can only collide with robots that were originally at most a distance of 1 from the current robot. However, because the pattern is scaled, this distance is now increased to 10, meaning that any overlap

between the robot's paths into the pattern and any previously placed robots is removed. □

LEMMA 4.2.8. *The Pattern Formation phase with scaling uses one new color.*

PROOF. By construction, the algorithm uses only a single new color during this phase: for a robot to store that it has reached its final position. □

LEMMA 4.2.9. *The Pattern Formation phase with scaling takes $O(n)$ rounds.*

PROOF. In the Pattern Formation phase, the robots move from the line to the target pattern one by one. The leader moves at most three times to move next to the robot it wants to move and it uses three rounds to move this robot to its final position in the pattern. Hence, it takes a constant number of rounds to move one robot from the line to the target pattern. As a result, in total we require at most $O(n)$ rounds to move all the robots to the target pattern. □

Using Lemmas 4.2.7, 4.2.8, and 4.2.9, we get the following theorem.

THEOREM 4.2.10. *The Pattern Formation phase with scaling takes $O(n)$ rounds, avoids collisions, and uses one new color.*

Using Theorems 4.2.1, 4.2.2, 4.2.6, and 4.2.10, we can now conclude the following.

THEOREM 4.2.11. *Our algorithm solves the Pattern Formation problem when scaling the target pattern is allowed for n unit disk robots in $O(n) + O(q \log n)$ rounds with probability at least $1 - n^{-q}$ without collisions in the fully synchronous setting using 10 colors.*

Finally, we analyze the Pattern Formation phase if scaling is not allowed.

LEMMA 4.2.12. *In every round of the Pattern Formation phase without scaling, only one robot that is not the leader moves from the line to be positioned in the target pattern avoiding collisions.*

PROOF. By construction, a robot only moves once it is activated by the leader. Hence, only one robot moves at a time, and since the leader always picks the topmost robot on the line and moves it vertically away from the line, no collisions with the other robots on the line can occur. Since the leader knows the pattern and can determine the last robot placed based on the placement order of the pattern and the visible robots, it can also compute how to push the robot into the pattern to avoid collisions (since the pattern is built right to left from top to bottom, a left to right push exists). By pushing the robot this way, there are thus no collisions. □

LEMMA 4.2.13. *The Pattern Formation phase without scaling uses two new colors.*

PROOF. By construction, the algorithm uses two new colors: one to signal that a robot is being pushed, and one for the robot to store it has reached its final position. □

LEMMA 4.2.14. *The Pattern Formation phase without scaling takes $O(n)$ rounds.*

PROOF. In the Pattern Formation phase, the robots move from the line to the target pattern one by one. The leader uses at most three moves to position itself above the topmost robot on the line and then another three rounds to move this robot to its final position in the pattern. Therefore, it takes a constant number of rounds to move one robot from the line to the target pattern. As a result, the total number of rounds used to move all the robots to the target pattern is $O(n)$. □

Using Lemmas 4.2.12, 4.2.13, and 4.2.14, we get the following theorem.

THEOREM 4.2.15. *The Pattern Formation phase without scaling takes $O(n)$ rounds, avoids collisions, and uses two new colors.*

Using Theorems 4.2.1, 4.2.2, 4.2.6, and 4.2.15, we obtain our final result.

THEOREM 4.2.16. *Our algorithm solves the Pattern Formation problem when scaling the target pattern is not allowed for n unit disk robots in $O(n) + O(q \log n)$ rounds with probability at least $1 - n^{-q}$ without collisions in the fully synchronous setting using 11 colors.*

4.3 Improving the Number of Colors

In this section, we improve the number of colors used to solve the Pattern Formation problem by reusing some of the colors used in the different phases discussed in the previous sections.

The Mutual Visibility phase uses two colors: *off* for non-corner robots and *corner* for corner robots. The Leader Election phase uses three new colors to keep track of status of the different robots as *competing*, *non-competing*, and *leader*. We start by arguing that instead of using a new color for non-competing robots, we can use the color *off* instead.

LEMMA 4.3.1. *The* off *color can be reused as the* non-competing *color in the Leader Election phase.*

PROOF. We need to argue that the Leader Election phase still works as intended and that reusing the color does not cause any problems in later phases.

When the robots activate in the Leader Election phase, they change their color to *competing*. During this phase, unsuccessful robots change their

color to *off* as non-competing robots. During this process all robots are mutually visible and thus the non-competing robots can always see either a competing robot or the leader robot,[2] which have colors that help them identify the phase. Hence, they can conclude that they are in the Leader Election phase and thus act accordingly.

Since some robots have the *non-competing* color during part of the Line Formation phase, we now argue that changing this to the *off* color does not cause any issues. We note that any robot that can see a robot with the *leader* color or a robot with the *on the line* color can conclude that it should not do anything unless the leader activates it and thus these robots cannot cause issues in the execution of the algorithm. However, if a robot sees neither of these colors it can mistakenly think that it is in the Mutual Visibility phase. We note that the robot would conclude that it is a corner and thus set its color to *corner*. As a corner robot, it would move to expand the convex hull away from where the line is being built (as any robot that would expand the convex hull towards the line can see the line). As robots are removed from the convex hull by the leader, every robot will eventually see the line and at that point it can conclude the correct phase again and stop moving. We note that since the robots move to expand the convex hull away from the other robots, no collisions can occur. Hence, while the robots can mistake the phase they are in, the fact that they would stop the moment they see the leader or a robot on the line implies that this does not cause issues for our approach. □

The Line Formation phase uses four new colors in order to allow the leader to activate a robot to follow it, for robots to store whether they are following the leader, for the leader to indicate that a robot should stop following it, and for a robot to store that it reached its position on the line. We argue that we

[2] If at any point all robots become non-competing, they could incorrectly conclude that they are in the Mutual Visibility phase, but since they would restart the Leader Election phase anyway, this is no issue.

can reuse the *competing* color from the Leader Election phase as the color to indicate that a robot is following the leader.

LEMMA 4.3.2. *The* competing *color can be reused as the* following the leader *color in the Line Formation and Pattern Formation phase.*

PROOF. The *competing* color was originally used to elect a single leader, so there was no leader before. Now since the robot with the *competing* color sees the leader, it knows a leader has already been elected, and thus it can conclude that this is the Line Formation or Pattern Formation phase, where it has to follow the leader. Therefore, the *competing* color can be reused as the *following the leader* color in these phases. □

Next, we argue that we can also reuse the *leader* color as the *do not follow* color in these phases.

LEMMA 4.3.3. *The* leader *color can be reused as the* do not follow *color in the Line Formation and Pattern Formation phase.*

PROOF. The function of both the *leader* color in the Leader Election phase and the *do not follow* color in the Line Formation and Pattern Formation phases is to indicate which robot is the leader, while ensuring that the other robots do not move. Hence, both colors allow the leader to move around without affecting moving the other robots, allowing it to get into position to guide the robots one at a time to their new positions on the line or in the pattern. □

The above reuse of colors implies the following theorems.

THEOREM 4.3.4. *Our algorithm solves the Pattern Formation problem when scaling the target pattern is allowed for n unit disk robots in $O(n)+O(q \log n)$ rounds with probability at least $1 - n^{-q}$ without collisions in the fully synchronous setting using 7 colors.*

THEOREM 4.3.5. *Our algorithm solves the Pattern Formation problem when scaling the target pattern is not allowed for n unit disk robots in $O(n) + O(q \log n)$ rounds with probability at least $1 - n^{-q}$ without collisions in the fully synchronous setting using 8 colors.*

CHAPTER 5

The Pattern Formation for Fat Robots with Memory

In this chapter, we consider the Pattern Formation problem in the robots with memory model.

5.1 Mutual Visibility

Our algorithm works in three phases. The first phase is the Mutual Visibility phase. The goal of this phase is to move the robots such that at the end of the phase every robot can see all other $n - 1$ robots. The algorithm we present forms a convex hull with each robot positioned on a corner of the convex hull (see Figure 5.1 for an example). We will argue later that our approach runs in $O(n)$ rounds and uses $O(1)$ memory space.

FIGURE 5.1: An example of the Mutual Visibility phase: (a) an initial configuration and (b) the end configuration of this phase.

5.1.1 Algorithm

On a high level, our Mutual Visibility algorithm is similar to the algorithm of Chapter 3. However, our algorithm works in the classical robots model

(i.e., without the lights required in Chapter 3. Their algorithm used two colors to distinguish (and communicate) which robots are corners of the convex hull and which robots are in the interior. Their algorithm proceeds to gradually expand the convex hull by moving the corner robots one unit away from the interior of the convex hull using the angle bisectors of consecutive robots on the hull. This ensures that the edges of the hull will become long enough to let all interior robots move through them to the outside of the convex hull to become new corners without collisions. The algorithm in Chapter 3 showed that this process completes in $O(n)$ rounds.

We first observe that robots can easily determine whether they themselves are interior or corner robots. Corner robots are those robots on the vertices of the convex hull with interior angles less than 180°. There could initially be robots with interior angle equal to 180°, but since all the corner robots move in every round to expand the convex hull these robots will become interior robots almost immediately.

Let us look a bit more closely at how algorithm in Chapter 3 moved the interior robots. They define *eligible edges* for each robot as the set of convex hull edges of length at least 3 for which no other robot is closer to the edge. If an interior robot has eligible edges, it picks the closest one and moves perpendicular to this edge through it to become a new corner of the convex hull. In every round, there are at most two interior robots that move through any single edge of the convex hull. By construction, the movements of the robots through different edges do not interfere each other. Therefore, there is no collision among interior robots. Since all the corner robots move in every round, the edges of the hull are long enough to move the interior robots through without any collisions. Therefore, there is no collision among all robots. Since a robot can determine whether it is interior or not, and all other computations for this process are based on the locations

of the other robots it sees, this process does not require the colors and can be mimicked in our algorithm.

Hence, it remains to argue that the robots can correctly determine when this phase has ended. Since there are no lights to signal this (all robots having the corner robots light does this in the original algorithm) and the robots do not know n, it is difficult for the robots to know whether the phase has ended even if the robots have a little memory. Specifically, there are cases where from a corner robot's perspective all other visible robots are in convex position. For example, in Figure 5.2 robot r_i cannot see robot r_k since robot r_j blocks its view to that robot. However, since all robots that are visible to r_i are in convex position, it can now incorrectly conclude that it can see all robots.

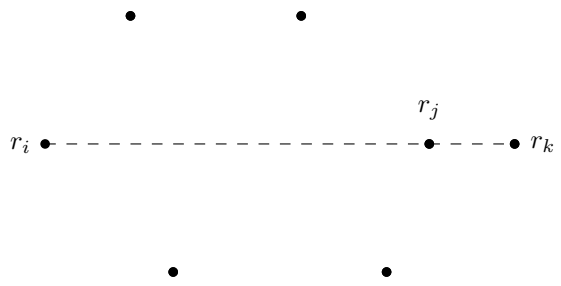

FIGURE 5.2: An example configuration where a corner robot r_i believes all robots are in convex position when there is an interior robot r_j left which blocks visibility to robot r_k.

Fortunately, the above problem can only occur for a single robot in the configuration. All other robots can see robot r_j as the interior robot it is. In the next section, we will prove this claim. Now, what we need to avoid is robot r_i incorrectly and prematurely ending the Mutual Visibility phase for itself, as we cannot correct this easily due to the lack of communication between robots. Hence, when a corner robot believes the phase is over because it observed that all other visible robots are in convex position, it starts a counter with value $4k + 2$, where k is the number of visible corner robots. It stores both the value k and the counter in its memory. Every round, it continues expanding the convex hull as normal (i.e., it continues

to behave the same way as if not all robots are in convex position yet), but it also decrements the counter. This is to ensure that if at any point it realizes that it made a mistake, we do not have to deal with correcting the situation or computing where it should have moved due to a mistake a certain number of rounds ago. If it sees more than k corner robots within these $4k + 2$ rounds, it stops its current counter and concludes that it was previously incorrect. Because the robot acted as normal, it is already in the exact same location as it would have been if it had not made this mistake and thus the algorithm can proceed as normal (and if it again observes all visible robots in convex position, it will start a new counter using the new higher value of k). In the next section, we will argue that this process ensures that every corner robot correctly concludes that the Mutual Visibility phase has ended and that they all conclude this in the same round. As a result, the Mutual Visibility problem has been solved. All robots store this in their memory.

5.1.2 Analysis

We proceed to prove that our algorithm solves the Mutual Visibility problem in a linear number of rounds using a constant amount of memory, while avoiding collisions between the robots.

Since our algorithm is similar to the algorithm in Chapter 3, the main parts of the proof can be found there. We therefore focus on the differences between the two algorithms.

We start by recalling that contrary to the original algorithm, robots cannot always determine whether they are in convex position. Specifically, if multiple robots are collinear, some might observe all others to be in convex position, while globally this is not yet the case (see Figure 5.2). We first

argue that there can be at most one robot that incorrectly concludes that all robots are in convex position.

LEMMA 5.1.1. *If there exists an interior robot, then there exists at most one corner robot that believes all robots are in convex position.*

PROOF. We observe that for a robot to not be visible to a corner robot, three robots have to be collinear. This implies that all robots except the two extreme ones on this line can conclude that they are not in convex position.

Now consider such an extreme robot. If there are one or more interior robots that block the view of one corner robot, this corner robot observes the position of at least one interior robot by using the line segments connecting each pair of corner robots (see Figure 5.3).

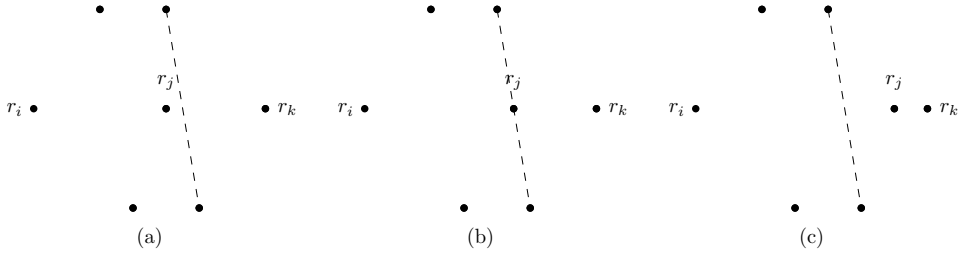

FIGURE 5.3: Robot r_j blocks visibility between r_i and r_k: (a) r_i concludes that r_j is an interior robot, (b) r_i concludes that the robots are not yet in convex position, and (c) r_i incorrectly concludes that the robots are in convex position, but r_k still correctly concludes that this is not the case.

If an interior robot r_j lies before a line segment between a pair of corner robots (see Figure 5.3a), the corner robot r_i observes that they are not in convex position. If the interior robot r_j lies on the line segment between a pair of corner robots (see Figure 5.3b), the corner robot r_i observes that there are three collinear robots and thus the robots are not yet in convex position. If an interior robot lies behind the line segment between a pair of corner robots (see Figure 5.3c), corner robot r_i may believe the robots are in convex position because the interior robot blocks the view to the corner

robot r_k that lies behind it. Hence, from r_i's perspective, all robots are in convex position. However, note that from r_k's perspective r_j lies before this same line segment and thus r_k correctly concludes that the robots are not in convex position yet. Hence, only one robot can incorrectly conclude that all robots are in convex position and the lemma follows. □

Next, we argue that when a robot starts a counter, within $4k + 2$ rounds either it sees more corner robots or it can correctly conclude that mutual visibility has been achieved.

LEMMA 5.1.2. *If a robot r starts a counter with value $4k + 2$ (where k is the number of currently visible corner robots) then within that number of rounds either it sees more than k corner robots or mutual visibility has been achieved, and r terminates this phase.*

PROOF. Lemma 3.2.14 in Chapter 3 in this thesis shows that it takes at most $4k$ rounds to make enough space for the interior robots to move outside the current convex hull. In addition to that, it takes two more rounds to move at least one interior robot outside the current convex hull. Therefore, if there is an interior robot inside the hull, it takes at most $4k + 2$ rounds for a new corner robot to appear. Since all other corner robots remain visible, this implies the first part of the lemma. Otherwise, if there is no interior robot inside the hull, then mutual visibility has been achieved, and r correctly terminates this phase. □

Now that we know that at some point the robots conclude that the phase has concluded, we proceed to show that all robots terminate in the same round.

LEMMA 5.1.3. *All robots terminate the Mutual Visibility phase in the same round.*

PROOF. Since by Lemma 5.1.1, only one corner robot can incorrectly believe that all robots are in convex position, only this robot can start a counter with value $4k + 2$ by Lemma 5.1.2. If there is an interior robot, by Lemma 5.1.2 it will conclude this within $4k + 2$ rounds.

Once there are no interior robots, every corner robot starts a counter with value $4n + 2$ when the last corner robot joins the convex hull, i.e., in the same round. Therefore, all robots terminate the Mutual Visibility phase in the same round. □

It remains to analyze the time complexity of the Mutual Visibility phase.

LEMMA 5.1.4. *The Mutual Visibility phase takes $O(n)$ rounds.*

PROOF. This lemma follows from Lemma 3.2.14 in Chapter 3 in this thesis, which states that the original algorithm takes $O(n)$ rounds, and Lemma 5.1.2, which adds an additional $4n+2$ (i.e., $O(n)$) rounds for all robots to correctly conclude that they are in convex position. □

LEMMA 5.1.5. *The Mutual Visibility phase uses $O(1)$ memory.*

PROOF. In the Mutual Visibility phase, every robot needs to store the termination counter and the number of visible robots at that time. Both of these are upper bounded by $O(k)$ which is at most $O(n)$ and can thus be stored in $O(\log n)$ bits, i.e., $O(1)$ memory space. The additional bits needed to store the current phase do not affect this bound. □

Therefore, we conclude the following theorem.

THEOREM 5.1.6. *The Mutual Visibility phase finishes in $O(n)$ rounds without collisions in the fully synchronous setting using $O(1)$ memory.*

5.2 Leader Election

The second phase of our algorithm is the Leader Election phase. The goal of the Leader Election phase is for a single robot to be elected as a leader. It is known that this problem cannot be solved with a deterministic algorithm in an anonymous distributed system [7]. We present an algorithm which is an adaptation of the slotted-Aloha protocol [56].

5.2.1 Algorithm

After the Mutual Visibility phase, every robot sees all other $n - 1$ robots and thus n is known and can be stored by all robots. This implies that at this point all robots can determine whether it is possible to build the given pattern with the number of available robots: if the number of robots in the given pattern is at most n, then the robots proceed and otherwise, they all terminate. Hence, in the remainder, we assume that the number of robots required to build the pattern is at most n.

The next phase of our algorithm aims to elect a single robot to be the leader in the final Pattern Formation phase. The leader will store that it is the leader and when a leader is elected all robots will be aware of this fact. However, but since all robots are identical, once a leader is elected, the other (non-leader) robots will not be able to store which robot is the leader. All that each of these robots can remember is that it is not the leader.

In order to elect a leader all robots first compute the centroid c of the convex hull formed by the robots. Next, they all determine the distance d from the centroid to the robot farthest from the centroid. Since all robots are visible to each other, all robots will compute the same c and d. The leader is selected from among all robots on the circle centered at c having radius d. In

other words, all robots on the boundary of the circle are *competing* to become the leader and all robots in the interior of the circle are *non-competing*. By construction, there are no robots outside the circle and there is at least one robot on the boundary of this circle, but there could be multiple and thus we need a mechanism to pick one of them.

To facilitate this, the following process is repeated until a leader has been elected: All competing robots flip a coin where the probability of success is $1/k$, where k is the number of competing robots. If a robot is successful, then the robot remains in its current position on the boundary of the circle. Otherwise, it stores its current location on the circle and moves inside the circle some distance without crossing the straight line between its two neighbours, to ensure that the convex hull remains convex and thus all robots remain visible to each other (see Figure 5.4). If there are 0 or more than 1 robot on the boundary of the circle, the robots conclude that this iteration failed to elect a leader and the robots that were competing in the previous round return to their positions on the boundary of the circle using their stored locations to repeat the algorithm at the next iteration. When there is exactly one competing robot left on the boundary of the circle, this robot becomes the leader and it stores this information. Note that all robots can observe this event, as we keep the convex hull convex and thus the robots remain mutually visible throughout this process. Hence, all robots can determine whether a leader has been elected and whether they are the leader. Every robot stores this information, along with the fact that the Leader Election phase has ended and proceeds to the next phase of the algorithm.

5.2.2 Analysis

We start by arguing the correctness of the Leader Election phase.

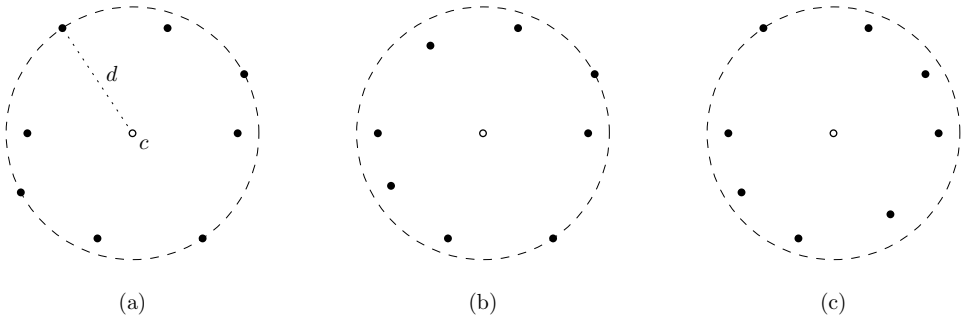

FIGURE 5.4: An example of the Leader Election phase: (a) centroid c and distance d are computed and used to form a circle, (b) an unsuccessful iteration where two robots remain on the boundary of the circle, and (c) a successful iteration where a leader is elected and the phase ends.

LEMMA 5.2.1. *Leader election can be solved in the memory model with n robots.*

PROOF. The analysis is identical to that in [56] with the lights replaced by being on the circle or not being on the circle. As this can be observed by all robots at all time, the lemma follows. □

Next, we consider the memory required by our algorithm.

LEMMA 5.2.2. *The Leader Election phase uses $O(1)$ memory.*

PROOF. In the Leader Election phase, every robot needs to store the circle, i.e., the center and the radius. Since both are a linear combination of the locations of the robots, these can indeed be computed and stored in $O(\log n)$ bits. In order to allow competing robots to move back to their original positions when an iteration is unsuccessful, we store their previous location, which takes $O(\log n)$ bits. Finally, every robot needs a constant number of bits to store whether it is a leader and whether the phase has ended. In total, this requires $O(\log n)$ bits of memory, i.e., $O(1)$ memory space. □

Finally, since the expected running time of the Leader Election phase has already been analyzed in the asynchronous setting [56], and the fully synchronous setting takes at most the same number of rounds, we obtain the following result.

THEOREM 5.2.3. *For any $q > 0$, the Leader Election phase finishes in $O(q \log n)$ rounds, with probability at least $1 - n^{-q}$, without collisions in the fully synchronous setting using $O(1)$ memory.*

5.3 Pattern Formation

The final phase of our algorithm is the Pattern Formation phase. The goal of this phase is to move the robots to form the given (target) pattern and thus solve the original problem.

5.3.1 Algorithm

Since we now have a leader, this robot will facilitate the other robots moving and the other robots will not move unless instructed to do so by the leader. Our algorithm depends on the leader's position on the convex hull at the end of the Leader Election phase. Since the leader is positioned on the convex hull of the robots, this means that at least one of its quadrants does not contain any robots. Once the leader has determined its empty quadrant, it proceeds to build the target pattern in that quadrant. Without loss of generality, we assume that the leader's top left quadrant is empty, and thus we want to build the pattern there. We define an ordering on the robots (excluding the leader) $r_1, \ldots, r_{n-1}$ and an ordering on the positions in the pattern $p_1, \ldots, p_k$ ($k \leq n$). Both orderings are left to right, with ties being broken top to bottom. The high-level idea is that the leader moves robot r_i to pattern position p_i repeatedly until the full pattern is built. If needed, the leader itself will fill position p_n.

To avoid collisions during this process, the Pattern Formation phase allows the given pattern to be scaled, as allowed by the problem. This is to ensure that after leading a robot to its position in the pattern, the leader can move away to get the next robot without colliding with previously placed robots in the process. Since the pattern is given as part of the input, every robot knows it and thus also knows the original size of the target pattern. The leader computes the size of the pattern after scaling by multiplying the input original size of the pattern by the scaling factor (say 5) to get the final target pattern after scaling. Note that the scaled pattern does not need to be stored, as the leader can simply recompute it when needed. The leader stores the scaling factor, as well as the first and last locations of the pattern after scaling in order to reconstruct the placement of the robots in the scaled pattern.

In order to precisely define the robots' movements, we need to determine some coordinates. After the Leader Election phase, let the leader be positioned at (x, y). Let y_{max} be the y-coordinate of the topmost robot on the convex hull and let x_{min} be the x-coordinate of the leftmost robot on the convex hull. To ensure that there is no overlap between the convex hull and the scaled target pattern, the leader will build the pattern such that p_k is at $(x_{min} - 100, y_{max} + 100)$ in the leader's coordinate system.

Next, we describe how the leader moves the robots in more detail (see also Figure 5.5). Initially the leader lies on the circle of the Leader Election phase and in order to move away from that without colliding with any other robot, the leader first moves perpendicular to the circle until it reaches the x-coordinate $x_{min} - 1$ in one round. It then moves to $(x_{min} - 1, y(r_1))$, i.e., one position to the left of the leftmost topmost robot r_1 in the ordering. Robot r_1 observes that there is a robot (the leader) touching it and concludes that this robot must be the leader and that it needs to follow this robot. The problem now is that the leader cannot communicate with robot r_1 where it

needs to go and since p_1 can essentially be arbitrarily far away, guiding the robot using repeated unit distance steps is not feasible.

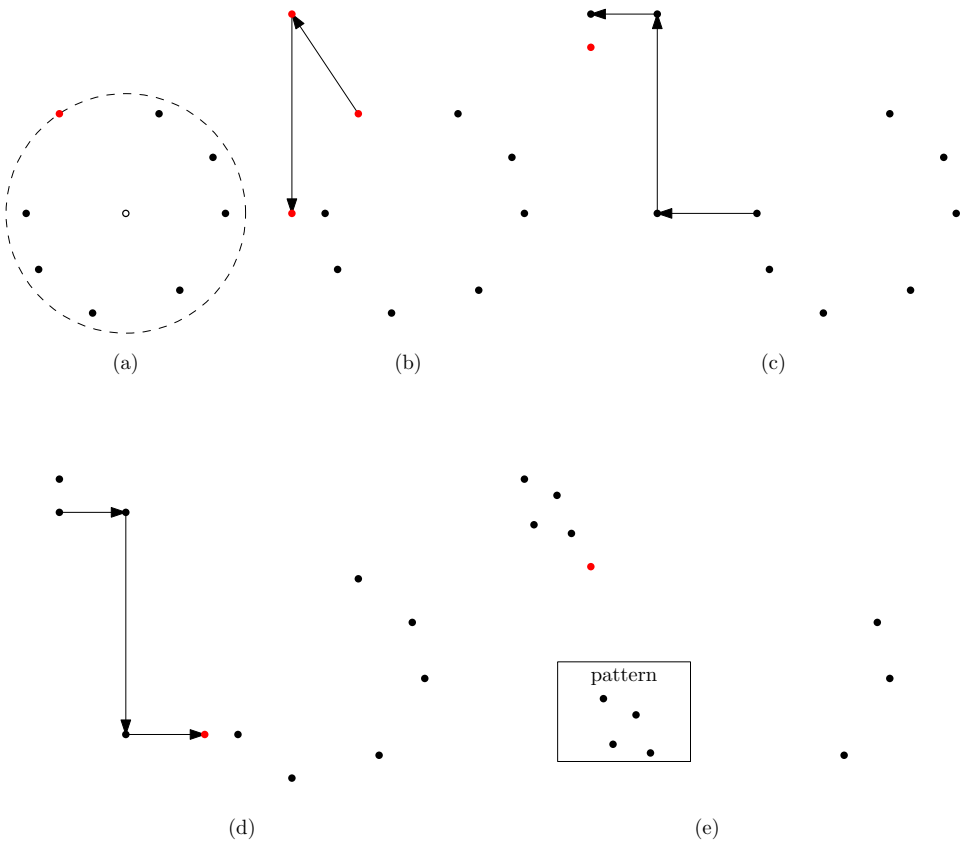

FIGURE 5.5: An example of the Pattern Formation phase (leader is colored red): (a) the situation after a leader has been elected, (b) the leader moves next to r_1, (c) the leader moves r_1 to p_1, (d) the leader moves next to the next robot, and (e) the pattern is built and the phase ends.

To work around this issue, the leader "instructs" the robot in two rounds. In the first, it places itself next to the robot in the direction it wants the robot to move in. Robot r_1 observes this and stores this direction, for example using the location of the leader. In the next round, the leader moves the intended distance in the direction of intended movement. Robot r_1, having stored the direction, sees how far the closest robot in that direction is and stores this distance. Once the leader has moved out of the way, robot r_1 is now free to move to the indicated location. Whenever the leader wants to move a robot

somewhere this process is executed a constant number of times, to facilitate multiple sequential moves.

Moving a robot r_i to its position p_i always starts with the leader moving one position to the left of robot r_i, to $(x(r_i)) - 1, y(r_i))$. The leader uses the above approach to guide robot r_i horizontally left to $(x_{min} - 90, y(r_i))$. The leader then moves to $(x_{min} - 90, y(r_i) + 1)$ (one position above the robot) to let robot r_i store the next direction and follows the leader until it reaches $(x_{min} - 90, y(p_i))$ by moving vertically up. The leader is now at $(x_{min} - 91, y(p_i))$ (again one position to the left of r_i) to guide r_i to its final position p_i at $(x(p_i), y(p_i))$ by moving horizontally left. In order to ensure that r_i does not follow the leader any further, the leader moves two positions down to $(x(p_i), y(p_i) - 2)$. Since the target pattern is scaled, this extra movement of the leader cannot cause collisions with other robots in the target pattern, as in the original pattern robots are at least a distance of 1 apart and thus now they are at least a distance of 5 apart, which is more than the required 2 units of movement plus two times the radius of a robot. Once robot r_i has reached p_i, the leader moves horizontally right to $(x(r_{i+1}) - 1, y(p_i) - 2)$, then vertically down to $(x(r_{i+1}) - 1, y(r_{i+1}))$ (the position to the left of robot r_{i+1}).

The above process is repeated until a robot is placed at p_k or the final robot r_{n-1} is positioned at p_{k-1} when $k = n$. To create a clean pattern, without the leader remaining in it, the leader moves horizontally right to x-coordinate $x_{min} - 90$. If $k < n$, the phase now ends. Otherwise $k = n$ and the leader moves vertically to $(x_{min} - 90, y(p_n))$, and finally to p_n at $(x(p_n), y(p_n))$.

5.3.2 Analysis

We now argue the correctness of the Pattern Formation phase.

LEMMA 5.3.1. *In every round of the Pattern Formation phase, only one non-leader robot moves from the convex hull to be positioned in the target pattern avoiding collisions.*

PROOF. In the Pattern Formation phase, the leader moves to the left of the leftmost topmost robot r_i a distance one outside the convex hull. Robot r_i observes this and follows the leader to its final position. Once the leader moves away from r_i, r_i stops at its final position. Since the leader moves only one robot at a time and a robot moves only once activated by the leader, only one non-leader robot moves in each round. Furthermore, due to the order in which the robots are moved (left to right) and the first move being horizontally to the left, no collisions can occur. □

Next, we analyze the time complexity of the Pattern Formation phase.

LEMMA 5.3.2. *The Pattern Formation phase takes $O(n)$ rounds.*

PROOF. By Lemma 5.3.1, the leader activates and moves each robot from the convex hull to the target pattern one robot at a time. The leader moves next to a robot in a constant number of rounds and it moves a robot to its final position in three moves, so in six rounds. Therefore, one robot moves from the convex hull to the target pattern in a constant number of rounds. As a result, the total number of rounds required to move all robots from the convex hull to the target pattern is $O(n)$. □

Next, we prove the memory space that our algorithm needs in total for all phases.

LEMMA 5.3.3. *The Pattern Formation phase uses $O(1)$ memory.*

PROOF. During the Pattern Formation phase, the leader stores the first and last locations of the pattern after scaling and the scaling factor. As

these are simply locations in the plane, they can be stored using $O(\log n)$ bits. Every other robot stores the direction and distance while moving to its final location in the target pattern. As these can be stored as locations of the leader, this also requires $O(\log n)$ bits. Hence, $O(1)$ memory suffices for all robots. □

Lemmas 5.3.1, 5.3.2, and 5.3.3 together imply the following result.

THEOREM 5.3.4. *The Pattern Formation phase finishes in $O(n)$ rounds without collisions in the fully synchronous setting using $O(1)$ memory.*

Theorems 5.1.6, 5.2.3, and 5.3.4 now imply our final result.

THEOREM 5.3.5. *Our algorithm solves the Pattern Formation problem for n unit disk robots in $O(n) + O(q \log n)$ rounds with probability at least $1 - n^{-q}$ without collisions in the fully synchronous model using $O(1)$ memory space.*

CHAPTER 6

Shortest Paths of Mutually Visible Robots

In this chapter, we consider the problem of maintaining mutual visibility between point robots in a simple polygon while they move from a starting line segment S to a target line segment T.

6.1 Model and Preliminaries

Consider a set of $n \geq 1$ point robots $R = \{r_1, r_2, \ldots, r_n\}$ operating in a simple polygon P. The robots need to move along their shortest paths from their starting positions s_i to their targets t_i, where s_i and t_i are known by each robot r_i, for all $1 \leq i \leq n$. Two robots are mutually visible if the line segment connecting them does not intersect any obstacle (i.e., any edge of the simple polygon). In other words, their visibility is obstructed, where a robot cannot see another robot, if the straight line segment between the robots intersects P. We aim to find a sequence of (possibly synchronized) robot movements along the shortest path between s_i and t_i such that at all times every pair of robots is mutually visible.

We consider two different cases: (a) the start and target line segments do not intersect (Section 6.2), and (b) the start and target line segments intersect (Section 6.3).

6.2 Non-Crossing Start and Target Lines

In this section, we present a centralized algorithm that solves the SPMV problem when the start and target line segments, S and T, do not intersect.

Let π_i be the shortest path within P from the start point s_i on S of robot r_i to its target point t_i on T, for $1 \leq i \leq n$. Let Q be the minimal, possibly degenerate polygon within P that includes the start line segment S, the target line segment T, and the set of π_i's, $1 \leq i \leq n$, as shown in Figure 6.2.

If all paths π_i, $1 \leq i \leq n$, share a single polygon vertex, we refer to this vertex as their intersection point, as shown in Figure 6.1(a). If they share multiple vertices, then we say that the first of these vertices along the paths is the intersection point, see Figure 6.1(b).

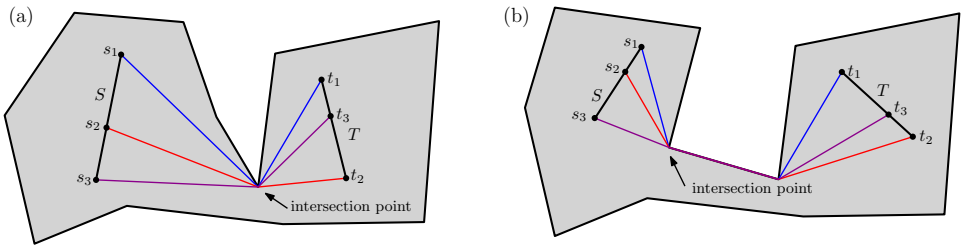

FIGURE 6.1: Illustrating the definition of an intersection point.

The following observation is immediate.

OBSERVATION 6.2.1. *If the shortest paths have no intersection point then Q consists of S, T and two concave paths connecting the end points of S and T, see Figure 6.2(a). If the paths π_i, $1 \leq i \leq n$, have an intersection point then Q is a degenerate polygon consisting of two funnels; one containing S and one containing T, and the two funnels are connected by either a joint intersection point or a single path, as shown in Figure 6.2(b).*

To simplify the description of the algorithm, rotate Q so that S is entirely to the left of T and neither S nor T are horizontal. Furthermore, assume

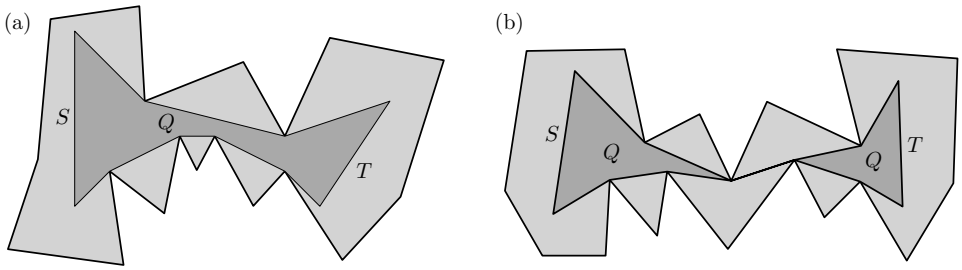

FIGURE 6.2: Illustrating Observation 6.2.1. (a) Q consists of S, T and two concave chains connecting S and T. (b) Q is a degenerate polygon consisting of two funnels; one containing S and one containing T, and the two funnels are connected by either a joint intersection point or a single polygonal path.

without loss of generality that $s_1, \ldots, s_n$ are ordered from top-to-bottom along S.

Let $U(Q) = \langle s_1 = u_1, u_2, \ldots, u_k \rangle$ denote the top boundary of Q connecting s_1 with the top-most endpoint of T and let $V(Q) = \langle s_n = v_1, v_2, \ldots, v_\ell \rangle$ denote the bottom boundary of Q connecting s_n with the bottom-most endpoint of T. Note that if Q is a degenerate polygon with an intersection point then at least one vertex of Q will be in both $U(Q)$ and $V(Q)$.

Our algorithm starts by computing a triangulation $\mathcal{T}$ of Q in linear time [20]. Note that since Q might contain an intersection point, a triangle in $\mathcal{T}$ might degenerate to a single segment.

All robots are initially positioned on S. Let τ be the unique triangle in $\mathcal{T}$ with the segment $S = (s_1, s_n)$ as one of its sides and u_2 and/or v_2 as its third vertex, according to Observation 6.2.1. Assume without loss of generality that u_2 is the third vertex of τ.

With the above notation, we next describe the algorithm.

Consider the initial configuration when all the robots are positioned on S. If $u_2 = v_2$ then move every robot r_i along π_i to u_2. Otherwise, move every robot r_i along π_i from the segment (s_1, s_n) to the segment (s_n, u_2). More

formally, we move robot r_i to the intersection of (s_n, u_2) and π_i. We note that since u_2 lies on $U(Q)$ and s_n lies on $V(Q)$, every π_i intersects (s_n, u_2).

In a generic step of the algorithm, the robots are positioned on a (possibly degenerate) segment (u_a, v_b) of $\mathcal{T}$. The algorithm considers four cases.

(a): If $u_a = v_b$ and $u_{a+1} = v_{b+1}$ then every robot r_i moves from u_a to u_{a+1}.

(b): If $u_a = v_b$ and $u_{a+1} \neq v_{b+1}$ then every robot r_i moves along π_i from u_a to the segment (u_{a+1}, v_{b+1}).

(c): If $u_a \neq v_b$ and $u_{a+1} = v_{b+1}$ then every robot r_i moves along π_i from (u_a, v_b) to u_{a+1}.

(d): Otherwise, if $u_a \neq v_b$ and $u_{a+1} \neq v_{b+1}$ then let τ be the unique triangle in $\mathcal{T}$ with (u_a, v_b) as one of its sides and either u_{a+1} or v_{b+1} as its third vertex. Assume without loss of generality that u_{a+1} is the third vertex of τ. Move every robot r_i along π_i from segment (u_a, v_b) to segment (u_{a+1}, v_b).

This iterative process continues until all robots have reached T.

The algorithm runs in $O(mn)$ time since the triangulation can be computed in linear time, and the shortest paths computation requires $O(mn)$ time, where m is the complexity of the polygon, and n is the number of robots.

It remains only to prove that the robots are mutually visible to each other during their movement from S to T.

LEMMA 6.2.2. *The algorithm described above guarantees that every robot r_i moves from s_i to t_i along π_i while being visible to every other robot.*

PROOF. Within each (possibly degenerate) triangle τ of $\mathcal{T}$, the robots move along their shortest paths from one segment (or point) to another segment (or point) of τ. Since a triangle is convex, all robots are mutually

visible within τ. Therefore, what is left to prove is that all the shortest paths visit the same set of triangles in the exact same order. From Observation 6.2.1, it follows that every non-degenerate triangle in $\mathcal{T}$ has at least one vertex from $V(Q) \setminus U(Q)$ and one vertex from $U(Q) \setminus V(Q)$. That implies that (1) any path from a point on S to a point on T must intersect every triangle in $\mathcal{T}$, and (2) every robot r_i following π_i from s_i to t_i must visit the triangles in the same order. Consequently, all the robots must visit all the triangles in the same order, and as a result, every r_i is visible to every other robot during the movement along their shortest paths. □

To summarize this section, we obtain the following theorem.

THEOREM 6.2.3. *In the case when the starting positions and the target positions lie on two non-intersecting segments, the* SPMV *problem can be solved in* $O(nm)$ *time.*

6.3 Crossing Start and Target Lines

At first sight one might think that the case when S and T intersect is not much harder than the non-intersecting case. However, we will argue that it is. Consider the instance shown in Figure 6.3(a). In Lemma 6.3.1, we show that any algorithm that does not move the robots along a rotating line segment around the intersection point q between S and T cannot terminate in polynomial time with respect to the input size (assuming v_2 and v_5 are very close to q). However, any algorithm that keeps the robots on a line segment pivoting around q can also get stuck (Lemma 6.3.2). This implies that any algorithm for the case when S and T intersect cannot use a single strategy algorithm, i.e., it will have to analyse the input instance and depending on the instance select an appropriate approach.

We will need the following notations before formalising the above observations.

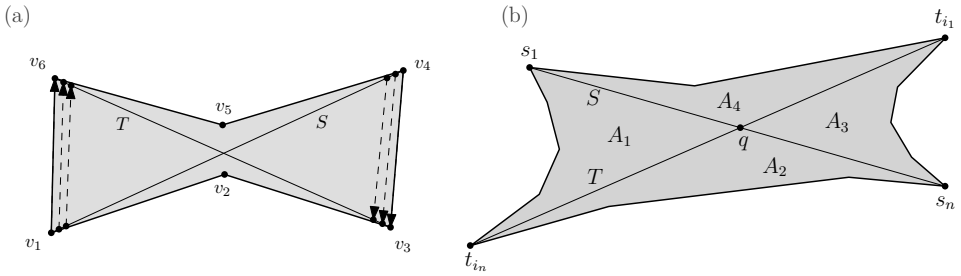

FIGURE 6.3: (a) An instance showing that the number of steps of any algorithm that does not move all the robots at the same time cannot be bounded by n and m. (b) An example illustrating the concave paths together with the regions A_1, A_2, A_3, and A_4.

Let q be the intersection of the start and target line segments S and T, as illustrated in Figure 6.3(b). Define Q as in the previous section. Furthermore, assume that $s_1, \ldots, s_n$ are ordered from top-to-bottom along S, and let $t_{i_1}, \ldots, t_{i_n}$ denote the target points along T ordered from top-to-bottom.

Let A_1 be the simple polygon bounded by the two segments (s_1, q) and (t_{i_n}, q), and the concave chain corresponding to the shortest path in Q from s_1 to t_{i_n}. Symmetrically, we define A_2, A_3, and A_4, see Figure 6.3(b).

Note that a robot r_i moving from $s_i \in S$ to $t_i \in T$ will stay within one of the regions A_1, A_2, A_3, or A_4 during their movement. Hence, we can partition the robots into four sets $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, and $\mathcal{R}_4$ such that $\mathcal{R}_j$, $1 \leq j \leq 4$, contains the robots in $\mathcal{R}$ moving through A_j. If a robot r_i only moves along the boundary of two regions (not in their interior), then r_i can be assigned to either of the two sets.

Next, we argue that any algorithm for the case when S and T intersect that does not move all robots at the same time requires an unbounded number of steps.

LEMMA 6.3.1. *There exist instances where any algorithm that does not move all robots of $\mathcal{R}_1$ and $\mathcal{R}_3$ at the same time requires a number of steps that is unbounded in n and m.*

PROOF. Consider the instance shown in Figure 6.3(a). It consists of six vertices $v_1, \ldots, v_6$, where v_1, v_3, v_4 and v_6 are at the endpoints of S and T while v_2 and v_5 are arbitrarily close to q.

Let all robots in $\mathcal{R}_1$ start arbitrarily close to v_1 and move to a position arbitrarily close to v_6, and let all robots in $\mathcal{R}_3$ start arbitrarily close to v_4 and move arbitrarily close to v_3.

We observe that for the robots in $\mathcal{R}_1$ and $\mathcal{R}_3$ to be visible to each other, they all need to be contained in a narrow strip defined by two almost[1] parallel line segments through v_2 and v_5.

Hence, in each step where not all robots are moved, the robots can move at most twice the width of this narrow strip. Since this width is determined by the distance of v_2 and v_5 to q, this can be arbitrarily small, leading to a number of steps not related to n and m. □

Next, we prove that any algorithm that moves the robots along a rotating line segment around q can also get stuck.

LEMMA 6.3.2. *If S and T intersect, then any algorithm that keeps the robots in $\mathcal{R}_1$ and $\mathcal{R}_3$ on a line segment through q or that keeps $\mathcal{R}_2$ and $\mathcal{R}_4$ on a line segment through q can get stuck and be unable to proceed.*

PROOF. We first show that any algorithm that keeps the robots in $\mathcal{R}_1$ and $\mathcal{R}_3$ on a line segment can get stuck, before extending this to also include $\mathcal{R}_2$ and $\mathcal{R}_4$.

[1] As v_2 and v_5 approach q, these line segments become more and more parallel.

Let a and b denote the top and bottom endpoints of S, respectively, and let c and d denote the top and bottom endpoints of T, respectively. We have four robots: r_1 starting at a and moving to d, r_2 starting at a and moving to c, r_3 starting at b and moving to c, and r_4 starting at b and moving to d.

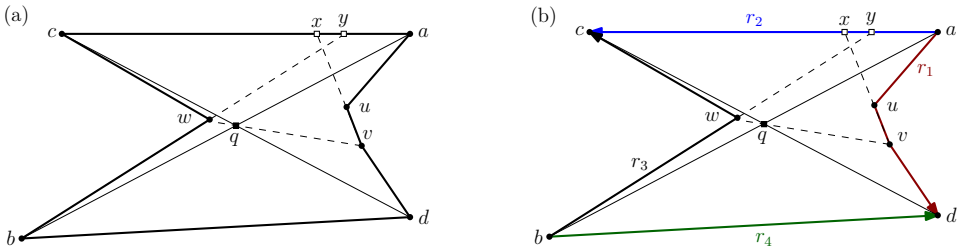

FIGURE 6.4: (a) An input instance showing that any algorithm that keeps the robots on a line segment through q can reach a situation where it is unable to proceed. (b) Illustrating the shortest paths for the four robots.

Consider the polygon in Figure 6.4(a) where a, b, c, and d are vertices of P, (a, c) and (b, d) are edges of the polygon, a and d are connected by a concave chain $\langle a, u, v, d\rangle$, and b and c are connected by a concave chain $\langle b, w, c\rangle$.

In order to ensure that r_1 and r_2 remain visible, r_1 cannot move past v before r_2 is on or past the projection x of $\overline{uv}$ onto (a, c). This also means that since r_1 and r_3 lie on a line segment through q, r_3 cannot reach w before r_1 passes v. However, to maintain visibility between r_2 and r_3, r_2 needs to be on or before the projection y of $\overline{bw}$ onto (a, c).

We now observe that since y occurs before x along (a, c), to maintain visibility, r_2 needs to be both on or before y and on or after x, meaning that there is no position along (a, c) that maintains visibility of r_2 with both r_1 and r_3 if we require r_1 and r_3 to be on a line segment through q.

However, in the above example, keeping r_2 and r_4 on a line segment through q would still work, so we now extend the above construction to create a

similar situation for that line segment while maintaining the one for r_1 and r_3. The construction is shown in Figure 6.5(a).

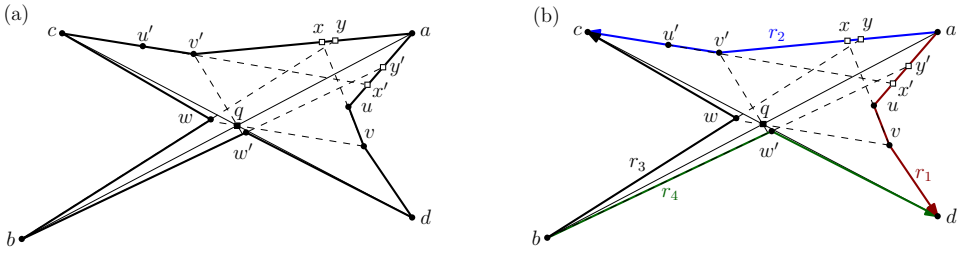

FIGURE 6.5: (a) An input instance showing that any algorithm for the crossing case that keeps the robots on a line segment through q can reach a situation where it is unable to proceed. (b) Illustrating the shortest paths for the four robots.

Compared to the construction in Figure 6.4, we replace the edge (a, c) with the concave chain $\langle a, v', u', c\rangle$, and the edge (b, d) with the concave chain $\langle b, w', d\rangle$. Using an analogous argument to the previous one, y still comes before x along the path from a to c, and using u', v' and w', we can construct x' and y' analogous to their counterparts to ensure that y' comes before x' along the path from a to d. Thus, neither the line segment connecting r_1 and r_3 through q, nor the line segment connecting r_2 and r_4 through q can move from v or v', respectively.

Since we assumed nothing of the algorithm other than that it keeps either S_1 and S_3 or S_2 and S_4 on a line segment through q, we can conclude that any such algorithm will reach a situation in which it is unable to proceed. □

The above results indicate that any polynomial-time algorithm for the SPMV problem cannot rely on a single strategy. Instead, it must take the input instance into account when choosing which strategy to choose.

The SPMV problem requires that all robots move along their shortest path and that the robots are visible to each other at all time. We note that if we remove either condition then polynomial-time algorithms are trivial.

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

We studied a number of problems related to robotics and computational geometry.

We studied the Mutual Visibility problem for a system of autonomous fat robots of unit disk size in the robots with lights model. We described an algorithm for this problem that uses two colors and works for fully synchronous computation of fat robots under rigid movements. Our solution is optimal with respect to the number of colors used since even for point robots at least two colors are required [49]. Also, our algorithm solves the Mutual Visibility problem in $O(n)$ rounds.

We also studied the Pattern Formation problem for unit disk robots in the robots with lights model under obstructed visibility. We described two algorithms for this problem, depending on the assumptions made to solve this problem. If the target pattern is allowed to be scaled, our initial algorithm used 10 colors, which we subsequently improved to 7 colors. If scaling is not allowed, our algorithm needs one additional color: initially 11, which we then improved to 8. Our algorithms run in $O(n) + O(q \log n)$ rounds, where $q > 0$ is a parameter of the Leader Election phase, which takes $O(q \log n)$ rounds with probability at least $1 - n^{-q}$. Interestingly, unlike previous work, our algorithms do not require any additional assumptions

on the capabilities of the robots or any shared information or coordinate system.

Furthermore, we studied the Pattern Formation problem for a system of n autonomous fat robots of unit disk size in the classical model where all robots have a small persistent memory. We described an algorithm that solves the Pattern Formation problem and works under the fully synchronous model and under obstructed visibility. Our algorithm solves the Pattern Formation problem in $O(n) + O(q \log n)$ rounds, where $q > 0$ is related to Leader Election, which takes $O(q \log n)$ rounds with probability at least $1 - n^{-q}$. The Pattern Formation problem uses $O(1)$ memory space. As a by-product, we also solve the Mutual Visibility problem in this memory model.

Finally, we studied the SPMV problem for n point robots. We gave an efficient $O(nm)$ time algorithm for the case where the start and target line segments do not intersect. We also argued that any polynomial-time algorithm for the case where these line segments do intersect has to take the input polygon into consideration to determine the strategy to move the robots. In particular, we showed that no single strategy will give a polynomial-time algorithm for all simple polygons.

7.2 Future Work

There are a number of interesting directions which we could consider for future work.

First of all, it is interesting to extend our Mutual Visibility algorithm for non-rigid movements of robots and also for semi-synchronous and asynchronous computations.

Secondly, it is interesting to extend the results for Pattern Formation with lights in several aspects. For example, we have no lower bounds indicating that the number of colors we use is optimal and thus the natural open problems are both trying to improve the number of colors or showing that this is not possible using a lower bound.

Additionally, it is interesting to extend our Pattern Formation algorithms in both models to the semi-synchronous and asynchronous settings. Furthermore, it is interesting to extend our algorithms to solve the Pattern Formation problem in $O(n)$ time complexity, which requires removing the Leader Election phase.

Finally, there are still many open problems related to the SPMV problem. Most importantly, is there a polynomial-time algorithm for the general case, or even just for the case where S and T intersect? If there is not, can one prove any lower bounds?

Bibliography

[1] Chrysovalandis Agathangelou, Chryssis Georgiou, and Marios Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 250–259, 2013.

[2] Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 56:836–859, 2016.

[3] Oswin Aichholzer, Thomas Hackl, Matias Korman, Alexander Pilz, and Birgit Vogtenhuber. Geodesic-preserving polygon simplification. *International Journal of Computational Geometry & Applications*, 24(04):307–323, 2014.

[4] Rusul J. Alsaedi, Joachim Gudmundsson, and André van Renssen. The mutual visibility problem for fat robots. In *Proceedings of the 18th Algorithms and Data Structures Symposium (WADS 2023)*, volume 14079 of *Lecture Notes in Computer Science*, pages 15–28, 2023.

[5] Boris Aronov, Matias Korman, Simon Pratt, André van Renssen, and Marcel Roeloffzen. Time-space trade-off algorithms for triangulating a simple polygon. *Journal of Computational Geometry*, 8(1):105–124, 2017.

[6] Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry: Theory and Applications*, 46(8):959–969, 2013.

[7] Hagit Attiya and Jennifer Welch. *Distributed computing: Fundamentals, simulations, and advanced topics*. John Wiley & Sons, 2004.

[8] Sang Won Bae, Matias Korman, and Yoshio Okamoto. The geodesic diameter of polygonal domains. *Discrete & Computational Geometry*, 50:306–329, 2013.

[9] Sang Won Bae, Matias Korman, and Yoshio Okamoto. Computing the geodesic centers of a polygonal domain. *Computational Geometry: Theory and Applications*, 77:3–9, 2019.

[10] Sang Won Bae, Matias Korman, Yoshio Okamoto, and Haitao Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. *Computational Geometry: Theory and Applications*, 48(6):495–505, 2015.

[11] Luis Barba, Matias Korman, Stefan Langerman, Kunihiko Sadakane, and Rodrigo I. Silveira. Space–time trade-offs for stack-based algorithms. *Algorithmica*, 72:1097–1129, 2015.

[12] Mordechai Ben-Ari and Francesco Mondada. Robots and their applications. In *Elements of Robotics*, pages 1–20. Springer, 2018.

[13] Kálmán Bolla, Tamás Kovacs, and Gábor Fazekas. Gathering of fat robots with limited visibility and without global navigation. In *International Conference on Swarm and Evolutionary Computation (SIDE)*, volume 7269 of *Lecture Notes in Computer Science*, pages 30–38, 2012.

[14] Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, Archak Das, and Buddhadeb Sau. Distributed pattern formation by autonomous robot swarm. In *23rd International Conference on Distributed Computing and Networking (ICDCN)*, pages 242–243, 2022.

[15] Kaustav Bose, Ranendu Adhikary, Manash Kumar Kundu, and Buddhadeb Sau. Arbitrary pattern formation by opaque fat robots with lights. In *Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, volume 12016 of *Lecture Notes in Computer Science*, pages 347–359, 2020.

[16] Kaustav Bose, Archak Das, and Buddhadeb Sau. Pattern formation by robots with inaccurate movements. In *25th International Conference on Principles of Distributed Systems (OPODIS)*, volume 217 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:20, 2021.

[17] Kaustav Bose, Manash Kumar Kundu, Ranendu Adhikary, and Buddhadeb Sau. Arbitrary pattern formation by asynchronous opaque robots with lights. *Theoretical Computer Science*, 849:138–158, 2021.

[18] Sruti Gan Chaudhuri, Swapnil Ghike, Shrainik Jain, and Krishnendu Mukhopadhyaya. Pattern formation for asynchronous robots without agreement in chirality. *arXiv preprint arXiv:1403.2625*, 2014.

[19] Sruti Gan Chaudhuri and Krishnendu Mukhopadhyaya. Leader election and gathering for asynchronous fat robots without common chirality. *Journal of Discrete Algorithms*, 33:171–192, 2015.

[20] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6:485–524, 1991.

[21] Serafino Cicerone, Gabriele Di Stefano, and Alfredo Navarra. Solving the pattern formation by mobile robots with chirality. *IEEE Access*, 9:88177–88204, 2021.

[22] Mark Cieliebak, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Distributed computing by mobile robots: Gathering. *SIAM Journal on Computing*, 41(4):829–879, 2012.

[23] Reuven Cohen and David Peleg. Local spreading algorithms for autonomous robot systems. *Theoretical Computer Science*, 399(1-2):71–82, 2008.

[24] Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer auf der Heide, Christoph Raupach, Kamil Swierkot, Daniel Warner, Christoph Weddemann, and Daniel Wonisch. Collisionless gathering of robots with an extent. In *37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 6543 of *Lecture Notes in Computer Science*, pages 178–189, 2011.

[25] Jurek Czyzowicz, Leszek Gasieniec, and Andrzej Pelc. Gathering few fat mobile robots in the plane. *Theoretical Computer Science*, 410(6-7):481–499, 2009.

[26] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609:171–184, 2016.

[27] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Spinger, 2008.

[28] Giuseppe Antonio Di Luna, Paola Flocchini, S Gan Chaudhuri, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. Mutual visibility by luminous robots without collisions. *Information and Computation*, 254:392–418, 2017.

[29] Ayan Dutta, Sruti Gan Chaudhuri, Suparno Datta, and Krishnendu Mukhopadhyaya. Circle formation by asynchronous fat robots with limited visibility. In *Distributed Computing and Internet Technology (ICDCIT)*, volume 7154 of *Lecture Notes in Computer Science*, pages 83–93, 2012.

[30] Joel Fenwick and Vladimir Estivill-Castro. Optimal paths for mutually visible agents. In *Proceedings of the 16th International Symposium on Algorithms and Computation (ISAAC)*, volume 3827 of *Lecture Notes in Computer Science*, pages 869–881, 2005.

[31] Joel Fenwick and Vladimir Estivill-Castro. Mutually visible agents in a discrete environment. In *Proceedings of the 30th Australasian Conference on Computer Science (ACSC)*, volume 62, pages 141–150, 2007.

[32] Paola Flocchini. Computations by luminous robots. In *14th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW)*, volume 9143 of *Lecture Notes in Computer Science*, pages 238–252, 2015.

[33] Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. *Distributed Computing by Oblivious Mobile Robots*. Synthesis Lectures on Distributed Computing Theory (SLDCT). Springer Cham, 2012.

[34] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1-3):412–447, 2008.

[35] D Floreano, J Godjevac, A Martinoli, F Mondada, and JD Nicoud. Design, control and applications of autonomous mobile robots. *Microprocessor Based and Intelligent Systems Engineering*, 18:159–186, 1999.

[36] Nao Fujinaga, Yukiko Yamauchi, Hirotaka Ono, Shuji Kijima, and Masafumi Yamashita. Pattern formation by oblivious asynchronous mobile robots. *SIAM Journal on Computing*, 44(3):740–785, 2015.

[37] Satakshi Ghosh, Pritam Goswami, Avisek Sharma, and Buddhadeb Sau. Move and time optimal arbitrary pattern formation by asynchronous robots on infinite grid. *International Journal of Parallel, Emergent and Distributed Systems*, 38(1), 2023.

[38] Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

[39] Manash Kumar Kundu, Pritam Goswami, Satakshi Ghosh, and Buddhadeb Sau. Arbitrary pattern formation by opaque fat robots on infinite grid. *International Journal of Parallel, Emergent and Distributed Systems*, 37(5):542–570, 2022.

[40] Giuseppe Antonio Di Luna, Paola Flocchini, Sruti Gan Chaudhuri, Nicola Santoro, and Giovanni Viglietta. Robots with lights: Overcoming obstructed visibility without colliding. In *16th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 8756 of *Lecture Notes in Computer Science*, pages 150–164, 2014.

[41] Giuseppe Antonio Di Luna, Paola Flocchini, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. The mutual visibility problem for oblivious robots. In *26th Canadian Conference on Computational Geometry (CCCG)*, 2014.

[42] Ulrich Nehmzow. Mobile robotics: Research, applications and challenges. *Proceeding of Future Trends in Robotics*, 2001.

[43] Debasish Pattanayak, Klaus-Tycho Foerster, Partha Sarathi Mandal, and Stefan Schmid. Conic formation in presence of faulty robots. In *16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, volume 12503 of *Lecture Notes in Computer Science*, pages 170–185, 2020.

[44] David Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In *7th International Workshop on Distributed Computing (IWDC)*, volume 3741 of *Lecture Notes in Computer Science*, pages 1–12, 2005.

[45] Pavan Poudel, Gokarna Sharma, and Aisha Aljohani. Sublinear-time mutual visibility for fat oblivious robots. In *Proceedings of the 20th International Conference on Distributed Computing and Networking (ICDCN)*, pages 238–247, 2019.

[46] Jörg-Rüdiger Sack and Jorge Urrutia. *Handbook of Computational Geometry*. Elsevier, 1999.

[47] Gokarna Sharma, Rusul Alsaedi, Costas Busch, and Supratik Mukhopadhyay. The complete visibility problem for fat robots with lights. In *Proceedings of the 19th International Conference on Distributed Computing and Networking (ICDCN)*, pages 1–4, 2018.

[48] Gokarna Sharma, Costas Busch, and Supratik Mukhopadhyay. Bounds on mutual visibility algorithms. In *27th Canadian Conference on Computational Geometry (CCCG)*, pages 268–274, 2015.

[49] Gokarna Sharma, Costas Busch, and Supratik Mukhopadhyay. Mutual visibility with an optimal number of colors. In *11th International Symposium on Algorithms and Experiments for Wireless Sensor Networks (ALGOSENSORS)*, volume 9536 of *Lecture Notes in Computer Science*, pages 196–210, 2015.

[50] Gokarna Sharma, Costas Busch, and Supratik Mukhopadhyay. How to make fat autonomous robots see all others fast? In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3730–3735, 2018.

[51] Gokarna Sharma, Costas Busch, Supratik Mukhopadhyay, and Charles Malveaux. Tight analysis of a collisionless robot gathering algorithm. *ACM Transactions on Autonomous and Adaptive Systems*, 12(1):1–20, 2017.

[52] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L Trahan. Constant-time complete visibility for asynchronous robots with lights. In *Proceedings of the 19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 265–281, 2017.

[53] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. MIT press, 2011.

[54] Csaba D Toth, Joseph O'Rourke, and Jacob E Goodman. *Handbook of Discrete and Computational Geometry*. CRC press, 2017.

[55] Ramachandran Vaidyanathan, Costas Busch, Jerry L. Trahan, Gokarna Sharma, and Suresh Rai. Logarithmic-time complete visibility for robots with lights. In *29th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 375–384, 2015.

[56] Ramachandran Vaidyanathan, Gokarna Sharma, and Jerry Trahan. On fast pattern formation by autonomous robots. *Information and Computation*, 285:104699, 2022.