Distributed Algorithms to Solve MIS Filling, Mutual Visibility and Uniform Partitioning with Luminous Mobile Robots

Thesis submitted in partial fulfilment of

the requirements for the degree of

Doctor of Philosophy

by

Subhajit Pramanick

(Roll No. 186123017)

under the supervision of

Prof. Partha Sarathi Mandal

to the

Department of Mathematics

Indian Institute of Technology Guwahati

Guwahati - 781039, India

May 2024

CERTIFICATE

This is to certify that this thesis entitled "**Distributed Algorithms to Solve MIS Filling, Mutual Visibility and Uniform Partitioning with Luminous Mobile Robots**" being submitted by Mr. Subhajit Pramanick to the Department of Mathematics, Indian Institute of Technology Guwahati, is a record of bonafide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

The results contained in this thesis have not been submitted in part or full to any other university for the award of any degree or diploma.

IIT Guwahati
May 2024

(**Prof. Partha Sarathi Mandal**)
Department of Mathematics
Indian Institute of Technology Guwahati
Guwahati - 781039, Assam, India

ACKNOWLEDGEMENTS

I am deeply grateful to all those who have contributed to the completion of this study.

First and foremost, I express my heartfelt gratitude to my supervisor, Prof. Partha Sarathi Mandal, for his invaluable guidance and unwavering support throughout my PhD journey. His mentorship, encouragement, and insightful feedback have been instrumental in shaping this thesis.

I extend my sincere appreciation to my co-authors, Dr. Debasish Pattanayak, Saswata Jana, and Adri Bhattacharya, for their enthusiastic collaboration and constructive discussions. Dr. Debasish Pattanayak's mentorship at the onset of my research journey has been invaluable, laying a solid foundation for my work. Saswata and Adri's dedication and expertise have enriched our discussions, offering fresh perspectives and innovative solutions to research challenges.

The guidance and feedback from the doctoral committee members, Prof Diganta Goswami, Prof Ashok Singh Sairam, and Prof Sushanta Karmakar, have been invaluable. Their insights have enriched this thesis and deepened my understanding of the subject. The role of the Indian Institute of Technology Guwahati has been pivotal in facilitating the research work. I want to thank the Ministry of Human Resource and Development (MHRD), Government of India, for the financial assistance provided. I sincerely thank all the staff of the Department of Mathematics, who have helped me with the administrative procedures and technical assistance.

My connection with the Department of Mathematics began during my M.Sc. days, and its faculty members, including Prof Gautam Kumar Das, Prof Kalpesh Kapoor, Prof Rajen Sinha, Dr Vinay Wagh, Dr Anjan Chakrabarty, and Dr Subhamay Saha, have provided a sense of belonging as I transitioned into the PhD program.

I extend my heartfelt gratitude to Debanjali for her indispensable support and understanding, which have been pillars of strength during challenging times. Additionally, I am immensely thankful to all my colleagues and friends, especially Anwita, Samadrita, Suvajit, Subhendu, and Malay, for their unwavering support. I also appreciate the help and sup-

port extended by Dr Dibakar Saha. I cannot be thankful enough for the care and concern shown by Dr. Debarati Mitra.

I feel blessed to be in the company of some wonderful teachers since my school days. Without their guidance and belief, I would not be here.

My parents have always supported me unconditionally throughout my life and sacrificed their comfort to accommodate my academic pursuits. Their role in my journey cannot be expressed in words. The blessings from my family, especially my uncle and aunt, have been a constant source of inspiration. I can never thank them enough.

Finally, I extend my gratitude to all my well-wishers for their support and encouragement throughout this journey.

IIT Guwahati (Subhajit Pramanick)

ABSTRACT

The coordination among a large number of autonomous mobile robots, also known as swarm robots, has gained significant interest in recent years. Under the framework of *Look-Compute-Move* cycles, the robots can perform various tasks such as exploration, gathering, pattern formation, dispersion, scattering and others. This thesis mainly deals with three different types of problems using a swarm of autonomous, anonymous and luminous mobile robots. The first problem falls into the class of graph exploration problems where we study the graph filling problem, named *MIS Filling Problem*. For an arbitrary connected graph with specific vertices, called *Doors*, the problem aims to fill the vertices using robots with limited visibility, entering the graph through the Door, such that the set of the occupied vertices forms a maximal independent set (MIS) of the graph. We propose algorithms for two versions of the problem: (1) Single Door under the asynchronous setting and (2) Multiple Doors under the semi-synchronous setting. We also discuss the lower bound for the problem for the single Door case.

The second problem in this thesis is one of the fundamental problems in swarm robotics is the *Mutual Visibility Problem*. This problem is widely studied under various constraints, such as axes agreement, fault tolerance, adversarial schedulers, etc. Opaqueness is one of the most interesting components in the model, which allows robots to obstruct the line of sight of other robots. The aim of the problem is to achieve a configuration of the robots such that any two robots are visible to each other. We introduce a new flavour to the model where we consider that the robots may exhibit inaccurate movement and have an agreement on one coordinate axis. Inaccuracy is defined as an angular deviation from the robot's target point. More specifically, if the robot r chooses t_r as its target point for the movement, inaccuracy makes the robot, move to a point t'_r such that the distance between r and t_r is the same as the distance between r and t'_r. We propose an algorithm under the semi-synchronous setting, which is optimal with respect to the number of colors. We extend the algorithm for the asynchronous setting using 3 colors. We also examine the problem from the aspect of fault tolerance under the asynchronous setting in addi-

tion to the inaccuracy in the robots' movement. All the algorithms under this model take $O(N)$ epochs to achieve the goal, where N is the number of robots. We further discuss the impossibility of the problem for the angular inaccuracy to be greater than 90°.

Another version of the model primarily investigates the mutual visibility problem from the perspective of the mobility failure of the robots with no agreement on the coordinate axes. No axes agreement adds up more challenges in tolerating mobility failure in the context of mutual visibility. We show that the problem becomes unsolvable for a specific initial configuration of the robots under this model where the initial configuration is symmetric. We present an algorithm that tolerates any number of faulty robots when the robots are operating under the fully-synchronous setting. The algorithm runs in $O(N^2)$ rounds. We propose another algorithm in the presence of a single faulty robot that requires only 2 colors in fully-synchronous and 5 colors in the asynchronous setting.

The third type of problem addresses the *Uniform Partitioning* for a given bounded region using a set of N of oblivious mobile robots. We initiate the study of partitioning a region, taking inspiration from our daily life, where our natural tendency is to divide work equally among ourselves to achieve faster results. The problem requires the robots to partition the whole region into sub-regions of equal area, each of which contains exactly one robot. Due to application-oriented motivation, we consider the region to be well-known geometric shapes such as rectangle, square and circle. To the best of our knowledge, this is the first attempt to study the Uniform Partitioning problem using oblivious opaque robots working under asynchronous settings. We propose three algorithms considering three different regions. The algorithms proposed for rectangular and square regions run in $O(N)$ epochs whereas the algorithm for circular regions runs in $O(N^2)$ epochs. The algorithms for the rectangular, square and circular regions require 2 (which is optimal), 5 and 8 colors, respectively.

Contents

Chapter 1

Introduction

1.1 Motivation

Distributed algorithms and their applications have permeated various facets of modern life, ranging from our everyday interactions on social media platforms to critical domains such as military operations and household chores, as well as advancements in nanotechnology. From single-processor systems to multi-processor systems, the journey was not very long, and within this progression, the concept of robots was revolutionary. Developers are increasingly drawing inspiration from insects and animals, striving to create robots that are not only compact but also cost-effective. Instead of a complex big-size robot, a multi-robot system can perform efficiently. Different types of cost-effective robots, like mobile agents, luminous robots, and Unmanned Aerial Vehicles (UAVs), are used in different scenarios. They can collaboratively achieve some tasks like rescue operations, pattern formation, gathering, building a structure, etc. What was once theoretical is now manifest in practical applications. Autonomous programmable robot swarms like Kilobots [62] and RoboBees [19] and innovations such as robotic valet parking [45] and dragon-inspired autonomous drones [75] are a reality now.

A group of identical robots is called a swarm of robots, popularly known as swarm robots [33]. Swarm robots are easy to manufacture, cost-efficient and have similar capabilities. Sometimes, these robots do not even need to be recovered after deployment

to achieve some goal. Scalability is one of the primary reasons behind their popularity, which makes the swarm robots resilient to faults to some extent. The motive is to design distributed algorithms to solve a specific problem to ensure decentralization so that each robot in the swarm can perform its task independently.

This thesis focuses on the LCM robot model, a widely studied swarm robot model that has garnered attention from researchers over several decades. This model aims to strike a balance between the capabilities of the robots and the solvability of specific problems. The following section explains the characteristics of the model in detail, which are common for all the subsequent chapters.

1.2 LCM Robot Model

Designing a robot model primarily focuses on achieving characteristics that closely resemble those of the real world. We study different problems using these robots, taking certain limitations into consideration.

1.2.1 Computational Model

Different models are proposed for swarm robots that are comprised of different capabilities and constraints. We present a computational model for the problems considered in this thesis. Here, we mention some facets of the robot model which are common to all the problems.

General Characteristics of Robots

In this thesis, we model the robots as points on the Euclidean plane, which is quite popular in the literature. Each of the robots in the swarm is

- *autonomous*: having no external control.
- *anonymous*: no external unique identifier.
- *homogeneous*: identical and runs the same algorithm.

- *luminous:* having an externally visible light that can assume a color from a color set.

The light is a form of weak communication between the robots. Besides the externally visible light, the robots do not possess any medium (wired or wireless) of communication.

Domain

We consider two different types of domains where the robots are deployed. First is the discrete domain, where the robots are placed in a graph. The robots can traverse the graph through the edges and occupy the vertices of the graph. Second is the continuous domain, where the robots are distributed at distinct points on the Euclidean plane and free to move anywhere from their current location.

Coordinate System and Orientation

Each of the robots has its own local coordinate system, the origin of which is the current location of the specific robot. They perceive a specific point based on their local coordinate systems. Two robots looking at the same point can have different coordinates. The robots may or may not agree on a common direction, for example, the direction of the y-axis. If they agree on one coordinate axis, we refer to it as *one-axis agreement*. In this case, all the robots have the same positive and negative direction for the specific axis. Orientation, also known as *chirality*, is a relative order of the positive x and y-axis on the Euclidean plane, such as the clockwise or the counterclockwise direction. If the robots agree on a common clockwise direction, we call them *oriented*. Otherwise, they are called *disoriented*.

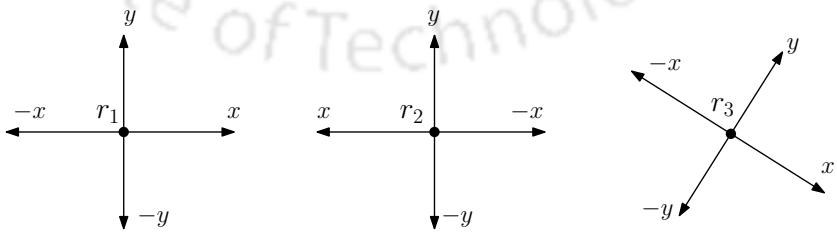


Figure 1.1: Axis Agreement and Orientation

For example, the robots [1] r_1 and r_2 agree on the y-axis, whereas r_3 does not agree on any

[1] We denote the robots as $\{r_1, r_2, \cdots r_N\}$ if N robots are considered. We generally refer to a robot using lower-case r.

coordinate axis with the two robots, as depicted in Fig 1.1. The robots r_1 and r_3 agree on the clockwise orientation (have the same chirality), but the orientation of r_2 is different than r_1 and r_3.

Opaqueness

Non-transparent robots are said to be *opaque robots*. Opaqueness is a more realistic assumption for the robot model than transparency. This allows the robots to obstruct the line of sight of other robots. For example, two robots r_i and r_j cannot see each other if a third robot r_k lies on the line segment joining r_i and r_j.

Visibility

The robots can be modelled to have infinite visibility. However, if the robots are opaque, a robot, even with infinite visibility, may not be able to see all the robots in the domain. This is often referred to as *obstructed visibility*. The robots may also have a finite visibility range. Such a robot is called *myopic* robot. A robot can see all the robots within its visibility range, which are not obstructed by other robots. It is common to assume that a robot is visible to itself.

Notion of Distance and Angle

In this thesis, we explore the problems assuming the robots have a notion of distance. Each robot has its own unit of measure depending on the local coordinate system. They can accurately calculate the distance between two visible points in the domain with respect to their local coordinate systems. Moreover, the notion of angle enables a robot r to compute the angle $\angle p_1 r p_2$ for two points p_1 and p_2 within its visibility range.

Activation Cycles

A robot is either active or inactive. The robot executes a *look-compute-move* (popularly known as LCM) cycle whenever it gets activated. Each cycle consists of three following phases.

- *look*: The robot captures a snapshot of its surroundings to gather data.
- *compute*: It runs the algorithm and finds a target point for its movement based on the data gathered in the look phase of its current cycle. If necessary, the robot also changes the color of the light in this operation.
- *move*: It moves to the target point in the same cycle or remains in place.

Activation Scheduler and Run-time

In the literature, there are three main kinds of schedulers under which the robots are activated.

- *Fully-Synchronous* (FSYNC): Under this scheduler, the time is divided into discrete global rounds. All the robots get activated simultaneously in each round and execute an LCM cycle in sync.
- *Semi-synchronous* (SSYNC): This is the same as the FSYNC scheduler except for the fact that a subset of robots gets activated simultaneously in SSYNC instead of all of them. All the activated robots execute their LCM cycles in sync. Clearly, the FSYNC scheduler is a special case of the SSYNC scheduler when all the robots are activated.
- *Asynchronous* (ASYNC): This is the weakest among all activation schedulers. There is no notion of a global time under this scheduler. Any robot can be activated at any time, with the fairness assumption that every robot is activated infinitely often.

Since a robot might be inactive for an arbitrarily large amount of activation cycles in both SSYNC and ASYNC settings, time is measured in terms of *epochs*, which is the smallest time interval in which every robot gets activated and executes its LCM cycle at least once.

Memory

The robots have their local memory to store the data gathered in the look phase of the current LCM cycle. Note that the collected data for the current cycle is erased after the end of the current cycle. The computation and movement of the robots rely only on local

computations on the basis of snapshots taken in the current look phase. Such a robot is called *oblivious* or a memoryless robot.

For luminous robots, the prefixed set of colors can be seen as a persistent memory. The current color of the light does not get changed until the robot switches it to a different color in the compute phase. Besides this, the robots do not possess any other persistent memory to store the information from their past cycles. It is customary for a robot with a constant-sized persistent memory to call it oblivious in the sense that when it transitions from one cycle to another, all its local memory, except the light, is reset.

Multiplicity

When two or more robots are collocated at a point simultaneously, the point is termed as *multiplicity point*. A multiplicity point leads to a *collision* between the robots. Detecting multiplicity is an extra cost for the robot model. In this thesis, we do not allow collision or crossing between robots' paths during their move phase.

Faults

In the context of swarm robots, fault tolerance is one of the important parts, as autonomous robots have drawn attention for their scalability. We are keenly interested in examining the solvability of a problem in the presence of faulty robots. In literature, two different types of faults are mainly considered.

- *Crash Faults*: The fault may occur at any time, which stops a robot from doing anything further.
- *Byzantine Faults*: Byzantine fault makes a robot behave maliciously and unpredictably.
- *Mobility Fault*: In the case of the luminous robot model, the crash fault is even more categorized. A widely popular fault model is *mobility fault*, which makes the robots immobile after the fault. However, the rest of the tasks remain unaltered.

Whenever we say fault in this thesis, we mean the mobility fault in the subsequent chapters.

Movement of the Robots

There are two types of movement of the robots.

- **Rigid Movement**: In this case, a robot moves to its target point in the same LCM cycle.
- **Non-rigid Movement**: A robot in motion might be stopped before reaching its destination. The robot can move at least a prefixed distance δ in an LCM cycle. The robot reaches its destination if the distance is less than a predefined distance δ. Otherwise, it moves at least δ towards the destination along its path in one cycle.

1.2.2 Common Problems

Autonomous mobile robots have a wide range of applications, which is why many standard problems are explored under this model. We mention some of the popular problems as follows.

- **Mutual Visibility:** Starting from any initial deployment of the robots, the aim of the problem is to achieve an arrangement of the robots such that any two robots present in the domain are visible to each other.
- **Pattern Formation:** The problem, which is extensively studied under this model, requires the robots to form a given pattern from an initial configuration of the robots within finite time.
- **Gathering:** The robot needs to gather at a point from any initial configuration. This problem can also be seen as a point formation problem for a system of robots.
- **Dispersion:** The problem is generally studied in graphs and aims at an arrangement of the robots such that every robot is placed at a distinct vertex from an arbitrary placement on the vertices of the graph.

- **Patrolling:** The task is to ensure that a border composed of segments with varying priorities is consistently traversed, with each segment's priority considered. The goal is to minimize the longest duration during which high-priority points remain unvisited while ensuring that low-priority points are visited continually.
- **Leader Election:** The task is to agree on a common leader.
- **Flocking**: This problem is an extension of the pattern formation problem where the goal is maintaining a specific pattern while in motion.

1.3 Organization and Contributions of the Thesis

The rest of the thesis is organized as follows. We extensively discuss all the relevant works in Chapter 2 for this thesis. Chapter 3 presents different models and preliminaries corresponding to the four problems considered in this thesis. In Chapter 4, we study the problem MIS Filling Problem on arbitrary graphs with myopic robots. Chapter 5 addresses the problem of mutual visibility in the presence of oblivious robots exhibiting angular inaccuracy in their movements. Chapter 6 discusses the mutual visibility problem in another model where the robots do not agree on any coordinate axes, along with the mobility failure of the robots. In Chapter 7, we present a relatively new problem, named uniform partitioning, using a swarm of oblivious, disoriented robots whose target is to collaboratively partition the given bounded region. Finally, we conclude the thesis in Chapter 8 along with the future scope of this thesis. We summarize the results presented in each chapter.

1.3.1 MIS Filling Problem on a Graph

We present a problem named *MIS Filling Problem* in Chapter 4 on a connected port-labelled graph with special vertices called Door. We consider the robots to be myopic, and they enter the graph through the door vertices. The aim of the problem is to fill a set of vertices such that the set of occupied vertices forms a maximal independent set (MIS) of the graph. In this chapter, we primarily investigate the power of myopic robots on the filling problem. We present two algorithms: (1) for a single Door under the ASYNC setting and

(2) for multiple Doors under the SSYNC setting. Both algorithms require $O(m^2)$ epochs to form an MIS of the given graph, where m is the number of robots that form an MIS of the graph. For a single-door case, we present a lower bound on the time complexity. Further, we give counterexamples to prove that the problem is not solvable if the robots have 2 hops of visibility range. We show that 4 hops of visibility is insufficient for the multiple-door case to achieve our goal. Chapter 4 is based on the publication [60].

1.3.2 Mutual Visibility Despite Angular Inaccuracy

Chapter 5 explores the fundamental problem in swarm robotics, which is popular as the *mutual visibility problem* in the presence of oblivious luminous mobile robots. To the best of our knowledge, this is the first work in the field of mutual visibility where we deal with inaccuracy in robots' movement. The inaccuracy is defined in terms of angular deviation from the target point chosen by the robots. The inaccuracy does not allow the classical solutions given in various literature to achieve mutual visibility among the robots. When there is no faulty robot, we present two algorithms, a color optimal algorithm in the SSYNC setting and a 3 color algorithm in the ASYNC setting. We extend the study to the mobility failure of the robots. In this case, a robot can become faulty at any time along with the inaccuracy in its movement. We propose an algorithm that tolerates any number of fault which requires 10 colors to achieve mutual visibility. All the algorithms presented in this chapter run in $O(N)$ epochs, where N is the total number of robots. Finally, we also show that after a certain extent of the angular inaccuracy, the problem becomes impossible to solve. Chapter 5 is based on the publication [58].

1.3.3 Fault-tolerant Mutual Visibility in Local Coordinates

In Chapter 6, we revisit the problem of mutual visibility using a swarm of luminous mobile robots deployed over the Euclidean plane. We investigate the problem of fault tolerance under the microscope, assuming that the robots do not agree on any global coordinate axes or orientation. Moreover, the robots are unaware about the total number of robots N present on the plane. By fault, we mean mobility failure which makes the robots immobile

after a fault. However, the light of a faulty robot remains unaffected. We show the impossibility of the problem in case of a specific initial configuration of the robots which is symmetric. We propose an algorithm that tolerates f ($< N$) number of faulty robots and uses a constant number of colors in the FSYNC setting. To be specific, the algorithm requires 21 colors and runs in $O(N^2)$ rounds. We present another algorithm that is much simpler than the prior one but can tolerate a single faulty robot. This algorithm needs only 2 colors in the FSYNC and 5 colors in the ASYNC setting. Chapter 6 is based on the publication [59].

1.3.4 Uniform Partitioning of a Bounded Region

In Chapter 7, we deal with a problem named *Uniform Partitioning* where a given bounded region needs to be partitioned into equal sub-regions using a swarm of oblivious luminous robots. To the best of our knowledge, we initiate the study of partitioning a region uniformly using opaque robots under the ASYNC setting. Along with opaqueness and asynchronous activation, the main challenge of the robots in this problem is the agreement among the robots regarding the partitioning method, as a bounded region can be divided in several ways. Because of the application-oriented point of view, we choose common geometric shapes such as rectangle, square and circle to partition them using the robots. We achieve a color-optimal algorithm for the rectangular region and a 5 color algorithm for the square region. The proposed algorithms for these two regions require $O(N)$ epochs in the ASYNC setting. We further present an algorithm for the circular region which needs $O(\log(N))$ epochs using a constant number of colors. We discuss how we can think of the trade-off between the number of colors required and the time complexity by proposing another algorithm for the circular region that has a higher time complexity. Chapter 7 is based on the publication [57].

Chapter 2

Literature Review

In this chapter, we explore the relevant literature for this thesis. Additionally, we highlight works that are of interest and closely aligned with the problems we are addressing in this thesis. We first discuss the literature related to the filling problem in graphs. After that, we explore the relevant work for the mutual visibility problem, and finally, we mention the related work for the uniform partitioning problem.

2.1 Graph Properties and Graph Filling using Robots

Exploring a graph using autonomous robots has been of great interest to researchers for many decades. In the recent past, many problems related to graph exploration have been studied, such as dispersion [5, 44, 46, 47], filling graph vertices [6, 7, 35, 36], maximal independent set and dominating set [8, 18, 30, 50, 55], searching [12, 13, 14, 15], etc. These problems give us a comprehensive idea of designing movement strategies for a swarm of autonomous robots on graphs.

2.1.1 Dispersion Problem

The dispersion problem was initiated by Augustine and Moses Jr. [5], where they studied the trade-offs between the two metrics: (i) memory requirement for the robots and (ii) the time to achieve the dispersion. They also provided the lower bound for both the metrics

of the problem. Later, the problem became popular among researchers and was studied to improve the gap between the lower and upper bound of the memory required and the time complexity under various models [40,41,42,43,70]. The problem is also studied in grid graphs where Kshemkalyani et al. [44] provided algorithms that are optimal with respect to time and memory complexity. The problem is also extended from the perspective of fault-tolerance [46, 47] where the byzantine fault is considered. A closely related work to our problem is the distance-2-dispersion, proposed by Kaur and Mondal [40]. Starting from a configuration where all the robots are collocated, the robots arrange themselves such that none of the adjacent vertices are occupied by the robots. Their algorithm does not always guarantee the formation of a maximal independent set (MIS) while achieving the goal of dispersion.

2.1.2 Independent and Dominating Sets on Graphs

A closely related problem to a set of problems on graph such as maximal independent set and dominating set is the scattering of robots [8, 30, 55]. Scattering of robots gives us an extensive plan in designing the movement of a swarm of robots in a graph which requires robots to place themselves on distinct vertices of the graph. Kamei and Tixeuil [39] solve two variations of the maximum independent set (MAX_IS) placement problem for grid networks under the ASYNC setting. The first one assumes knowledge of port numbering for each node. It uses three colors of light and a visibility range of two. The other one removes the port-numbering assumption and uses seven colors and a visibility range of three. Pattanayak et al. [50] presented algorithms for finding an MIS for different graph classes, such as trees, grids, general graphs, etc., considering different initial configurations of the robots. The robots operate under synchronous rounds. For the rooted configuration, where the robots are placed at the same vertex, the proposed algorithm finds an MIS for a general graph in $O(N\Delta)$ time and requires $O(\log N)$ bits of persistent memory, where N is the number of robots deployed on the graph and Δ is the maximum degree of the graph. For the dispersed configuration where the robots are at distinct vertices, the algorithm requires $O(N\Delta \log N)$ time and $O(\log N)$ bits of memory.

Another classical graph problem is the dominating set. In the recent past, Chand et al. [18] studied the problem using autonomous mobile robots. Two algorithms, a $O(m)$ time algorithm and a $O(l\Delta\log(\lambda)+Nl+m)$ time algorithm, were presented to place the robots for creating a dominating set of a graph, where m is the number of edges, l is the number of clusters of the robots initially, λ is the maximum ID-length of the robots.

2.1.3 Filling Problem

Another variation of graph exploration is the filling problem, which has a wide range of applications in the real world. The problem is thoroughly explored under various models. Barrameda et al. [6] proposed algorithms for uniform dispersal or filling problems on any simply connected orthogonal space using identical asynchronous sensors. They present two algorithms: one for the single door, where sensors have one unit of visibility range and two bits of persistent memory, and the other for multiple doors, where sensors have two units of visibility and a constant amount of persistent memory. They also prove that oblivious sensors cannot solve the problem deterministically, even if they have unlimited visibility. For multiple doors, they show that the problem is unsolvable if the visibility range is less than two, even if sensors have unbounded memory. Further, even with unbounded visibility and memory, they show that the problem is unsolvable if the sensors are identical. Barrameda et al. [7] extended the problem of uniform dispersal for orthogonal domains with holes. They solve the problem when robots have a visibility range of six without any direct communication among themselves. Later, they solve the problem using direct communication among robots to reduce the visibility radius without increasing the memory requirement. Hideg and Lukovszki [35] solve the filling problem in orthogonal regions, where the robots enter the region through entry points called doors. They propose two algorithms with run-time $O(n)$, one for single door and the other for multiple door case, where n is the number of vertices of the graph. Later, Hideg and Lukovszki [36] presented the Filling problem for arbitrary connected graphs in the ASYNC settings where the goal is to fill the graph using myopic luminous robots.

We extend the filling problem further to a problem called the *MIS Filling Problem*, using

Algorithm	Schedular	Problem	Topology	Number of Doors	Visibility Range	Memory (in bits)	Number of Colors	Time Complexity
PACK [36]	ASYNC	Filling Problem	Connected graph	Single	1 hop	$O(\log\Delta)$	$\Delta+4$	$O(n^2)$ epochs
BLOCK [36]	ASYNC	Filling Problem	Connect graph	Multiple (k)	2 hops	$O(\log\Delta)$	$\Delta+k+4$	$O(n)$ epochs
Algorithm 1 [39]	ASYNC	MAX_IS Placement	Grid network	Single	2 hops	$O(1)$	3	$O(n(L+l))$ steps
Algorithm 2 [39]	ASYNC	MAX_IS Placement	Grid network	Single	3 hops	$O(1)$	7	$O(n(L+l))$ steps
IND	ASYNC	*MIS Filling*	Connected graph	Single	3 hops	$O(\log\Delta)$	$\Delta+8$	$O(N^2)$ epochs
MULTIND	SSYNC	*MIS Filling*	Connected graph	Multiple (k)	5 hops	$O(\log(\Delta+k))$	$\Delta+k+7$	$O(N^2)$ epochs

Table 2.1: The state of the art of previous results PACK [36], BLOCK [36] and MIS placement on grid [39] with our proposed algorithms IND and MULTIND. The steps represent the total movement of robots throughout the execution of the algorithm. The number of nodes is n and the grid has a dimension of $L \times l$.

luminous robots with limited visibility. The problem requires the robots to enter the graph and eventually get settled on the vertices such that all occupied vertices form a maximal independent set (MIS) of the graph. The state-of-the-art results and ours' are presented in Table 2.1, where N is the number of robots that form MIS.

2.2 Mutual Visibility

The capabilities of the robots under LCM Robot model is examined for various problems such as gathering [49, 52], pattern formation [23, 34, 51, 61], mutual visibility [10, 66, 72], searching [14, 15], etc. In the LCM Robot model, the mutual visibility problem is one of the most common problems. The problem is studied with a swarm of autonomous mobile robots having various abilities. The exploration of this problem extends across different domains (continuous and discrete) and also includes strategies for tolerating faults in the robots.

2.2.1 Continuous Domain

Usually, the robots are modelled as points on the Euclidean plane [28, 29, 66, 69]. A point robot is assumed not to occupy any space on the plane. However, a more realistic model called *fat robot* model, is considered to inspect the mutual visibility problem. A fat robot is modelled as a unit disk on the 2D plane [9, 17, 56].

Point Robots

Di Luna et al. [29] introduced the problem of mutual visibility on the Euclidean plane using luminous point robots under the SSYNC setting where N, the total number of robots, is a global knowledge. Later in [28], they investigated the problem using luminous robots with rigid and non-rigid movement under both SSYNC and ASYNC settings and proposed algorithms for both cases. Their algorithms require 3 colors to achieve mutual visibility among the robots. Using the same model as in [28], Sharma et al. [66] presented an algorithm that improves the number of colors to 2 in SSYNC and ASYNC settings. Vaidyanathan et al. [72] presented a $\mathcal{O}(\log n)$ algorithm with FSYNC luminous robots considering a model that assumes chirality and allows collision between robots. Later, a constant time algorithm under the SSYNC setting is proposed by Sharma et al. [69] using robots with rigid movement that is free from collision. Bhagat et al. [10] investigated mutual visibility with semi-synchronous robots in FSTATE model [27] where the robots use a persistent memory to remember only the previous internal state. Sharma et al. [68] presented a constant time algorithm with ASYNC luminous robots, where robots do not have any agreement on coordinate axes.

Fat Robots

The problem of mutual visibility is also studied using fat robots in [9, 17, 56]. Sharma et al. [65] solved the mutual visibility using fat luminous robots under fully synchronous settings. Bose et al. [17] proposed an algorithm for the mutual visibility problem using fat luminous robots having both axes agreement under semi-synchronous settings where the camera on the robot is slim in the sense that the camera is a disk with a smaller radius than

the robot. Bhagat et al. [9] explored the problem using fat luminous robots with one axis agreement among them and proposed a linear time algorithm under this model.

2.2.2 Discrete Domain

The problem is not restricted to the Euclidean plane. It is also studied in discrete domains, such as grids, trees, and general graphs. Adhikary et al. [2] presented an algorithm for luminous point robots where robots are deployed on an infinite grid and operate under the ASYNC setting. Later, Sharma et al. [67] proposed a randomized algorithm using luminous point robots on a grid domain of unbounded size. The problem is examined further in finite grids and trees by Cicerone et al. [21] where the aim is to place the robots on distinct vertices of the graph in such a way that there exists a shortest path between every pair of robots with no third robot lying on it.

2.2.3 Fault-tolerance

The problem is also studied under the microscope of fault tolerance. Mainly, the fault is considered to be mobility failure, which means that the faulty robots become immobile after the fault. Aljohani and Sharma [4] presented an algorithm tolerating a single faulty robot under the SSYNC setting and with the assumption of agreement on both coordinate axes. Poudel et al. [54] gave an algorithm that tolerates a general number of mobile faulty robots with an extra assumption that the robots agree on one coordinate axis. Sharma et al. [64] solved mutual visibility for SSYNC luminous robots considering a different type of fault, where the light of the robots becomes faulty, but the mobility of the robots is not affected.

2.2.4 Inaccurate Movement

Few papers in the literature address the inaccuracy in robots' movement, which is comparatively harder to handle in the case of the mutual visibility problem. Souissi et al. [71] proposed an algorithm for the gathering problem using two ASYNC robots having inaccurate compasses. They considered that the compasses of the two robots can diverge as

Paper	No. of faults	Inaccuracy	Scheduler	Global Knowledge	Colors	Time Complexity
Di Luna et al. [29]	None	No	SSYNC	N	-	-
Di Luna et al. [28]	None	No	ASYNC	None	3	-
Vaidyanathan et al. [72]	None	No	FSYNC	Chirality	12	$O(\log n)$
Aljohani & Sharma [4]	1	No	SSYNC	Both axes	3	-
Poudel et al. [54]	f	No	SSYNC	One axis	2	$O(N)$
Poudel et al. [54]	f	No	ASYNC	One axis	3	$O(N)$
Sharma et al. [68]	None	No	ASYNC	None	47	$O(1)$
SSYNC_MV_INACC	None	Yes	SSYNC	One axis	2	$O(N)$
ASYNC_MV_INACC	None	Yes	ASYNC	One axis	3	$O(N)$
FAULT_MV_INACC	f	Yes	ASYNC	One axis	10	$O(N)$
FAULT_MV$_{LC}$	f	No	FSYNC	None	21	$O(N^2)$
FAULT_MV$_{LC}^{f=1}$	1	No	SSYNC	None	2	-
			ASYNC		5	

Table 2.2: The state of the art of previous works related to mutual visibility problem on the Euclidean plane

much as $\pi/4$. Later, Yamashita et al. [74] extended the same problem, taking the angular divergence between the compasses of the two robots to any angle less than π.

This thesis studies the mutual visibility problem under two separate models. In the first one, we explore the problem with point robots exhibiting inaccuracy in their movement. We further integrate mobility failure in this model and solve the problem under the assumption of one-axis agreement. In the second model, we assume that the robots do not have any agreement on the coordinate axes and are susceptible to mobility failure. Table 2.2 provided a comparison between some of the major literature in mutual visibility and our works.

2.3 Distributed Uniform Partitioning Problem

Partitioning a bounded region using an autonomous robot swarm is a relatively less explored area. The problem has a keen correlation with pattern formation that also deals with repositioning a swarm of robots from any arbitrary initial configuration, although the primary difference is the knowledge of N which is known to the robots.

2.3.1 Pattern Formation

Pattern formation using the LCM robot model has grabbed a lot of attention for many years now. In various literature [11, 22, 34, 48, 61, 73], the pattern formation problem is studied with mobile robots with different capabilities. Uniform circle formation (popularly known as UCF) is one of the most popular problems in this area, and it can be useful for partitioning a region of a specific geometrical shape. In the recent past, Feletti et al. [31] presented a linear time algorithm for the UCF problem using ASYNC luminous robots. Later in [32], they proposed an improved algorithm that runs in $O(\log(N))$ epochs using a constant number of colors. Biswas et al. [16] explored the formation of multiple uniform circles using robots with limited visibility working under the FSYNC setting.

2.3.2 Partitioning a Region

The uniform partitioning problem has a wide range of applications, such as surveillance of a particular region, painting a bounded region and many more. Saha et al. [63] inspected surveillance of uneven surfaces using drones, which falls into the category of coverage problem. Their target is to provide compact coverage of the area so that the diameter of the drone network is minimized. Das and Mukhapadhyaya [24] proposed an ASYNC algorithm for distributed painting in a rectangular region with a robot swarm where robots have agreement on a global line. The algorithm divides the region into uniform horizontal strips, but one of the advantages of this paper is that the robots are transparent, which enables them to see all other robots in the region in every activation. Later in [25], they studied the distributed painting with robots having limited visibility and a global coordinate system. Robots are not completely oblivious and transparent, so they can see all the robots within the visibility range. Das et al. [26] extended the distributed painting when the rectangular region has opaque obstacles in it. They consider the robots to be transparent and work under SSYNC settings, having total agreement in the direction and orientation of their local coordinate systems. Pavone et al. [53] proposed an algorithm for equitable partitioning of an environment using synchronous mobile agents, where equitable partitioning means the load for every robot is similar for each agent. Their algorithm uses

Papers	Problem	Region	Scheduler	Opaque Robots	Axis Agreement	Number of Colors	Time Complexity
Das and Mukhopadhyaya [24]	Distributed painting	Rectangle	ASYNC	No	Both axes	None	-
Das and Mukhopadhyaya [25]	Distributed painting with limited visibility	Rectangle	ASYNC	No	Both axes	None	-
Das et al. [26]	Distributed painting with obstacles	Rectangle	SSYNC	No	Both axes	None	-
Acevedo et al. [1]	Non-uniform Partitioning	Known Environment	FSYNC	Yes	Both axes	None	-
Feletti et al. [32]	Uniform circle formation	Euclidean plane	ASYNC	Yes	None	64	$O(\log N)$
RECTANGLE_PARTITION	Uniform Partitioning	Rectangle	ASYNC	Yes	None	2	$O(N)$
SQUARE_PARTITION	Uniform Partitioning	Square	ASYNC	Yes	None	5	$O(N)$
CIRCLE_PARTITION	Uniform Partitioning	Circle	ASYNC	Yes	None	17	$O(\log N)$
CIRCLE_PARTITION	Uniform Partitioning	Circle	ASYNC	Yes	None	8	$O(N^2)$

Table 2.3: The state of the art of some previous works related to uniform partitioning of a bounded region

the Voronoi-based partitioning approach but does not guarantee equal partitioning of the region. Acevedo et al. [1] gave an algorithm for partitioning a known region using aerial robots, assuming the whole area is divided as grids, but the partitioning is not uniform.

We explore the *Uniform Partitioning* problem where the aim is to partition the given bounded region into sub-regions of equal area using opaque ASYNC robots such that each sub-region has exactly one robot at the end. Table 2.3 gives us an overview of the related work to the uniform partitioning problem and provides a comparison with our work.

Chapter 3

Models and Preliminaries

In this chapter, we formally list down the robot models and preliminaries required for the problems in the subsequent chapters. As mentioned in Chapter 1, this thesis deals with four problems, out of which one is on the discrete domain (arbitrary graph) and the other three consider a continuous domain (the usual Euclidean plane). In this chapter, we are going to mention the domain and any assumption related to the domain explicitly in the following sections.

3.1 MIS Filling Problem on a Graph

We consider the *MIS Filling Problem* in Chapter 4. In this problem, we model the environment as a graph. We say that the graph contains a set of vertices that are connected to Doors from where robots can enter. The number of Doors in the graph is unknown to the robots, but they are equipped with colors to distinguish themselves if they enter the graph from different Doors. We assume a maximum of k Doors are attached to the graph. The goal of the robots is to fill a maximal independent set of vertices of the given graph.

Graph: We consider an anonymous graph, i.e., the nodes of the graph are indistinguishable from each other. Each vertex v of the graph contains port numbers corresponding to the incident edges from $[1,2,\ldots,\delta(v)]$, where $\delta(v)$ is the degree of the vertex v. Given an anonymous connected port labeled graph $H = (V', E')$, we construct a graph $G = (V, E)$ with k Doors, that adds two auxiliary vertices $\{d_i, d_i'\}$ corresponding to each Door that is con-

nected to distinct vertices $\{v_1, v_2, \ldots, v_k\} \subset V'$. We have a path $d_i \rightarrow d_i' \rightarrow v_i$ corresponding to each Door d_i. The robots enter the graph through the Door, and a new robot appears at the Door immediately after it becomes empty. We say a vertex is *free* if none of the vertices adjacent to it are occupied by any robot. Since we add a buffer vertex corresponding to each Door, all the vertices in the original graph H are free vertices in the beginning.

Robots: The focus is on completing the task using robots with minimal capabilities operating under certain adversarial conditions. The robots are autonomous, anonymous and homogeneous. In this chapter, we consider the robots to be myopic which means that they have a finite visibility range. Additionally, the robots are luminous and follow the LCM activation cycles. The light of a robot can be seen from any direction. Moreover, a robot moves two hops in a single move phase.

Assumptions: We have the following assumptions regarding the knowledge of a robot and the properties of the underlying graph.

- The robots have no knowledge of the graph but an upper bound of Δ, the maximum degree of the graph.
- For a robot placed at v with a visibility range of z, the port numbers of all the vertices in its visibility range are visible. We say that the robot has z hops of visibility.
- Movement of robots is non-instantaneous.
- As soon as a robot appears at the Door, it is activated immediately and starts the look phase.
- A robot recognizes all the colors corresponding to the doors, but it can only display the color corresponding to the Door via which it enters the graph along with the colors common to all robots. A robot entering through i-th Door cannot display the colors corresponding to the j-th Door. Note that, at most one robot from the set of robots that entered the graph from a given Door, shows the color of the Door at a time. We use unique colors for each Door to determine a hierarchy among them.

Note that we use directions and port numbers interchangeably throughout this chapter. Each port number corresponds to a `DIR` color, also the direction towards Successor or

Predecessor.

3.2 Mutual Visibility Despite Angular Inaccuracy

Chapter 5 deals with a fundamental problem in swarm robotics, popularly known as the *mutual visibility problem*, where the robots are required to arrange themselves so that no three of them are collinear. The domain considered in this problem is the Euclidean plane. In this chapter, we study the problem in the presence of robots exhibiting angular inaccuracy in their movement.

Robots: We consider a set of N autonomous, anonymous and homogeneous point robots $R = \{r_1, r_2, \cdots r_N\}$ on the Euclidean plane, where r_i denotes a robot for $i = 1, 2, \cdots N$. They are opaque and operate in LCM cycles. All the robots are oblivious. There are no means of communication except for a persistent externally visible light on them, which can take any color from a constant-sized color set. None of the robots have knowledge about N. The robots perceive all the points on the plane by their local coordinate system. However, we assume that the robots agree on one coordinate axis. Without loss of generality, we consider the y-axis. This means that the robots know the positive and negative directions of the y-axis, but they do not know the direction of the x-axis. The positive and negative direction of the x-axis might be different for different robots. This enables a robot to determine the horizontal and vertical lines passing through it.

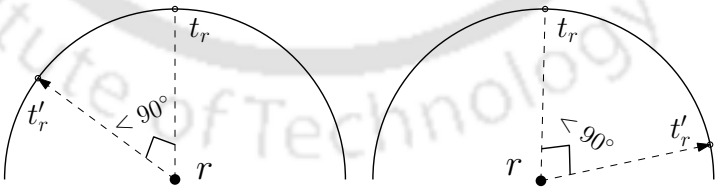


Figure 3.1: Illustrating the inaccurate movement of a robot r_i which chooses to move to t_r but moves to t'_r

Inaccurate Movement: For any robot r, the inaccuracy in movement is defined as an angular deviation from the intended target to which it has chosen to move. As shown in Fig. 3.1, if r decides to move to the point t_r, it actually moves to the point t'_r due to the inaccuracy in its movement. We call the angle $\theta_r = \angle t'_r r\, t_r$ the *angular inaccuracy* of the robot r. The

angle θ_r is assumed to be less than 90°. Two robots don't need to have the same angular inaccuracy.

Configuration: We define a configuration C_t to be the set $\{(r_1^t, r_1^t.color), (r_2^t, r_2^t.color), \cdots, (r_N^t, r_N^t.color)\}$, where r_i^t is the position of the robot r_i at time t (≥ 0) and $r_i^t.color$ is the color shown by the light on the robot r_i at the time t. Since the visibility of the robots can be obstructed, all the robots might not be visible to a particular robot. So we define the local configuration for a robot $r_i \in R$ at any time t to be set of all tuples of the form $(r_j^t, r_j^t.color)$, where r_j is a visible robot to r_i. Since a robot is visible to itself, the local configuration for the robot r_i includes the robot itself. For simplicity, we write r_i and $r_i.color$ instead of r_i^t and $r_i^t.color$ respectively. We misuse the notation r_i to represent the robot r_i and the current location of it.

Notations: Here, we mention the notations and conventions that we follow throughout this chapter.

- The y-coordinate of the location of the robot r_i is denoted by $r_i.y_{\text{coord}}$.
- For two robots r_i and r_j, $\overline{r_i r_j}$ is the line segment joining them, whereas $\overleftrightarrow{r_i r_j}$ represents the infinite line passing through r_i and r_j.
- For any non-perpendicular line (a line which is not parallel to the y-axis) L passing through the robot r, we call the open half plane delimited by the line L that contains the positive y-axis of the local coordinate system of r as the *upper half plane* and denote it by H_L^{Up}. Similarly, the open half plane delimited by the line L that contains the negative y-axis of the local coordinate system of r is called the *lower half plane* and denoted by H_L^{Low}.
- The shortest distance between a point r_i and a line $\overleftrightarrow{AB}$ is the perpendicular distance between them and denoted by $d(r_i, \overleftrightarrow{AB})$. We use the same notation $d(r_i, r_j)$ to represent the shortest distance between the two points r_i and r_j.
- For convenience, we use the notation $\mathcal{VAR}_r^{Prev}$ to denote any variable $\mathcal{VAR}_r$ in the previous LCM cycle for any robot r.

Collinearity: Any three robots r_i, r_j and r_k are collinear if they lie on the same line. We denote this collinearity by $\overline{r_i r_j r_k}$. For three collinear robots r_i, r_j and r_k, we call the infinite line passing through them as the *line of collinearity* (we use LOC henceforth), and it is represented by $\overleftrightarrow{r_i r_j r_k}$. We classify collinearity into two types. The first one is *horizontal collinearity*, where the LOC $\overleftrightarrow{r_i r_j r_k}$ is a horizontal line. The second type is *non-horizontal collinearity* in which $\overleftrightarrow{r_i r_j r_k}$ is a non-horizontal line. In the latter case, if $r_i.y_{\text{coord}} > r_j.y_{\text{coord}} > r_k.y_{\text{coord}}$, then r_i, r_j and r_k are called the *first*, *middle* and *last robot of collinearity*, respectively. Notice that the middle robot r_j of the collinearity $\overline{r_i r_j r_k}$ is the only robot that can detect the collinearity because both the robots r_i and r_k are visible to it. This enables the middle robot to determine the LOC.

3.3 Fault-tolerant Mutual Visibility without any Axis Agreement

In Chapter 6, we revisit the *mutual visibility problem* with a different model than the previous. The goal is to achieve mutual visibility among the non-faulty robots, i.e., any two non-faulty robots should be visible to each other.

Robots: We consider N autonomous, anonymous, homogeneous and disoriented point robots denoted by $r_1, r_2, \cdots r_N$ on the 2D plane, each of which operates under LCM cycles. These robots are *opaque* and do not know N. Robots have no agreement on the coordinate axes or the orientation. By fault, we mean the mobility failure of the robots. When a fault occurs, the robot is unable to move henceforth. The fault can occur at any time. A faulty robot cannot be identified even by itself.

Configuration: The definition of a configuration C_t of all the robots remains the same. We denote the convex hull of all the robots by $\mathcal{CH}$. Initially, the robots do not necessarily know $\mathcal{CH}$ because of the obstructed visibility. We denote $C_t(r)$ to be the configuration of all the robots that are visible to r at the time t. The convex hull of the robots visible to r, sometimes called *the local convex hull of* r, is denoted by $\mathcal{CH}_r$.

3.4 Uniform Partitioning of a Bounded Region

In Chapter 7, we introduce a problem, named as *uniform partitioning problem* of a bounded region, where the goal is to divide the given region into N uniform partitions with the help of N mobile robots so that each of the partitions contains exactly one robot.

Robots: A set of N disoriented and opaque luminous mobile robots $\{r_1, r_2, \cdots r_N\}$ is deployed at distinct points within a bounded region. This region can be thought of as a subset of the Euclidean plane. The robots can detect the boundary of the region. All the robots operate in LCM cycles and are activated under a fair ASYNC scheduler. A robot is visible to itself but might be unable to see all the robots inside the region due to obstructed visibility. Moreover, robots do not know N and have no agreement on their coordinate axes.

Region: We are given with a bounded region, denoted by $\Re$. The *interior* of the region $\Re$, denoted by $Int(\Re)$ is defined to be the part of $\Re$ without the boundary. In this chapter, we consider the region to be a standard geometric region such as a rectangle, square or circle. A robot lying on $Int(\Re)$ is called an *interior robot*. A robot is a *boundary robot* when it lies on the boundary of $\Re$ but not on the corners. When $\Re$ has corners, robots lying on them are called *corner robots*. The boundary of $\Re$ is identifiable by the robots from any point in the region, which enables them to identify whether the region is a rectangle, square or circle. For any two points A and B in the region, $\overline{AB}$ denotes the line segment joining A and B. $\overleftrightarrow{AB}$ is the line passing through the two points. The length of a line segment $\overline{AB}$ is represented by $len(\overline{AB})$. We denote the distance between two points A and B by $d(A,B)$. A similar notation $d(p,L)$ is used to represent the shortest distance between a point p and a line L.

Chapter 4

MIS Filling Problem on a Graph

4.1 Introduction

The maximal independent set (MIS) of a network graph plays a significant role in decomposing the network into clusters of low diameter, which is often very useful in designing and implementing a distributed divide-and-conquer algorithm. MIS vertices can also be used as a network backbone for deploying communication infrastructure. For example, information dissemination in a low latency system where all robots form a network should be located at MIS vertices so that all other vertices are just one hop away.

The Filling problem, introduced by Hsiang et al. [38], considers the robots enter via particular vertices and fill an environment (graph) composed of pixels (vertices), and robots occupy every pixel (vertex). Later, Hideg et al. [36] presented the Filling problem for an arbitrary connected graph. It is of interest to cover the entire graph but using a smaller number of robots. Thus, forming an MIS by the robots that enter the graph becomes a natural extension. We call this problem the *MIS Filling problem*. In this problem, the robots enter the graph one by one through a specific vertex called the *Door* and move in the graph along the edges from one vertex to another while avoiding a collision. The objective is to occupy only those vertices that form an MIS. We solve two flavours of the problem: graphs with a single Door under an ASYNC scheduler and graphs with multiple Doors under an SSYNC scheduler. Having multiple Doors instead of just one offers redundancy in situa-

tions where a Door can be blocked. We define the model and the problem formally in the subsequent section.

4.1.1 Contributions

In this chapter, we propose two algorithms, IND and MULTIND, corresponding to single and multiple doors.

- Algorithm IND solves the *MIS Filling problem* in graphs with a single Door under an ASYNC scheduler using robots with a visibility range of 3, $\Delta + 8$ number of colors, $O(\log \Delta)$ bits of persistent storage in $O(N^2)$ epochs, where Δ is the maximum degree of the graph, and N is the number of robots that form an MIS.
- Algorithm MULTIND solves the *MIS Filling problem* in graphs with multiple Doors under an SSYNC scheduler using robots with a visibility range of 5, $\Delta + k + 7$ number of colors, $O(\log(\Delta + k))$ bits of persistent storage in $O(N^2)$ epochs, where k is the number of Doors.
- We show a lower bound of $\Omega(n)$ of the *MIS Filling problem* with single Door, where n is the number of vertices of the graph. We also discuss the minimality of the visibility range of a robot required for the problem which is 3 hops for the single Door case and 5 hops for the multiple Doors case.

4.2 Model and Preliminaries

In this chapter, we model the environment as a graph. We say that the graph contains a set of vertices that are connected to Doors from where robots can enter. The number of Doors in the graph is unknown to the robots, but they are equipped with colors to distinguish themselves if they enter the graph from different Doors. We assume a maximum of k Doors are attached to the graph.

Graph: We consider an anonymous graph, i.e., the nodes of the graph are indistinguishable from each other. Nodes do not possess any computation power or memory. Each

vertex v of the graph contains port numbers corresponding to the incident edges from $[1,2,\ldots,\delta(v)]$, where $\delta(v)$ is the degree of the vertex v. Given an anonymous connected port labeled graph $H = (V', E')$, we construct a graph $G = (V, E)$ with k Doors, that adds two auxiliary vertices $\{d_i, d_i'\}$ corresponding to each Door that is connected to distinct vertices $\{v_1, v_2, \ldots, v_k\} \subset V'$. We have a path $d_i \rightarrow d_i' \rightarrow v_i$ corresponding to each Door d_i. The robots enter the graph through the Door, and a new robot appears at the Door immediately after it becomes empty. We say a vertex is *free* if none of the vertices adjacent to it are occupied by any robot. Since we add a buffer vertex corresponding to each Door, all the vertices in the original graph H are free vertices in the beginning.

Robots: The focus is on completing the task using robots with minimal capabilities operating under certain adversarial conditions. The robots are autonomous, anonymous and homogeneous. In this chapter, we consider the robots to be myopic which means that they have a finite visibility range. Additionally, the robots are luminous and follow the LCM activation cycles. The light of a robot can be seen from any direction. Moreover, a robot moves two hops in a single move phase.

Assumptions: We have the following assumptions regarding the knowledge of a robot and the properties of the underlying graph.

- The robots have no knowledge of the graph but an upper bound of Δ, the maximum degree of the graph.
- For a robot placed at v with a visibility range of z, the port numbers of all the vertices in z hops distance are visible. We say that the robot has z hops of visibility.
- Movement of robots is non-instantaneous.
- As soon as a robot appears at the Door, it is activated immediately and starts the look phase.
- A robot recognizes all the colors corresponding to the doors, but it can only display the color corresponding to the Door via which it enters the graph along with the colors common to all robots. A robot entering through i-th Door cannot display the colors corresponding to the j-th Door. Note that, at most one robot from the set of

robots that entered the graph from a given Door, shows the color of the Door at a time. We use unique colors for each Door to determine a hierarchy among them.

Note that we use directions and port numbers interchangeably throughout this chapter. Each port number corresponds to a DIR color, also the direction towards Successor or Predecessor. We now formally define the problem as follows.

Problem 1 (MIS Filling Problem): Given an anonymous connected port labelled graph $G = (V, E)$ with k Doors, the objective is to relocate robots that appear at Doors such that at termination, the robots occupy a set of vertices V_1 ($V_1 \subset V$) that forms a maximal independent set of G.

4.3 Algorithm for MIS Filling with Single Door

We now describe the algorithm called IND, which is inspired by the PACK algorithm [36] and uses the concept of Virtual Chain Method [37]. The robot's movement throughout the graph is similar to the depth-first search (DFS). We assume that the robots operate under an asynchronous (ASYNC) scheduler, where time is measured in terms of epochs. Each robot requires a visibility range of 3 hops, $O(\log \Delta)$ bits of persistent memory, and $\Delta + 8$ colors.

4.3.1 Algorithmic Preliminaries

Variables: Table 4.1 lists all the variables used by a robot r in our algorithms. Each robot r has four states. $r.State$ is set to None when r appears at the Door in the beginning. $r.State$ is set to Follower when r has a predecessor robot. $r.State$ is set to Leader when r is the current leader. $r.State$ is set to Finished when r is stuck and transfers the leadership to its successor. The variable $r.Successor$ (resp. $r.Predecessor$) is a tuple (p_1, p_2) of port numbers corresponding to the direction of the successor (resp. predecessor) robot from the current location of r. The variable $r.color$ represents the color of the robot r at any time. Any robot r moves two hops to a vertex from its current location. If r needs to take the port p_1 first and then p_2 to reach its target vertex, it stores p_1 in $r.Target.One$ and p_2

in $r.Target.Two$. When the predecessor communicates its target using different colors, r stores them in $r.NextTarget$ in the same way as $r.Target$, so that it can follow the path taken by its predecessor.

Variable	Description
$r.State$	State of the robot - None/Follower/Leader/Finished
$r.Successor$	A tuple (p_1, p_2) of port numbers where if the robot takes the ports p_1 and p_2, in a sequence it reaches the successor robot.
$r.Predecessor$	A tuple (p_1, p_2) of port numbers where if the robot takes the ports p_1 and p_2, in a sequence it reaches the predecessor robot.
$r.color$	Color displayed by robot's light
$r.Target$	A tuple $(Target.One, Target.Two)$ of port numbers that corresponds to the direction of a vertex 2 hops away from the current position of the robot
$r.NextTarget$	A tuple $(NextTarget.One, NextTarget.Two)$ of port numbers that corresponds to the direction of a vertex to which the predecessor robot is moving (Used by robots in the Follower state)

Table 4.1: List of variables used by a robot r in algorithms IND and MULTIND.

Colors: The colors used by the robots are described here.

- `OFF` - Used initially when a robot arrives at the door.
- `DIR` - Δ colors corresponding to a port number in $[1, \Delta]$.
- `CONF` - Used to confirm that first `DIR` color has been seen and received.
- `CONFC` - Used to confirm that `CONF` color has been seen and received.
- `CONF2` - Used to confirm that second `DIR` color has been seen and received.
- `CONF3` - Used to confirm that the Packed state is achieved.
- `WAIT` - Used by a Leader while waiting for the Packed State.
- `MOVE` - Used when a robot is in movement.
- `FINISH` - Used by a robot in the Finished state.

During the execution of the algorithm, a robot r needs to know the color of its successor and predecessor to execute its next action. For any robot r', we define $Color_r(r')$ as the color of r' that r sees in its look phase. The variable $r.color$ is clearly different from $Color_r(r')$ as $r.color$ is the variable used by the robot r to represent its own color.

Definition 4.3.1. (k hops Neighbourhood of a vertex v): For a vertex v, we define k-hops Neighbourhood of v to be the set of all vertices that are exactly k hops away from v and denote it by N_v^k.

Definition 4.3.2. (k hops Visibility Set of a robot r): For a robot r placed at a vertex of the graph, we define the k hops Visibility Set of r to be the set of all vertices which are within k hops from the current location of r. We denote it by V_r^k.

Packed State: We define *Packed state* as the state of a chain where all the alternative vertices are occupied by robots. We define it formally as follows:

Definition 4.3.3. (Packed state) Let L be a positive odd integer and $P = \{v_1, v_2, \ldots, v_L\}$ be a path starting from the Door at v_1 and the leader at v_L such that every second vertex of P, i.e., $v_1, v_3, \ldots, v_{L-2}$ was visited by the Leader. A chain of robots is in a Packed state if the vertices $v_1, v_3, \ldots, v_{L-2}$ are occupied by follower robots.

4.3.2 IND Algorithm

We now present the IND algorithm for *MIS filling problem* with a Single Door. We say a vertex is *free* if none of its neighbours contains a robot. The first robot that enters the graph becomes the first Leader robot. Any robot that enters the graph after the Leader is a follower robot until the current leader is stuck and transfers the leadership. Every robot r chooses a target vertex to explore the graph. Before moving to the target vertex, r communicates the direction of the target to its successor, so that the successor can follow the path r takes. We describe the communication process in Section 4.3.2. After that, r moves to the target vertex with color `MOVE`. After the movement, the chain of robots needs to confirm that the Packed state is achieved which is described in Section 4.3.2. the current leader

initiates the next movement only if the chain of robots is in a Packed state. Section 4.3.2 describes the actions taken by the current leader robot when it has no vertex to move to.

We first explain some of the important subroutines of the algorithm before going to the main description for better comprehension of our algorithm. We demonstrate the communication between the Predecessor and Successor and the restoration of the Packed state after the movement with an example as shown in Fig. 4.1.

Communicating the movement directions: The robots establish the Predecessor and Successor relationship among them by their order of appearance from the Door. A Predecessor r communicates its destination $r.Target$ to a Successor by showing the colors corresponding to the port numbers at the vertex. Suppose r_1, r_2 and r_3 are located at e, c and a, respectively as shown in Fig. 4.1.

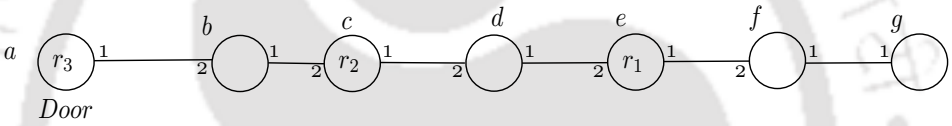


Figure 4.1: Communication of color from r_1 to r_2

r_1 determines that it will move to vertex g. Now, it has to communicate that to r_2, so that r_2 can follow r_1. First r_1 shows the `DIR` color of port 1 corresponding to the edge ef as it wants to take the path $ef - fg$, and r_2 responds by showing `CONF` color. Then r_1 sets its color to `CONFC` to confirm that it has seen the `CONF` color. Now, r_2 also sets its color to `CONFC` to inform r_1 that it is ready to receive the second color. (`CONFC` color is used to distinguish between two `DIR` colors that can be the same.) Next, r_1 shows the `DIR` color of port 1 corresponding to edge fg. r_2 confirms that it has seen the color by setting its color to `CONF2`. Once r_1 sees `CONF2`, it can move to g after setting its own color to `MOVE`. Now r_2 sets e as its target (which was the old position of r_1 Then, r_2 does the same process to communicate `DIR` colors to r_3.

Moving to a target vertex: If the robot r sets its target and stores it in $r.Target$, it first takes the port $r.Target.One$ to reach a vertex v, which is 1 hop away from the current location. It then takes the port $r.Target.Two$ from v to reach the target.

Restoration of Packed state after movement: We illustrate this module using Fig. 4.1.

Once the leader r_1 transitions to g with the MOVE color, e becomes vacant, causing the Packed state to become distorted. So r_1 changes its color to WAIT. The WAIT color is utilized by the Leader to signify that it is waiting for the chain to be in the Packed state. Now, r_2 moves to e with color MOVE after communicating DIR colors to r_3. So, c becomes empty. After that, r_3 moves to c with color MOVE without communicating DIR colors as it does not have any successor at this movement. As soon as r_3 leaves the Door, a robot r_4 is placed at the Door with color OFF. After r_3 reaches c, it sees r_4 with color OFF and $r_3.color$ = MOVE. So, it changes its color to CONF3. Now, r_2 also changes its color to CONF3 after seeing $r_3.color$ = CONF3 and $r_2.color$ = MOVE. The leader r_1 sees r_2 with color CONF3 and $r_1.color$ = WAIT, so it understands that the Packed state is achieved. Now, r_1 looks for new target to move to.

The WAIT color is necessary in scenarios like the one depicted in Fig. 4.2, where r_1 can select the vertex c as its target while r_3 is yet to move to c.

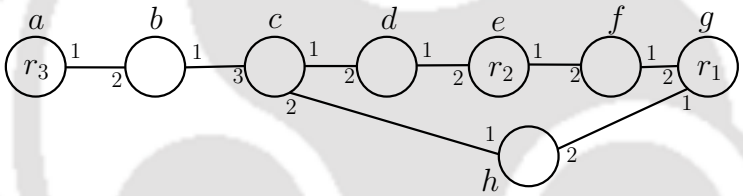


Figure 4.2: r_1 chooses c if it does not wait for the chain to be in Packed state. In Packed state, r_3 is supposed to be at the vertex c

Transferring the leadership: The current leader r_1 transfers its leadership if either of the two scenarios occurs. First, r_1 gets stuck, i.e. there are no other free vertices left to move to from the current vertex of the leader. Secondly, for each free vertex $v \in V^2_{r_1}$, r_1 finds at least one vertex $v' \in N^2_v \cap V^3_{r_1}$ with a robot not in the Finished state. In both of the above cases, r_1 transfers the leadership to its successor r_2 by communicating the direction that points towards r_2. We illustrate the process of transferring the leadership step by step in Fig. 4.4 of Lemma 4.3.13.

Detailed description: When a robot first appears at the Door, it initializes to the None state and sets color OFF. Let r_1 be the first robot that appears at the Door. r_1 does not find a robot on any adjacent vertices, so it changes its state to Leader. Now r_1 is the Leader and chooses a target vertex two hops away and moves to it with $r_1.color$ = MOVE. As soon as the Leader

r_1 leaves the Door, the next robot r_2 appears at the Door. At this time, r_1 is still in motion and is nearest to r_2. r_2 sees this and becomes the follower [1] of r_1 and sets $r_2.color$ = OFF. After r_1 reaches its target, it first sets its successor by $r.Successor.p_1 = r.Target.One$ and $r.Successor.p_2 = r.Target.Two$. Then, it sets $r_1.color$ = WAIT to indicate that it is waiting for the Packed state. Similarly, r_2 stores the port numbers corresponding to the direction of r_1. When r_2 finds $Color_{r_2}(r_1)$ = WAIT and $r_2.color$ = OFF, r_2 changes its color to CONF3. r_1 now chooses a new free vertex (if any) as $r_1.Target$ and communicate its directions to r_2 as described in Section 4.3.2. When r_1 gets confirmation from r_2 ($Color_{r_1}(r_2)$ = CONF2), r_1 moves to $r_1.Target$ with $r_1.color$ = MOVE. It then updates $r_1.Successor$ and changes $r_1.color$ to WAIT to indicate that it is waiting for the chain to be in the Packed state.

In general, when the chain is in the Packed state, the leader r_1 chooses a free vertex v from $V^2_{r_1}$ as the target if all $v' \in N^2_v \cap V^3_{r_1}$ are either unoccupied or having robots at Finished state. Then, r_1 communicates the directions of $r_1.Target$ to r_2 as follows: r_1 displays DIR color corresponding to $r_1.Target.One$ by setting $r_1.color$. r_2 sees this and stores the direction in $r_2.NextTarget.One$. r_2 confirms that it has seen the first DIR color by displaying CONF color. r_1 confirms that it has seen CONF on r_2 by setting $r_1.color$ to CONFC (confirmation of confirmation). r_2 sees CONFC on r_1 and in turn, sets $r_2.color$ to CONFC to show that it is ready to receive the second DIR color. The second DIR color $r_1.Target.Two$ is displayed by r_1 and is seen by r_2. r_2 stores this second direction in $r_2.NextTarget.Two$ and sets $r_2.color$ to CONF2 to send confirmation that the second direction of r_1 is received. Once r_1 has seen CONF2 in r_2, it moves two hops to $r_1.Target$ in the move phase (which is explained in Section 4.3.2) by setting $r_1.color$ = MOVE. After reaching to $r_1.Target$, r_1 updates $r_1.Successor$ and sets $r_1.color$ = WAIT. Now r_2 sets $r_2.Target$ based on the information stored in $r_2.Predecessor$ as r_2 needs to reach the old position of r_1. It can start moving towards $r_2.Target$ only if it is empty. After the movement, it also updates $r_2.Predecessor$ and $r_2.Successor$.

Finally, when the Leader r_1 can no longer find free vertices to move to, it communicates this information to its follower r_2 using the DIR color that points towards r_2. Then,

[1] The robots are anonymous, and hence do not possess identifiers. The successor and predecessor relationship is established from the direction of movement and the variables essentially store the port numbers corresponding to the location of the successor or predecessor.

r_1 changes its color to FINISH and goes into the Finished state, and r_2 becomes the Leader and continues exploring the graph.

Example Execution of IND Algorithm: Now, we illustrate the execution procedure of Algorithm IND with Fig. 4.3. The location of the Door and the initial setting are shown in Fig. 4.3(a). The robot r_1 first appears at the Door in Fig. 4.3(b). As soon as r_1 moves away from the Door, robot r_2 appears there and sets r_1 as its predecessor. This situation is shown in Fig. 4.3(c). In Fig. 4.3(d), r_1 reaches its target vertex but r_2 does not move until r_1 has moved away from its current position. r_1 reaches a new vertex and r_2 follows it, r_3 appears at the Door and follows r_2 which is shown in Fig. 4.3(e). Observe that r_1 is stuck so it transfers its leadership to its follower, i.e., r_2. In Fig. 4.3(f), r_2 reaches a new target vertex and its followers follow.

4.3.3 Analysis of IND Algorithm

In this section, we present a few lemmas and theorems that establish the behaviour of the robots and the correctness of the algorithm. We also analysed the time complexity of the IND algorithm.

Lemma 4.3.4. *There can be at most one Leader robot, and the Leader robot r_1 moves to a free vertex $v \in V^2_{r_1}$ such that every $v' \in N^2_v \cap V^3_r$ is either unoccupied or occupied by robots in Finished state.*

Proof. In the rules for *Transferring the Leadership*, when a Leader r_1 signals to its successor r_2 that it is stuck, r_2 can become the Leader only after the previous Leader r_1 has switched to Finished state (recognized by FINISH color on r_1). The first robot placed becomes the Leader, and the robots appearing next can become a Leader only after the previous one reaches the Finished state. Therefore, there can be at most one Leader at any time during the dispersion.

A free vertex has none of its adjacent vertices occupied by a robot. As the current Leader r_1 can only move when the chain is in a Packed state, it chooses a vertex in $V^2_{r_1}$ that is free by checking all the vertices adjacent to potential target vertex are not occupied by any robot. This can be done since the robots have a visibility range of 3. Further, there is no possibility

Algorithm 1: IND

```
1  if r.State is Leader then
2      if r.Target is not set then
3          if Color_r(r.Successor) = CONF3 then
4              if ∃ a vertex v ∈ V_r^2 such that all v' ∈ V_r^3 ∩ N_v^2 is either unoccupied or
                occupied with robots in Finished state then
5                  r sets v as r.Target by setting r.Target.One and r.Target.Two
6                  COMMUNICATE()
7                  UPDATE()
8                  Set r.color = WAIT
9                  r clears r.Target by clearing r.Target.One and r.Target.Two
10                 PACKED_STATE()
11             else
12                 LEADERSHIP_TRANSFER()
13 else if r.State is Follower then
14     if r.NextTarget is not set then
15         Receive()
16     else
17         if r.Target is set then
18             COMMUNICATE()
19             UPDATE()
20             PACKED_STATE()
21 else // r.State is None & r.color =OFF
22     if r does not find any other robot then
23         Change r.State to Leader
24         Find a vertex v ∈ V_r^2
25         Set v as r.Target by setting r.Target.One and r.Target.Two
26         Set r.color = MOVE
27         r moves to r.Target // r moves to r.Target.Two
28         Set r.color = WAIT
29     else
30         r stores the port numbers in r.Predecessor corresponding to the nearest
            robot
31         Change r.State to Follower
32         if Color_r(r.Predecessor) = WAIT & r.color =OFF then
33             Set r.color = CONF3
```

Algorithm 2: COMMUNICATE()

if $r.Target$ *is set* & $Color_r(r.Successor) =$ `CONF3` **then**
  Set $r.color$ to match $r.Target.One$ // Showing the first DIR color
else if $r.Target$ *is set* & $Color_r(r.Successor) =$ `CONF` **then**
  Set $r.color =$ CONFC // Confirming the confirmation sent by successor
else if $r.Target$ *is set* & $Color_r(r.Successor) =$ `CONFC` **then**
  Set $r.color$ to match $r.Target.Two$ // Showing the second DIR color
else // $r.Target$ is set and empty & $Color_r(r.Successor) =$CONF2
  Set $r.color =$ MOVE // Moving towards the target
  Move to $r.Target$

Algorithm 3: RECEIVE()

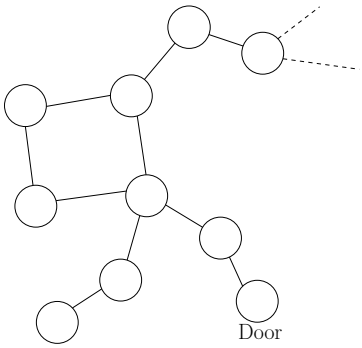
if $r.color =$ `CONF3` & $Color_r(r.Predecessor)$ *is a* `DIR` *color* **then**
  Store that shown color as $r.NextTarget.One$
  Set $r.color =$ CONF // Confirmation for the first DIR color
else if $r.color =$ `CONF` & $Color_r(r.Predecessor) =$ `CONFC` **then**
  Set $r.color =$ CONFC // Ready to receive the second DIR color
else // $r.color =$ CONFC & $Color_r(r.Predecessor)$ is a DIR color
  Store that shown color as $r.NextTarget.Two$
  Set $r.color =$ CONF2 // Confirmation for the second DIR color

Algorithm 4: PACKED_STATE()

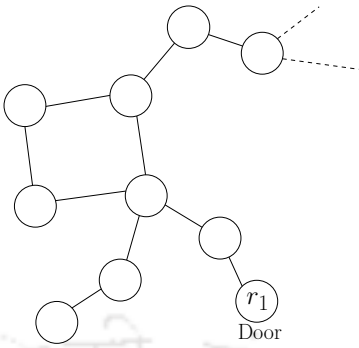
if $r.color =$ `MOVE` & $Color_r(r.Successor) =$ `OFF` **then**
  Set $r.color =$ CONF3 // For $r.Successor$ being at the Door
else if $r.color =$ `MOVE` & $Color_r(r.Successor) =$ `CONF3` **then**
  Set $r.color =$ CONF3 // For any other pair of predecessor and successor
else // $r.color =$ WAIT & $Color_r(r.Successor) =$ CONF3
  return // For the leader and its successor

Algorithm 5: UPDATE()

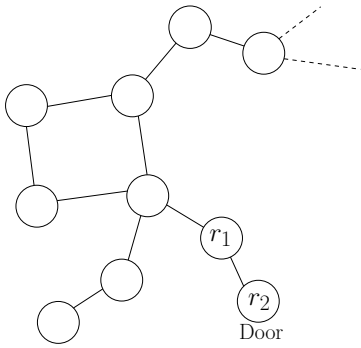
$r.Successor.p_1 = r.Target.One$
$r.Successor.p_2 = r.Target.Two$
if $r.State \neq Leader$ **then**
  $r.Predecessor.p_1 = r.NextTarget.One$
  $r.Predecessor.p_2 = r.NextTarget.Two$
Clear $r.NextTarget.One$ and $r.NextTarget.Two$



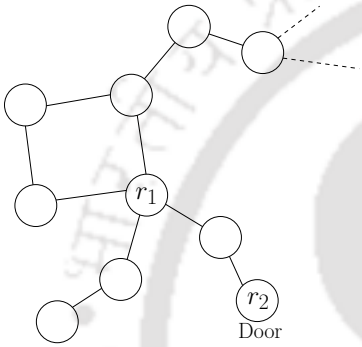
(a) Location of the Door vertex and the initial setting of the graph



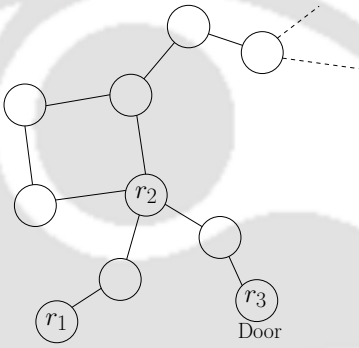
(b) The robot r_1 appears at the Door and becomes the Leader



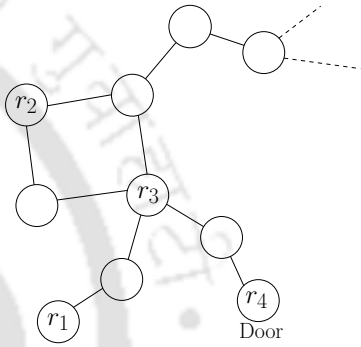
(c) As soon as r_1 moves away from the Door, r_2 becomes its follower



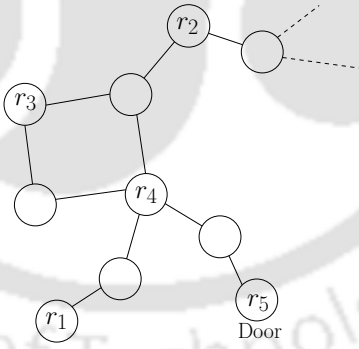
(d) r_1 reaches its target vertex after the movement from the Door vertex



(e) r_1 transfers its leadership to r_2 as there are no free vertex for it.



(f) r_2 is now the new Leader and moves to new free vertices and r_3 follows r_2



(g) r_2 keeps exploring the graph.

Figure 4.3: An example execution of the IND algorithm

of this target vertex or any vertex adjacent to it being occupied by any other robot as the Leader is allowed to move only when the chain is in the Packed state. If there is a free vertex $v \in V_{r_1}^2$ as a potential target, then every 2 hop neighbour v' of v ($v' \in N_v^2 \cap V_{r_1}^3$), that is visible to r_1 needs to be either free or occupied by robots in Finished state. Due to this condition,

Algorithm 6: LEADERSHIP_TRANSFER()

```
if r.Successor is set then
    r sets DIR color to point towards its Successor
    r.color =FINISH
    Change r.State to Finished
else
    r.color =FINISH
    Change r.State to Finished
```

a chain does not cross itself, as shown in Fig. 4.5. If no such free vertex v is found, then r_1 transfers the leadership to r_2 and switches to the Finished state. □

Corollary 4.3.5. *The leader r moves to a free vertex v such that every vertex $v' \in N_v^2 \cap V_r^3$ is either unoccupied or occupied by a robot in Finished state. Consequently, the chain never crosses itself.*

Lemma 4.3.6. *The Robots do not collide.*

Proof. The consecutive robots in the chain establish a predecessor-successor relationship between them if they are not in the Finished state. Only the current leader has no predecessor. Considering a predecessor and successor pair, the predecessor robot moves first after communicating the movement direction. The successor starts moving towards the old position of its predecessor only if the vertex is empty. For any robot r, $r.Successor$ stores the port numbers corresponding to the direction of its successor, using which r avoids moving in the direction of its successor. Hence, none of the predecessor-successor pairs can meet a collision. Moreover, Corollary 4.3.5 ensures that the chain of robots does not cross itself, and hence the collision never happens. □

Lemma 4.3.7. *No two robots in the Finished state occupy adjacent vertices.*

Proof. A robot can go to the Finished state only from the Leader state. From Lemma 4.3.4, we know that a Leader robot moves only to free vertices. As only free vertices are occupied, the robots are not present in adjacent nodes. □

From Lemma 4.3.7, we can say that when a robot enters the Finished state, none of

the vertices adjacent to it are occupied by a robot, which means eventually, all the vertices occupied by robots form an independent set.

Remark 4.3.8. If a vertex is occupied by a robot in the packed state, it remains occupied thenceforth.

Theorem 4.3.9. *Algorithm* IND *fills a maximal independent set.*

Proof. From Lemma 4.3.7, the filled vertices form an independent set. Suppose the independent set is not maximal. As the graph is connected, there exists a vertex v which is free and has a robot r_1 in the Finished state two hops away since the execution of the algorithms is done. Before the robot r_1 entered the Finished state, it was in the Leader state. The Leader r_1 switches into the Finished state in two cases. First, it cannot find any free vertex two hops away. Secondly, it finds a free vertex v such that there exists a neighbour v' of v in $N_v^2 \cap V_{r_1}^3$ occupied by a robot not in Finished state. If all the neighbours of v in $N_v^2 \cap V_{r_1}^3$ are unoccupied or have robots in Finished state, r_1 has a clear path to v. Hence, it cannot switch to Finished state, which leads to a contradiction. If the 2 hops neighbour v' of v has a robot not in Finished state, r_1 does not have a clear path to v. In that case, r_1 transfers the leadership to its successor by communicating the direction pointing towards its successor and eventually the robot on v' will become the leader. Since v' is 2 hops away from v, the robot on v' cannot switch to Finished state as it gets a clear path to the free vertex v, which again leads to contradiction. □

Lemma 4.3.10. *Algorithm* IND *fills an MIS of G with luminous robots having visibility range of 3 hops, $O(\log\Delta)$ bits of memory, and $\Delta+8$ colors.*

Proof. The visibility range of 3 hops for robots executing IND algorithms is necessary. Else, the robots cannot check if the target vertex is free or not.

The robots require $O(\log\Delta)$ bits of memory to store the following: *State* (4 states: 2 bits), *Target* (directions to the target vertex: $2\lceil\log\Delta\rceil$ bits [2]), *NextTarget* (directions to the vertex to which the robot has to move after the *Target* vertex is reached: $2\lceil\log\Delta\rceil$ bits).

The colors used by the robots are Δ colors to show the directions to the target of the robot, including the special color that points in the direction of the Follower to switch to

[2]Two port numbers are needed to be stored corresponding to the movement.

the Finished state. Initially when the robot is placed at the Door, the robot sets its color to OFF. There are four additional colors (CONF, CONFC, CONF and CONF3) for confirming that the robot saw the signalled direction and confirmations of the Predecessor or the Successor and the Packed state is achieved, one color WAIT used when the Leader is waiting for the Packed State, one color MOVE used during the movement and the FINISH color. □

Now we analyze the time complexity of the algorithm in terms of epochs. To find the total time required by the algorithm, we first individually establish the time-bound on the movement of robots. Consider a chain containing i robots $\{r_1, r_2, \ldots, r_i\}$, where r_1 is the Leader and the foremost robot in the chain. The robots $r_1, r_2, \ldots, r_i$ are on alternative vertices on the path from the vertex occupied by the Leader to the Door vertex. Suppose the chain is in the Packed state. We determine the time required for two consecutive movements of the Leader. We first determine the time required for all the robots in the chain to occupy the position of their Predecessor. As a result, a new robot appears at the Door, increasing the size of the chain. Next, we find the time required for the robots in the chain to set their colors to CONF, indicating that they have received the movement direction of their Predecessor. We also find the time required for the chain to reach in the Packed state again after the leader of the chain moves.

Lemma 4.3.11. *Algorithm* IND *takes $O(i)$ epochs for all the robots in a chain of length i to perform one move operation.*

Proof. Once the chain is in the Packed state, the leader robot r_1 communicates the movement direction to its successor r_2 using colors and change its color to MOVE at the end of the compute phase. It then moves to its target vertex in one epoch and r_1 sets its color to WAIT. By the next epoch, r_2 observes that r_1 has left its previous vertex, so it moves to v after communicating movement direction to its successor. In a cascading manner, all Successors move to their Predecessors' location. Thus in the worst case, it takes $O(i)$ epochs for all the robots in the chain to move. □

Lemma 4.3.12. *Algorithm* IND *takes at most $O(i)$ epochs for the robots in a chain of length i to reach the Packed state after movement.*

Proof. As per the algorithm description, the communication between a Predecessor r_p and its Successor r_q by showing a series of seven colors: first DIR color at r_p, CONF at r_q, CONFC at r_p, CONFC at r_q, second DIR color at r_p, CONF at r_q and CONF3 at r_q. A leader can only move when the chain is in the Packed state. For a chain to be in the Packed state, the Successor of the leader robot has to show CONF3 color. The process of communication of colors starts from the new robot r_{i+1} that appears at the Door. It takes at most seven epochs for the communication between r_i and r_{i+1}. Similar communication happens between all i pairs of consecutive robots on the chain. So it takes at most $O(7i) \approx O(i)$ epochs for the chain to reach Packed state. □

Lemma 4.3.13. *Algorithm* IND *takes at most 4 epochs to have a transfer of leadership from a leader to its Successor.*

Proof. Consider the situation where the Leader r_1 cannot find any free vertices two hops away, as shown in Fig. 4.4. If r_1 is at the Door, it sets its color to FINISH and switches to Finished state; the maximal independent set is filled. Otherwise, r_1 switches to the Finished state by setting the DIR colors pointing towards its follower. r_2 sees the DIR color and sets $r_2.color$ to CONF. r_1 sees the CONF color, it switches to color FINISH. r_2 becomes the new Leader once it sees the color FINISH at r_1. In total, this needs at most 4 epochs because of the sequence of colors (DIR, CONF and FINISH) and final state change at r_2. □

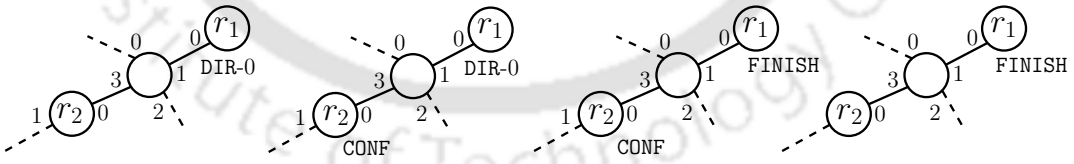


Figure 4.4: Leadership transfer: (First figure from left) illustrating that r_1 is stuck and shows DIR color pointing toward r_2; (second) r_2 shows CONF color; (third) r_1 sets its color to FINISH; (last) r_2 changes its state to Leader.

Theorem 4.3.14. *The algorithm* IND *runs in* $O(N^2)$ *epochs.*

Proof. For an MIS of size N, we can have a chain of length at most N. For each increase in a chain of length i, it takes at most $O(i)$ epochs from Lemma 4.3.11 and 4.3.12. Also, we can have at most N leadership transfers. From Lemma 4.3.13, each transfer of leadership takes

at most 4 epochs. So in total we need, $\sum_{i=1}^{N} O(i) + 4N = O(N^2)$ epochs. Therefore, after at most $O(N^2)$ epochs, an MIS of vertices of the graph becomes filled. □

The corollary below follows from Corollary 1 in [36].

Corollary 4.3.15. *(i) Assume that there are no inactive intervals between the LCM cycles and that every LCM cycle of every robot takes at most t_{max} time. Then the running time of the* IND *algorithm is $O(N^2 t_{max})$. (ii) The* IND *algorithm needs $O(N^2)$ LCM cycles under an FSYNC scheduler.*

4.4 Algorithm for MIS Filling with Multiple Doors

The MULTIND is largely similar to algorithm IND with a few modifications. It works under a *Semi Synchronous* (SSYNC) scheduler, where a subset of robots is activated in each round, where each activated robot finishes its LCM cycle in the same round. We define an epoch similarly as the minimum number of rounds where all robots are activated at least once. Note that an epoch can have a variable number of rounds. The graph has a maximum of k number of Doors. The robots do not have any knowledge about the number of Doors. The visibility range of the robots is five hops. Each robot has $O(\log(\Delta + k))$ bits of persistent memory, and $\Delta + k + 7$ colors where Δ number of `DIR` colors, `CONF`, `CONFC`, `CONF`, `CONF3`, `MOVE`, `OFF`, `FINISH` and k number of `WAIT` colors denoted by `WAIT-1`, `WAIT-2`, ..., `WAIT-`k representing that a robot is waiting as well as their rank. All the robots entering from a particular Door can only display the `WAIT` color corresponding to that Door. The `WAIT` colors can be compared against each other to establish dominance between the robots. Initially, a robot has color `OFF` when it is placed at the Door. The proposed MULTIND runs in O(N^2) epochs.

4.4.1 The MULTIND Algorithm

The Leader robots need to avoid collision with other Leader robots and the follower robots in chains. The robots make use of the hierarchy among the k `WAIT` colors to avoid collision with another Leader robot. The Leader robots also avoid cutting through a chain to avoid

collision with follower robots in another chain. In the multiple Door situation, the Leaders display the WAIT-i color instead of the WAIT color used in a single Door case. In the *Look* phase, if a Leader robot r_i with color WAIT-i sees any other Leader r_j with color WAIT-j such that $j < i$, then we say that the Leader robot r_j *dominates* Leader robot r_i. The Leader robot r_j is said to be the *dominating* and the Leader robot r_i is said to be *dominated*. Note that a dominating Leader robot can be dominated by another Leader robot at the same time. If a Leader robot is not dominated by any other robot, it can choose a target.

The rest of the model is the same as in the Single Door case. The robots can be in any one of these states during execution: *None*, *Leader*, *Follower*, *Finished*. When the robots appear at the Door, they are initialized with the None state and with the color OFF. We define $\mathcal{P}_k(v_i, v_j)$ as the set of vertices that are part of all the paths from v_i to v_j of length k. Note that there can be multiple paths of length k, and we include vertices of all those paths. The additional rules are applicable to the Leader as follows.

Movement of a Leader robot: If a chain is in a Packed state, the corresponding leader r_L chooses a free vertex in $V^2_{r_L}$ as the target in one of the following ways.

- If $V^5_{r_L}$ has a dominating leader r'_L of some other chain, then a free vertex $v \in V^2_{r_L}$ is chosen such that $v \notin \mathcal{P}_5(r_L, r'_L)$. If no such vertex v is found, then r_L transfers the leader to its successor by pointing the direction towards the successor and goes into the Finished state by changing its color to FINISH.

- If r_L is the leader dominating some other leader r'_L in $V^5_{r_L}$, then a free vertex $v \in V^2_{r_L}$ is chosen such that every vertex $v' \in N^2_v \cap V^2_{r_L}$ is either unoccupied or occupied with robot in Finished state. If no such vertex v is present, r_L transfers its leadership to its successor.

- If $V^5_{r_L}$ does not have any other leader robot, then a free vertex is chosen as the target such that every vertex $v' \in N^2_v \cap V^2_{r_L}$ is either unoccupied or occupied with a robot in Finished state. If no such free vertex is there, r_L transfers the leadership to its successor.

After the target is fixed for a leader r_L, it communicates the target to its successor using DIR colors. After getting confirmation from its successor, it moves to the target with color

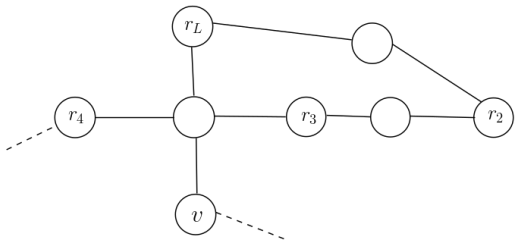


Figure 4.5: A chain $r_L - r_2 - r_3 - r_4 \cdots$ does not cross itself

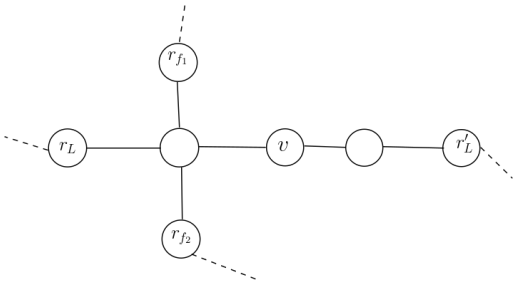


Figure 4.6: Two chains of robots do not cross each other

MOVE and waits for the chain to be in the Packed state with the respective WAIT color. Additionally, irrespective of whether a Leader robot is dominated or not, the target chosen is such that the Leader robot does not cut through a chain while moving to that target vertex, i.e., the vertex between the target vertex and the current vertex does not have more than one of its neighbouring vertices occupied at least one of which is an active robot (a robot not displaying FINISH color) other than the current Leader robot. As shown in Fig. 4.6, r_L and r'_L are the dominating and dominated Leader at an instance and v is a free vertex which is 2 hops away from r_L. r_{f_1} and r_{f_2} are some followers of some other chain. r_L does not reach to v even if r_L is the dominating Leader, as r_{f_1} and r_{f_2} is not in Finished state. As shown in Fig. 4.5, a Leader r_L sees a free vertex v two hops away. However, r_L cannot reach v (self cross) as the two hops neighbouring vertices are occupied by r_3 and r_4.

Example Execution of MULTIND Algorithm: We use an example to illustrate the execution of the MULTIND algorithm. Consider the graph with respective Door positions in Fig. 4.7(a). Door 1 ranks higher in the hierarchy than Door 2, which ranks higher than Door 3. Assume that all the robots are activated in each epoch. The positions of the robots in the next epoch are shown in Fig. 4.7(b). Each Leader robot makes a move. Observe that Leader-1 and Leader-2 are within the visibility range of each other. Similarly, Leader-2 and Leader-3 are within the visibility range. Leader-2 is dominated by Leader-1 and hence, its target vertex should not lie on the path to Leader-1. A similar argument applies to Leader-3, which is dominated by Leader-2. Fig. 4.7(c) shows the positions of the robots after the next epoch. All the Leaders are now stuck and have to transfer their Leadership. The result of that is shown in Fig. 4.7(d). Leader-2 has no possible target vertices and has to transfer its

leadership again. Leader-3 is dominated by Leader Leader-2. In the next epoch shown in Fig. 4.7(e), Leader-1 and Leader-3 make a move. Leader-2 is at its Door. In the next epoch, Leader-2 goes into *Finished* state. Similarly, in the next epoch, Leader-1, and Leader-3 transfer their leaderships which is shown in Fig. 4.7(f). Leader-3 makes one final move in Fig. 4.7(g).

4.4.2 Analysis of MULTIND Algorithm

Lemma 4.4.1. *A Leader robot always moves to and occupies free vertices.*

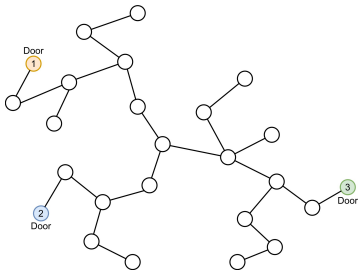
Proof. Since the robots have a visibility range of 5 hops, a Leader robot can choose a target vertex such that none of its neighbouring vertices are occupied by a robot that is not a Leader. Also, two leaders do not occupy adjacent vertices. For a leader r_L, if there is another Leader r'_L within 5 hops visibility of r_L, one of them dominates the other. Let r_L be the dominating robot. MULTIND algorithm ensures that r'_L does not choose a vertex as its target in the path where it is being dominated. So, either r'_L finds a target on some other path where is not getting dominated or transfers the leadership (in case of no such target is there). So, the dominating Leader r_L can choose a free vertex as its target from the path $\mathcal{P}_5(r_L, r'_L)$. Hence, the robots cannot choose target vertices such that they are adjacent. □

Lemma 4.4.2. *The Robots do not collide.*

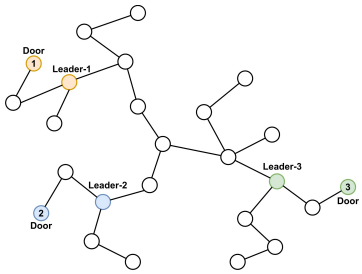
Proof. The proof for collision avoidance within a chain is same as Lemma 4.3.6 for a single Door case. In the *Look* phase of the robots, if a Leader robot encounters another Leader robot within its visibility range, one of the robots gets dominated and will choose a target such that it does not lie on any path (of maximum length of 5 hops) connecting the two Leader robots. Thus, any two Leader robots avoid collision with each other. A Leader robot avoids collision with robots in another chain by avoiding cutting through a chain. When a leader robot finds a free vertex v, it checks every vertices in $N^2_v \cap V^2_{r_L}$. If at least one of the vertex in $N^2_v \cap V^2_{r_L}$ is occupied by a robot not in Finished state, the Leader does not move. This prevents the crossover of two chains. Hence, a leader cannot collide with a follower robot from another chain. □

Algorithm 7: MULTIND

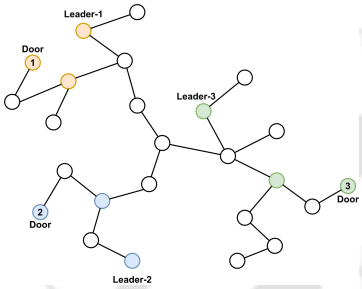
```
1  if r.State is Leader then
2      if r.Target is not set then
3          if r.Entry is set and Color_r(r.Successor) = CONF3 then
4              MULTIND_FINDTARGET()
5              COMMUNICATE()
6              UPDATE()
7              Set r.color = WAIT-i // Wait color corresponding to the Door-i
8              r clears r.Target by clearing r.Target.One and r.Target.Two
9              PACKED_STATE()
10 else if r.State is Follower then
11     if r.NextTarget is not set then
12         RECEIVE()
13     else
14         if r.Target is set then
15             COMMUNICATE()
16             UPDATE()
17             r clears r.Target by clearing r.Target.One and r.Target.Two
18             PACKED_STATE()
19 else // r.State is None & r.color =OFF
20     if r does not find any other robot then
21         Change r.State to Leader
22         Set r.color = Wait-i // Wait color corresponding to the Door-i
23         MULTIND_FINDTARGET()
24         Set r.color = MOVE
25         Move to r.Target
26         Set r.color = WAIT-i
27     else
28         r stores the port numbers in r.Predecessor corresponding to the nearest
            robot
29         Change r.State to Follower
30         if Color_r(r.Predecessor) = WAIT-i & r.color =OFF then
31             Set r.color = CONF3
```



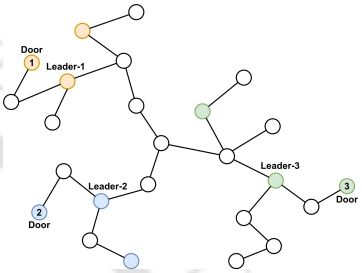
(a) Location of the Doors and initial setting



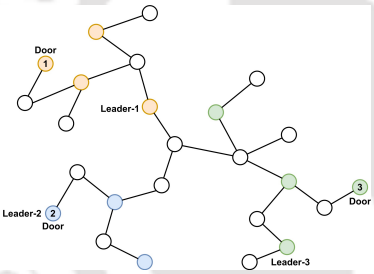
(b) All the Leaders make a move



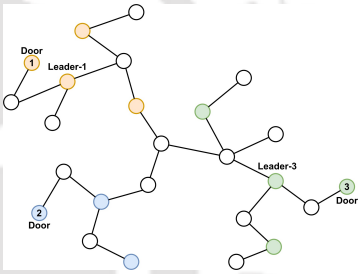
(c) Leader-2 and Leader-3 avoid moving toward their dominating robot



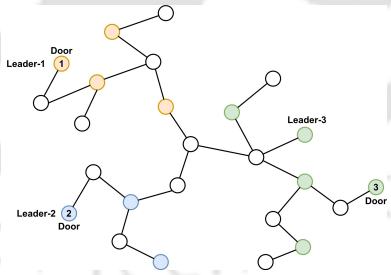
(d) Leadership transfer for each chain of robots occurs



(e) Leader-1 and Leader-3 make a move. Leader-3 avoids moving toward Leader-1



(f) Leadership transfer occurs again after the current leader is stuck



(g) Leader-3 makes one final move

Figure 4.7: An example execution of the MULTIND algorithm

Consequently, we can state following corollary.

Corollary 4.4.3. *A chain does not cross itself. Moreover, two chains do not cross each other.*

Lemma 4.4.4. *No two robots in the Finished state occupy adjacent vertices.*

Algorithm 8: MULTIND_FINDTARGET()

```
1  if ∃ a dominating leader r' ∈ V_r^5 then
2      if ∃ a free vertex v ∈ V_r^2 such that v ∉ P(r, r') then
3          Set v as r.Target
4      else // No such free vertex to move to
5          LEADERSHIP_TRANSFER()
6  else
7      if ∃ a free vertex v ∈ V_r^2 such that all v' ∈ N_v^2 ∩ V_r^2 is either unoccupied or occupied
         with robots in Finished state then
8          Set v as r.Target
9      else
10         LEADERSHIP_TRANSFER()
```

Proof. A robot goes into the Finished state only after becoming a Leader. According to Lemma 4.4.1, a Leader only moves to and occupies free vertices. Hence, when a robot enters the Finished state, none of the vertices adjacent to it are occupied. □

Theorem 4.4.5. *Let G be a connected graph G with k Doors. Algorithm* MULTIND *fills an MIS of vertices in $O(N^2)$ epochs of G under an SSYNC scheduler, without collisions, by mobile luminous robots having the following capabilities: visibility range of 5 hops, persistent storage of $O(\log(\Delta+k))$ bits, and $\Delta+k+7$ colors.*

Proof. By Lemma 4.4.2 and Lemma 4.4.4 and the similar arguments described in Theorem 4.3.9 for the single door case, the filled vertices in G form a MIS and the filling is done without collisions. The robots require $O(\log(\Delta+k))$ bits of memory to store the following: *State* (4 states: 2 bits), *Target* (directions to the target vertex: $2\lceil\log\Delta\rceil + \lceil\log k\rceil$ bits), *NextTarget* (directions to the vertex to which the robot has to move after the *Target* vertex is reached: $2\lceil\log\Delta\rceil$ bits).

The colors used by the robots are Δ colors to show the directions to the target of the robot, which also acts as a special color to switch to the Finished state. Initially, when a robot is placed for the first time at the Door, it is colored with `OFF`. There are k numbers of `WAIT` colors - one for each Door. There are three additional colors (`CONF`, `CONFC`, and `CONF2`) for confirming that the robot saw the signalled direction and confirmations of the

Predecessor or the Successor, one color MOVE used during the movement, one color CONF3 to indicate that the chain is in Packed state and the FINISH color.

Consider a graph where all the k vertices that are connected to the Doors node form a clique. In this case, only the Leader corresponding to the highest color Door would occupy one of the nodes of the k-clique. In this particular case, only one of the Doors remains active, and robots at all other Doors would go into the Finished state. So, the multi-door case would behave as a single-door case and thus replicate the time-bound for a single-door case. Thus, from Theorem 4.3.9 it follows that, MULTIND solves the *MIS Filling problem* in a graph with multiple Doors in $O(N^2)$ epochs. □

4.5 Discussion

4.5.1 Lower bound for MIS Filling with Single Door

Theorem 4.5.1. *MIS Filling problem with Single Door takes at least* $\Omega(n)$ *time using robots having* 3 *hops of visibility.*

Proof. The robots enter the graph one by one. A new robot can enter the graph through the door when the current robot at the door moves away. Since, each robot is equipped with 3 hops of visibility, robots need to visit at least $n-3$ vertices in worst case, to see whole graph. For example, if the graph is a line graph with n vertices $v_1, v_2, \cdots v_n$ and the door is placed at one endpoint v_1 of the graph, then none of the robot can see v_n by visiting till the vertex v_{n-4}. Since, the robots enters the graph one by one, it takes at least $n-3$ epochs to explore the graph with robots having 3 hops of visibility. □

4.5.2 On the requirement of the colors

We need at least Δ colors. If there are less than Δ colors, there exists two ports that are marked by the same color, resulting in a collision on movement to an already visited port, or one port may never get visited. In the absence of k colors corresponding to the k Doors, it may lead to a deadlock or collision between different chains. We use only a constant number of colors for communicating the directions for the robot movements.

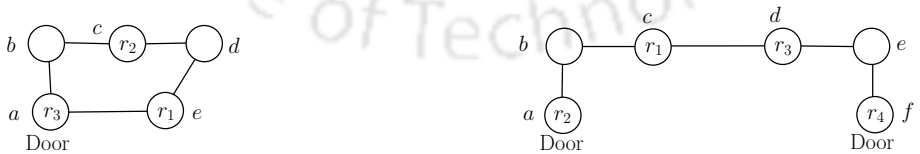
While we need Δ colors to communicate the port numbers, we can minimize the number of colors at the cost of increasing round complexity. We can use Hoffman encoding to reduce the number of colors used for the port numbers from Δ to a constant number of colors. Now, a sequence of colors would represent a port number instead of a particular color and increases time complexity by a factor of the size of the largest encoding.

4.5.3 One hop vs Two hops movement

Our model considers that a robot moves two hops in one LCM cycle. While moving, the color of the robot is set to `MOVE`. However, we can easily avoid this 2 hops movement of a robot by replacing the color `MOVE` with two colors, `MOVE1` and `MOVE2`. After a robot chooses its target, it sets its color to `MOVE1` for the first hop and then changes its color to `MOVE2` before reaching the target.

4.5.4 Minimality of visibility range

For the single Door case, a robot having a visibility range of two fails to avoid placing robots in adjacent nodes. Consider a graph as shown in Fig. 4.8(a). Initially, a robot appears at the Door vertex A; then it moves to C, a vertex two hops away. If the robots only have a visibility range of two, then r_1 can go to E without realizing that E and A are connected, resulting in a configuration that is not an independent set. With a visibility range of two, a robot cannot determine whether a robot is present at the neighbour of the target vertex. Hence, we need a visibility range of three for a single-door case.



(a) Single Door and 2 hops of visibility

(b) Multiple Doors and 4 hops of visibility.

Figure 4.8: Robots occupies adjacent vertices with 2 and 4 hops visibility in Single and Multiple Door cases respectively.

Similar to the single Door case, consider the graph in Fig. 4.8(b) for the multi-Door case. If the robots have a visibility range of four, then the robots at A and F may simultaneously

move to occupy C and D and result in a configuration with robots occupying adjacent vertices.

4.6 Conclusion

We presented two collision-free algorithms for solving two versions of the problem of filling a maximal independent set of vertices in an arbitrary connected graph using luminous mobile robots. The first algorithm, IND, for graphs with a single Door, works under an asynchronous scheduler. It uses robots having three hops of visibility range, $\Delta + 8$ number of colors, and $O(\log \Delta)$ bits of persistent storage and solves the problem in $O(N^2)$ epochs. The second algorithm, MULTIND, works in graphs with k (> 1) Doors. It forms an MIS under a semi-synchronous scheduler using robots with five hops of visibility range, $\Delta + k + 7$ number of colors, and having $O(\log(\Delta + k))$ bits of persistent storage, completing in $O(N^2)$ epochs. We explain both the algorithms by example execution with figures and give the corresponding pseudocodes. We show a lower bound of $\Omega(n)$ for the case of single Door, where n is the number of vertices of the graph. We further discuss the requirement of the colors and the minimality of the visibility range of the robots for both models. It is open to extend the second algorithm to the generalized asynchronous scheduler. The model of attaching splitting Doors to a graph is a new direction in multi-robot coordination problems, and many other graph problems can be explored under the same model.

Chapter 5

Mutual Visibility Despite Angular Inaccuracy

5.1 Introduction

One of the fundamental problems in swarm robotics is the *Mutual Visibility Problem*. From any initial configuration of the robots on the Euclidean plane, the problem aims to arrange the robots in a configuration such that any two robots are visible to each other. This chapter studies the problem using N opaque luminous point robots with inaccurate movements. An inaccuracy of a robot is defined as an angular deviation from its target point. To the best of our knowledge, this is the first work that considers inaccuracy in robots' movement to investigate the mutual visibility problem. Along with obstructed visibility and obliviousness of the robots, inaccurate movement adds up the challenge for the problem. Inaccuracy may occur at any time due to some technical failure in robots (for example, misalignment of the direction of motion). In addition to that, we also study the problem by considering the mobility failure of the robots.

So far, the mutual visibility problem has been investigated under different models in the literature but has not been studied under the assumption of inaccurate movement. Most of the solutions provided heavily rely on the concept of the convex hull, where the aim is to arrange the robots at the vertices of a convex hull. Starting from a configuration, robots

either expand or contract the convex hull with their movements. When all the robots become the vertices of a convex hull, three robots cannot lie on the same line, which leads to mutual visibility. But, when the movements are not accurate, the concept fails to solve the problem, as the inaccurate movement may distort the convex hull itself. In the presence of inaccurate movements of the robots, we cannot guarantee that all robots will be on the vertices of a convex hull. A different type of solution is proposed by Poudel et al. [54] under the assumption of one axis agreement, where robots find a polygonal region on the plane, called *visible area*. The idea is to place a robot in its corresponding *visible area* while others are stationary to avoid collinearities with other robots. This also does not solve the problem in case of inaccurate movement of the robots, as a robot might move to a point that is not a part of its *visible area*. The same logic applies when the problem is studied under fault tolerance with inaccuracy. When we integrate inaccuracy with the fault model defined in [54], the existing solution fails to achieve mutual visibility.

5.1.1 Contributions

Our contributions in this chapter are listed below.

- We propose an algorithm SSYNC_MV_INACC for the SSYNC setting that solves the mutual visibility problem in the presence of robots with inaccurate movements under one axis agreement. The algorithm uses 2 colors which are optimal in terms of the number of the colors, and runs in $\mathcal{O}(N)$ epochs where N is the number of robots.
- We further present an algorithm ASYNC_MV_INACC in ASYNC settings, keeping all other model components intact. This algorithm uses 3 colors and runs in $\mathcal{O}(N)$ epochs.
- We propose a fault-tolerant algorithm FAULT_MV_INACC that can tolerate any number of mobility faults along with inaccurate movements. The algorithm takes $\mathcal{O}(N)$ epochs using 10 colors in the ASYNC settings and 9 colors in the SSYNC settings.
- We also discuss the impossibility when the angular inaccuracy θ_r is greater than or equal to 90° for any robot r.

5.2 Model and Preliminaries

Robots: We consider a set of N autonomous, anonymous and homogeneous point robots $R = \{r_1, r_2, \cdots r_N\}$ on the Euclidean plane, where r_i denotes a robot for $i = 1, 2, \cdots N$. They are opaque and operate in LCM cycles. All the robots are oblivious. There are no means of communication except for a persistent externally visible light on them, which can take any color from a constant-sized color set. None of the robots have knowledge about N. The robots perceive all the points on the plane by their local coordinate system. However, we assume that the robots agree on one coordinate axis. Without loss of generality, we consider the y-axis. This means that the robots know the positive and negative directions of the y-axis, but they do not know the direction of the x-axis. The positive and negative direction of the x-axis might be different for different robots. This enables a robot to determine the horizontal and vertical lines passing through it.

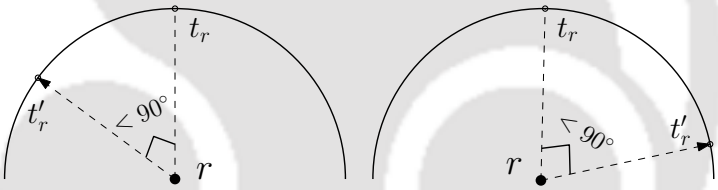


Figure 5.1: Illustrating the inaccurate movement of a robot r_i which chooses to move to t_r but moves to t'_r

Inaccurate Movement: For any robot r, the inaccuracy in movement is defined as an angular deviation from the intended target to which it has chosen to move. As shown in Fig. 5.1, if r decides to move to the point t_r, it actually moves to the point t'_r due to the inaccuracy in its movement. We call the angle $\theta_r = \angle t'_r r\, t_r$ the *angular inaccuracy* of the robot r. The angle θ_r is assumed to be less than 90°. Two robots don't need to have the same angular inaccuracy.

Configuration: We define a configuration C_t to be the set $\{(r_1^t, r_1^t.color), (r_2^t, r_2^t.color), \cdots, (r_N^t, r_N^t.color)\}$, where r_i^t is the position of the robot r_i at time t (≥ 0) and $r_i^t.color$ is the color shown by the light on the robot r_i at the time t. Since the visibility of the robots can be obstructed, all the robots might not be visible to a particular robot. So we define the local configuration for a robot $r_i \in R$ at any time t to be set of all tuples of the form

$(r_j^t, r_j^t.color)$, where r_j is a visible robot to r_i. Since a robot is visible to itself, the local configuration for the robot r_i includes the robot itself. For simplicity, we write r_i and $r_i.color$ instead of r_i^t and $r_i^t.color$ respectively. We misuse the notation r_i to represent the robot r_i and the current location of it.

Notations: Here, we mention the notations and conventions that we follow throughout this chapter.

- The y-coordinate of the location of the robot r_i is denoted by $r_i.y_{\text{coord}}$.
- For two robots r_i and r_j, $\overline{r_i r_j}$ is the line segment joining them, whereas $\overleftrightarrow{r_i r_j}$ represents the infinite line passing through r_i and r_j.
- For any non-perpendicular line (a line which is not parallel to the y-axis) L passing through the robot r, we call the open half plane delimited by the line L that contains the positive y-axis of the local coordinate system of r as the *upper half plane* and denote it by H_L^{Up}. Similarly, the open half plane delimited by the line L that contains the negative y-axis of the local coordinate system of r is called the *lower half plane* and denoted by H_L^{Low}.
- The shortest distance between a point r_i and a line $\overleftrightarrow{AB}$ is the perpendicular distance between them and denoted by $d(r_i, \overleftrightarrow{AB})$. We use the same notation $d(r_i, r_j)$ to represent the shortest distance between the two points r_i and r_j.
- For convenience, we use the notation $\mathcal{VAR}_r^{Prev}$ to denote any variable $\mathcal{VAR}_r$ in the previous LCM cycle for any robot r.

Collinearity: Any three robots r_i, r_j and r_k are collinear if they lie on the same line. We denote this collinearity by $\overline{r_i r_j r_k}$. For three collinear robots r_i, r_j and r_k, we call the infinite line passing through them as the *line of collinearity* (we use LOC henceforth), and it is represented by $\overleftrightarrow{r_i r_j r_k}$. We classify collinearity into two types. The first one is *horizontal collinearity*, where the LOC $\overleftrightarrow{r_i r_j r_k}$ is a horizontal line. The second type is *non-horizontal collinearity* in which $\overleftrightarrow{r_i r_j r_k}$ is a non-horizontal line. In the latter case, if $r_i.y_{\text{coord}} > r_j.y_{\text{coord}} > r_k.y_{\text{coord}}$, then r_i, r_j and r_k are called the *first*, *middle* and *last robot*

of collinearity, respectively. Notice that the middle robot r_j of the collinearity $\overline{r_i r_j r_k}$ is the only robot that can detect the collinearity because both the robots r_i and r_k are visible to it. This enables the middle robot to determine the LOC.

Now, we formally define the problem statement as follows.

Problem 2 (Mutual Visibility with Inaccurate Movement): We are given a set of N oblivious opaque point robots on the Euclidean plane. Each robot operates in LCM cycles and has a persistent light on it that can assume a fixed number of colors. They agree on one coordinate axis. The movement of the robots can be inaccurate with an angular deviation of less than 90° from their target position. Starting from any arbitrary configuration, the robots must reposition themselves such that no three robots are collinear.

5.3 Mutual Visibility with Inaccurate Movement Under SSYNC Settings

In this section, we present Algorithm SSYNC_MV_INACC that solves the mutual visibility with robots having inaccurate movements. The algorithm is collision-free and runs under the SSYNC setting. The algorithm is the same for every robot, and we describe the algorithm for a robot r. There are two colors used in this algorithm: `OFF` (initial color) and `FINISH` (used at termination).

5.3.1 Highlevel Idea of the Algorithm

In the initial configuration C_0, the robots are located at distinct points on the plane, each with color `OFF`. A common approach in the existing literature addressing the mutual visibility problem involves arranging all robots on the vertices of a convex hull. However, this approach is impractical when considering robots with inaccurate movements. To address this limitation, we have developed a technique aimed at sequentially relocating the robots to break the collinearities. In this process, we ensure that no robot inadvertently creates new collinearities after its movements with two other robots with which it was not collinear. Additionally, we assume y-axis agreement where robots are aware of the north

(positive y-axis) and south (negative y-axis) directions. Upon activation, a robot r needs to check whether it is eligible to execute a movement. This can be done by checking the colors of all the visible robots located in the northward direction. If all of them exhibit the color `FINISH`, r checks if it is a part of some collinearity or not. We take advantage of the fact that a middle robot of a collinearity always detects the collinearity. If r finds itself as a part of multiple collinearity, it chooses a target to move vertically northward from its current location. It is possible that r remains collinear with two robots due to the inaccurate movement. We make sure that r is a part of not more than one collinearity, after this movement. When r detects itself in exactly one collinearity $\overline{r_i r r_j}$ for two other robots r_i and r_j, it executes another movement with respect to the line $\overleftrightarrow{r_i r_j}$ to break the collinearity. In case r does not detect any collinearity, it changes its color to `FINISH` and terminates.

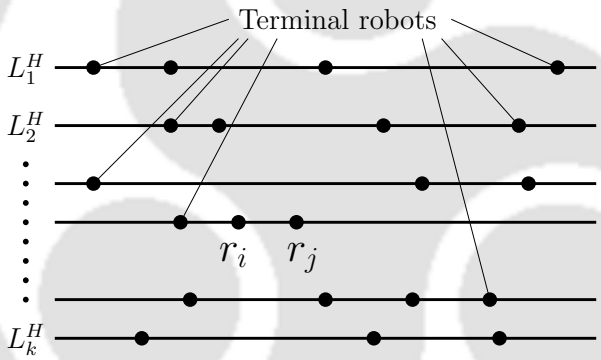


Figure 5.2: The horizontal lines $L_1^H, L_2^H, L_3^H, \cdots, L_k^H$ pass through at least one robot in the initial configuration. The robots r_i and r_j are neighbours of each other.

5.3.2 Description of the Algorithm

Initially, all robots are static and arbitrarily located at different points on the plane with color `OFF`. As a result of an agreement on the y-axis, we can assume that all the robots lie on k ($\leq N$) distinct horizontal lines $L_1^H, L_2^H, \cdots, L_k^H$. Each line passes through at least one robot on the plane. As shown in Fig. 5.2, these k lines cover all the robots on the plane. We call two robots r_i and r_j neighbours if they lie on the same horizontal line and are visible to each other. A robot r is called a *terminal* robot on a horizontal line L_j^H if r has at most one neighbour.

Algorithmic Preliminaries: We list some of the important notations that we will use to describe our algorithm in this chapter.

- L_r^H and $L_r^\perp$ are the horizontal and perpendicular lines passing through the robot r.
- $\mathcal{V}_r = \{r_i \in R \mid r_i \neq r$ and $\nexists\, r_j \in R$ such that $\overline{r r_j r_i}\}$ is the set of all visible robots to r.
- $\mathcal{NB}_r = \{r_i \in \mathcal{V}_r \mid L_{r_i}^H = L_r^H\}$ is the set of neighbours of r. Note that there can be at most two neighbours for any robot r.
- $\mathcal{M}_r = \{m_{r_i} \mid m_{r_i}$ is the midpoint of the line segment $\overline{r r_i}$ for $r_i \in \mathcal{NB}_r\}$ is the set of midpoints of the line segments joining r and its neighbours. For example, $\mathcal{M}_r = \{m_{r_1}, m_{r_2}\}$ in Fig. 5.3.
- $\mathcal{NCV}_r = \{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \nexists\, r_j \in \mathcal{V}_r$ such that $\overline{r_i r r_j}\}$ is the set of all non-neighbour visible robots to r such that no two robots from this set forms a collinearity with r where r is the middle robot of that collinearity. For example in Fig. 5.3, $\mathcal{NCV}_r = \{r_k\}$.
- $\mathcal{CV}_r = \{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \exists\, r_i^c \in \mathcal{V}_r$ such that $\overline{r_i r r_i^c}\}$ is the set of all non-neighbour visible robots to r such that for every $r_i \in \mathcal{CV}_r$, there exists a unique robot $r_i^c \in \mathcal{CV}_r$ where r is the middle robot of the collinearity $\overline{r_i r r_i^c}$. In this case, we call r_i^c the *conjugate robot* of r_i for the collinearity $\overline{r_i r r_i^c}$. Intuitively, $\mathcal{CV}_r$ is the set of all visible robots with whom r is non-horizontally collinear as a middle robot of collinearity. So, in this set, robots appear in pairs. For example, $\mathcal{CV}_r = \{r_l, r_l^c, r_j, r_j^c\}$, as illustrated in Fig. 5.3, since r is a middle robot of two collinearities $\overline{r_l r r_l^c}$ and $\overline{r_j r r_j^c}$.

We now define an *eligible robot* as follows, who can execute a movement according to the algorithm. Note that all other robots who are not eligible, wait till they become eligible.

Definition 5.3.1. (Eligible Robot): Any robot r is called an eligible robot if it satisfies all of the following conditions. (1) All robots in $H_{L_r^H}^{Up}$ are with color `FINISH`. (2) $r.color$ is set to `OFF`. (3) r is either a terminal robot or has two neighbours, out of which at least one is with color `FINISH`.

Table 5.1: List of Notations used in this chapter

Notation	Specification
$\overline{r_i r_j r_k}$	The robots r_i, r_j and r_k are collinear
$\overleftrightarrow{r_i r_j r_k}$	The line of collinearity for the robots r_i, r_j and r_k
H_L^{Up}	Open upper half plane delimited by a non-perpendicular line L passing through r that contains the positive y-axis of r
H_L^{Low}	Open lower half plane delimited by a non-perpendicular line L passing through r that contains the negative y-axis of r
$d(r_i, \overleftrightarrow{AB})$	Shortest distance between r_i and the line $\overleftrightarrow{AB}$
L_r	The horizontal line passes through r
$\mathcal{V}_r$	The set of all visible robot to r
$\mathcal{NB}_r$	The set of neighbors of r
$\mathcal{M}_r$	The set of mid-points of the line segments joining r and its neighbors
$\mathcal{NCV}_r$	$\{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \nexists\, r_j \in \mathcal{V}_r$ such that $\overline{r_i r r_j}\}$
$\mathcal{CV}_r$	$\{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \exists\, r_i^c \in \mathcal{V}_r$ such that $\overline{r_i r r_i^c}\}$
$R(r)$	$\min\{d(r, \overleftrightarrow{r_i r_j}) \mid r_i \neq r_j \neq r_i^c; r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r; r_j \in \mathcal{NCV}_r \cup \mathcal{CV}_r \cup \mathcal{M}_r\}$
$D(r)$	$\min\{d(r, L_{r_i}) \mid r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r\}$
$N(r)$	$\min\{d(r, m_{r'}) \mid m_{r'} \in \mathcal{M}_r\}$
$Radius_r$	$\frac{1}{2}\min\{R(r), D(r), N(r)\}$

The subsequent strategy is followed by a robot r in the compute phase of its LCM cycle to calculate the distance ($Radius_r$) of the target point from its current location.

Strategy FIND_TARGET_DISTANCE: Based on the data gathered from the snapshot taken in the look phase of the current activation cycle, r computes three distances $R(r), D(r)$ and $N(r)$ which are defined as follows.

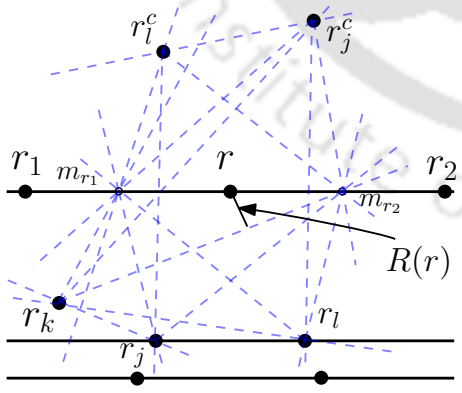


Figure 5.3: Illustrating $R(r)$ by which r avoids moving to any of the blue lines not to create new collinearities

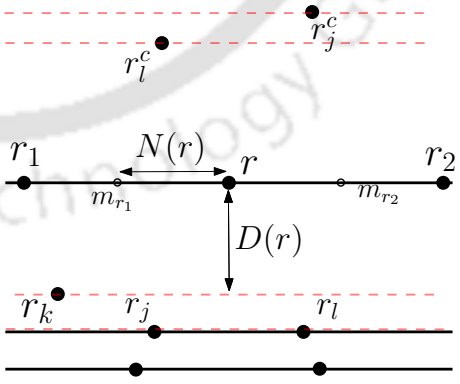


Figure 5.4: Illustrating $D(r)$ by which r avoids crossing horizontal red lines & $N(r)$ for no collision with neighbour

$R(r) = \min\{d(r, \overleftrightarrow{r_i r_j}) \mid r_i \neq r_j \neq r_i^c; r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r; r_j \in \mathcal{NCV}_r \cup \mathcal{CV}_r \cup \mathcal{M}_r\}$. The robot

r considers a line passing through any two non-neighbour visible robots r_i and r_j which are not collinear with r. This is to avoid the possible collinearity $\overline{r_i r r_j}$, after the movement. It also avoids moving to a line that passes through r_i and $m_{r'} \in \mathcal{M}_r$ to prevent itself from getting collinear due to the simultaneous movement of another robot r' from L_r^H. As shown in Fig. 5.3, each blue line is a possible collinearity for r and $R(r)$ is the smallest of all distances from r to these blue lines.

$D(r) = \min\{d(r, L_{r_i}^H) \mid r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r\}$. r avoids the horizontal line passing through any visible robot r_i to ensure that it does not cross the line. This is to prevent r from being a first or last robot of collinearity with a FINISH-colored robot. For example, r and r' are moving simultaneously to t_r and $t_{r'}$ respectively, as shown in Fig. 5.6. Since the three points t_r, r_i and $t_{r'}$ lie on the same line, r becomes collinear with r_i and r' after the movement. To prevent this, $D(r)$ is considered so that r does not cross the horizontal line passing through r_i. Fig. 5.4 illustrates that $D(r)$ is the minimum among all distances from r to the horizontal red lines passing through the visible robots. We also plan to make r a terminal robot after movement by considering $D(r)$.

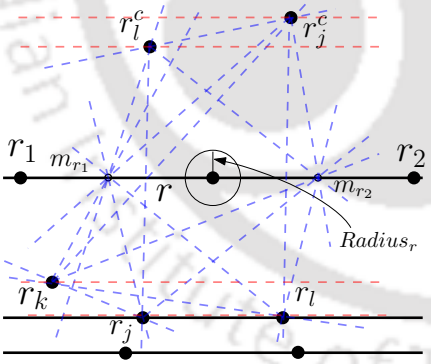


Figure 5.5: r calculates $Radius_r$ which is the distance between the target point of r and its current position.

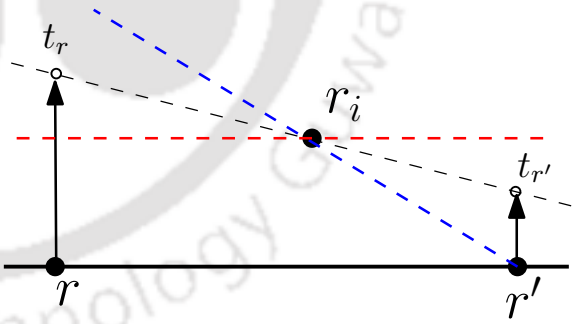


Figure 5.6: The movement of r to t_r creates a new collinearity if the horizontal line $L_{r_i}^H$ is not taken into consideration.

$N(r) = \min\{d(r, m_{r'}) \mid m_{r'} \in \mathcal{M}_r\}$ ensures that there is no collision between r and its neighbour in case of simultaneous movement.

Finally, r computes the distance $Radius_r = \frac{1}{2}\min\{R(r), D(r), N(r)\}$, as shown in Fig 5.5.

We now define a line called *reference line* that r uses to execute its movement with respect to the line. We explain later in this section how r computes this line in the current LCM cycle.

Definition 5.3.2. (Reference Line): For a robot r in its current compute phase, a line L passing through it, is called the reference line if r chooses L to execute its movement to the target point t_r in the same LCM cycle such that $\overline{r\,t_r}$ is perpendicular to the line L.

The movement of the robot r depends on the choice of the reference line in the current LCM cycle and the target point in that cycle. The reference line can be either a horizontal line or a non-horizontal line passing through the robot r. Based on this, we distinguish following two types of movements for a robot r. The main difference in these two types of movement is the choice of the reference line. When the reference line is horizontal, the movement executed by r with respect to the line is called the Type I movement. When the reference line is non-horizontal, the movement is called Type II movement.

Type I Movement: When the reference line L, selected by r is the horizontal line L_r^H, it chooses a point t_r as the target point in the upper half plane $H^{Up}_{L_r^H}$ such that the distance of the target t_r from r is $Radius_r$ and $\overline{r\,t_r}$ is perpendicular to the reference line L_r^H. The objective of r is to move to t_r by following the straight line segment $\overline{r\,t_r}$. This movement is referred to as the Type I movement.

Type II Movement: When the reference line (say L) selected by the robot r is a non-horizontal line, r needs to check whether the reference line L is perpendicular to L_r^H or not. If perpendicular, the target point t_r is chosen such that it is $Radius_r$ distance away from the current location of r in the left direction (local left with respect to the coordinate system of r) and $\overline{r\,t_r} \perp L$. If it is not perpendicular, the target point t_r is chosen in the upper half plane H_L^{Up}, rather than in the leftward direction. Even in this case, the distance between r and t_r remains $Radius_r$ and $\overline{r\,t_r} \perp L$. Finally, r aims to move to t_r by following the straight line segment $\overline{r\,t_r}$. This movement is identified as a Type II movement.

We will now describe how the robot r uses the above strategies to determine the reference line for its movement and the type of movement it executes in various scenarios.

After activation, r first determines whether it is an eligible robot or not. If it is not an

eligible robot, it remains stationary and does not change its current color. When r finds itself as an eligible robot, there can be two cases.

- **Case 1 ($\mathcal{CV}_r$ is empty):** When this case arises, r might be a terminal robot or a non-terminal robot on the current horizontal line L_r^H. Since the set $\mathcal{CV}_r$ is empty, r can only be a part of horizontal collinearity with its neighbours. Our target is to move the robot r when it is a middle robot of the horizontal collinearity. So, we differentiate two sub-cases here.
  - **Case 1.1 (r is a terminal robot on L_r^H):** In this scenario, the robot r is not a middle robot of some collinearity, as $\mathcal{CV}_r = \emptyset$ and it is a terminal robot on L_r^H. r changes its color to `FINISH` and terminates with no change in its current position.
  - **Case 1.2 (r is a non-terminal robot on L_r^H):** Even if $\mathcal{CV}_r$ is empty, r is a middle robot of the horizontal collinearity with its neighbours, as shown in Fig 5.7. Note that one of the neighbours of r must be with color `FINISH` in this case, as otherwise, r would not be an eligible robot. r follows the strategy FIND_TARGET_DISTANCE to calculate the distance $Radius_r$ and executes a Type I movement by choosing the horizontal line L_r^H as the reference line. During the movement, it maintains the current color `OFF`.

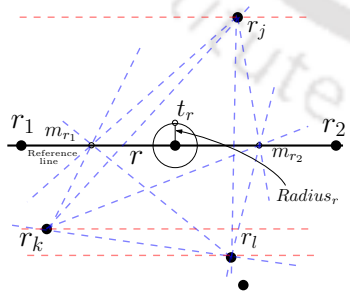


Figure 5.7: The horizontal line is the reference line for r to and it executes a Type I movement to t_r.

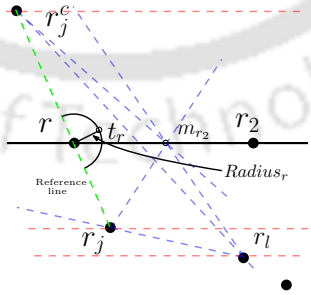


Figure 5.8: The LOC is the reference line for a Type II movement when r is terminal and $\mathcal{CV}_r$ has two robots.

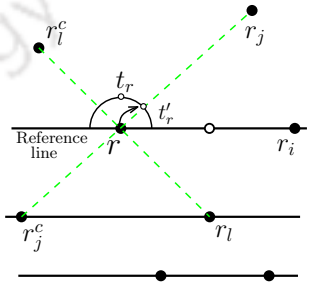


Figure 5.9: r remains collinear after the movement to t_r', instead of t_r, because of inaccuracy

- **Case 2 ($\mathcal{CV}_r$ is non-empty):** Recall that the set $\mathcal{CV}_r$ always has even number of elements in it. So, depending on the number of elements in $\mathcal{CV}_r$, we change our strategy for the robot r. We differentiate two sub-cases here.

  - **Case 2.1 ($\mathcal{CV}_r$ has exactly two robots and r is a terminal robot):** Let r_j and r_j^c be the two robots in $\mathcal{CV}_r$ such that r is a middle robot of collinearity $\overline{r_j r r_j^c}$ (illustrated in Fig 5.8). In this case, the choice of the reference line for r is simple as the current $\mathcal{CV}_r$ has exactly two elements. So, r sets the line $L = \overleftrightarrow{r_j r_j^c}$ as the current reference line. This line L is a non-horizontal line as r is a terminal robot. Also, r follows the strategy FIND_TARGET_DISTANCE and finds $Radius_r$, which is the distance of the target point from the current location of r. In consequence, r executes a Type II movement with no change in the current color OFF.

  - **Case 2.2 ($\mathcal{CV}_r$ has more than two elements) or ($\mathcal{CV}_r$ has two elements but r is a non-terminal robot):** In this case, r must be a part of multiple collinearity as a middle robot. As shown in Fig. 5.5, r is collinear with r_j and r_j^c as well as r_l and r_l^c. In such a case, r chooses the horizontal line L_r^H as the reference line and finds the distance $Radius_r$ by FIND_TARGET_DISTANCE. Then r executes a Type I movement to the target point t_r without changing the current color OFF. This move eliminates horizontal collinearity of r, if any. In case r does not detect any collinearity after getting activated again, which means $\mathcal{CV}_r$ becomes empty, it terminates by setting its color $r.color$ = FINISH. But it is possible that the movement might not be accurate enough that r actually reaches t_r. Instead, r might reach to a point t_r' after the Type I movement, as illustrated in Fig. 5.9. Notice that r was a part of two collinearities (indicated by the green lines) before the Type I movement and it remains collinear even after the movement, due to the inaccuracy. At this point, the current $\mathcal{CV}_r$ contains two robots r_j and r_j^c. In consequence, it sets the line $L = \overleftrightarrow{r_j r_j^c}$ as the reference line, so that it breaks the collinearity by executing another movement with respect to L. Since L is a non-horizontal line, r executes a Type II movement, if it is an eligible robot, as

described in Case 2.1 with color OFF.

Algorithm 9: SSYNC_MV_INACC

if $r.color = \texttt{OFF} \wedge$ *all robots in* $H^{Up}_{L^H_r}$ *are with color* FINISH $\wedge$ r *has either no neighbour or one neighbour or both neighbours out of which at least one is with color* FINISH **then**
  $\mathcal{V}_r = \{r_i \in R \mid r_i \neq r$ and $\nexists\, r_j \in R$ such that $\overline{rr_jr_i}\}$
  $\mathcal{NB}_r = \{r_i \in \mathcal{V}_r \mid L^H_{r_i} = L^H_r\}$
  $\mathcal{M}_r = \{m_{r_i} \mid m_{r_i}$ is the midpoint of the line segment $\overline{rr_i}$ for $r_i \in \mathcal{NB}_r\}$
  $\mathcal{NCV}_r = \{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \nexists\, r_j \in \mathcal{V}_r$ such that $\overline{r_irr_j}\}$
  $\mathcal{CV}_r = \{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r \mid \exists\, r^c_i \in \mathcal{V}_r$ such that $\overline{r_irr^c_i}\}$
  if $\mathcal{CV}_r$ *is empty* **then**
    if r *is a terminal robot on* L^H_r **then**
      Change $r.color$ to FINISH and terminate
    else
      Follow FIND_TARGET_DISTANCE() and choose $L = L^H_r$ as the reference line
      Follow MOVE_TO_TARGET() with color OFF
  else
    if $\mathcal{CV}_r$ *has two robots* r_j *and* $r^c_j \wedge r$ *is a terminal robot* **then**
      Choose $L = \overleftrightarrow{r_jr^c_j}$ as the reference line and follow FIND_TARGET_DISTANCE()
      Follow MOVE_TO_TARGET() with color OFF
    else // $\mathcal{CV}_r$ has more than two elements or it has two elements but r is non-terminal
      Choose $L = L^H_r$ as the reference line and follow FIND_TARGET_DISTANCE()
      Follow MOVE_TO_TARGET() with color OFF
else
  No change in color and no movement

5.3.3 Analysis of the Algorithm

In this section, we prove the correctness and the time complexity of SSYNC_MV_INACC algorithm. We also show that the movements of the robots are free from collision.

Lemma 5.3.3. *Any two robots moving simultaneously lie on the same horizontal line before the movement.*

Proof. Let r and r' be the two robots lying on different horizontal lines. Without loss of generality, we assume r' lies below L^H_r. If the two robots are visible to each other and $r.color \neq$

FINISH, r' cannot move. If $r.color$ = FINISH and the robots r and r' are visible to each other, r cannot move thenceforth. If they are not visible to each other, it means that there is another robot r'' lying between them. Let r' execute a movement. Then $r''.color$ must be FINISH, as otherwise r' would not be eligible for movement. Since, r lies in $H^{Up}_{L^H_{r''}}$ and r'' is with color FINISH, r must be with color FINISH. Therefore, r cannot move when r' is executing its movement. □

Corollary 5.3.4. *If r executes a movement in an LCM cycle, no robot from the set $\mathcal{NCV}_r \cup \mathcal{CV}_r$ executes a movement in the same LCM cycle.*

Proof. Since, a robot r' from the set $\mathcal{NCV}_r \cup \mathcal{CV}_r$ does not lie on the horizontal line L^H_r, both of r and r' cannot move simultaneously, by Lemma 5.3.3. □

Algorithm 10: FIND_TARGET_DISTANCE()

// The subroutine calculates the distance between the target and the current position of r

1 Calculate $R(r) = \min\{d(r, \overleftrightarrow{r_i r_j}) \mid r_i \neq r_j \neq r_i^c$ and $r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r$ and $r_j \in \mathcal{NCV}_r \cup \mathcal{CV}_r \cup \mathcal{M}_r\}$

2 Calculate $D(r) = \min\{d(r, L^H_{r_i}) \mid r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r\}$

3 Calculate $N(r) = \min\{d(r, m_{r'}) \mid m_{r'} \in \mathcal{M}_r\}$

4 Calculate $Radius_r = \frac{1}{2}\min\{R(r), D(r), N(r)\}$

Algorithm 11: MOVE_TO_TARGET()

1 **if** $L = L^H_r$ **then** // Type I Move

2 Choose a point t_r in $H^{Up}_{L^H_r}$ such that $d(r, t_r) = Radius_r$ and $\overline{r\,t_r} \perp L^H_r$

3 **else if** $L \perp L^H_r$ **then** // Type II Move

4 Choose a point t_r in the local left direction such that $d(r, t_r) = Radius_r$ and $\overline{r\,t_r} \perp L$

5 **else** // Type II Move

6 Choose a point t_r in H^{Up}_L such that $d(r, t_r) = Radius_r$ and $\overline{r\,t_r} \perp L$

7 Move to t_r by following $\overline{r\,t_r}$

Lemma 5.3.5. *If r_i and r_j are two robots lying on the same horizontal line and r lies on a different horizontal line, r cannot be on a point on $\overleftrightarrow{r_i r_j}$ after the movement.*

Proof. By Lemma 5.3.3, neither of r_i and r_j cannot move while r is executing its movement. We can assume that r_i and r_j both are visible to r. Since, the inequality $Radius_r < D(r) \leq d(r, L^H_{r_i})$ holds, r cannot move to the line $L^H_{r_i} = \overleftrightarrow{r_i r_j}$. If r_i or r_j or both are not visible to r, the above inequality holds true. □

Lemma 5.3.6. *At most two robots can move simultaneously.*

Proof. We prove this by contradiction. Let r_1, r_2 and r_3 be three robots moving simultaneously. By Lemma 5.3.3, these robots must be on the same horizontal line. Let r_2 lie between r_1 and r_3 and it has two neighbours r_4 and r_5. Since r_2 is eligible for movement, at least one of r_4 and r_5 is with color FINISH. Let us assume that $r_4.color$ = FINISH. r_1 being an eligible robot for movement, cannot have the color FINISH. So r_1 and r_4 must be two different robots. Also r_4 cannot be a terminal robot, as r_2 lies between r_1 and r_3 and r_2 is a neighbour of r_4. When r_4 sets the color FINISH, r_1 and r_2 together cannot be on the same horizontal line as r_4, by the eligibility criteria described in the algorithm. Without loss of generality, let us consider that r_1 is the robot that lies on a different horizontal line. It is not possible for r_1 to move to the line $\overleftrightarrow{r_2 r_4}$, which is a horizontal line, by Lemma 5.3.5. This leads to a contradiction. □

The purpose of a movement from the reference line is to break the existing collinearity. We prove in the following two lemmas (Lemma 5.3.7 and 5.3.8) that if r chooses a reference line L in the current LCM cycle and executes a movement, it cannot move to a point on that reference line, even if the movements are inaccurate.

Lemma 5.3.7. *After a Type I movement, r cannot move to a point on L_r^H even with inaccuracy, where the horizontal line passing through L_r^H is the current reference line.*

Proof. Due to inaccuracy in the movement, r might move to a point t_r' whereas t_r is its chosen target such that $\overline{r\,t_r}$ is perpendicular to the reference line L_r^{HPrev}. Since, the angle $\angle t_r\, r\, t_r' < 90°$ for any robot, t_r' cannot be on L_r^H. □

Lemma 5.3.8. *After a Type II movement, r cannot move to a point on the reference line $L = \overleftrightarrow{r_j r_j^c}$, where r is collinear with r_j and r_j^c as a middle robot.*

Proof. By a similar argument as Lemma 5.3.7, we can prove this statement. □

Remark 5.3.9. If r is a part of a collinearity $\overline{r_i\, r\, r_j}$, it can detect the collinearity only when it is the middle robot of that collinearity. In that case, $r_i, r_j \in \mathcal{CV}_r$ and both are conjugates of each other.

Before proving that our algorithm SSYNC_MV_INACC eliminates any existing collinearity for a robot, we prove in the subsequent lemmas (Lemma 5.3.10-5.3.13) that r does not create any new collinearity after a movement, even with inaccuracy.

Lemma 5.3.10. *r cannot be collinear with any other two robots r_i and r_j after a movement, where $r_j \neq r_i^c$ with $r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r$ and $r_j \in \mathcal{NCV}_r \cup \mathcal{CV}_r$.*

Proof. Note that r_i and r_j are stationary when r executes its movement by Corollary 5.3.4. Since, both r_i and r_j are visible to r, r considers the line $\overleftrightarrow{r_i r_j}$ and calculates the distance $d(r, \overleftrightarrow{r_i r_j})$. r chooses a point t_r such that $d(r, t_r) = Radius_r$. By definition, $Radius_r < R(r) \leq d(r, \overleftrightarrow{r_i r_j})$. So, we have $d(t_r, \overleftrightarrow{r_i r_j}) > 0$, which implies t_r does not lie on $\overleftrightarrow{r_i r_j}$. Even if r moves inaccurately to a point t_r' instead of t_r, the above inequality holds true as $d(r, t_r') = Radius_r$. □

Lemma 5.3.11. *r cannot be collinear with other two robots r_i and r_j after a movement, where $r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r$ and $r_j \in \mathcal{NB}_r$, even if r_j moves simultaneously with r.*

Proof. Corollary 5.3.4 ensures that r_i does not move when r moves. Let r_j be stationary when r is moving. Since, $d(r, m_{r_j}) = \frac{1}{2} d(r, r_j) < d(r, r_j)$ where $m_{r_j} \in \mathcal{M}_r$ is the midpoint of the line segment $\overline{r r_j}$, the inequality $Radius_r < \min\{d(r, \overleftrightarrow{r_i m_{r_j}}), d(r, L_{r_i}^H)\} < d(r, \overleftrightarrow{r_i r_j})$ holds. If r moves to a point t_r, it does not lie on $\overleftrightarrow{r_i r_j}$ as $d(r, t_r) = Radius_r$. In case of simultaneous movement of r and r_j, if r and r_j move to t_r and t_{r_j} respectively, as shown in Fig. 5.10, the inequality $d(r, \overleftrightarrow{r_i t_r}) < \min\{d(r, \overleftrightarrow{r_i m_{r_j}}), d(r, L_{r_i}^H)\} \leq \min\{d(r, \overleftrightarrow{r_i P}), d(r_j, L_{r_i}^H)\} < d(r, \overleftrightarrow{r_i t_{r_j}})$ holds. □

Corollary 5.3.12. *Statement of Lemma 5.3.11 holds when r_j is a non-neighbour robot of r with $L_r^H = L_{r_j}^H$.*

Lemma 5.3.13. *For two robots $r_i \notin \mathcal{V}_r$ and $r_j \in R$ with r_i, r_j not on L_r^H, r cannot be a middle robot of a collinearity $\overline{r_i r r_j}$ after its movement if it was not collinear with both of them in the previous LCM cycle.*

Proof. Since, $r_i \notin \mathcal{V}_r$, there is a robot $r_k \in \mathcal{V}_r$ such that $\overline{r r_k r_i}$. The line segment $\overline{r_i r_j}$ lies on one side of the line $\overleftrightarrow{r_k r_j}$ while r lies on the other side of $\overleftrightarrow{r_k r_j}$ as shown in Fig. 5.11

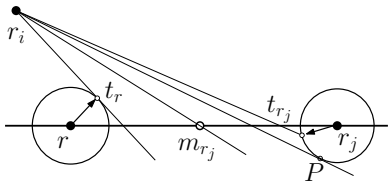


Figure 5.10: Concurrent movements do not create collinearity

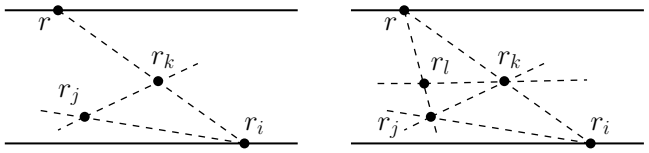


Figure 5.11: r cannot move to any point on the line segment $\overline{r_i r_j}$ because of $\overleftrightarrow{r_k r_j}$ (left) and $\overleftrightarrow{r_k r_l}$ (right)

(left). Since, r cannot cross $\overleftrightarrow{r_k r_j}$ as $Radius_r < R(r) \leq d(r, \overleftrightarrow{r_k r_j})$, the collinearity $\overline{r_i r r_j}$ is not possible. In case of $r_j \notin \mathcal{V}_r$, there must be a robot $r_l \in \mathcal{V}_r$ lying between r and r_j such that $\overline{r r_l r_j}$ holds, as described in Fig. 5.11 (right). Since r cannot cross the line $\overleftrightarrow{r_k r_l}$ to reach a point on the line segment $\overline{r_i r_j}$ as $Radius_r < R(r) \leq d(r, \overleftrightarrow{r_k r_l})$, the collinearity $\overline{r_i r r_j}$ is not possible. In both account, r cannot be a middle robot of the collinearity $\overline{r_i r r_j}$. □

Remark 5.3.14. Lemma 5.3.10, 5.3.11 and 5.3.13 ensure that r never becomes a middle robot of collinearity with two robots after a movement, with whom it was non-collinear in previous LCM cycle.

It is possible that r becomes collinear with two robots after the movement whom it was collinear with, in its previous LCM cycle. There can be two such situations. As shown in Fig. 5.9, r remains a middle robot of a collinearity $\overline{r_j r r_j^c}$ after the movement whereas it was non-horizontally collinear with the two robots r_j and r_j^c in the previous LCM cycle. The second situation is depicted in Fig. 5.12, where r was a part of the horizontal collinearity $\overline{r_k r r_i}$ before the movement, but the simultaneous inaccurate movement of the two robots r and r_i to the points P and Q respectively, makes r again collinear with r_k and r_i after the movement. The second situation may occur when a non-terminal robot moves simultaneously with a terminal robot. We prove in the subsequent Lemma 5.3.15 that r becomes a terminal after a movement.

Lemma 5.3.15. *After a movement, r can never be a non-terminal robot.*

Proof. We prove this by contradiction. Let r be a non-terminal robot after executing a movement. Then r must have two neighbours r_i and r_j. If in the previous LCM cycle, all three robots lie on the same horizontal line, they cannot be collinear again after the movement, as three of them cannot move together (Lemma 5.3.6) and the movement of

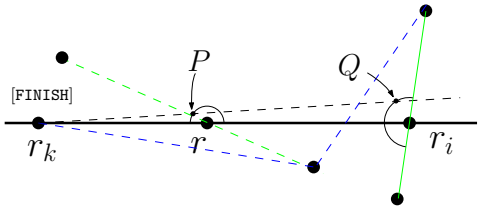


Figure 5.12: The simultaneous inaccurate movements of a terminal and a non-terminal robot may create a collinearity with a third stationary robot.

r^{prev} breaks the collinearity $\overline{r_i^{prev} r^{prev} r_j^{prev}}$. If, they do not lie on the same horizontal line in the previous LCM cycle, we can have two cases here. (1) All of them are collinear which means that each of them lies on a different horizontal line, so these three robots cannot move together in the previous LCM cycle to form a horizontal collinearity. (2) Three of them are not collinear which means that r cannot move to $\overleftrightarrow{r_i r_j}$ by Remark 5.3.14. In all of the cases, we get a contradiction. □

We now show that the set $\mathcal{CV}_r$ become empty for a terminal robot after at most two movements.

Lemma 5.3.16. *$\mathcal{CV}_r \subseteq \mathcal{CV}_r^{Prev}$, where r was a terminal robot in its previous LCM cycle.*

Proof. If $\mathcal{CV}_r = \emptyset$, we are done. Otherwise, let $r_i \in \mathcal{CV}_r$. Then there is $r_i^c \in \mathcal{CV}_r$ such that $\overline{r_i r r_i^c}$. If we assume $r_i \notin \mathcal{CV}_r^{Prev}$, then we get $r_i \notin \mathcal{V}_r^{Prev}$. Now we can have either $L_{r_i}^H \neq L_r^{HPrev}$ or $L_{r_i}^H = L_r^{HPrev}$. By Lemma 5.3.13, the collinearity $\overline{r_i r r_i^c}$ is not possible when $L_{r_i}^H \neq L_r^{HPrev}$. Since r^{prev} is terminal and $r_i \notin \mathcal{CV}_r^{Prev}$, $\overline{r_j r r_i}$ does not hold for any $r_j \in R$. So even if $L_{r_i}^H = L_r^{HPrev}$, r^{Prev} cannot be a middle robot of collinearity with r_i and r_i^c after the movement by Remark 5.3.14. So, we find a contradiction for both of the cases. □

Lemma 5.3.17. *After a Type I movement of a terminal robot r, $\mathcal{CV}_r$ contains at most two robots.*

Proof. Let $r_i, r_i^c \in \mathcal{CV}_r$ such that $\overline{r_i r r_i^c}$ after the movement. We prove this by contradiction. Let $r_j, r_j^c \in \mathcal{CV}_r$ be another pair of robots such that $\overline{r_j r r_j^c}$. Let r^{Prev} be the position of r in the previous LCM cycle before the movement. Note that $r \neq r^{Prev}$ even with inaccurate movement, as $Radius_r \neq 0$. By Lemma 5.3.16, $r_i, r_i^c, r_j, r_j^c \in \mathcal{CV}_r^{Prev}$. Hence, $\overline{r_i r_i^c}$ and $\overline{r_j r_j^c}$

intersect at r^{Prev} and r. This leads to a contradiction as two non-parallel lines intersect at exactly one point. □

Lemma 5.3.18. *After Type II movement of the robot r, $\mathcal{CV}_r$ contains no robot.*

Proof. Since r executes Type II movement, r^{Prev} is terminal and $\mathcal{CV}_r^{Prev}$ contains exactly one pair of robots, say r_i and r_i^c. By Lemma 5.3.8, r cannot move to the line $\overleftrightarrow{r_i r_i^c}$. Also by Lemma 5.3.16, $\mathcal{CV}_r \subseteq \mathcal{CV}_r^{Prev}$. Thus $\mathcal{CV}_r$ must be empty. □

Remark 5.3.19. For a robot r, Lemma 5.3.15, 5.3.17 and 5.3.18 guarantee that the set $\mathcal{CV}_r$ eventually becomes empty and r becomes a terminal which leads to the termination of r with color `FINISH`.

Lemma 5.3.20. *For a robot r with $r.color =$ `FINISH`, $\mathcal{CV}_r$ remains empty.*

Proof. r sets its color to `FINISH`, when $\mathcal{CV}_r$ becomes empty. On the contrary, let a robot r_i move in such a way that $\mathcal{CV}_r$ become non-empty for the first time. Then there exists a robot r_i^c for which $\overline{r_i r r_i^c}$. Before r_i moves, $\mathcal{CV}_r = \emptyset$, so $\overline{r_i^{Prev} r r_i^c}$ does not hold. Then r_i cannot be collinear with both r and r_i^c. This is a contradiction. □

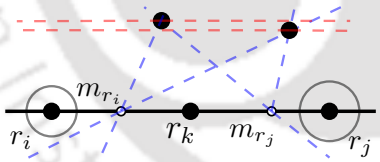


Figure 5.13: The circles centered at r_i and r_j are disjoint in presence of a common neighbour r_k

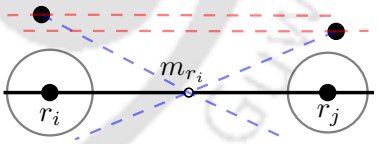


Figure 5.14: The circles are disjoint even if there is no common neighbour, leading to no collision

Theorem 5.3.21. *The movements of the robots are free from collision.*

Proof. Let r_i and r_j be any two robots in R lying on two different horizontal lines. By Lemma 5.3.3, they cannot move together which means they cannot meet a collision with each other. If r_i and r_j lie on the same horizontal line $L_{r_i}^H = L_{r_j}^H = L$, then they move simultaneously only if they both are eligible robots on L. If there is a robot r_k lying between them on L, as shown in Fig 5.13, then r_i and r_j move to two points t_{r_i} and t_{r_j} such that

$Radius_{r_i} = d(r_i, t_{r_i}) < d(r_i, m_{r_i})$ and $Radius_{r_j} = d(r_j, t_{r_j}) < d(r_j, m_{r_j})$, where m_{r_i} and m_{r_j} are the midpoints of $\overline{r_i r_k}$ and $\overline{r_k r_j}$ respectively. If no such robot r_k exists, it means r_i and r_j are neighbours on L, illustrated in Fig. 5.14. Then, $m_{r_i} = m_{r_j}$. Even if both of them move simultaneously, the above inequalities hold. Since, the circles of radius $Radius_{r_i}$ and $Radius_{r_j}$ are disjoint, both the robots cannot collide. □

Theorem 5.3.22. *Algorithm* SSYNC_MV_INACC *solves the mutual visibility under SSYNC setting in $\mathcal{O}(N)$ epochs using 2 colors, where N is the number of robots which have one axis agreement and might exhibit inaccuracy in their movement.*

Proof. Let us assume that after termination, there are three robots r_i, r_j and r_k such that $\overline{r_i r_k r_j}$. At termination, all robots are with color FINISH. Then, $r_k.color$ = FINISH. As $\overline{r_i r_k r_j}$, we have $\mathcal{CV}_{r_k} \neq \emptyset$, contradicting Lemma 5.3.20.

An eligible terminal robot r moves at most twice before termination. It is possible that r moves thrice before setting its color to FINISH when it is a non-terminal robot. Under the SSYNC setting, r executes a Type I movement if it detects any horizontal collinearity or if it is a part of multiple collinearities as a middle robot. A non-terminal robot takes one epoch to become terminal with a Type I movement with color OFF. A terminal robot takes one epoch to execute a movement so that the number of the elements in the set $\mathcal{CV}_r$ is at most 2, by Lemma 5.3.17 and 5.3.18. If $\mathcal{CV}_r$ contains two robots, r needs to execute a Type II movement to eliminate the collinearity, which needs one more epoch. Finally, it takes another epoch to change its color to FINISH from OFF. So it takes at most 3 epochs for a terminal robot to change its color to FINISH and 4 epochs for a non-terminal robot. Let us consider an initial configuration where N robots lie on N different horizontal lines $L_1^H, L_2^H, \cdots L_N^H$ and all of them are situated on the same non-horizontal line. In that case, robots on L_2^H to L_N^H cannot move till the robot on L_1^H is with color FINISH. The robot on L_1^H terminates in one epoch and then the robot on L_2^H gets the chance to move and ends up with FINISH color with at most three epochs, being a terminal robot. Proceeding this way, robots execute and reach to FINISH color one by one. Hence, it takes at most $3N$ epochs for all robots to reach at termination.

We can prove that this is the worst case configuration, where the algorithm takes maxi-

mum epochs. Since a non-terminal robot takes at most 4 epochs to terminate, one can assume that the algorithm may take more than $3N$ epochs to reach the termination. But, an eligible non-terminal robot r takes four epochs only if it has both neighbours, say r_1 and r_2, among them one (say r_1) with color FINISH, other moves simultaneously with r and $\overline{r_1 r r_2}$ holds after the movement, as referred in Fig. 5.12. In that case, r_1 will reach to FINISH in one epoch and in the same epoch r and r_2 will executes their movement simultaneously. After the movement of r and r_2, it can happen that they form different horizontal lines, but both of them are terminal robot by Lemma 5.3.15. After the movement r_2 must be above r and $\mathcal{CV}_{r_2}$ contains at most two robots, by Lemma 5.3.17. So r_2 takes at most two epochs to terminate while r waits at its position. r now takes at most three epochs to change its color to FINISH. Thus, six epochs are required in total for r, r_1 and r_2 to terminate, whereas it would take at most 9 epochs, if they lie on different horizontal lines. Therefore, if any initial configuration has a non-terminal robot, our algorithm will take less number of epochs than the configuration with all terminal robots. □

5.4 Mutual Visibility with Inaccurate Movement Under ASYNC Settings

This section discusses the Algorithm ASYNC_MV_INACC that solves the problem under the ASYNC setting. All other components of the model are the same as SSYNC settings, described in Section 5.3. There is a mechanism introduced by Das et al. [27], which can help us obtain a protocol in the ASYNC setting from any given SSYNC protocol with 6 colors. However, we propose an algorithm in this section that solves the mutual visibility problem using just 3 colors. Instead of applying the mechanism proposed by Das et al. [27] on our existing SSYNC algorithm, our target is to give an algorithm in the ASYNC setting that uses an even lesser number of colors. First, we are going to discuss the difficulty in applying the same algorithm SSYNC_MV_INACC in the ASYNC setting. Next, we describe the algorithm and its correctness.

5.4.1 Technical difficulties of ASYNC setting

The main difference between SSYNC and ASYNC schedulers is that a active robot operating under the ASYNC scheduler, might capture the snapshot in its look phase when some other robot is still in its move phase, but this cannot happen in the SSYNC setting. In the SSYNC setting, two active robots execute their respective LCM cycle together while an inactive robot stays inactive till the current round is finished. Cicerone et al. [20] (see Fig. 1) illustrates this difference between SSYNC and ASYNC settings for three robots.

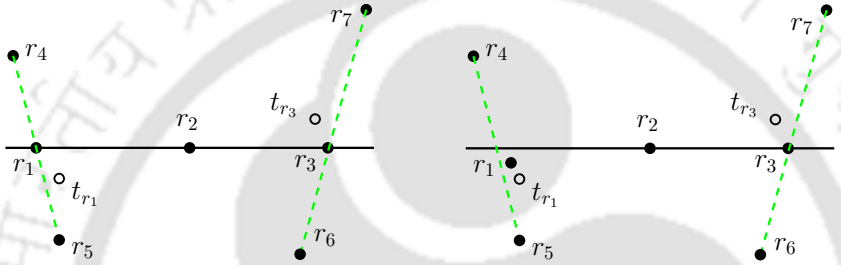


Figure 5.15: (Left) Illustrating the configuration before the movement of r_1 and r_3 where they compute their target to be t_{r_1} and t_{r_3} respectively. (Right) r_1 starts moving towards t_{r_1}, but in between r_2 gets activated and takes the snapshot, so r_2 thinks itself as eligible robot.

Consider the example shown in Fig. 5.15 where r_1 is collinear with two robots r_4 and r_5. Similarly, r_3 is collinear with other two robots r_6 and r_7. So both of them choose their respective reference lines $\overleftrightarrow{r_4 r_5}$ and $\overleftrightarrow{r_6 r_7}$ and choose their target points. Due to inaccuracy, let us assume that r_1 starts moving towards the point t_{r_1} and r_3 is about to move towards t_{r_3}. At this moment, r_2 gets activated and captures the snapshot. It is not possible for r_2 to understand that r_1 is still in a move, as r_1 is moving with color `OFF` according to the SSYNC_MV_INACC algorithm. Since r_1 moves away from the line $\overleftrightarrow{r_1 r_3}$, r_2 thinks of itself as an eligible robot and finds its current $\mathcal{CV}_r$ is empty. So, it changes its color to `FINISH`. During this process, both r_1 and r_3 reach to t_{r_1} and t_{r_3}. Since r_2 lies on the line segment $\overline{t_{r_1} t_{r_3}}$ and is with color `FINISH`, it cannot eliminate the collinearity by executing a movement. Moreover, r_1 and r_3 are unable to detect collinearity as neither of them is a middle robot.

To eliminate this possibility, we modify the previous algorithm with one more color. The algorithm ASYNC_MV_INACC is similar to the previous algorithm SSYNC_MV_INACC. We use 3 colors in this algorithm: `OFF` (used initially), `MOVE` (used while executing move-

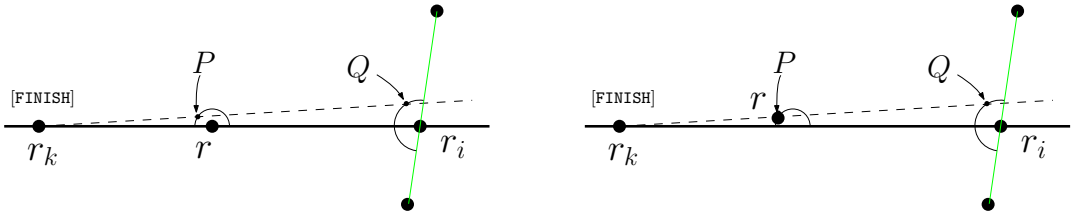


Figure 5.16: (left) illustrating the configuration before the movement of r. (right) illustrating the configuration when r has moved to P but r_i is yet to reach Q. At this situation r takes a snapshot after activation at P.

ment till termination) and FINISH (used at termination). Even if the two algorithms have similarities between them, the algorithm SSYNC_MV_INACC is not going to work in the case of ASYNC setting if we just incorporate the MOVE color into it. This is when the configuration is the same as the configuration depicted in Fig 5.16, as it is possible in the ASYNC setting that the robot r_i is yet to reach Q, but the robot r has reached P. In this situation, r, after activation at P, might take a snapshot while r_i is still below $H_{L_r^H}^{Low}$. r cannot figure out from the snapshot that the robot r_i is still in its move phase and determines itself as an eligible robot. After computing the current $\mathcal{CV}_r$, r does not find itself as a part of any collinearity which leads to the termination, but the collinearity $\overline{r_k r r_i}$ persists when r_i reaches to Q.

To deal with this situation, we change the eligibility conditions of a robot. In SSYNC_MV_INACC, a terminal robot r_k might terminate without any movement, as shown in Fig. 5.12. It is also possible that a non-terminal robot moves thrice (two Type I and one Type II movement) when a situation, shown in Fig 5.12, arises. In the algorithm ASYNC_MV_INACC, we make sure that a robot moves only when it is a terminal robot. This change ensures that a robot executes at most two movements.

5.4.2 Description of the Algorithm

Since we introduce a new color MOVE, we modify the definition of the eligible robot for this algorithm to incorporate this new color and not to allow a non-terminal robot to move. We change the Condition 2 and 3 in Definition 5.3.1 while others remain the same.

Definition 5.4.1. (Eligible Robot) Any robot r is called an eligible robot if it satisfies all of the following conditions. (1) All robots in $H^{Up}_{L^H_r}$ are with color FINISH. (2) $r.color$ is set to either OFF or MOVE. (3) r is a terminal robot.

All other definitions, notations and terminologies, defined in Section 5.3 are still the same. The first task for an active robot r is to determine if it is an eligible robot or not. If not, it waits till it becomes eligible. Once robot r is eligible, it needs to check its current color, which serves as an indicator of whether the robot has executed a movement. Based on the current color of r, two distinct cases can be distinguished.

- **Case 1 ($r.color$ is OFF):** When r is activated with color OFF, it indicates that r has not executed a movement before the current LCM cycle. Now, r selects the horizontal line L^H_r as the reference line for the movement. It then identifies the sets $\mathcal{NB}_r, \mathcal{M}_r, \mathcal{NCV}_r$ and $\mathcal{CV}_r$ from the snapshot taken in the look phase of the current LCM cycle and follows the strategy FIND_TARGET_DISTANCE to calculate the distance $Radius_r$ of the target point from the current location. At the end of the compute phase, r sets $r.color$ to MOVE. Finally, r executes a Type I movement to reach the target point in the move phase of the same LCM cycle.

- **Case 2 ($r.color$ is MOVE):** When r gets activated with color MOVE, it figures out that it has already executed a movement and determines the set $\mathcal{CV}_r$ based on the data gathered in the current look phase. If $\mathcal{CV}_r$ is empty, it terminates by setting its color FINISH. But, it is possible that r remains collinear with two robots i.e., $\mathcal{CV}_r \neq \emptyset$ due to inaccurate Type I movement. Since r is an eligible robot, it is a terminal robot. We prove in Lemma 5.3.17 that $\mathcal{CV}_r$ contains exactly two elements after the Type I movement of a terminal robot if it is non-empty. Let us consider that r_j and r^c_j are the two robots in $\mathcal{CV}_r$ and $L = \overleftrightarrow{r_j r^c_j}$. Now, r checks whether there is another robot r_i visible with color MOVE. If r_i lies in $H^{Up}_{L^H_r}$, r does not qualify to be an eligible robot. In that case, r waits till $r_i.color$ = FINISH. Otherwise, r selects L as the reference line and needs to compute $Radius_r$ by FIND_TARGET_DISTANCE. We modify the definition of $\mathcal{M}_r$ for a MOVE-color eligible robot as follows. $\mathcal{M}_r = \{m_{r_i} |$ m_{r_i} is the midpoint of the line segment $\overline{rr'}$ where $r_i.color$ =

MOVE and r' is the point of intersection of $L^{\perp}_{r_i}$ and L^{H}_{r}}. So r calculates $Radius_r$ with the new definition of $\mathcal{M}_r$. Note that the reference line L is non-horizontal, as r_j and r^c_j belong to $\mathcal{CV}_r$. So, r maintains the current color MOVE and executes a Type II movement to reach its target point.

Algorithm 12: ASYNC_MV_INACC

if $r.color = OFF \wedge$ *all robots in* $H^{Up}_{L^H_r}$ *are with color* $FINISH \wedge r$ *is terminal robot* **then**
  Identify the sets $\mathcal{V}_r, \mathcal{NB}_r, \mathcal{M}_r, \mathcal{NCV}_r$ and $\mathcal{CV}_r$
  if $\mathcal{CV}_r$ *is empty and* r *has no neighbour* **then**
    Change $r.color$ to FINISH and terminate
  else
    Follow FIND_TARGET_DISTANCE() and choose $L = L^H_r$ as the reference line
    Change $r.color$ to MOVE and follow MOVE_TO_TARGET()
else if $r.color = MOVE$ **then**
  Identify the sets $\mathcal{V}_r, \mathcal{NB}_r, \mathcal{M}_r, \mathcal{NCV}_r$ and $\mathcal{CV}_r$
  if *there are two robots* r_j *and* r^c_j *in* $\mathcal{CV}_r$ *such that* $\overline{r_j r r^c_j}$ **then**
    if r *sees a robot* r' *with color* $MOVE$ *in* $H^{Up}_{L^H_r}$ **then**
      No change in color and no movement.
    else
      Follow FIND_TARGET_DISTANCE() and choose $L = \overleftrightarrow{r_j r^c_j}$ as the reference line
      Follow MOVE_TO_TARGET() with color MOVE
  else
    Change $r.color$ to FINISH and terminate
else
  No change in color and no movement

5.4.3 Analysis of the Algorithm

Here we explain the correctness and the time complexity of the algorithm ASYNC_MV_INACC and prove that the movement of the robots is free from collision.

Lemma 5.4.2. *All the eligible robots must be on the same horizontal line.*

Proof. In contradiction, let us assume that r and r' be two eligible robots, where r lies in $H^{Up}_{L^H_{r'}}$. From the eligibility criteria, if r' is an eligible robot and visible to r, then $r.color$ must

be FINISH and thus r cannot be an eligible robot. So, r must not be visible to r', implying another robot r'' exists with color FINISH on $\overleftrightarrow{rr'}$. $r''.color$ = FINISH and r lies in $H^{Up}_{L^H_{r''}}$, both implying $r.color$ must be FINISH and thus r cannot be an eligible robot. Hence, we arrive at a contradiction. □

Remark 5.4.3. A OFF-colored robot cannot be eligible if another robot is in its move phase.

Lemma 5.4.4. *If any two robots lie on different horizontal lines, they cannot move to the same horizontal line after the movement.*

Proof. Let r and r' be two robots lying on different horizontal lines. These two robots cannot move together by Lemma 5.4.2. If t_r is the target point of r, calculated in the current compute phase, it is not possible for t_r to be on $L^H_{r'}$ as $Radius_r = d(r, t_r) < D(r) \leq d(r, L^H_{r'})$. □

Lemma 5.4.5. *If r_1 and r_2 lie on the same horizontal line, they cannot form a non-horizontal collinearity with another robot r_3 after their movement, where r_3, lying on a different horizontal line than that of r_1 and r_2, is stationary while the other two are executing a movement.*

Proof. Since r_3 lies on different horizontal line than that of r_1 and r_2, it will be either in $\mathcal{NCV}_{r_1} \cup \mathcal{CV}_{r_1}$ or not in $\mathcal{V}_{r_1}$. If $r_3 \in \mathcal{NCV}_{r_1} \cup \mathcal{CV}_{r_1}$, r_1 cannot be collinear with r_2 and r_3 after the movement, even if simultaneous movement of r_2, by Lemma 5.3.11 and Corollary 5.3.12. So, r_1 and r_2 cannot form the non-horizontal collinearity with r_3 after a movement. If $r_3 \notin \mathcal{V}_{r_1}$, then there exists $r_4 \in \mathcal{NCV}_{r_1} \cup \mathcal{CV}_{r_1}$ such that $\overline{r_3 r_4 r_1}$ holds and we can prove the statement of the lemma by similar argument as above by considering the robot r_4 instead of r_3. □

Remark 5.4.6. Two MOVE-colored robots cannot be collinear with a third robot.

Lemma 5.4.7. *There can be at most two eligible robots.*

Proof. By Lemma 5.4.2, all the eligible robots must be on the same horizontal line. Since an eligible robot must be a terminal robot and a horizontal line can have at most two terminal robots, there can be at most two eligible robots at the same time. □

Lemma 5.4.8. *There can be at most two robots with color* `MOVE` *at the same time.*

Proof. Let us assume that there are three robots r_1, r_2 and r_3 with color `MOVE` at a point in time. Let r_1 be the latest robot which changes its color to `MOVE` from `OFF` among the three robots. Let t be the time just before the change in color. Then r_1 is eligible at t. If $r_2.color$ = `OFF` at t, at this time r_2 is also an eligible robot. By Lemma 5.4.2, both r_1 and r_2 lie on the same horizontal line at t. If $r_2.color$ = `MOVE` at t, we now prove that they lie on the same horizontal line. On the contrary, we assume that r_1 and r_2 lie on different horizontal lines at the time t. Since r_1 is eligible and $r_2.color$ = `MOVE` at t, r_2 must not be visible to r_1 at this time, by the definition of an eligible robot. So, there exists a robot r_4 with color `FINISH` lying between r_1 and r_2. Let t' ($< t$) be the time just before r_4 changes its color to `FINISH`. At t', r_2 cannot lie on $\overleftrightarrow{r_1 r_4}$. Let P and Q be the positions of r_2 at t and t', respectively. We can prove that r_2 has to be `MOVE` at the time t'. If at t', $r_2.color$ = `OFF`, r_2 never comes to the line $\overleftrightarrow{r_1 r_4}$, as r_1 and r_4 are stationary in between the time t' and t, and r_2 avoids moving to the line $\overleftrightarrow{r_1 r_4}$ by the definition of $Radius_r$. This is a contradiction to the assumption that $\overline{r_1 r_4 r_2}$ holds. So $r_2.color$ must be set to `MOVE` at t'. If r_2 does not lie on the horizontal line $L^H_{r_4}$, r_4 cannot become eligible at t', as both of the robots r_2 and r_4 are with color `MOVE` at this time t'. Thus r_2 and r_4 must lie on the same horizontal line at t'. After any movement, r_2 and r_4 cannot form a non-horizontal collinearity with r_1, by Lemma 5.4.5. So, $\overline{r_1 r_4 r_2}$ must be a horizontal collinearity, as we have assumed that r_4 lies between r_1 and r_2 at the time t. This is a contradiction to the assumption that r_1 and r_2 lie on different horizontal lines at t. So, we proved that r_1 and r_2 lie on the same horizontal line at t when $r_2.color$ is either `OFF` or `MOVE`. Similarly, it can be proved that r_1 and r_3 lie on the same horizontal line at t, which implies all three robots r_1, r_2 and r_3 lie on the same horizontal line and they are eligible. This is again a contradiction to the Lemma 5.4.7, as there cannot be two eligible robots at t. □

Lemma 5.4.9. *A* `MOVE`*-colored robot r can see the other* `MOVE`*-colored robot, if exists.*

Proof. We prove this by contradiction. Let us consider that r' is the other robot with color `MOVE` and it is not visible to r. It implies that there is a robot r_i lying between them. Since, $Radius_{r^{Prev}} < D(r^{Prev}) \leq d(r^{Prev}, L^H_{r_i})$, it is not possible that L^H_r and $L^H_{r_i}$ are same. So $\overline{r r_i r'}$

is a non-horizontal collinearity. In addition, r and r' are two robots with color MOVE at the same time, which means they initially must lie on the same horizontal line. If r_i does not lie on $L^H_{r^{Prev}}$, any of them cannot cross the line $L^H_{r_i}$ to form the non-horizontal collinearity $\overline{r r_i r'}$. If $L^H_{r^{Prev}} = L^H_{r_i}$, none of them can move downwards from $L^H_{r^{Prev}}$ to form the collinearity $\overline{r r_i r'}$, as the movement is of Type I. This is a contradiction. □

As a summary of the above results, we can restate the following. There can be at most two MOVE-colored robots at any time (Lemma 5.4.8). They can see each other (Lemma 5.4.9). They cannot form a collinearity after the movement (Remark 5.4.6) and an OFF-colored robot cannot become eligible at this time (Remark 5.4.3). These results aid to the validity of all the lemmas from Lemma 5.3.16 to 5.3.20, mentioned in Section 5.3, that proves the correctness of the algorithm ASYNC_MV_INACC. It can also be proved by the argument presented in Theorem 5.3.21 that the movement of the robots is free from collision.

In this algorithm, only the terminal robots move and each terminal robot moves at most twice before termination. As proved in Theorem 5.3.22, the worst case time complexity stays the same, as the worst case scenario for this algorithm remains same as earlier. This takes at most $3N \approx O(N)$ epochs to terminate. So we can state the following theorem.

Theorem 5.4.10. *Algorithm* ASYNC_MV_INACC *solves the mutual visibility under ASYNC setting and terminates in* $O(N)$ *epochs using* 3 *colors where the movement of the robots might be inaccurate and the robots are assumed to have one axis agreement.*

5.5 Fault-Tolerant Mutual Visibility with Inaccuracy

This section presents the algorithm FAULT_MV_INACC that tolerates *mobility failure* of the robots along with inaccuracy in their movements under the ASYNC setting. This algorithm uses 10 colors. Table 5.2 lists down all the colors used in this algorithm with their specification. We also illustrate the transition between the colors by a flow chart in Fig. 5.26. When a fault occurs, this robot does not move thenceforth. However, it can change the color of its light without any trouble. A robot can become faulty at any point of time. All other

elements of the previous model in Section 5.2 remain unchanged. We revise the formal definition of the mutual visibility problem with inaccurate movement as follows.

Color	Specification
OFF	Used as initial color
MOVE1	Used when robot is executing first Type I movement
HALT	Used to indicate that the robot is stationary
WAITING	Used when the robot is asking for guidance from the FAULT-colored robots
MOVE2	Used when executing a movement after the initial one and $\mathcal{CV}_r$ contains either 0 or 4 robots
MOVE3	Used when executing a movement after the first one and $\mathcal{CV}_r$ contains 2 robots
FAULT	Used when the robot finds itself as faulty
COLLINEAR	Used by a FAULT-colored robot if it is collinear with a WAITING-colored robot
NON-COLLINEAR	Used by a FAULT-colored robot if it is non-collinear with every WAITING-colored visible robot
FINISH	Used by a robot at termination

Table 5.2: The list of colors used in the algorithms with their Specification

Problem 3 (Fault-Tolerant Mutual Visibility with Inaccurate Movement): N oblivious, opaque, luminous point robots, each of which operates in LCM cycles, are situated at distinct points on the plane. They agree on one coordinate axis. The movements of the robots can be inaccurate, with an angular deviation of less than 90° from their target position. Some robot may become faulty which means that such robots remain stationary thenceforth. The light of a robot remains functional even if it experiences fault. From any initial configuration, the robots must rearrange themselves such that no three non-faulty robots are collinear and no faulty robot lies between two non-faulty robots, even if f ($\leq N$) robots become faulty.

5.5.1 Technical difficulty and Overview

The definitions of Type I and Type II movements are same as earlier. Our target of reducing the number of elements from the set $\mathcal{CV}_r$ remains the same, but in case of mobility failure of the robots, it is possible that the middle robot of a collinearity is a faulty robot. Since

the faulty robots are unable to move and the middle robot is the only robot that can detect the collinearity, it is not enough just to reduce the number of robots in $\mathcal{CV}_r$. It may happen that $\mathcal{CV}_r$ is empty for the robot r, but r is still collinear with a faulty robot as the last robot of collinearity and the first robot of the same collinearity is also a non-faulty robot. So in this algorithm, whenever this situation occurs, we move the last robot of collinearity using the guidance of the faulty visible robots. This is done using colors from a prefixed color set. We still manage to use a constant number of colors to achieve our goal. Any non-faulty robot r executes movement at most four times. The first movement is a mandatory Type I movement with a specific color. There can be two situations after this movement. First is when $\mathcal{CV}_r$ might contain 4 robots after this movement, where the robot r needs to execute two other movements in the worst case, one Type I and another Type II to make $\mathcal{CV}_r$ empty. Another movement is needed to make r non-collinear in case of r being a last robot of collinearity with a faulty middle robot r_i. In that case, the faulty robot r_i acts as a guiding robot for r and tries to make r understand about the collinearity using some colors `FAULT`, `COLLINEAR` and `NON-COLLINEAR`. Second situation occurs when $\mathcal{CV}_r$ has at most two robots and r is collinear with a faulty robot. Here, r executes a Type II movement to make $\mathcal{CV}_r$ empty, followed by another Type II movement if it is collinear with a faulty robot. Same as earlier, we prevent r from making any new collinearity after any movement. We use three colors (`MOVE1`, `MOVE2` and `MOVE3`), each corresponds to a movement, with the help of which a robot understands whether it is a faulty robot or not.

5.5.2 Description of the Algorithm

As described in the previous algorithm ASYNC_MV_INACC, we consider distinct horizontal lines due to the assumption of y-axis agreement, shown in Fig 5.2. In that algorithm, a non-terminal robot cannot be eligible for a movement. But, in this case, the terminal robots on a horizontal line can be faulty, because of which a non-terminal robot keeps on waiting forever, even if it is non-faulty. So, we revise the eligibility criteria for a robot.

Definition 5.5.1. (Eligible robot): A robot r is called an *eligible robot* if both the following conditions are satisfied. (1) All the visible robots in $H^{Up}_{L^H_r}$ are either with color `FINISH` or

FAULT. (2) r is a terminal robot or has two neighbours with one of them having color FAULT or FINISH.

The set of all non-neighbour visible robots of r whose current colors are either FAULT or COLLINEAR or NON-COLLINEAR, is represented by $\mathcal{FV}_r$ and defined as $\mathcal{FV}_r = \{r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r |\ r_i.color$ = FAULT or COLLINEAR or NON-COLLINEAR$\}$. We modify the strategy FIND_TARGET_DISTANCE as follows. However, the definitions of $\mathcal{M}_r, \mathcal{V}_r, \mathcal{NB}_r, \mathcal{NCV}_r$ and $\mathcal{CV}_r$ remain same.

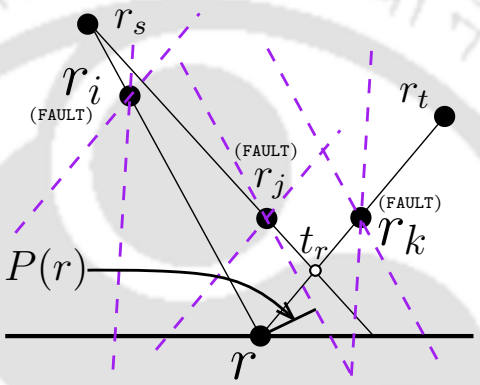


Figure 5.17: $P(r)$ prevents r moving to t_r

Strategy MODIFIED_TARGET_DISTANCE: r computes four distances $R(r), D(r), N(r)$ and $P(r)$, out of which $R(r), D(r)$ and $N(r)$ are calculated as described in FIND_TARGET_DISTANCE. If $\mathcal{FV}_r$ has exactly one element, r does not need to calculate $P(r)$. Otherwise,

- $P(r) = \min\{d(r,L) |\ L \parallel \overleftrightarrow{rr_i}$ and L passes through $r_j; \forall r_i, r_j \in \mathcal{FV}_r; r_i \neq r_j\}$. If r sees any two robots r_i and r_j with color FAULT, it considers a line L which is passing through r_j and parallel to the line $\overleftrightarrow{rr_i}$. It then calculates the distance from its current position to the line L. The minimum of all such distances is represented by $P(r)$. Fig. 5.17 illustrates the significance of calculating $P(r)$. In this particular example, $r_i, r_j, r_k \in \mathcal{FV}_r$. Since, $r_s \notin \mathcal{V}_r$ as r_i lies between r and r_s, the robot r might move to the point t_r which makes r collinear not only with r_t and r_k, but with r_s and r_j. As both of the faulty robot r_j and r_k will become middle robots of the respective collinearity if r moves to t_r, r cannot figure out a unique reference line after the movement to break both the collinearities.

Finally, r calculates $Radius_r = \frac{1}{2}\min\{R(r), D(r), N(r), P(r)\}$.

First Movement: When r gets activated, it first checks whether it is an *eligible robot* with $r.color$ = OFF or not. If not, r does not move or change its current color. Otherwise, it uses the strategy MODIFIED_TARGET_DISTANCE to calculate the distance $Radius_r$ and chooses the horizontal line L_r^H as the current reference line. It then executes a Type I movement after setting $r.color$ to MOVE1. Note that before this movement, $\mathcal{FV}_r$ consists of all robots having color FAULT. At this moment, there is no robot with color COLLINEAR or NON-COLLINEAR.

When r gets activated with $r.color$ = MOVE1, it recognizes this as an indication of completing its first movement. If r finds that the set $\mathcal{CV}_r$ has more than four robots, it implies that r encountered mobility failure and so it changes its color to FAULT, as $\mathcal{CV}_r$ of a non-faulty robot r cannot contain more than four elements after any movement, by Lemma 5.5.3. Otherwise, it changes its color to HALT without any movement to indicate other visible robots that r is stationary. Since two robots can be eligible at any time, r may observe another robot r' at different positions on the plane under ASYNC activation setting. r with the color $r.color$ = HALT, maintains status quo if one of the two situations occurs.

- (S1) r finds r' with the color MOVE1.
- (S2) r finds r' in $H^{Up}_{L_r^H}$ with $r'.color \in$ {HALT, WAITING, MOVE2, MOVE3}.

In case of (S1), since $r'.color$ = MOVE1, r lets r' finish its movement which gets confirmed once $r'.color$ becomes HALT. In case of (S2), r' lies in the upper half plane $H^{Up}_{L_r^H}$ because of which r' gets the priority to terminate before r. When r' becomes a robot either with color FINISH or FAULT, r gets the chance. If both of (S1) and (S2) do not occur, it means that either no such r' exists or r finds the HALT-colored robot r' lying on L_r^H or in the lower half plane $H^{Low}_{L_r^H}$. In this situation, r needs to figure out whether it is a middle robot of some collinearity or not, by checking its current $\mathcal{CV}_r$. Depending on the number of robot in the set $\mathcal{CV}_r$, we can distinguish three cases.

- **Case 1 ($\mathcal{CV}_r$ contains four robots):** This case occurs when two robots r and r' have executed the first Type I movement together from the line L_I^H, but r' lying in $H^{Up}_{L_r^H}$

terminates before r, as shown in Fig 5.18. When r gets the chance to execute the algorithm further, all other robots are either with color OFF, FAULT or FINISH and it finds that $r_k, r', r_j, r_j^c \in \mathcal{CV}_r$. In this case, we can prove in Corollary 5.5.4 that r, being a non-faulty robot, cannot be collinear with a FAULT-colored robot as a last robot of collinearity. Our objective here is to make r execute at most two more movements before termination. At this point, it is not possible for r to determine a unique reference line for movement. So, r follows the strategy MODIFIED_TARGET_DISTANCE to find $Radius_r$ and takes the horizontal line L_r^H passing through r as the reference line. It then changes its color to MOVE2 from HALT and executes a Type I movement (see Fig 5.19). After this movement, when r gets activated with the color MOVE2, it is still possible that $\mathcal{CV}_r$ has two elements, as the last movement might be inaccurate. Here, we identify three sub-cases.

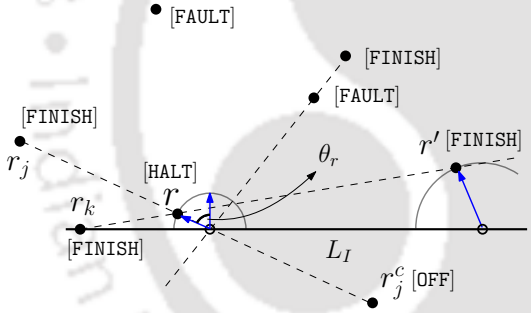


Figure 5.18: r and r' both executed Type I movements after which $\mathcal{CV}_r$ has 4 robots. r now gets the chance after $r'.color$ become FINISH

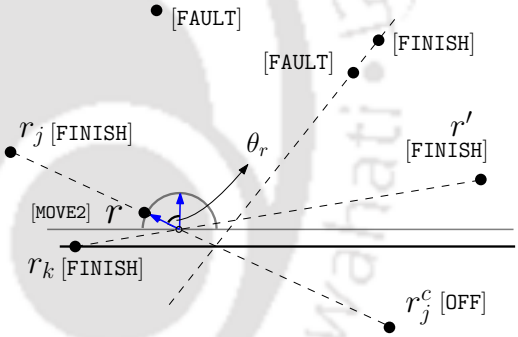


Figure 5.19: r executes another Type I movement with color MOVE2 when $\mathcal{CV}_r$ has 4 robots, so that $\mathcal{CV}_r$ has at most two robots after the movement

– **Case 1.1 ($\mathcal{CV}_r$ becomes empty):** This indicates that r is not a middle robot of any collinearity. Since the current color of r is MOVE2, r realizes that it is not collinear with any of the FAULT-colored robot. So, it changes its color to FINISH and terminates.

– **Case 1.2 ($\mathcal{CV}_r$ contains two elements):** At this stage, the current color of r is MOVE2 and the choice of the reference line is simple for r, as the LOC is unique. If r_j and r_j^c are the robots in $\mathcal{CV}_r$ as shown in Fig 5.20, r selects the line $\overleftrightarrow{r_j r_j^c}$ as

the reference line for the next movement and calculates $Radius_r$. r executes a Type II movement after setting its color to MOVE3 from MOVE2. The robot r might become faulty during this movement. So, we can again identify two sub-cases here.

* **Case 1.2.1** ($\mathcal{CV}_r$ **contains two robots):** After activation, when r finds this with color MOVE3, it understands that r was a part of collinearity as a middle robot in its previous LCM cycle. Since, r also finds two robots in $\mathcal{CV}_r$ at the current LCM cycle, it figures out that the movement was not successfully executed and it has encountered mobility failure. So, it changes its color to FAULT.

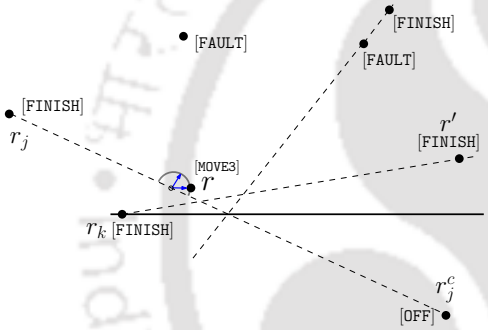


Figure 5.20: r now executes a Type II movement with color MOVE3 to break the collinearity with r_j and r_j^c when the reference line is unique

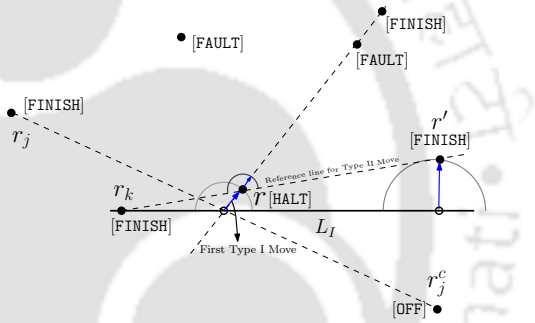


Figure 5.21: After the first movement, r might move to a point where $\mathcal{CV}_r$ contains 2 robots and r is collinear with a FAULT-colored robot

* **Case 1.2.2** ($\mathcal{CV}_r$ **is empty):** r changes the current color to HALT from MOVE3. After this, r does not change its color and position till (S1) or (S2) holds. Otherwise, it changes its color to WAITING to seek guidance from the FAULT-colored robots, which is described in GUIDING_SUBROUTINE.

– **Case 1.3** ($\mathcal{CV}_r$ **contains four robots):** This happens when r is actually faulty which means it could not executed the movement, but the color has been changed according to the algorithm. So, when r is active with the color MOVE2, it changes its color to FAULT.

• **Case 2** ($\mathcal{CV}_r$ **contains two robots):** Since the set $\mathcal{CV}_r$ has two robots, and $r_j, r_j^c \in \mathcal{CV}_r$,

r uniquely determines the LOC which is taken as the reference line in the current LCM cycle, as shown in Fig 5.21. So, r calculates $Radius_r$ and executes Type II movement after setting its color to MOVE3 from HALT. After this, there can be similar situation as described in Case 1.2.1 and Case 1.2.2 and r follows the respective strategy.

- **Case 3 ($\mathcal{CV}_r$ is empty):** It is still possible that the robot r remains collinear with a FAULT-colored robot. In this case, r detects the LOC with the help of the FAULT-colored robot. When (S1) and (S2) do not hold, r changes its color to WAITING from HALT and follows GUIDING_SUBROUTINE to terminate.

GUIDING_SUBROUTINE: When r is with color WAITING, it is possible that an other robot r' also moved and changed its color to WAITING in sync with r. In this situation, $L_r^H = L_{r'}^H$, as otherwise the robot with lower y-coordinate would have not changed its color to WAITING from HALT. They might execute the process to determine their respective LOCs and move. Since r itself cannot understand whether it is a part of any collinearity or not, as $\mathcal{CV}_r = \emptyset$, the robots with color FAULT act as guiding robots in determining the LOC by setting their color either to COLLINEAR or NON-COLLINEAR. Until all FAULT-colored robots change their color either to COLLINEAR or to NON-COLLINEAR, r maintains status quo. Since both r and r' are with the color WAITING, it is possible that two FAULT-colored robots change their color to COLLINEAR, one for r and the other for r'. Consider a configuration, shown in Fig

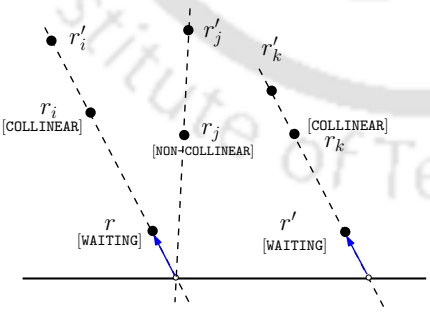


Figure 5.22: Two robots are with color WAITING and two robots with the color COLLINEAR

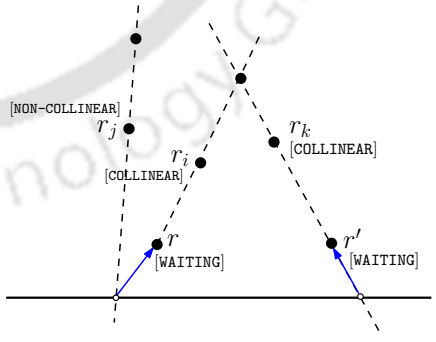


Figure 5.23: Two robots with color WAITING and two with COLLINEAR, but r'_k is not visible

5.22, where r was apart of two collinearities $\overline{r'_i r_i r}$ and $\overline{r'_j r_j r}$ before the Type I movement (which is also referred as the first movement). Similarly, we consider that r' was a part

of the collinearity $\overline{r'_k r_k r'}$ before the first movement. All of r_i, r_j and r_k have color FAULT. Due to inaccuracy, r and r' moved to the points on $\overleftrightarrow{r'_i r_i}$ and $\overleftrightarrow{r'_k r_k}$ respectively after the first movement. This makes r and r' retain their collinearity $\overline{r'_i r_i r}$ and $\overline{r'_k r_k r'}$. When r and r' are with the color WAITING, both r_i and r_k are able to detect the collinearities as middle robots, because of which both of them change their color to COLLINEAR. The robot r_j changes its color to NON-COLLINEAR as it is not a part of any collinearity either with r or r'. At this point, r sees two robots with color COLLINEAR and hence it has to distinguish that out of $\overleftrightarrow{r r_i}$ and $\overleftrightarrow{r r_k}$, which one is actually its own LOC. Depending on the number of robots with color COLLINEAR, we distinguish the following five cases. We will explain each of them with proper examples.

- **Case A (r sees r' with color WAITING and two other robots with color COLLINEAR):** As illustrated in Fig 5.22, if r sees a third robot r'_k on $\overleftrightarrow{r' r_k}$ (or on $\overleftrightarrow{r' r_i}$), it recognizes that the color COLLINEAR shown by r_k (or by r_i) is the indication for r', not for its own. Therefore, it chooses the line $\overleftrightarrow{r r_i}$ (or $\overleftrightarrow{r r_k}$) as its LOC. If r does not see any third robot on $\overleftrightarrow{r' r_k}$ and $\overleftrightarrow{r' r_i}$, it understands that the third robot must be either on the intersection of $\overleftrightarrow{r r_i}$ and $\overleftrightarrow{r' r_k}$ or on the intersection of $\overleftrightarrow{r r_k}$ and $\overleftrightarrow{r' r_i}$, as shown in Fig. 5.23. Since r sees the intersection of $\overleftrightarrow{r r_k}$ and $\overleftrightarrow{r' r_i}$, the third robot must be on the intersection of $\overleftrightarrow{r r_i}$ and $\overleftrightarrow{r' r_k}$. So, r chooses the line $\overleftrightarrow{r r_i}$ as its LOC.

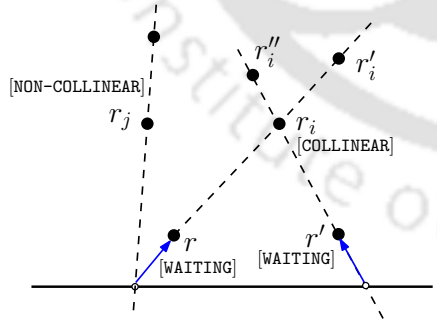


Figure 5.24: Two robots with color WAITING and one with COLLINEAR

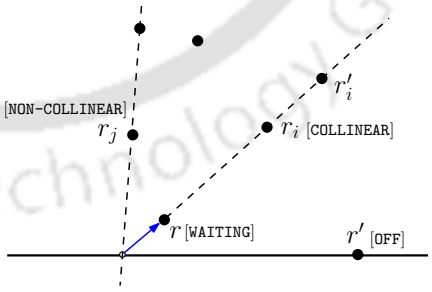


Figure 5.25: One robot with color WAITING and one with COLLINEAR

- **Case B (r sees r' with color WAITING and exactly one robot with color COLLINEAR):** Let us consider a configuration where r and r' are part of the collinearities $\overline{r'_i r_i r}$ and $\overline{r''_i r_i r'}$, as depicted in Fig. 5.24. Both r and r' are with color WAITING while r_i is with

color COLLINEAR, indicating both of the collinearities $\overline{r_i' r_i r}$ and $\overline{r_i'' r_i r'}$. The robot r_j is with color NON-COLLINEAR. In this case, r and r' respectively consider $\overleftrightarrow{r r_i}$ and $\overleftrightarrow{r' r_i}$ as the LOC. Note that even if r_i is only collinear with r', r assumes the line $\overleftrightarrow{r r_i}$ as its LOC. Importantly, this action does not result in new collinearity of r.

- **Case C (r sees no such r', but a robot with color** COLLINEAR**)**: If the robot r_i is with the color COLLINEAR, r simply considers the line $\overleftrightarrow{r r_i}$ as its LOC, as shown in Fig. 5.25.

- **Case D (r sees no robot with color** COLLINEAR**):** This case appears when all the FAULT-colored robots are not collinear with r. So, r changes its color to FINISH and terminates.

- **Case E (r sees no robot with** WAITING **but multiple robots with** COLLINEAR**) or (r sees another robot with** WAITING **but more than two robots with** COLLINEAR**):** We prove in the Lemma 5.5.6 that a WAITING-colored robot can be collinear with at most one FAULT-colored robot and so as at most COLLINEAR-colored robot. It means that r must be faulty in this case. Since $\mathcal{CV}_r$ is empty, r does not need to guide any robot in its $\mathcal{CV}_r$. So, it changes its color to FINISH and terminates.

After the LOC is detected, r calculates the $Radius_r$ and considers the current LOC as the reference line and executes a movement according to the type of the reference line (horizontal or non-horizontal) with color MOVE2. Note that at the stage, $\mathcal{CV}_r$ is always empty. So, r after its next activation, finds the set $\mathcal{CV}_r$ as empty and $r.color$ = MOVE2, so it terminates after changing its color to FINISH from MOVE2, similar to Case 1.1.

Action of FAULT**-colored robot:** r having the current color FAULT, acts as a guiding robot for non-faulty robots. Until r sees at least one WAITING-colored robot, it maintains its current color with no movement. When it sees a WAITING-colored robot r', it first checks whether there is a HALT-colored robot on $L^H_{r'}$. If such a robot r'' exists, r waits till r'' changes its color to WAITING or MOVE3. Now r changes its color to COLLINEAR from FAULT, if r is collinear with r' or r'' or both. In case of r not being collinear with both r' and r'', it changes its color to NON-COLLINEAR from FAULT. In case of r having either the color COLLINEAR or NON-COLLINEAR and no visible robot with color WAITING, it changes its color to FAULT

again. r being a FAULT-colored robot terminates when $\forall r' \in \mathcal{CV}_r$, $r'.color$ is either FINISH or FAULT.

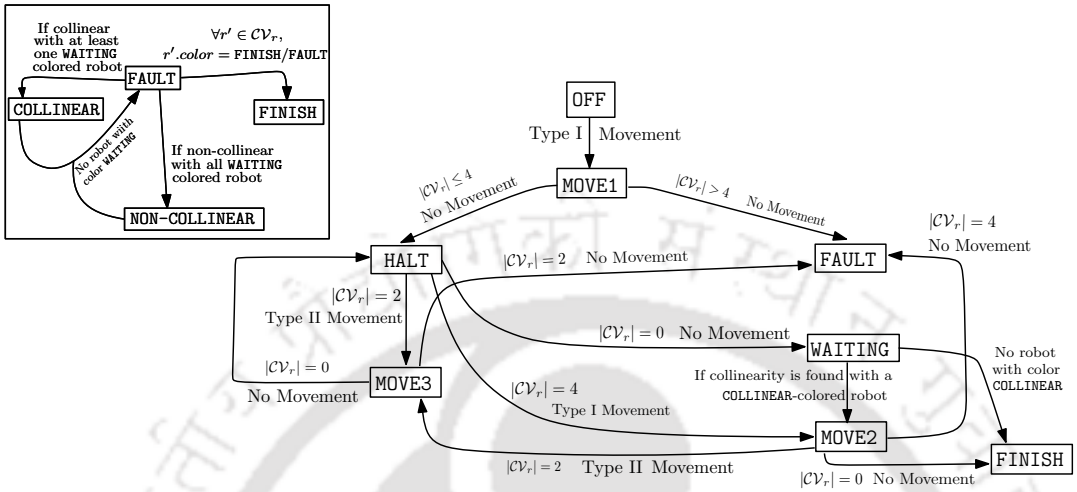


Figure 5.26: The color transitions throughout the algorithm FAULT_MV_INACC

5.5.3 Analysis of the Algorithm

Here, we discuss the correctness and time complexity of FAULT_MV_INACC. We take the help of the lemmas proved in the previous sections. Since we follow the same kind of movement strategy discussed in previous sections, we can prove that the movement of the robots are free from collision by a similar argument as Theorem 5.3.21. One of the main difference in the eligibility criteria of a robot r between Section 5.4 and 5.5 is that some of the robots in $H^{Up}_{L^H_r}$ might have color FAULT and some might have the color FINISH. Since a FAULT-colored robot remains stationary similar to a FINISH-colored robot, we only use a FAULT-colored robot as a guiding robot for some non-faulty robots. The arguments we put forth in Section 5.4 regarding FINISH-colored robots remain applicable even when the robots are colored FAULT. In Section 5.4, we have shown that some of the lemmas in Section 5.3 gets validated under ASYNC setting by introducing a new intermediate color MOVE. Here in this section, we also introduce new intermediate colors. Let us summarize which Lemmas are still valid for a non-faulty robot in case of fault-tolerant mutual visibility. At most two robots can move simultaneously (Lemma 5.3.6). A non-faulty robot r never forms a new collinearity as a middle robot after a movement (Remark 5.3.14) and it does not be-

Algorithm 13: FAULT_MV_INACC

if $r.color = \texttt{OFF} \wedge$ *all robots in* $H^{Up}_{L^H_r}$ *are with color* `FINISH` *or* `FAULT` $\wedge$ r *has either no neighbour or one neighbour or both neighbours out of which at least one is with color* `FAULT` *or* `FINISH` **then**
  Identify $\mathcal{M}_r$, $\mathcal{V}_r$, $\mathcal{NB}_r$, $\mathcal{NCV}_r$, $\mathcal{CV}_r$ and $\mathcal{FV}_r$ and follow MODIFIED_TARGET_DISTANCE()
  Choose L^H_r as the reference line and set $r.color$ to `MOVE1`
  Follow MOVE_TO_TARGET()
else if $r.color = \texttt{MOVE1}$ **then**
  Set $r.color$ to `HALT`
else if $r.color = \texttt{HALT}$ **then**
  if r *finds a robot* r' *in* $H^{Up}_{L^H_r}$ *with color either* `HALT` *or* `WAITING` *or* `MOVE2` *or* `MOVE3` **then**
    r does not change its color or position.
  else
    Identify $\mathcal{M}_r$, $\mathcal{V}_r$, $\mathcal{NB}_r$, $\mathcal{NCV}_r$, $\mathcal{CV}_r$ and $\mathcal{FV}_r$
    if $\mathcal{CV}_r$ *has more than 4 robots* **then**
      Set $r.color$ to `FAULT`
    else if $\mathcal{CV}_r$ *has exactly 4 robots* **then**
      Follow MODIFIED_TARGET_DISTANCE() and choose L^H_r as the reference line
      Follow MOVE_TO_TARGET() after setting $r.color$ to `MOVE2`
    else if $\mathcal{CV}_r$ *has exactly 2 robots* r_j *and* r^c_j **then**
      Choose the line $L = \overleftrightarrow{r_j r^c_j}$ as the reference line and follow MODIFIED_TARGET_DISTANCE()
      Follow MOVE_TO_TARGET() after setting $r.color$ to `MOVE3`
    else // $\mathcal{CV}_r$ `is empty`
      Set $r.color$ to `WAITING`
else if $r.color = \texttt{MOVE2}$ **then**
  if $\mathcal{CV}_r$ *has 4 robots* **then**
    Set $r.color$ to `FAULT`
  else if $\mathcal{CV}_r$ *has two robots* **then**
    Choose the line $L = \overleftrightarrow{r_j r^c_j}$ as the reference line and follow MODIFIED_TARGET_DISTANCE()
    Follow MOVE_TO_TARGET() after setting $r.color$ to `MOVE3`
  else
    Set $r.color$ to `FINISH` and terminate
else if $r.color = \texttt{MOVE3}$ **then**
  if $\mathcal{CV}_r$ *has two elements* **then**
    Set $r.color$ to `FAULT`
  else
    Set $r.color$ to `HALT`
else if $r.color = \texttt{FAULT}$ *or* `COLLINEAR` *or* `NON-COLLINEAR` **then**
  FAULT_COLOR_SUBROUTINE()

else if *r.color = WAITING and all the FAULT-colored robot change their color to COLLINEAR or NON-COLLINEAR* **then**
    FIND_LOC()
    if *LOC does not exist or LOC is not unique* **then**
        Set *r.color* to FINISH and terminate
    else
        Choose the LOC as the reference line and follow MODIFIED_TARGET_DISTANCE
        Follow MOVE_TO_TARGET() after setting *r.color* to MOVE2
else
    No change in color or position

Algorithm 14: MODIFIED_TARGET_DISTANCE()

// The subroutine calculates the distance between the target and the current position of r
Calculate $R(r) = \min\{d(r, \overleftrightarrow{r_i r_j}) \mid r_i \neq r_j \neq r_i^c; r_i \in \mathcal{NCV}_r \cup \mathcal{CV}_r; r_j \in \mathcal{NCV}_r \cup \mathcal{CV}_r \cup \mathcal{M}_r\}$
Calculate $D(r) = \min\{d(r, L^H_{r_i}) \mid r_i \in \mathcal{V}_r \setminus \mathcal{NB}_r\}$
Calculate $N(r) = \min\{d(r, m_{r_i}) \mid m_{r_i} \in \mathcal{M}_r\}$
Calculate $P(r) = \min\{d(r, L) \mid L \parallel \overleftrightarrow{rr_i}$ and L passes through $r_j; \forall r_i, r_j \in \mathcal{FV}_r; r_i \neq r_j\}$
Calculate $Radius_r = \frac{1}{2}\min\{R(r), D(r), N(r), P(r)\}$

come a non-terminal robot after a movement (Lemma 5.3.15). If r is a terminal robot, $\mathcal{CV}_r \subseteq \mathcal{CV}_r^{prev}$ (Lemma 5.3.16) and $\mathcal{CV}_r$ contains at most two robots after a Type I movement (Lemma 5.3.17). For a non-faulty robot r, the set $\mathcal{CV}_r$ becomes empty after a Type II movement (Lemma 5.3.18). Additionally, we prove following lemmas for the correctness of the algorithm FAULT_MV_INACC.

Lemma 5.5.2. *Two non-faulty robots with color WAITING lie on the same horizontal line and can see each other.*

Proof. A robot sets its color to WAITING without any movement, when the current color is HALT. Consider any two robots r and r' with color WAITING. By similar arguments given in Lemma 5.4.9 for MOVE-colored robots, we can prove that two WAITING-colored robots are visible to each other. So, r and r' are visible to each other, even when they are with the color HALT. At that time, they cannot change their color to WAITING, if they lie on different horizontal lines. □

Lemma 5.5.3. *After the first movement of a non-faulty robot r, $\mathcal{CV}_r$ contains at most four robots.*

Algorithm 15: FIND_LOC()

```
// This subroutine enables r to determine its LOC with the help of
FAULT-colored robots
if all robots in H^{Up}_{L^H_r} are with color either FINISH or COLLINEAR or NON-COLLINEAR then
    if there is no other robot with color WAITING and ∀ r_i ∈ FV_r, r_i.color ≠ COLLINEAR then
        LOC does not exist
    else if there is no other robot with color WAITING, but multiple robots with color COLLINEAR ∧
      there is one robot r' with color WAITING and more than two robots with color COLLINEAR then
        LOC is not unique
    else if there is at most one robot with color WAITING and exactly one robot r_i with color
      COLLINEAR then
        Choose ←→r_i r as the LOC.
    else if there is one robot r' with color WAITING and two other robots r_i and r_j with color
      COLLINEAR then
        if a third robot r_k is visible to r on ←→r_i r' then
            Choose ←→r_j r as LOC
        else if a third robot r_k is visible to r on ←→r_j r' then
            Choose ←→r_i r as LOC
        else
            P is the point of intersection of the lines ←→r r_i and ←→r' r_j
            if P is visible to r then
                Choose ←→r_j r as LOC
            else
                Choose ←→r_i r as LOC
    else
        LOC does not exist
else
    No change in color and position
```

Proof. We will prove this by contradiction. Note that the set $\mathcal{CV}_r$ always contains robots in pair. Let us assume that $\mathcal{CV}_r$ contains 6 robots $r_i, r_i^c, r_j, r_j^c, r_k$ and r_k^c. Among these robots, at most one, say r_i has moved simultaneously with r, as shown in Fig 5.18 (By Lemma 5.3.6). All the other robots must be stationary at this moment. By Remark 5.3.14, r cannot form a new collinearity as a middle robot. So, $r_j, r_j^c, r_k, r_k^c \in \mathcal{CV}_r^{Prev}$. This is a contradiction, as $\overleftrightarrow{r_j r_j^c}$ and $\overleftrightarrow{r_k r_k^c}$ cannot intersect at two points r and r^{Prev}. □

Corollary 5.5.4. *After the first movement of a non-faulty robot r, if $\mathcal{CV}_r$ contains 4 robots, r cannot be collinear with a FAULT-colored robot which is not a member of $\mathcal{CV}_r$.*

Proof. It follows from the similar argument given in Lemma 5.5.3. □

Algorithm 16: FAULT_COLOR_SUBROUTINE()

```
// This subroutine is for the FAULT, COLLINEAR and NON-COLLINEAR-colored
robots
if r.color is FAULT then
    if r sees a WAITING-colored robot r' and there is no HALT-colored robot on L^H_r' then
        No change in color
    else if r is collinear with at least one WAITING-colored robot then
        Set r.color to COLLINEAR
    else if r is non-collinear with all WAITING-colored robots then
        Set r.color to NON-COLLINEAR
    else if ∀r' ∈ CV_r, r'.color is either FINISH or FAULT then
        Set r.color to FINISH and terminate
    else
        No change in color
else // r.color is COLLINEAR or NON-COLLINEAR
    if no robot with color WAITING is visible then
        Set r.color to FAULT
    else
        No change in color
```

Lemma 5.5.5. *After at most three movements of the non-faulty robot r, $\mathcal{CV}_r$ becomes empty.*

Proof. Lemma 5.5.3 proves that the set $\mathcal{CV}_r$ has at most four robots, after the first movement. By Lemma 5.3.15, we prove that the robot r remains terminal after the first Type I movement. As discussed in Case 1, if $\mathcal{CV}_r$ contains 4 robots, r executes another Type I movement. Lemma 5.3.17 justifies that $\mathcal{CV}_r$ contains at most two robots after this movement. In worst case ($\mathcal{CV}_r$ still has two robots due to inaccurate movement), a Type II movement is needed to make its $\mathcal{CV}_r$ empty, which is justified in Lemma 5.3.18, as mentioned in Case 1.2 and Case 2. Thus at most three movements makes $\mathcal{CV}_r$ empty. □

Lemma 5.5.6. *A WAITING-colored non-faulty robot r can be collinear with at most one robot as the last robot of collinearity.*

Proof. Since $r.color$ = WAITING, r has already executed a movement and $\mathcal{CV}_r$ is empty at this point. We can prove this by contradiction. Let us assume that r_i and r_j are the two robots such that r is collinear as a last robot of the collinearities $\overline{r'_i r_i r}$ and $\overline{r'_j r_j r}$. Let r^0 be the initial position of r. When r was at r_0, it must be collinear with both r_i and

r_j, as otherwise, r cannot be collinear to them again, by Remark 5.3.14. So, $\overleftrightarrow{r_i' r_i}$ and $\overleftrightarrow{r_j' r_j}$ intersects at r_0 as well as at r. This is a contradiction. □

Lemma 5.5.7. *If a* `WAITING`*-colored non-faulty robot r is a part of the collinearity $\overline{r_i' r_i r}$, the color of r_i must be* `FAULT`*,* `COLLINEAR` *or* `NON-COLLINEAR`*.*

Proof. Since r_i lies above r, it must be of color either `FAULT` or `COLLINEAR` or `NON-COLLINEAR` or `FINISH`. It is not possible for r to become collinear with r_i if r was not collinear with it earlier. Also $r_i', r \in \mathcal{CV}_{r_i}$. So, r_i must have had the color `FAULT` earlier at some point in time. r_i cannot set its color to `FINISH` from `FAULT`, as $r.color \neq$ `FAULT` or `FINISH` at this time. So, $r_i.color$ must be `FAULT`, `COLLINEAR` or `NON-COLLINEAR`. □

Lemma 5.5.8. *If $r.color$ is* `FINISH` *for a non-faulty robot r, it cannot be either a middle robot or last robot or be a part of horizontal collinearity.*

Proof. The mandatory first movement eliminates the horizontal collinearity for r. Since the current color of r is `FINISH` and it is a non-faulty robot, it must have the color `MOVE2` and the corresponding $\mathcal{CV}_r$ is empty at some earlier point of time (Case 1.1, 1.2.2 and 3). So, r cannot be a middle robot of collinearity. Before Case 1.1 occurs, r must have had 4 robots in $\mathcal{CV}_r$ after its first movement. By Corollary 5.5.4, r cannot be a part of collinearity with a `FAULT`-colored robot as a last robot. It is also not possible for r to be a last robot of collinearity with a `FINISH`-colored robot r_i, as otherwise r_i would have not changed its color to `FINISH`. When Case 1.2.2 and 3 occur, r executes GUIDING_SUBROUTINE before it changes its color to `FINISH` from `MOVE2`. So, at some point of time, r must have the color `WAITING`. By Lemma 5.5.6 and 5.5.7, r can be collinear with at most one robot with color `FAULT`. This collinearity gets eliminated by a movement after detecting the LOC, as discussed in Cases A, B and C. So when $r.color$ becomes `FINISH`, r cannot be a last robot of some collinearity. □

Theorem 5.5.9. *Algorithm* FAULT_MV_INACC *solves the mutual visibility among the non-faulty robots under ASYNC setting in $\mathcal{O}(N)$ epochs using* 10 *colors, where N is the number of robots, out of which some of them might exhibit mobility failure as well as inaccurate movement and the robots agree on one coordinate axis.*

Proof. Let r and r' be two non-faulty robots. There cannot be a non-faulty robot lying between them, by Lemma 5.5.8. Let us assume that there is a faulty robot r'' for which $\overline{rr''r'}$ holds. This collinearity cannot be a horizontal collinearity by Lemma 5.5.8. So, one of r and r' has to be the last robot of collinearity which is also not possible by Lemma 5.5.8. Hence, at termination, any two non-faulty robots can see each other.

A non-faulty robot might take at most 15 epochs before termination when it is a non-terminal robot. A non-terminal robot r needs 3 movements if a configuration depicted in Fig 5.18 occurs, where another robot r' moves simultaneously with r and forms a collinearity after the first movement. r and r' complete their first movement with color `MOVE1` in 1 epoch and change the color to `HALT` in 1 epoch. At this moment, $\mathcal{CV}_r$ has four elements including r'. r now waits till $r'.color$ becomes either `FINISH` or `FAULT`. In the worst case, $\mathcal{CV}_{r'}$ might contain two robots after the first movement. So, r' takes 1 epoch to complete a Type II movement with color `MOVE3` and another epoch to change its color to `HALT`. Although at this point, $\mathcal{CV}_{r'}$ becomes empty, it is possible that r' is collinear as a last robot with a `FAULT`-colored robot. r' takes 1 epoch to change its color to `WAITING` and waits for 1 epoch in which all the visible `FAULT` colored robots convert their color either to `COLLINEAR` or `NON-COLLINEAR`. Thereafter, r' needs 1 epoch to execute a movement with color `MOVE2` and one more epoch to change its color either to `FAULT` or `FINISH`. After this, it is still possible that r' becomes faulty because of which r' is still in $\mathcal{CV}_r$. r requires 1 epoch to execute a Type I movement with color `MOVE2`, one more epoch to perform Type II movement with color `MOVE3` and another epoch to change its color to `HALT`. Afterwards, r might be collinear with a `FAULT`-colored robot. So, r needs 4 epochs, similar as r', to reach the termination. In total, r needs 15 epochs to reach the termination.

If r is a terminal robot, the set $\mathcal{CV}_r$ contains at most two robots after the first movement, by Lemma 5.3.17. So, it takes fewer epochs than the earlier case for r to reach termination. Since there can be at most two robots that can execute movement at any time, and each robot needs at most a constant number of epochs to terminate, the algorithm FAULT_MV_INACC takes $O(N)$ epochs to achieve mutual visibility. □

5.6 Discussion

5.6.1 Fault Tolerant SSYNC Algorithm

As it is not possible for a robot in the SSYNC setting to execute its look phase while another robot is still in the move phase, we can reduce the number of colors required for the algorithm FAULT_MV_INACC. The same algorithm can be applied in the case of the SSYNC setting with 9 colors. The reason for using an extra color in ASYNC setting is to ensure that the two eligible robots must be confirmed about the position of each other. We do not encounter this problem in SSYNC setting, they cannot see each other until the next round is started and the next round starts when all active robots finish their current LCM cycle.

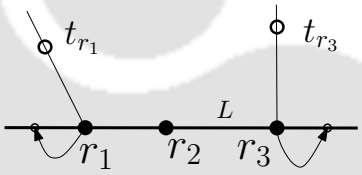


Figure 5.27: The robots always remain on the line L as the adversary chooses the angular inaccuracy to prevent them from breaking the collinearity

5.6.2 Impossibility on Angular Inaccuracy

This chapter assumes that the angular inaccuracy θ_r is less than 90°. If $\theta_r = 90°$, the problem becomes unsolvable. We can consider a simple configuration where r_1, r_2 and r_3 are the three robots lying on a line L. The adversary can choose the angular inaccuracy so that the robots never leave the line L. For example, if r_1 chooses to move to a point t_{r_1} such that $\angle t_{r_1} r_1 r_2 \leq 90°$, the adversary chooses θ_{r_1} to be $\angle t_{r_1} r_1 r_2$ which makes r_1 to move on the line L again. Even the parallel movements of multiple robots result in the same problem. Since the problem is unsolvable for $\theta = 90°$, it is also unsolvable for $\theta > 90°$.

5.6.3 Requirement of the Colors in Presence of Fault

In just one movement, it is not possible for the robot r to become non-collinear with all the robots due to inaccuracy in its movement. So, r needs to move at least two movements

to become non-collinear with others. Recall that each robot is oblivious and does not have knowledge of N, i.e., the total number of robots. If we restrict the number of colors to 2 (say `OFF` and `FINISH`), then a robot must look at some environmental parameter to reach termination. Let r' be another robot that has become faulty. Further, we consider a configuration that has a non-faulty robot r'' such that r is a part of the collinearity $\overline{r''r'r}$. As the robots are opaque, the robot r cannot understand the existence of r'' and vice versa. As a faulty robot, r' is unable to break the collinearity by executing a movement. So at least one of the robots r and r'' must execute a movement by considering $\overleftrightarrow{rr'r''}$ as a reference line, as otherwise inaccuracy in the robots' movement might bring the non-faulty robots again on the line $\overleftrightarrow{rr''}$. For r and r'' to consider the line $\overleftrightarrow{rr''}$ as the reference line, r' needs to show a third color (which is different than the prior two). Otherwise, the following happens. Let the initial color of all the robots be `OFF`. If r' does not change its color from `OFF`, it is not possible for r to figure out the collinearity $\overline{r''r'r}$ as there could be other visible robots to r which also are with color `OFF`. So, r cannot choose the line $\overleftrightarrow{rr''}$ as a reference line for its movement to break the collinearity. This is why r' must change it color to `FINISH` (as we have only two colors). Now, there could be two other faulty robots r_1 and r_2 such that $\overline{r_1r'r_1}$ is a collinearity. In this case, r' always maintains its color to `FINISH` to indicate the collinearity. So, r always thinks that it is collinear with r' and executes movement considering $\overleftrightarrow{rr'}$ as a reference line. Hence r is unable to terminate even if it is not collinear with two other robots. Similar arguments hold for the robot r''. So, more than two colors are needed to terminate the algorithm.

We used three more colors `WAITING`, `COLLINEAR`, `NON-COLLINEAR` to tackle this issue. When r shows the color `WAITING` to seek help from the faulty robot such as r', the color `COLLINEAR` is used by r' to indicate that r is collinear and the color `NON-COLLINEAR` is used to indicate when r is not collinear with r'. That is how r understands when to stop executing movement to break collinearities.

5.7 Conclusion

To the best of our knowledge, this is the first work that studies the problem of mutual visibility with robots that may exhibit inaccuracy in their movements. Inaccuracy may occur due to multiple reasons, such as hardware or software malfunction, geomagnetic hazards, etc. We consider luminous opaque robots that neither have any knowledge about the total number of robots nor agree on any global orientation. However, they agree on only one coordinate axes. The main challenge is to solve the problem using a constant number of colors and a constant number of movements for each robot. We present three algorithms. The first algorithm SSYNC_MV_INACC requires 2 colors to solve the problem of mutual visibility under the SSYNC setting when the movement of the robots can be inaccurate. The number of colors used in this algorithm is optimal, as having one color for the lights on the robots is the same as having no lights on them. A single color, shown through the light does not indicate any status or information of a robot. Our second algorithm ASYNC_MV_INACC solves the same problem under the ASYNC setting using 3 colors. The last algorithm FAULT_MV_INACC solves the problem when the robots can be faulty as well as exhibiting inaccuracy in movement. This algorithm uses 10 colors.

This study opens many interesting future directions. An immediate extension of this work is to reduce the number of colors in the fault-tolerant algorithm. It will be interesting to study the problem with no agreement on any coordinate axis. Tolerating fault with inaccuracy in movement is another interesting direction where the number of colors can be reduced. The problem can also be studied with fat robots, where a robot is considered as a unit disk.

Chapter 6

Fault-tolerant Mutual Visibility without any Axis Agreement

6.1 Introduction

In this chapter, we continue our study of the mutual visibility problem under a different model. Unlike the previous chapter, we assume no agreement on the coordinate axes among all the robots. The previous chapter addresses angular inaccuracy in robots' movement, whereas this chapter considers the movement of the robots to be accurate, but they are susceptible to mobility failure. However, the light of a robot remains functional.

So far, the mutual visibility problem is solved with the assumption of a one-axis (y-axis) agreement in the presence of faulty robots. One-axis agreement provides us with a serialization among the robots. For example, in case of y-axis agreement, the robots can be serialized from the bottom-most to the top-most with the help of colors. This becomes difficult when the robots have no agreement on coordinate axes, even in the FSYNC setting. Moreover, in this scenario, the problem cannot be solved for any initial configuration of the robots due to symmetry. If the total number of robots, N, is not known to the robots, the challenge is increased further to solve the problem of mutual visibility. We highlight the details of the model in the subsequent section.

6.1.1 Contributions

The fault-tolerant mutual visibility is studied with robots having both [4] and single [3] coordinate axes agreement. Relaxing the assumptions in the above-mentioned works, we study the fault-tolerant mutual visibility problem under a relatively weaker model. Our contributions are listed below in this chapter.

- We show an initial configuration of robots that makes the mutual visibility unsolvable in the presence of one faulty robot.
- We present Algorithm FAULT_MV$_{LC}$ that solves the mutual visibility problem in the presence of f ($\leq N$) faulty robots under the fully-synchronous setting where the robots do not agree on any global coordinate system or orientation. The algorithm requires 21 colors, runs in $O(N^2)$ rounds and is collision-free.
- We propose another algorithm FAULT_MV$_{LC}^{f=1}$, which is simpler than the prior one and tolerates a single faulty robot. This algorithm requires 2 colors in the FSYNC and SSYNC setting and 5 colors in the ASYNC setting.

6.2 Model and Preliminaries

Robots: We consider N autonomous, anonymous, homogeneous and disoriented point robots denoted by $r_1, r_2, \cdots r_N$ on the 2D plane, each of which operates under LCM cycles. These robots are *opaque* and do not know N. Robots have no agreement on the coordinate axes or the orientation. By fault, we mean the mobility failure of the robots. When a fault occurs, the robot is unable to move henceforth. The fault can occur at any time. A faulty robot cannot be identified even by itself.

Configuration: The definition of a configuration C_t of all the robots remains the same. We denote the convex hull of all the robots by $\mathcal{CH}$. Initially, the robots do not necessarily know $\mathcal{CH}$ because of the obstructed visibility. We denote $C_t(r)$ to be the configuration of all the robots that are visible to r at the time t. The convex hull of the robots visible to r is denoted by $\mathcal{CH}_r$.

Now, we can formally define the problem as follows.

Problem 4 (Fault-tolerant Mutual Visibility): We are given N oblivious, disoriented and opaque luminous robots, out of which f ($\leq N$) robots can be faulty. The fault is considered to be mobility failure. However, the lights of the faulty robots remain functional. Robots neither have any agreement on the coordinate axes nor have any knowledge about N. Starting from any arbitrary configuration of the robots where they are situated at distinct points on the Euclidean plane, the aim of the problem is to achieve a configuration where none of the three non-faulty robots is collinear, and no faulty robot lies between two non-faulty robots.

6.3 An Impossibility Result

Here we explain a configuration where N is odd and $(N-1)$ non-faulty robots are on the vertices of a regular $(N-1)$-gon, and the faulty robot is at the center of that, for which mutual visibility is unsolvable. When N is odd, any diagonal of the $(N-1)$-gon connecting the i-th and $i+\frac{N-1}{2}$-th vertices ($1 \leq i \leq \frac{N-1}{2}$) passes through the center of the polygon. In that case, the center robot blocks the view of the robot on the i-th vertex, which cannot see the robot on the $i+\frac{N-1}{2}$-th vertex. When N is even, this does not happen and mutual visibility automatically gets achieved in such a configuration.

Theorem 6.3.1. *For a regular $(N-1)$-gon, if N is odd and the faulty robot lies at the center of the $(N-1)$-gon and all other $(N-1)$ robot lie on the vertices of the $(N-1)$-gon, then the mutual visibility problem using N robots having no common coordinate system is impossible.*

Proof. The schedule of activation of the robots is controlled by adversary. Also, since the robots do not have access to a global coordinate system, the adversary has the power to choose the initial configuration and the direction of the local coordinate axes of each robot. For such a configuration, the adversary can choose the direction of the local coordinate axes in such a way that every synchronous movement of the non-faulty robots makes the convex hull of the robots again a regular polygon with the faulty robot in the center. One

such choice of the local coordinate system is the following.

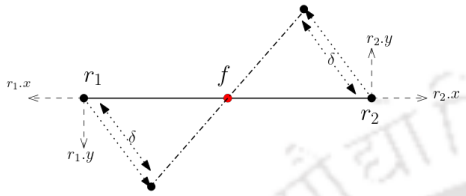


Figure 6.1: For $N = 5$, the collinear configuration is intact even after movement of non-faulty robots

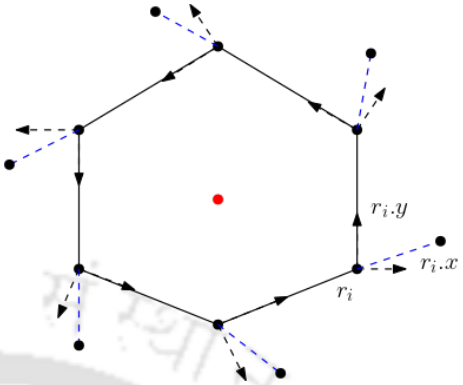


Figure 6.2: The symmetry of the $(N-1)$ robots is maintained even after movement of them

Let the robots $r_1, r_2, \cdots, r_{N-1}$ be situated on the vertices of $(N-1)$-gon in the clockwise order and r_N is at the center. Let the initial configuration be denoted as C_0. A side $S_i = (r_i, r_{i+1}), i \in \{1, 2, \cdots, N-1\}$ is a pair of consecutive robots r_i and r_{i+1} in the ordering. Note that the robots do not know this order. For a robot r_i, the direction of positive y-axis is along the side S_i and the positive x-axis is along the outward direction of the convex hull, as shown in the Fig. 6.2. Let the configuration be C_1 after a synchronous movement of the corner robots due to some deterministic algorithm $\mathcal{A}$ that solves the fault-tolerant mutual visibility problem. In Fig 6.2, the movements of the robots are indicated in dashed lines from the vertices of the polygon and the arrowhead lines are their respective coordinate axes. Since C_0 is a regular polygon (which is symmetric), the view of all the corner robots are same before they reach to C_1. Also because of the homogeneity, the robots execute the same algorithm $\mathcal{A}$. So all the robots behave identically as their views are identical. When all the $(N-1)$ corner robots get activated synchronously, their movements are also identical. So, the configuration C_1 is also identical with the previous configuration C_0 and robots cannot distinguish the current configuration C_1 from C_0, as they are oblivious. Also the robot at the center cannot move as it is faulty. So the symmetry of the configuration remains intact. Any pair of diagonally opposite robots are not visible to each other due to the obstruction by the faulty robot even after the movements of the robots. The adversary can keep repeating the synchronous movement of the corner robots so that the robots can

never be in a configuration by which the symmetry can be broken. Hence, the mutual visibility will never be achieved. □

6.4 Algorithm for Fault-tolerant Mutual Visibility

6.4.1 Technical Difficulty

Recall that the robots do not agree on any coordinate axis or orientation. When the robots agree on one coordinate axis, it becomes easier to serialize them. For example, if the direction of the y axis is the same for all the robots, we can start the algorithm for the robots starting from north-most to south-most robots, considering virtual horizontal lines. In the case of robots, without any agreement on the coordinate system, it becomes very difficult to serialize them. Moreover, it is difficult to achieve the goal for the opaque robots if all the robots are executing and moving simultaneously. In that case, the serialization of the robots becomes useful in designing algorithms to achieve mutual visibility. We introduce the concepts of layers (which are formally defined later in this section), which have a similar concept to the virtual horizontal lines in the case of y-axis agreement. We start moving the non-faulty robots starting from the outermost layer (refer to Fig 6.4) towards the innermost layer. The challenge comes when all of the robots on a layer are faulty. In that case, the robots on the next layer need to be made to understand that it is their turn to move towards the innermost layer. In the following section, we present the algorithm FAULT_MV$_{LC}$ that solves the problem under the FSYNC setting. However, the algorithm cannot solve the problem for a specific initial configuration when the innermost layer is composed of a single faulty robot. We disregard this configuration while proposing an algorithm that can tolerate any number of faulty robots using robots without any coordinate axis agreement.

In this section, we propose an algorithm FAULT_MV$_{LC}$ that runs under the FSYNC setting, where all the robots are activated simultaneously and execute their LCM cycles in sync. The movement of the robots in this algorithm is also free from collision. We take the help of a strategy called *visible area*, proposed by Poudel et al. [54]. The intuitive idea of this strategy is to reposition a specific robot to a point such that it can see all other stationary

robots present on the plane. If the initial configuration of all the robots is a line, we can make a non-linear configuration in one step as follows. After activation, if a robot r sees at most two robots, then r can understand that all the robots are in a line. In that case, the robots that see exactly two other robots will move. A robot r that sees exactly two robots after activation moves to a small fixed distance δ in the perpendicular direction of the line $\overline{rr'}$, where r' is one the visible robot to r. After this step, we get a non-linear configuration of the robots.

We now define some notations and terminologies that will be used throughout the algorithm.

Algorithmic Preliminaries:

- $\mathcal{V}_r$ is the set of all visible robots to r.
- **Layers:** The boundary of the convex hull $\mathcal{CH}$ (refer to Fig 6.3) of all the robots present on the plane, denoted by $\mathcal{CH}^0$, is the *outer-most* layer of the robots. For $j \geq 1$, $\mathcal{CH}^j$ is the boundary of the convex hull of all the robots excluding the robots on $\mathcal{CH}^0 \cup \mathcal{CH}^1 \cup \cdots \cup \mathcal{CH}^{j-1}$. We refer to $\mathcal{CH}^j$ as the *j-th layer*. $\mathcal{CH}^k$ represents the *inner-most layer* such that $\mathcal{CH}^{k+1}$ does not exist. This is illustrated in Fig 6.4.

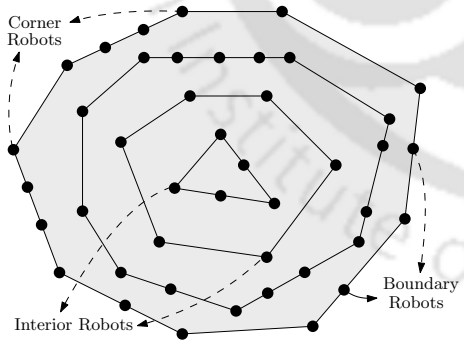


Figure 6.3: Illustrating the convex hull $\mathcal{CH}$ of all the robots and corner, boundary and interior robots

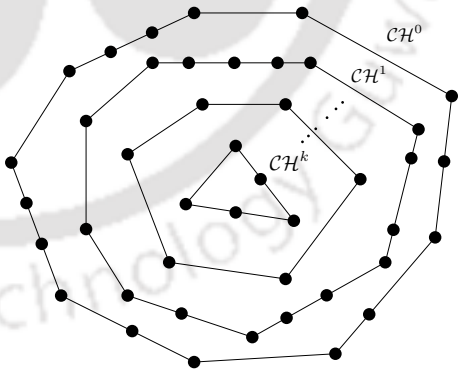


Figure 6.4: Illustrating the layers where $\mathcal{CH}^0$ and $\mathcal{CH}^k$ are the outer and inner-most layers, respectively

- $\mathcal{CH}_r^*$ is the boundary of the convex hull of all the robots in $\mathcal{V}_r$ such that none of them has color `FAULT`.

- We denote the convex hull of all the visible robots to r with the color $\#color$ as $\mathcal{CH}_r^{\#color}$.
- For two points a and b, the line segment joining them is denoted by $\overline{ab}$ and the line passing through the two points is denoted by $\overleftrightarrow{ab}$.
- For a line segment LS, e_{LS}^1 and e_{LS}^2 represent the two endpoints of it. We denote the line containing LS by $\overleftrightarrow{LS}$.
- For a line L, H_L^1 and H_L^2 represent the two closed (including L) half-planes delimited by L.
- On the convex hull $\mathcal{CH}_r$, two robots r and r' are *neighbours* of each other if the line segment $\overline{rr'}$ lies on the boundary of $\mathcal{CH}_r$ and there is no other robot lying on $\overline{rr'}$.
- **Distance Metric:** We use $d(a,b)$ as the Euclidean distance between the two points a and b. For a line L, we denote the Euclidean distance between the point a and the line L as $d(a,L)$. For a line segment LS, we denote the distance between the point a and LS as $d(a,LS)$ and define it as follows. If the foot of the perpendicular from the point a to $\overleftrightarrow{LS}$ lies on LS, then $d(a,LS) = d(a,\overleftrightarrow{LS})$. Otherwise, $d(a,LS) = \min\{d(a,e_{LS}^1), d(a,e_{LS}^2)\}$.
- $\mathcal{E}(O,a_e,b_e,L)$ denotes the ellipse whose center is O, the length of the major axis is $2a_e$ lying on the line L and the length of the minor axis is $2b_e$.

Note that the robots do not have any knowledge about the layers. We use these notations for our convenience. Table 6.1 lists down the notations which are going to be used throughout this chapter.

High-level Idea of the Algorithm: First our aim is to identify the inner-most layer of the robots. The concept of layers provides a serialization among the robots. Starting from the outer-most layer, a robot r moves to a side of the subsequent inner layer $Next\_\mathcal{CH}_r^*$. Our purpose is to eventually reach $\mathcal{CH}^{k-1}$ (when the inner-most layer is linear) or $\mathcal{CH}^k$ (when the inner-most layer is non-linear). After this, the robots move in such a way that they

Variables	Specifications
t_r	The target point of r
$\mathcal{V}_r$	The set of all visible robot to r
k	The total number of layers
$\mathcal{CH}^0$	The outer-most layer of robots
$\mathcal{CH}^j$	The j-th layer of robots
$\mathcal{CH}^k$	The inner-most layer of robots
H_L^1, H_L^2	The two closed half planes delimited by the line L
$\mathcal{CH}_r^*$	The boundary of the convex hull of all visible the robots to r without the color FAULT
$Next\_\mathcal{CH}_r^*$	The boundary of the convex hull either of all INNER-colored robots or of all the robots in $\mathcal{V}_r \setminus \mathcal{CH}_r^*$ without the color FAULT
$\mathcal{CH}_r^{\#color}$	The convex hull of the robots in $\mathcal{V}_r$ with the $\#color$
$\overline{ab}$	The line segment with the endpoints a and b
$\overleftrightarrow{ab}$	The line passing through the points a and b
e_{LS}^1, e_{LS}^2	The two endpoints of the line segment LS
a_r	The half length of the major axis of the target ellipse of r
b_r	The half length of the minor axis of the target ellipse of r
cen_r	The center of the target ellipse of r
$\mathcal{E}(cen_r, a_r, b_r, L)$	The target ellipse of r whose major axis lies on L
TS_r	The target side of r
$len(LS)$	The length of the line segment LS
$d(a, L)$	The Euclidean distance between a and L
LIC	The linear inner-most configuration
$\mathcal{CH}_r^{LIC}$	The convex hull of all the robots without the color FAULT lying on the same half plane delimited by $\overleftrightarrow{LIC}$ where r is situated.
r_{Nbr_1}, r_{Nbr_2}	The two neighbours of r on its convex hull
p_{mid_1}, p_{mid_2}	The midpoints of $\overline{r r_{Nbr_1}}$ and $\overline{r r_{Nbr_2}}$

Table 6.1: The list of notations and conventions used throughout the algorithm

either become a corner of a common convex hull inside $\mathcal{CH}^k$ or reach on some points on an ellipse.

6.4.2 Description of the Algorithm

Initially, all the robots are situated at distinct points on the plane, and all of them have the color OFF. We first define a corner, boundary and an interior robot (refer to Fig 6.3).

Definition 6.4.1. (**Corner Robot, Boundary Robot, Interior Robot**): A robot r is called a *corner robot* if there exist two distinct lines, L_1^r and L_2^r such that r is situated at the point

Color	Specification
OFF	The initial color of all the robots
OUTER	Used for all the robots does not lie on the inner-most layer
WAIT	Used when a robots finds itself on the inner-most layer
INNER	Used for all the robots lying on the non-linear inner-most layer
LINEAR	Used for robots lying on the linear inner-most layer
INTERIM	An interim color for detecting the middle robots on linear inner-most layer
NON-CENTER	Used for robots which are other than middle robots on the linear inner-most layer
CENTER1	Used for middle robots on the linear inner-most layer with even number of robots
CENTER2	Used for middle robot on the linear inner-most layer with odd number of robots
CORNER	Used for a corner robot which is moving to its visible area
MOVE-O	Used for movement of CORNER-colored robots not moving to the inner-most layer
MOVE-I	Used for movement of CORNER-colored robots moving to the inner-most layer
FAULT	Used when a MOVE-O or MOVE-I-colored robot finds itself as corner robot
BOUNDARY	Used for MOVE-I-colored boundary robots
CORNER1	Used for a corner robot on the inner-most layer
BOUNDARY1	Used for a boundary robot on the inner-most layer
MOVE-C	Used for a CORNER1-colored robot moving inside the convex hull
MY-TURN	Used for a BOUNDARY1-colored robot with a CORNER1-colored neighbour
CORNER2	Used for a robot before it moves to the ellipse in case of linear inner-most layer
INSPECT	Used for the movement of a NON-CENTER-colored robot to its visible area
FINISH	Used at termination

Table 6.2: The list of colors with their specification

of intersection of these two lines and all other visible robots are on $H^i_{L^1_r} \cap H^j_{L^2_r}$ for some $i, j \in \{1,2\}$. Otherwise, r is either a boundary or an interior robot. If there exists a line L_r passing through r such that all the visible robots lie either on $H^1_{L_r}$ or on $H^2_{L_r}$, then r is categorized as a *boundary robot*. If no such line exists, the robot r is called an *interior robot*.

In other words, r is a corner robot if it lies on some vertex of $\mathcal{CH}_r$. First, we identify the innermost layer using colors. Then, our target is to bring all the non-faulty robots to the inner-most layer, starting from the outer-most layer $\mathcal{CH}^0$. We first discuss the strategy, named DETECT_INNERMOST_LAYER for detecting the inner-most layer using colors of the

robots.

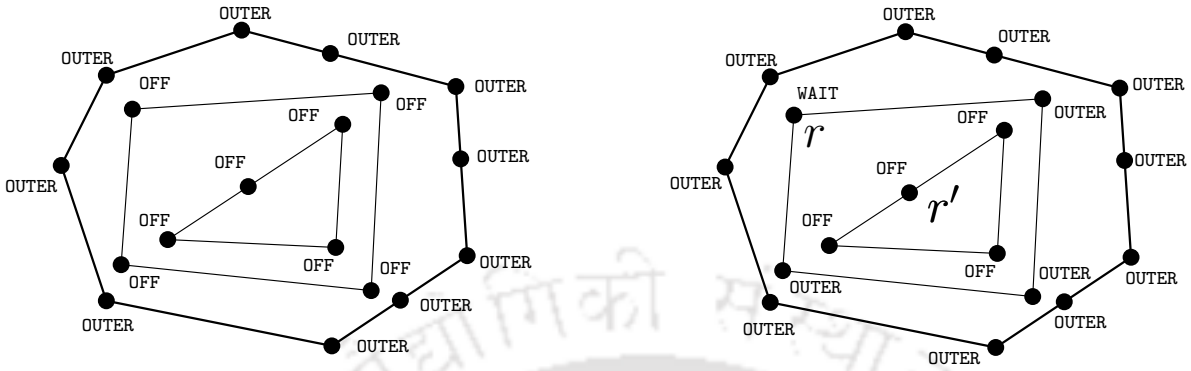


Figure 6.5: (Left) The robots on $\mathcal{CH}^0$ switches their color to OUTER from OFF. (Right) After finding no interior robots, the robot r on $\mathcal{CH}^1$ changes its color to WAIT.

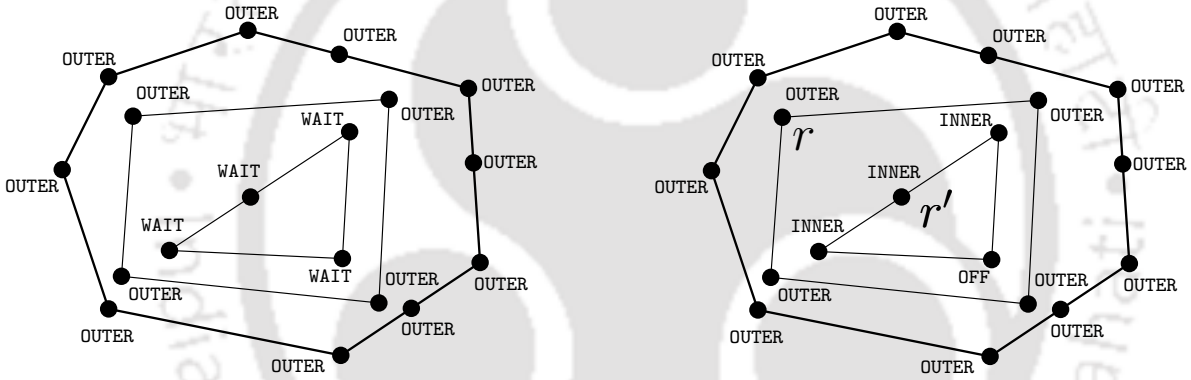


Figure 6.6: (Left) r corrects itself in the next round after seeing OFF-colored robots. (Right) The robots on the innermost layer change their color WAIT.

Strategy DETECT_INNERMOST_LAYER: If r gets activated with the color OFF, it considers the convex hull $\mathcal{CH}_r^{\texttt{OFF}}$ of all the OFF-colored visible robots. If r is an interior robot on $\mathcal{CH}_r^{\texttt{OFF}}$ it remains stationary and maintains its current color OFF. Otherwise, if r finds itself as a corner or a boundary robot with some interior robot on $\mathcal{CH}_r^{\texttt{OFF}}$, it sets $r.color$ to OUTER without any movement, as depicted in Fig. 6.5. If no interior robot is found on the convex hull $\mathcal{CH}_r^{\texttt{OFF}}$, r changes its color either to LINEAR or to WAIT with no movement depending on the configuration of $\mathcal{CH}_r^{\texttt{OFF}}$ (linear or non-linear). If $\mathcal{CH}_r^{\texttt{OFF}}$ is a linear configuration, r changes its color to LINEAR, otherwise it changes to WAIT. In case of $\mathcal{CH}_r^{\texttt{OFF}}$ being a non-linear configuration, it is possible that the robot r misinterpreted the convex hull $\mathcal{CH}_r^{\texttt{OFF}}$ as the inner-most layer, but there must

be a robot r' on $\mathcal{CH}_r^{\texttt{OFF}}$ in this round which does not change its color to WAIT, as it is an interior robot on $\mathcal{CH}_{r'}^{\texttt{OFF}}$ (refer to Fig. 6.5). When r is activated with the color WAIT, it checks whether there is any OFF-colored robot in $\mathcal{V}_r$ or not. If such a robot exists, r changes the current color from WAIT to OUTER, as illustrated in Fig. 6.6. Otherwise ($r.color$ = WAIT and does not find OFF-colored robots), r changes the current color from WAIT to INNER.

Description of the Algorithm for Non-linear Inner-most Layer

In case of r being activated with color OUTER, it waits till it sees any WAIT and OFF-colored robot in $\mathcal{V}_r$. Seeing such a robot signifies that the innermost layer is not yet detected. Otherwise, we aim to bring all the corner OUTER-colored robots to the layer comprised of INNER-colored robots. For this, a corner robot r on $\mathcal{CH}_r^*$ (we illustrate this convex hull by the shaded region in Fig 6.7) first moves to a point so that all the stationary robots become visible and then computes $Next\_\mathcal{CH}_r^*$ to move to some point on it. The movement of the corner robots creates new corners on $\mathcal{CH}_r^*$, and the new corners eventually move to the next inner layer.

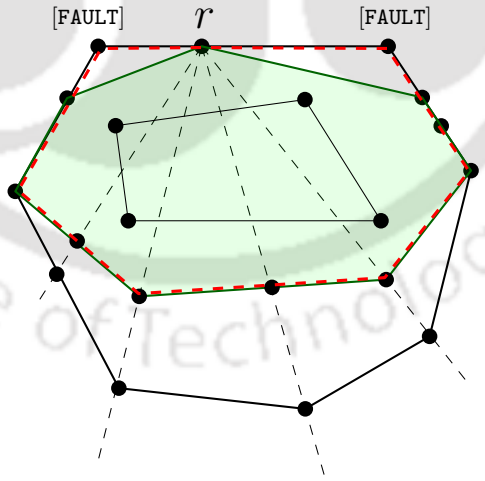


Figure 6.7: The region with the red boundary is $\mathcal{CH}_r$ whereas the green shaded region is $\mathcal{CH}_r^*$

Movement of a Corner Robot r from $\mathcal{CH}_r^*$ to Next Layer: For better comprehension, we consider an example where r is a corner robot on $\mathcal{CH}_r^*$ and, r_{Nbr_1} and r_{Nbr_2} are the two

neighbours of the robot r on $\mathcal{CH}_r^*$, as depicted in Fig. 6.8. All of them are with color OUTER. After activation, if r finds that it is a corner of $\mathcal{CH}_r^*$ (note that initially $\mathcal{CH}_r = \mathcal{CH}_r^*$) with $r.color$ = OUTER and does not have any robot in $\mathcal{V}_r$ with the color from the set {CORNER, MOVE-O,MOVE-I}, r computes its *visible area* (refer to Section 2.6 in [54]) *Visible_Area*(r, $\mathcal{CH}_r^*$) and changes its current color to CORNER from OUTER. It then moves to a point in *Visible_Area*(r, $\mathcal{CH}_r^*$). Poudel et al. [54] proved that r remains a corner robot after this movement (refer to Lemma 2.1). In the same round, r_{Nbr_1} and r_{Nbr_2}, after activation, determine that they are boundary robots on $\mathcal{CH}^*_{r_{Nbr_1}}$ and $\mathcal{CH}^*_{r_{Nbr_1}}$, respectively with color OUTER. Theses robots along with all other interior robots remain stationary with their current color OUTER. Since we only move the corner robots on $\mathcal{CH}_r^*$ and make the other robots stationary in that round, r can see the boundary robots and the interior robots after the movement, by Lemma 2.1 [54].

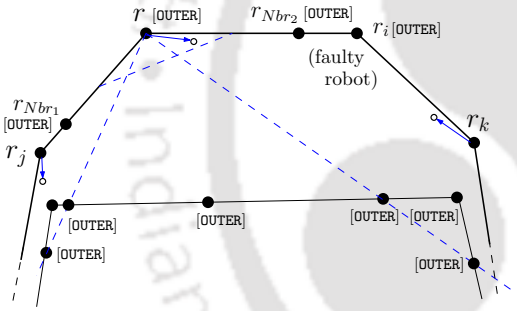


Figure 6.8: r moves to a point inside its visible area *Visible_Area*(r, $\mathcal{CH}_r^*$) so that it can see the next inner layer

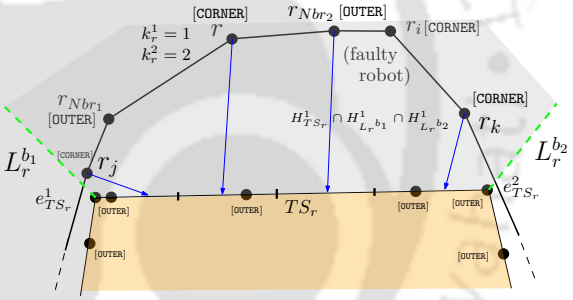


Figure 6.9: A CORNER-colored robot r moves to the target side TS_r on the next inner layer $Next\_\mathcal{CH}_r^*$

When r gets activated with color CORNER in the next round, it understands that it has already moved to a point in the visible area of its previous position which indicates that it is now able to see all the stationary robots without the color CORNER on the plane. In this case, it waits till it does not see any WAIT or OFF-colored robots. In other respects, r first compute $Next\_\mathcal{CH}_r^*$ as follows: If $\mathcal{CH}_r^*$ contains at least one robot with the color INNER, $Next\_\mathcal{CH}_r^*$ is the boundary of the convex hull of the all INNER-colored robots. Otherwise, $Next\_\mathcal{CH}_r^*$ is the boundary of the convex hull of all the robots without the color FAULT, excluding all the robots on $\mathcal{CH}_r^*$. Then it determines a target side TS_r on $Next\_\mathcal{CH}_r^*$, as shown in Fig 6.9, such that r lies on $(H^1_{\overleftarrow{TS_r}} \cap H^1_{L_r^{b_1}} \cap H^1_{L_r^{b_2}})$, where $H^1_{\overleftarrow{TS_r}}$ is the closed half plane

delimited by the line $\overleftrightarrow{TS_r}$ in which r is situated. $H^1_{L_r^{b_1}}$ (respectively $H^1_{L_r^{b_2}}$) is the closed half plane delimited by the line $L_r^{b_1}$ (respectively $L_r^{b_2}$) in which r resides, where $L_r^{b_1}$ (respectively $L_r^{b_2}$) is the bisector of the angle created by two adjacent sides of $Next\_\mathcal{CH}_r^*$ with common endpoint as $e^1_{\overleftrightarrow{TS_r}}$ (respectively $e^2_{\overleftrightarrow{TS_r}}$). In case of r having two such sides on $Next\_\mathcal{CH}_r^*$, it chooses any one of them as its target side. It now changes its current color either to MOVE-O or to MOVE-I depending on the color of robots lying on the endpoints of TS_r. So, we can identify two following cases. (i) If the robots lying on the endpoints of TS_r are with color OUTER, r changes its color to MOVE-O. (ii) If the robots lying on the endpoints of TS_r are with color INNER, r then changes its color to MOVE-I. After that, r follows the movement strategy MOVE_TO_NEXTLAYER (which is described later in this section) to move to the boundary of the next layer $Next\_\mathcal{CH}_r^*$.

In the next round, when r gets activated with the color MOVE-O, it changes its color either to OUTER or FAULT. If r finds itself as a non-corner robot on $\mathcal{CH}_r^*$, then it changes its color to OUTER. If r finds itself as a corner robot on $\mathcal{CH}_r^*$, it changes its color to FAULT, as the current color MOVE-O indicates that it has executed a movement from its previous layer, but it failed to move. Similarly, if r finds $r.color$ = MOVE-I, it changes its color either to BOUNDARY when it is a non-corner robot on $\mathcal{CH}_r^*$ or to FAULT when it is a corner robot on $\mathcal{CH}_r^*$. After this round, r_{Nbr_1} and r_{Nbr_2} get activated with color OUTER and finds itself as the corner robot of $\mathcal{CH}^*_{r_{Nbr_1}}$ and $\mathcal{CH}^*_{r_{Nbr_2}}$ respectively with no robots in $\mathcal{V}_r$ with the color from the set {CORNER, MOVE-O, MOVE-O}. They calculate their respective visible areas *Visible_Area*$(r_{Nbr_1},\mathcal{CH}^*_{r_{Nbr_1}})$ and *Visible_Area*$(r_{Nbr_2},\mathcal{CH}^*_{r_{Nbr_2}})$ respectively, and move to some points after changing their current color to CORNER. Similar to earlier, this movement creates new corner robots.

Strategy MOVE_TO_NEXTLAYER: r considers the convex hull $\mathcal{CH}_r^{\texttt{CORNER}}$ containing all the CORNER-colored robots on $H^1_{\overleftrightarrow{TS_r}} \cap H^1_{L_r^{b_1}} \cap H^1_{L_r^{b_2}}$ and the endpoints $e^1_{TS_r}$, $e^2_{TS_2}$ of the line segment TS_r. Let r_j and r_k be the two CORNER-colored robots on $\mathcal{CH}_r^{\texttt{CORNER}}$, which are neighbours of the points $e^1_{TS_r}$ and $e^2_{TS_2}$ respectively. Let us also assume that k_r^1 (respectively k_r^2) represents the number of CORNER-colored robots excluding r on $\mathcal{CH}_r^{\texttt{CORNER}}$ start counting from the robot r_j (respectively r_k) towards r in the opposite direction of r_j to $e_1^{TS_r}$ (respectively $e_2^{TS_r}$), as shown in Fig 6.9. r chooses any one of the points $e^1_{TS_r}$ and $e^2_{TS_r}$ as the

reference point. Without loss of generality, we assume that r chooses the point $e^1_{TS_r}$. It then divides the side TS_r into $k^1_r + k^2_r + 1$ segments, each of length $\frac{len(TS_r)}{k^1_r+k^2_r+1}$. Now r selects the $(k^1_r + 1)$-th segment on TS_r from the endpoint $e^1_{TS_r}$ and moves to one of the points on that segment (excluding the endpoints), which does not contain any robot. Note that if r_{Nbr_1} and r_{Nbr_2} do not exist, then $k^1_r = k^2_r = 0$ and r moves to a point on TS_r where no other robot is situated.

Movement of Robots Lying on the Inner-most Layer: If r is an `INNER` or `BOUNDARY`-colored robot with at least one visible robot having the color from the set {`OUTER`, `CORNER`, `MOVE-O`, `MOVE-I`}, r does not change its color or position, as we want such a robot to move on the boundary of the convex hull of the `INNER`-colored robots. If $r.color$ = `INNER` and r finds all the robots on the convex hull $\mathcal{CH}^*_r$ with the color {`INNER`, `CORNER1`, `BOUNDARY`, `BOUNDARY1`}, it changes its color either to `CORNER1` if it is a corner robot or to `BOUNDARY1` if it is a boundary robot on $\mathcal{CH}^*_r$. Similarly, if $r.color$ = `BOUNDARY` and r finds all the robots on the convex hull $\mathcal{CH}^*_r$ with the color from the set {`INNER`, `CORNER1`, `BOUNDARY`, `BOUNDARY1`}, it changes its color to `BOUNDARY1`. Note that a `BOUNDARY`-colored robot can not be the corner robot of $\mathcal{CH}^*_r$.

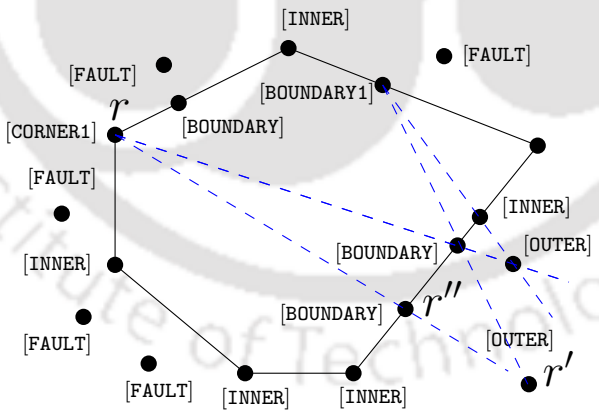


Figure 6.10: Some robots color themselves `CORNER1` and `BOUNDARY1` as they do not see robots from the outer layers

Now, our target is to make all the non-faulty robots corner of the same convex hull. When r gets activated with $r.color$ = `CORNER1` or `BOUNDARY1`, it first checks whether all the robots on $\mathcal{CH}^*_r$ also have the color either `CORNER1` or `BOUNDARY1`. If not, it remains stationary with the current color. This is because there can be a robot r' with color {`OUTER`,

MOVE-O, MOVE-I} which is not visible to r and it is yet to move to the inner-most layer, as shown in Fig 6.10. The robot r' is not visible because of a robot r'' lying on the inner-most layer which must be with the color either INNER or BOUNDARY. Otherwise, we implement the following strategy.

Strategy MAKING_CORNERS_NONLINEARCASE: We differentiate four cases depending on the current color of the robot r and also demonstrate the process with a simple example in Fig 6.11.

- **Case NL1** ($r.color$ = CORNER1)**:** If one of its neighbours r_{Nbr_1} on $\mathcal{CH}_r^*$ with color BOUNDARY1, we want r to move to some point inside the inner-most layer such that it remains a corner of the convex hull $\mathcal{CH}_r^*$ and r_{Nbr_1} also becomes a corner robot. In this case, it follows the strategy MOVING_INSIDE_CONVEXHULL(r,$\mathcal{CH}_r^*$) after changing its color to MOVE-C. If none of its neighbours is with color BOUNDARY1, it changes its current color to FINISH.

- **Case NL2** ($r.color$ = BOUNDARY1)**:** If r detects itself as a corner on $\mathcal{CH}_r^*$, it switches its color to CORNER1. Otherwise, r changes its color to MY-TURN if r sees a neighbour with color CORNER1. If both the neighbours of r are with the color BOUNDARY1, r does not change its current color. It maintains its current color also when it has one neighbour with color MOVE-C and r itself is a boundary robot.

- **Case NL3** ($r.color$ = MOVE-C)**:** r waits if it finds one of its neighbour with the color MY-TURN. If none of the neighbours has the color MY-TURN, but one of them is with the color BOUNDARY1, r sets $r.color$ to FAULT. If both of its neighbours are with the color either CORNER1 or FINISH, r changes its color to FINISH.

- **Case NL4** ($r.color$ = MY-TURN)**:** If r finds itself as a boundary robot on $\mathcal{CH}_r^*$, it switches its color to BOUNDARY1. In case of r finding itself as a corner on $\mathcal{CH}_r^*$, it changes its color to CORNER1.

Strategy MOVING_INSIDE_CONVEXHULL(r,$\mathcal{CH}(r)$): A corner robot r considers its two neighbours r_{Nbr_1} and r_{Nbr_2} on the convex hull $\mathcal{CH}(r)$. Since both of these two robots are visible to r, it computes the two midpoints p_{mid_1} and p_{mid_2} of $\overline{rr_{Nbr_1}}$ and $\overline{rr_{Nbr_2}}$,

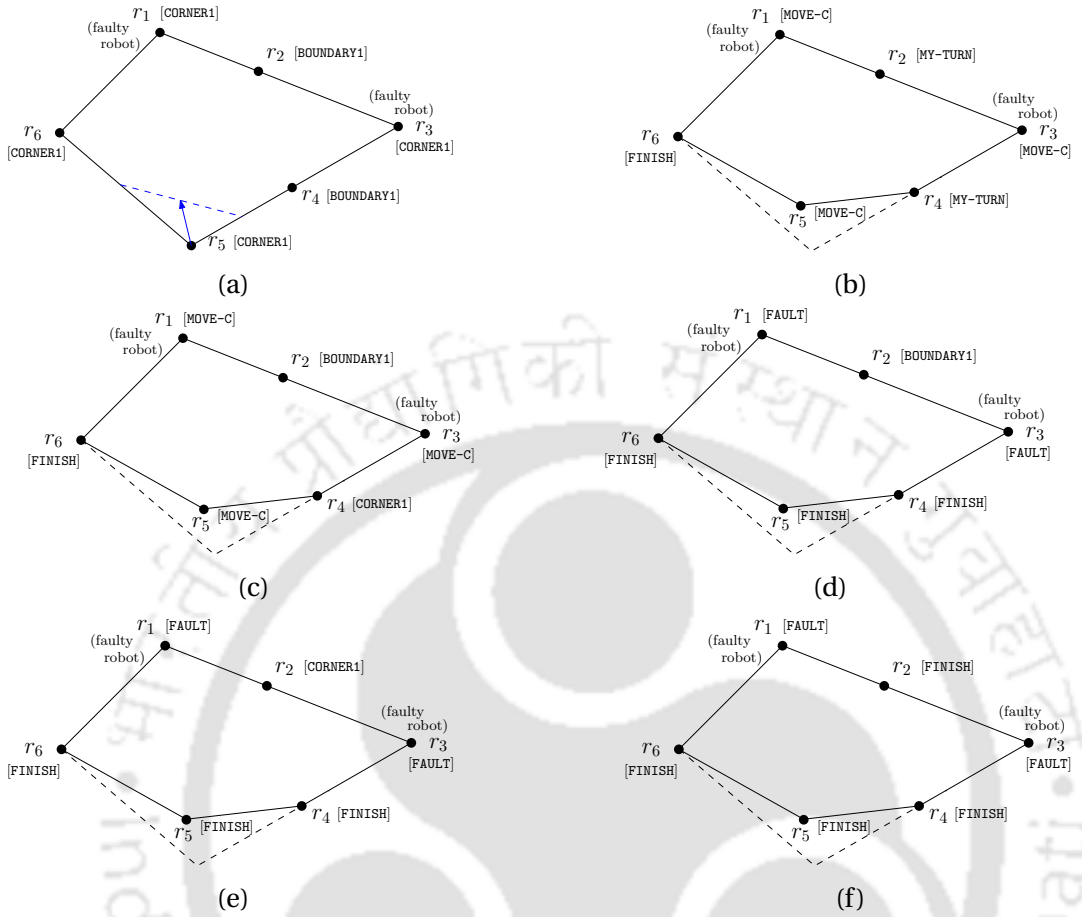


Figure 6.11: Color transitions of all the robots after they present on the inner-most layer with color {CORNER1, BOUNDARY1}

respectively. Finally, the robot r considers the triangle $\Delta r\, p_{mid_1}\, p_{mid_2}$ and moves to the midpoint of the line segment $\overline{p_{mid_1}\, p_{mid_2}}$. For example, as demonstrated in Fig 6.11(a), r_5 moves to a point inside the convex hull.

Description of the Algorithm for Linear Inner-most Layer

In this case, an OUTER-colored robot moves to the next layer (which is not the inner-most layer) similarly described in Section 6.4.2. Recall that when an OUTER-colored robot r finds itself on the corner of $\mathcal{CH}^*_r$, it first moves to the visible area with the color CORNER and then moves to one of the sides of the next layer with the color MOVE-O. In this way, the non-faulty robots move through the layers one by one until they reach the inner-most layer of the robots. Clearly, when the inner-most layer is a linear configuration (i.e. a line segment),

we cannot make the robots move to a side of the inner-most layer, as otherwise all the non-faulty robots become collinear. In that case, we perceive a different strategy which we describe in this section.

We call the robots lying on the endpoints of the linear inner-most configuration (we call it LIC henceforth) the *terminal robots*. In this case, all the robots on the inner-most layer switch their color to LINEAR. Now, we aim to find the robots with the color CENTER1 or CENTER2 which are essentially the middle-most robot on the inner-most layer. The following strategy discusses how these robots change their color either to CENTER1 or to CENTER2.

Strategy FIND_CENTER: If r is a terminal robot on LIC with the color LINEAR, it changes its color to INTERIM. If r is a LINEAR-colored robot with one of its neighbour with color INTERIM, then r also changes its color to INTERIM. If $r.color$ = INTERIM and r finds one of the neighbours with color LINEAR, it changes its color to NON-CORNER. When r gets activated with the color INTERIM and also sees one of the neighbours with the color INTERIM, it sets $r.color$ as CENTER1. In case of r getting activated with the color INTERIM with both the neighbours having color NON-CENTER, r changes its color CENTER2.

We now identify two conditions here, based on the existence of CENTER1 or CENTER2-colored robots on the LIC.

- **Case EVEN (There exists CENTER1-colored robots):** This scenario happens when there are even number of robots on the LIC. In this case, there must be a pair of CENTER1-colored robots on the LIC.
- **Case ODD (There exists CENTER2-colored robot):** This happens when there are odd number of robots on LIC. Only one CENTER2-colored robot can be there in this case.

Now, our target is to bring all the robots not lying on LIC to a common ellipse. The robots first need to agree to a common ellipse for which they need to visible to each other by following the strategy MAKING_CORNERS_LINEARCASE. After that, they move to the agreed ellipse together in one round. It is possible that the robots lying on one half plane

$H^1_{\overleftrightarrow{LIC}}$ delimited by the line $\overleftrightarrow{LIC}$ (the line containing the LIC) agree on a different ellipse than that of the robots on $H^2_{\overleftrightarrow{LIC}}$, due to obstructed visibility. Even in that case, we ensure that the two ellipses have the same center and the vertices. At last, we move the robots on the LIC to the ellipse after all other non-faulty robots finish their movement. The detailed description of this strategy is as follows.

Strategy MAKING_CORNERS_LINEARCASE: The robot r first checks whether either of the following two conditions are satisfied. (1) $Next\_\mathcal{CH}^*_r$ is empty and one side of $\mathcal{CH}^*_r$ contains robots with color from the set {LINEAR, INTERIM, NON-CENTER, CENTER1, CENTER2} (2) $Next\_\mathcal{CH}^*_r$ is made of robots with colors from the set {LINEAR, INTERIM, NON-CENTER, CENTER1, CENTER2}. If $r.color$ = CORNER or OUTER and r is a corner robot with either of the above conditions is satisfied, it changes its color to CORNER1. If $r.color$ = OUTER and r is a boundary robot with either of the above conditions is satisfied, it changes its color to BOUNDARY1. In the next round, r considers the convex hull $\mathcal{CH}^{LIC}_r$ of all the robots without the color FAULT lying on the same half plane delimited by $\overleftrightarrow{LIC}$ where r is situated. r waits till all the robots in $\mathcal{CH}^{LIC}_r$ color themselves with the color from the set {CORNER1, BOUNDARY1, LINEAR, INTERIM, NON-CENTER, CENTER1, CENTER2}. After that, r follows the similar strategy to MAKING_CORNERS_NONLINEARCASE with slight change. It considers $\mathcal{CH}^{LIC}_r$ instead of $\mathcal{CH}^*_r$ and also considers the color CORNER2 in place of FINISH. All other components of the strategy MAKING_CORNERS_NONLINEARCASE remain intact.

Movement of the CORNER2-colored robots: After the above strategy, all the robots not lying on LIC color themselves with CORNER2. A CORNER2-colored robot r waits till it finds a CENTER1 or CENTER2-colored robot. If r finds a CENTER1 or CENTER2-colored robot, it first calculates the center cen_r of its target ellipse, which is (i) the midpoint of the two CENTER1-colored robots (for Case EVEN), as illustrated in Fig 6.12 and (ii) the midpoint of the NON-CENTER-colored robots which are the neighbours of the CENTER2-colored robot (for Case ODD), as shown in Fig 6.13. We now describe how the robot r calculates the length of the two axes (major and minor) of the target ellipse. Let a_r and b_r denote the length of the major and minor axes of the ellipse for the robot r, respectively. a_r is either the

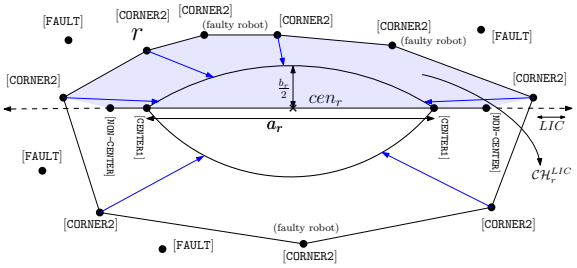


Figure 6.12: A CORNER2-colored robot r moves to a point on the ellipse for Case EVEN

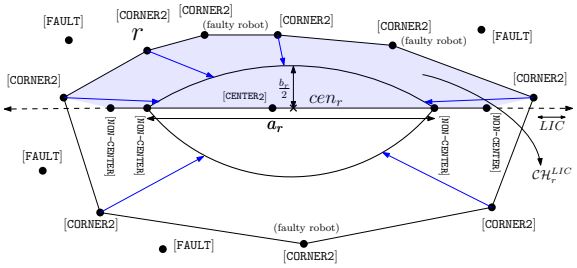


Figure 6.13: A CORNER2-colored robot r moves to a point on the ellipse for Case ODD

distance between the two CENTER1-colored robots for Case EVEN or the distance between the two NON-CENTER-colored neighbours of the CENTER2-colored robot for Case ODD. Also $b_r = \frac{1}{2}\min\{d(cen_r, r') | r'.color \in \{\texttt{CORNER2}, \texttt{CENTER1}, \texttt{CENTER2}, \texttt{NON-CENTER}\}$ and $r' \in \mathcal{CH}_r^{LIC}$ and $r' \neq cen_r\}$. r chooses the ellipse with a_r and b_r as the length of the major and minor axes respectively and the major axis lies on the line passing through the two robots with the color from the set {CENTER1, CENTER2, NON-CENTER}. Then, r moves to the point of intersection of the computed ellipse and the line segment $\overline{rcen_r}$ after changing the color to FINISH. We proved in Lemma 6.4.16 that all the non-faulty robots with color CORNER2 lying on the same open half-plane delimited by $\overleftrightarrow{LIC}$ move to the common ellipse.

Movement of the robots lying on the LIC: If r is a NON-CENTER-colored robot, it waits till it finds all the visible robots with the color from the set {FAULT, FINISH, NON-CENTER, CENTER1, CENTER2}. r considers itself as a terminal robot on the LIC if it finds exactly one robot r' with the color from the set {NON-CENTER, CENTER1, CENTER2}. We segregate two cases here for a terminal NON-CENTER-colored robot r with r' as its neighbour on the LIC.

- **Case L1 ($r'.color$ = NON-CENTER or CENTER1):** It computes the line $L^{\perp}_{\overline{rr'}}$ which is perpendicular to $\overline{rr'}$ and passing through itself. It now considers a set $S_{\overline{rr'}}$ consisting all the robots lying on the open half plane delimited by the line $L^{\perp}_{\overline{rr'}}$ where r' is situated. Finally, it computes the convex hull $\mathcal{CH}_{\overline{rr'}}$ of all the robots from the set $S_{\overline{rr'}} \cup \{r\}$ without color FAULT and moves to its visible area $Visible\_Area(r, \mathcal{CH}_{\overline{rr'}})$ with the color INSPECT, as shown in Fig 6.14. If r is a non-faulty robot, it can see all the CENTER1 and CENTER2-colored robot after its movement to the visible area, as the CENTER1 and CENTER2-colored robots are stationary in this round. When it gets

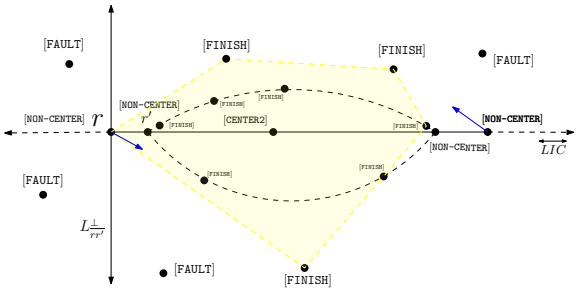


Figure 6.14: A terminal NON-CENTER-colored robot r on LIC moves to its $Visible\_Area(r, \mathcal{CH}_{\overline{rr'}})$

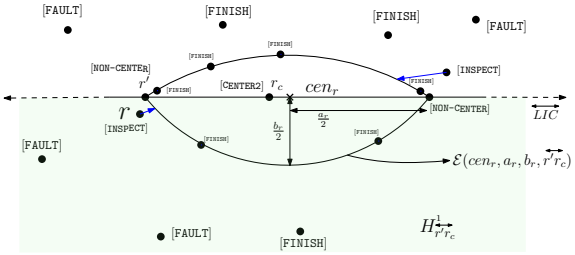


Figure 6.15: An INSPECT-colored robot r moves to a point on ellipse $\mathcal{E}(cen_r, a_r, b_r, \overleftrightarrow{r'r_c})$

activated with the color INSPECT, it checks whether a CENTER1 or CENTER2-colored robot $r_c(\neq r')$ is visible or not. If not, it changes its color to FAULT. Otherwise, it calculates cen_r and a_r (similarly as a CORNER2-colored robot does). It further calculates $b_r = \min\{b_{r_i} |\ r_i \in H^1_{\overleftrightarrow{r'r_c}}$, and $r_i.color =$ FINISH$\}$, where $H^1_{\overleftrightarrow{r'r_c}}$ is the half plane delimited by the line $\overleftrightarrow{r'r_c}$ such that r lies on it and $2b_{r_i}$ is the length of the minor axis of the ellipse $\mathcal{E}(cen_r, a_r, b_{r_i}, \overleftrightarrow{r'r_c})$ passing through r_i . Now r moves to the nearest empty point on the ellipse $\mathcal{E}(cen_r, a_r, b_r, \overleftrightarrow{r'r_c})$ with the color FINISH, as depicted in Fig 6.15.

- **Case L2** ($r'.color =$ CENTER2)**:** In this case, r is already on a ellipse where the non-faulty FINISH-colored robots are situated. So, it changes it color to FINISH and terminates.

We now demonstrate the movement of the CENTER1 or CENTER2-colored robots to a point, when they find all other visible robots with the color from the set {FINISH, FAULT, CENTER1}. If r itself is a CENTER1-colored robot, it terminates with the color FINISH, as it is already on a vertex of the ellipse. In case of $r.color =$ CENTER2, r calculates a *movable area* by avoiding the line $\overleftrightarrow{r_ir_j}$ for all $r_i, r_j \in \mathcal{V}_r$. Now it moves to a point t_r inside the movable area such that $0 < d(r, t_r) < \min\{d(r, \overleftrightarrow{r_ir_j}) |\ r_i, r_j \in \mathcal{V}_r$ and $\overleftrightarrow{r_ir_j}$ does not pass through $r\}$ with the color FINISH.

Description of the Algorithm when the Inner-most Layer is a Point

Recall that we disregard the case if the inner-most layer is made of a single robot r and it is a faulty robot. In case of r being a non-faulty robot lying on the inner-most layer, it changes

its color to LINEAR by following the strategy DETECT_INNERMOST_LAYER. When all the non-faulty robots position themselves on the layer $\mathcal{CH}^{k-1}$ and color with either CORNER1 (for corner robots) or BOUNDARY1 (similarly as described in Section 6.4.2), r moves to an empty point on the boundary of the convex hull of all the CORNER1 and BOUNDARY1-colored robots after changing its color to BOUNDARY1. Afterwards, all the robots start following the strategy MAKING_CORNERS_NONLINEARCASE to terminate themselves.

6.4.3 Analysis of the Algorithm

In this section, we analyze the correctness and the time complexity of the above algorithm. We also prove that the robots do not meet any collision during the algorithm.

Lemma 6.4.2. *Only the robots lying on the inner-most layer eventually color themselves as INNER when the inner-most layer is a non-linear configuration.*

Proof. If there is exactly one layer of robots, all the robots first change their color to WAIT in one round. After this round when all the robots find no robot with color OFF, they change their color to INNER.

We now discuss the proof of the above statement when there is more than one layer of robots. Without loss of generality, we consider that there are exactly two layers $\mathcal{CH}^0$ and $\mathcal{CH}^1$. In the first round, all the robots on $\mathcal{CH}^0$ change their respective colors either to OUTER or to WAIT. When a robot lying on $\mathcal{CH}^0$ changes its color to OUTER, it must have found an OFF-colored interior robot on its local convex hull which indicates that the robot correctly identified that there is an inner layer of robots. A robot r on $\mathcal{CH}^0$ which has changed its color to WAIT, must have not found any interior robot on $\mathcal{CH}_r^{\texttt{OFF}}$, leading to an incorrect detection of the layer. So, there exists a robot r' on $\mathcal{CH}^1$ which obstructed the line of sight of r to see another robot r'' on $\mathcal{CH}^0$, as similarly shown in Fig 6.5. Since r' can identify itself as an interior robot, it does not change its color to WAIT, when the robot r has changed its color to WAIT in the same round. r' maintains its color to OFF in that round. In the next round, when r gets activated with the color WAIT and finds r' with color OFF, it simply changes its color to OUTER, which leads to the correct identification of the layers. After this round, r' along with all other robots on $\mathcal{CH}^1$, changes their color to WAIT, as they do

not find any interior robot on their respective convex hulls of all the OFF-colored robots. Finally, all the robots on $\mathcal{CH}^1$ change their color to INNER, as they do not find any OFF-colored robot. □

Remark 6.4.3. Only the robots lying on the inner-most layer eventually color themselves as LINEAR when the inner-most layer is a linear configuration.

Lemma 6.4.4. *All the robots in the inner-most layer color themselves as INNER in $O(N)$ rounds when the inner-most layer is a non-linear configuration.*

Proof. A robot changes its color to INNER when its current color is WAIT and does not find any OFF-colored robot. A robot switches its color to WAIT, if it does not find any interior robot on the convex hull made by all OFF-colored visible robots.

Let us assume that there are $k(>1)$ layers of robots $\mathcal{CH}^0, \mathcal{CH}^1, \cdots \mathcal{CH}^k$ where $\mathcal{CH}^0$ and $\mathcal{CH}^k$ are the outer-most and the inner-most layer, respectively. In one round, all the robots in $\mathcal{CH}^0$ change their color to OUTER from OFF. The robots on $\mathcal{CH}^j$ $(1 \leq j \leq k-1)$ set their color to OUTER in at most two rounds after all the robots lying on $\mathcal{CH}^{j-1}$ with the color OUTER. A robot r on $\mathcal{CH}^j$ changes its color WAIT from OFF after seeing no interior robot on $\mathcal{CH}_r^{\texttt{OFF}}$. In the next round, it must finds at least one OFF-colored robot on $\mathcal{CH}^{j+1}$ and switches its color to OUTER. So, all the robots except on the inner-most layer changes their color to OUTER in $O(k)$ rounds. After that, the robot on the inner-most layer changes their color to INNER in 2 rounds. Hence, all the robot on inner-most layer changes their color to INNER in $O(N)$ rounds, as in the worst case, $k = O(N)$. □

Remark 6.4.5. All the robots in the inner-most layer color themselves as LINEAR in $O(N)$ rounds when the inner-most layer is a linear configuration.

Lemma 6.4.6. *If a non-faulty OUTER-colored robot is a corner robot on $\mathcal{CH}_r^*$ and it moves to some point on its visible area* Visible_Area$(r, \mathcal{CH}_r^*)$ *with the color CORNER, it remains a corner robot on $\mathcal{CH}_r^*$ and sees all the stationary robots on the plane.*

Proof. It follows from Lemma 2.1 in [54]. □

Lemma 6.4.7. *If a* CORNER*-colored non-faulty robot r decides to move on the side TS_r of $Next\_\mathcal{CH}_r^*$ (where $Next\_\mathcal{CH}_r^*$ is not a line), it can see all the other* CORNER*-colored robots which are also eligible to move on TS_r.*

Proof. Let $H^1_{\overleftrightarrow{TS_r}}$ be the closed half plane delimited by the line $\overleftrightarrow{TS_r}$ that contains the robot r. Let r' be another non-faulty robot with color CORNER which has also chosen the side TS_r as its target side. Since all the corner robots on $Next\_\mathcal{CH}_r^*$ are stationary, $Next\_\mathcal{CH}_r^* = Next\_\mathcal{CH}_{r'}^*$, by Lemma 6.4.6. Also, $Next\_\mathcal{CH}_r^*$, being a non-linear, all the robot which choose the side TS_r as their target sides must be on one of the half plane delimited by TS_r. So, r and r' lie on the $H^1_{TS_r}$.

Now we prove that r and r' are visible to each other. On contrary, let us assume that there exists a robot r'' such that r'' lies on the line segment $\overline{rr'}$. Since r and r' lie on the same half plane $H^1_{\overleftrightarrow{TS_r}}$, so is r'' which indicates that r'' is not a robot on $Next\_\mathcal{CH}_r^*$. If $r''.color =$ CORNER, the robot r'' must be a corner robot on $\mathcal{CH}^*_{r''}$, which lead to a contradiction to the assumption that r'' lies on the line segment $\overline{rr'}$. If $r''.color \neq$ CORNER, r'' must be stationary in the previous round when r and r' decided to move to their respective visible region with the color CORNER. Without loss of generality, we further assume that r'' is a neighbour of r in the previous round. After the movement of r, the robot r'' must have been a corner robot in this round, which again leads to a contradiction. □

Lemma 6.4.8. *If two* CORNER*-colored robots r and r' choose the target sides TS_r and $TS_{r'}$ respectively for their movement, then r and r' execute their movement without any collision.*

Proof. Without loss of generality, we assume that r and r' do not lie on any of the angle bisectors $L_r^{b_1}$, $L_r^{b_2}$, $L_{r'}^{b_1}$ and $L_{r'}^{b_2}$. If $TS_r \neq TS_{r'}$, the two robots r and r' cannot collide as the two regions $H^1_{\overleftrightarrow{TS_r}} \cap H^1_{L_r^{b_1}} \cap H^1_{L_r^{b_2}}$ and $H^1_{\overleftrightarrow{TS_{r'}}} \cap H^1_{L_{r'}^{b_1}} \cap H^1_{L_{r'}^{b_2}}$ intersect at most one of the bisectors and they move to their target sides within their respective regions.

We now prove the lemma when $TS_r = TS_{r'}$. By Lemma 6.4.7, r and r' can see each other along with all other CORNER-colored robot whose target side is also TS_r. Moreover, $Next\_\mathcal{CH}_r = Next\_\mathcal{CH}_{r'}$. So, $(k_r^1 + k_r^2 + 1) = (k_{r'}^1 + k_{r'}^2) + 1$, which is the total number of CORNER-colored robots all of whose target side is TS_r. Thus r and r' divide TS_r into the same number of segments of length $\frac{len(TS_r)}{k_r^1+k_r^2+1}$. Further, we assume that r and r' neighbours

of each other on $\mathcal{CH}_r^*$ with r situated nearer to $e^1_{TS_r}$ than r'. If r and r' both choose $e^1_{TS_r}$ and $e^1_{TS_{r'}}$ as their reference point with $e^1_{TS_r} = e^1_{TS_{r'}}$, r selects the $k^1_r + 1$-th segment on TS_r whereas r' selects $k^1_r + 2$-th segment on TS_r from $e^1_{TS_r}$. On the other hand, if $e^1_{TS_r} \neq e^1_{TS_{r'}}$, $k^2_{r'} = k^1_r + 1$. In this case, r selects the $k^1_r + 1$-th segment on TS_r from the point $e^1_{TS_r}$ and r' selects $k^1_{r'} + 1$-th segment on TS_r from $e^1_{TS_{r'}}$, which is the $k^2_{r'} + 1$-th segment on TS_r from $e^2_{TS_{r'}} = e^1_{TS_r}$. As $k^2_{r'} = k^1_r + 1$, we get $k^2_{r'} + 1 > k^1_r + 1$. Since r is nearer to $e^1_{TS_r}$ than r' and the chosen segment is also nearer to $e^1_{TS_r}$ than that of r', the paths followed by the two robots do not cross each other. Hence the movement of the two robots are collision-free. □

Lemma 6.4.9. *If the inner-most layer is a non-linear convex hull, then all the* `OUTER`*-colored non-faulty robots eventually settle themselves on the boundary of the convex hull made by all the* `INNER`*-colored robots without any collision.*

Proof. If r is a non-faulty corner robot on $\mathcal{CH}^0$, by Lemma 6.4.8, r eventually moves to the next layer $\mathcal{CH}^1$ without collision. If r is a non-faulty boundary robot on $\mathcal{CH}^0$, we can prove that r eventually become a corner robot on $\mathcal{CH}_r^*$ and eligible to move to the next layer $\mathcal{CH}^1$. We assume that r' is a corner robot on $\mathcal{CH}^0$ with r as one of its neighbour. In one round, r' changes its color to `CORNER` and moves to a point in *Visible_Area*$(r', \mathcal{CH}^*_{r'})$. If r' is a non-faulty robot which executes the movement successfully, then r becomes a corner robot on $\mathcal{CH}_r^*$. Otherwise (r' is a faulty robot), r' eventually changes its color to `MOVE-O` or `MOVE-I` from `CORNER`. In the next round, when it finds itself as a corner robot on $\mathcal{CH}^*_{r'}$, it changes its color to `FAULT`. After this round, r becomes a corner robot on $\mathcal{CH}_r^*$ avoiding all the `FAULT`-colored robot in $\mathcal{V}_r$. So, r eventually moves to $\mathcal{CH}^1$ without collision by Lemma 6.4.8. Whenever r becomes a corner robot on $\mathcal{CH}_r^*$ with $Next\_\mathcal{CH}_r^*$ as the boundary of the convex hull made by the `INNER`-colored robots, it moves to the boundary of it with the color `MOVE-I` after finite rounds. Thus all the robots eventually settle themselves on the boundary of the inner-most layer. □

Lemma 6.4.10. *For two consecutive layers $\mathcal{CH}^{j-1}$ and $\mathcal{CH}^j$ $(j > 1)$, if $\mathcal{CH}^{j-1}$ contains N' robots, it takes $O(N')$ rounds for the robots on $\mathcal{CH}^{j-1}$ to move to $\mathcal{CH}^j$ after all the robots on the inner-most layer change their color to* `INNER`*.*

Proof. We assume that all the robots on $\mathcal{CH}^0, \mathcal{CH}^1, \cdots \mathcal{CH}^{j-2}$ move to some points on $\mathcal{CH}^{j-1}$. After that, all the corner robots on $\mathcal{CH}^{j-1}$ move together to the layer $\mathcal{CH}^j$ in two rounds with the color MOVE-O or MOVE-I. It needs one more round for them to change their color either to FAULT, or OUTER or BOUNDARY. Some of the boundary robots on $\mathcal{CH}^{j-1}$ has now become corner robots and follow the same strategy to move to $\mathcal{CH}^j$. In worst case, there can be $O(N')$ many robots on one of the sides of $\mathcal{CH}^j$. So, it needs $O(N')$ rounds for them to become corner robots and in those many round, they move to $\mathcal{CH}^j$. □

Remark 6.4.11. All the non-faulty robots move to the inner-most layer $\mathcal{CH}^k$ in $O(N^2)$ rounds, when the inner-most layer is a non-linear configuration.

Lemma 6.4.12. *If a non-faulty corner robot r follows the strategy* MOVING_INSIDE_CONVEXHULL*$(r, \mathcal{CH}^*_r)$, r remains a corner robot and the neighbouring robot r_{Nbr_1} on $\mathcal{CH}^*_r$ also becomes a corner robot after the movement.*

Proof. Let r_{Nbr_2} be the neighbour of r on $\mathcal{CH}^*_r$ other than r_{Nbr_1}. Since r is a corner robot before the movement, all the other robots must lie on the open half-plane delimited by $\overleftrightarrow{p_{mid_1}p_{mid_2}}$ different from the one occupied by r. After its movement, r lies on $\overline{p_{mid_1}p_{mid_2}}$ whereas all the other robot are still on one of the open half-plane delimited by $\overleftrightarrow{p_{mid_1}p_{mid_2}}$. So, r remains corner after the movement. Now, if r_{Nbr_1} is stationary when r executes its movement, then r_{Nbr_1} must be boundary at that time and it becomes corner as r move away from the line $\overleftrightarrow{r r_{Nbr_1}}$. If r_{Nbr_1} and r move with sync, r_{Nbr_1} being a corner robot, remains corner after the movement. □

Lemma 6.4.13. *If the inner-most layer is a non-linear convex hull, then all the non-faulty robots eventually positioned themselves as the corner of the same convex hull without collision.*

Proof. Lemma 6.4.9 proves that the non-faulty robots eventually place themselves on the inner-most layer of the robots. After this, all the robots on the inner-most layer must be with the color INNER or BOUNDARY. When a robot r with color INNER or BOUNDARY does not find any robot with the color from the set {OUTER, CORNER, MOVE-O, MOVE-I}, it changes its color to CORNER1 (if it is a corner robot) or to BOUNDARY1 (if it is a boundary robot).

It is possible that there exists at least one robot r' (which has not yet changed its color to BOUNDARY or FAULT) with $r.color \in$ {OUTER, MOVE-O, MOVE-I}. Due to this, some of the robots on the inner-most layer have not changed their color to CORNER1 or BOUNDARY1. In this case, r must see a robot r'' on the inner-most layer with $r''.color$ = INNER or BOUNDARY, and r waits till all the robots in $\mathcal{CH}^*_r$ are with the color either CORNER1 or BOUNDARY1. Thus, all the robots on the inner-most layer eventually start the strategy MAKING_CORNERS_NONLINEARCASE together with all the corner robots colored CORNER1 and the boundary robot colored BOUNDARY1.

Now, we argue that the strategy MAKING_CORNERS_NONLINEARCASE successfully place all the non-faulty robots on the vertices of some convex hull from the inner-most layer. If r is a corner robot and both of its neighbours are also corner robots (colored with CORNER1), it changes its color to FINISH. If r is a corner having one of the neighbours r_{Nbr_1} with the color BOUNDARY1, it moves inside the inner most layer with color MOVE-C by following the strategy MOVING_INSIDE_CONVEXHULL($r, \mathcal{CH}^*_r$). By Lemma 6.4.12, both of r and r_{Nbr_1} become corner robots on $\mathcal{CH}^*_r$ without any collision if r is a non-faulty robot. In the same round, r_{Nbr_1} sets its color to MY-TURN. In the next round, r stays with the current color MOVE-C after seeing one of its neighbour with MY-TURN whereas r_{Nbr_1} changes its color either to CORNER1 (if it finds itself as the corner of $\mathcal{CH}^*_{r'}$) or to BOUNDARY1 (if it finds itself as the boundary of $\mathcal{CH}^*_{r'}$). In the next round, when r finds r_{Nbr_1} with the color BOUNDARY1 and $r.color$ = MOVE-C, it changes its color to FAULT, as it failed to execute the movement to make the robot r_{Nbr_1} a corner. Otherwise, when $r.color$ = MOVE-C and $r_{Nbr_1}.color$ = CORNER1, r changes its color to FINISH only if the other neighbour of r on $\mathcal{CH}^*_r$ is with color CORNER1 or FINISH. From next round onward, r_{Nbr_1} follows the strategy similar to r and eventually becomes a corner of some convex hull with the color FINISH. □

Lemma 6.4.14. *If all the robots are settled on the inner-most layer, each with the color either CORNER1 or BOUNDARY1, they need $O(N)$ rounds to terminate on the vertices of the same convex hull.*

Proof. If r is a CORNER1-colored robot on the inner-most layer, it needs 3 rounds to transitioned through the colors to reach the termination with the color FINISH or FAULT, as we

argue in Lemma 6.4.13. In the worst case, it is possible that the inner-most layer has exactly 3 vertices and one of the sides of it contains rest of the $N-3$ robots. The $\lceil\frac{N-1}{2}\rceil$-th robot on that particular side needs $O(N)$ many rounds to become a corner robot after all the other robot becoming corner robots. Hence, it needs $O(N)$ rounds to terminate on the vertices of the same convex hull. □

Lemma 6.4.15. *When the inner-most layer is a linear configuration, one of the following two holds.*
(i) If the LIC has even number of robots, there must be exactly two robots on the LIC which color themselves with `CENTER1` *and visible to each other.*
(ii) If the LIC has a odd number of robots, exactly one robot on LIC colors itself with `CENTER2`*.*

Proof. By Remark 6.4.3, all the robots on the inner-most layer color themselves `LINEAR`. For (i), if we assume that there are $2s$ robots on the LIC, the strategy FIND_CENTER finds two `CENTER1`-colored robots in $s+1$ rounds, as all the robots on LIC change their color to `INTERIM` in s rounds, two at a time in each round, and then in one round the s and $s+1$-th robots on LIC change their color to `CENTER1`. Moreover, there are no robots lying between the s and $s+1$-th robots on the LIC which implies that they are visible to each other. For (ii), if there are $2s+1$ robots on LIC, the $s+1$-th robot changes its color to `CENTER2` in $s+1$ rounds. □

Lemma 6.4.16. *All the non-faulty* `OUTER`*-colored robots lying on one of the open half planes delimited by the LIC, eventually get settled on a common ellipse without collision, when the inner-most layer is a linear configuration.*

Proof. By the similar argument presented in Lemma 6.4.9, we can prove that all the non-faulty `OUTER`-colored robots eventually place themselves on $\mathcal{CH}^{k-1}$ without any collision. After this, any robot r on $\mathcal{CH}^{k-1}$ satisfies one of the following two conditions. (1) $Next\_\mathcal{CH}^*_r$ is empty and one side of $\mathcal{CH}^*_r$ contains robots with color from the set {`LINEAR`, `INTERIM`, `NON-CENTER`, `CENTER1`, `CENTER2`} (2) $Next\_\mathcal{CH}^*_r$ is made of robots with colors from the set {`LINEAR`, `INTERIM`, `NON-CENTER`, `CENTER1`, `CENTER2`}. The robot r fol-

lows the strategy MAKING_CORNERS_LINEARCASE, which is similar to the strategy MAKING_CORNERS_NONLINEARCASE. The movement of the robot is similar in these two strategies. The main difference is that the robot r considers $\mathcal{CH}_r^{LIC}$ in the former case instead of $\mathcal{CH}_r^*$ in the later one. So, by Lemma 6.4.13, we can conclude that all the non-faulty OUTER-colored robots lying on one of the same half plane delimited by the $\overleftrightarrow{LIC}$ eventually become corners with the color CORNER2 of the same convex hull without any collision. This enables those robots to see each other along with the CENTER1 or CENTER2-colored robots. Such a robot r calculates the ellipse $\mathcal{E}(cen_r, a_r, b_r, \overleftrightarrow{LIC})$ and any other robot r', which are on the same half plane delimited by the $\overleftrightarrow{LIC}$ same as r, agrees on the same ellipse as they are visible to each other. r now considers the point of intersection $\overline{cen_r r}$ and the ellipse as its target point. Since, $cen_r = cen_{r'}$ and $r \neq r'$, $\overline{cen_r r}$ and $\overline{cen_{r'} r'}$ cannot meet at two points, which implies r and r' cannot collide during their movement towards their respective target points. □

Lemma 6.4.17. *All the non-faulty robot lying on the LIC eventually see all the non-faulty robots.*

Proof. On the LIC, there are four types of robots. First, a robot can be a NON-CENTER-colored robot which does not see the CENTER2-colored robot. When such a robot r becomes a terminal robot on the LIC, it moves to the visible area with the color INSPECT. It is guaranteed that all the FINISH-colored robots (all are stationary now) are visible to r after the movement, by Lemma 2.1 [54]. Let us now consider two robots r_i and r_j which lie on the half plane $H^1_{\overleftrightarrow{LIC}}$ same as r. If they both are non-faulty, by Lemma 6.4.16, they move to the common ellipse with $b_{r_i} = b_{r_j}$. We assume that one of them is faulty. Without loss of generality, r_j is the faulty robot among them before executing its movement. Before the movement of r_i and r_j, $b_{r_i} = b_{r_j} < \min\{d(cen_{r_i}, r_i), d(cen_{r_i}, r_j)\}$ holds. So they must be outside their common target ellipse before the movement. Since r_i has executed movement but r_j could not, r_i must be on the ellipse whereas r_j lies outside of it. It implies $b_{r_i} < b_{r_j}$ holds for the robot r. So, r accurately calculates the $2b_r$ which is also the length of the minor axis of the ellipse where all the non-faulty FINISH-colored robots of $H^1_{\overleftrightarrow{LIC}}$ reside. Since r can determine the ellipse, it can move to a point on the ellipse. So, all the

non-faulty robots on the LIC with the color NON-CENTER, which do not see any CENTER2-colored robot, eventually move to some points on a ellipse and color themselves FINISH. After this, r can see all the FINISH-colored non-faulty robots except the robot lying on the line $\overleftrightarrow{rr_c}$ with $r_c.color$ = CENTER2.

Second, a robot is with color NON-CENTER and sees a robot r_c with the color CENTER2. Such a robot r is a vertex of an ellipse where the non-faulty robots are situated. So, it can see all the non-faulty FINISH-colored robots except the robot lying on $\overleftrightarrow{rr_c}$. Third, a robot is with the color CENTER1. It is also lying on a vertex of an ellipse. So, they can see all the non-faulty robots.

Fourth, a robot is with the color CENTER2. The robot r finds its movable area by avoiding the line $\overleftrightarrow{r_i r_j}$ for all $r_i, r_j \in \mathcal{V}_r$ and moves to a point in it. Since all the FINISH-colored robots are stationary at this time, it can see all the non-faulty FINISH-colored robots. □

Theorem 6.4.18. *Algorithm* FAULT_MV$_{LC}$ *solves the problem in* $O(N^2)$ *rounds without any collision and requires* $O(1)$ *colors.*

Proof. Table 6.2 lists down all the colors used in the algorithm FAULT_MV$_{LC}$ which is 21 in total.

When the inner-most layer is a non-linear configuration, it needs $O(N^2)$ rounds by Lemma 6.4.4 Remark 6.4.11 and Lemma 6.4.14 to solve the problem. When the inner-most layer is a non-linear configuration, the algorithm is free from collision by Lemma 6.4.8, 6.4.9 and 6.4.13.

When the inner-most layer is a linear configuration, Lemma 6.4.16 and 6.4.17 proves the correctness of the algorithm with no collision between the robots. In this case, it needs $O(N^2)$ rounds to reach the layer $\mathcal{CH}^{k-1}$ by similarly argument presented in Lemma 6.4.10. It also needs $O(N)$ rounds to color all the robots not lying on the LIC with CORNER2, by the similar argument as in Lemma 6.4.14. Additionally, it requires only 1 round for all such robot to reach on a common ellipse. A NON-CENTER-colored robot lying on the LIC takes at most 2 rounds to reach on the ellipse after it becomes terminal on LIC and a CENTER1 or CENTER2-colored robot needs 1 round to terminate. Hence, the algorithm needs $O(N^2)$ rounds. □

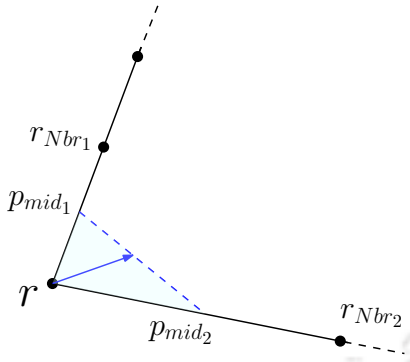


Figure 6.16: r moves to the midpoint of $\overline{p_{mid_1} p_{mid2}}$

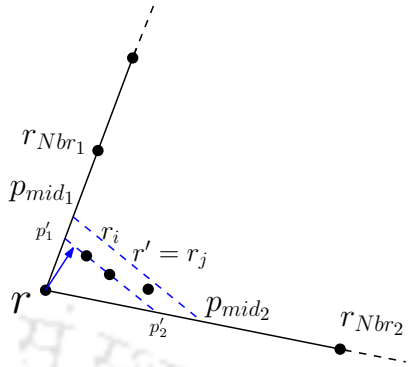


Figure 6.17: r moves to the midpoint of $\overline{p_1' r_i}$

6.5 Fault-tolerant Mutual Visibility in Presence of Single Fault

If we restrict the number of fault to 1, we can propose a simpler algorithm that requires much lesser number of colors. The algorithm is based on the contraction of the outer most convex hull of the robots. The goal is to achieve a configuration where all the robots are at the vertices of a convex hull. Every robot, when gets activated, checks if it is a corner, boundary or interior robot. A corner robot finds a target point inside the convex hull based on its neighbouring robot on the convex hull and moves to the point in the current LCM cycle. The detailed description of the algorithm FAULT_MV$^{f=1}_{LC}$ is as follows.

6.5.1 Description of the Algorithm

When the configuration is non-linear, r can determine whether it is a corner robot, boundary robot or an interior robot on the convex hull $\mathcal{CH}_r$. We identify two cases here.

Case 5.1 (r is a corner robot): Let r_{Nbr_1} and r_{Nbr_2} be the two neighboring robots of r on $\mathcal{CH}_r$. Also, r_{Nbr_1} and r_{Nbr_2} can be corner or boundary robots. Since both of r_{Nbr_1} and r_{Nbr_2} are visible to r, it computes two midpoints p_{mid_1} and p_{mid_2} of $\overline{r r_{Nbr_1}}$ and $\overline{r r_{Nbr_2}}$ respectively. If there is no robot inside the triangle $\Delta r\, p_{mid_1} p_{mid_2}$, r moves to the midpoint of the line segment $\overline{p_{mid_1} p_{mid_2}}$, as shown in Fig. 6.16. Otherwise, if there exists at least one robot inside the triangle $\Delta r\, p_{mid_1} p_{mid_2}$, let r' be the nearest

interior robot to r inside $\Delta r\, p_{mid_1} p_{mid_2}$. Let L' be the line passing through the robot r' and parallel to $\overline{p_{mid_1} p_{mid_2}}$. We further assume that the line L' intersects $\overline{r\, p_{mid_1}}$ and $\overline{r\, p_{mid_2}}$ at the points p'_1 and p'_2 respectively. Moreover, r_i and r_j are the two robots on $\overline{p'_1 p'_2}$ such that both $\overline{p'_1 r_i}$ and $\overline{p'_2 r_j}$ do not have any robots on them. For a single robot on the line segment $\overline{p'_1 p'_2}$, $r_i = r_j = r'$. The robot r moves either to the midpoint of $\overline{p'_1 r_i}$ or to the midpoint of $\overline{p'_2 r_j}$, whichever is closest to its current location, as shown in Fig 6.17.

Case 5.2 (r is an interior robot at the center of $\mathcal{CH}_r$ and $\mathcal{CH}_r$ is a regular polygon having no other interior robot): In this case, r selects the any of the sides of $\mathcal{CH}_r$ and calculates its midpoint p_{mid}. It finally moves to the midpoint of $\overline{r\, p_{mid}}$.

In all other cases, the interior and the boundary robots move until they become corners.

A simple execution of the above algorithm is illustrated in Fig 6.18, where we have taken 8 robots, out of which one is faulty.

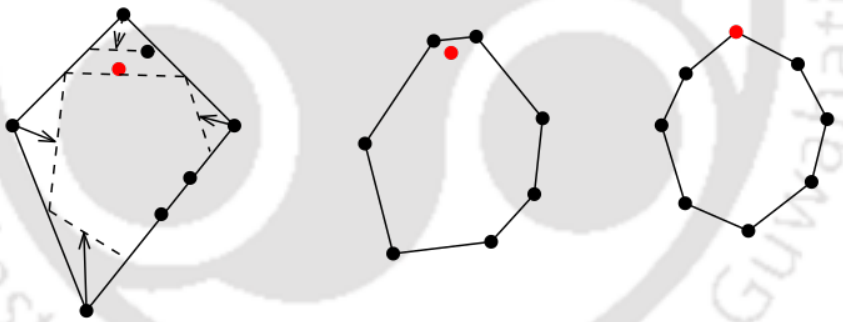


Figure 6.18: Illustrating execution of the algorithm: (left) the initial configuration with red colored robot being faulty; (middle) 3 new robots become corners after the movement of previous corners; (right) the faulty robot becomes a corner robot

This algorithm requires 2 colors in FSYNC and SSYNC setting, which is optimal with respect to the number of colors. All the robots start with the initial color `OFF`. A robot changes its color to `CORNER`, when it finds itself as a corner robot. The boundary robots and the interior robots maintains their color to `OFF` until they become corner robots. The algorithm finishes for a robot when it sees all other robots with the color `CORNER` along with itself.

The same movement strategy can also be followed in the ASYNC setting, in the presence of a single faulty robot, but with a little modification to ensure collision-free movement. In

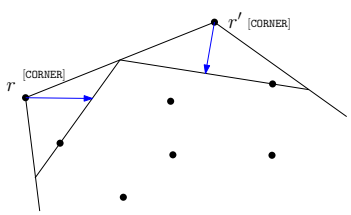


Figure 6.19: r and r' take the snapshots before any movement under asynchronous activation

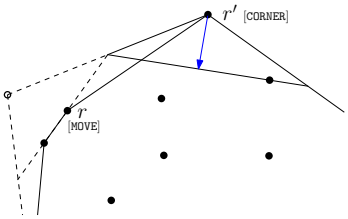


Figure 6.20: r moves before r' with the color MOVE, but sees its neighbour r' with color CORNER

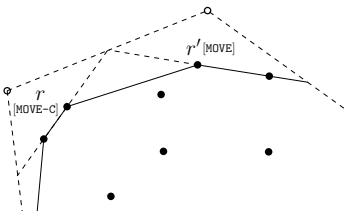


Figure 6.21: r waits until r' completes its movement and eventually changes its color to MOVE-C

the ASYNC setting, it is possible that a robot r takes a snapshot but is yet to move. Meanwhile, the adversary can create a situation where the other robots might execute multiple LCM cycles. This might compromise the collision-free movement of the robots. To tackle this issue, we can use a few extra colors (BOUNDARY, MOVE and MOVE-C) to prevent the multiple movements of one of the two neighbouring corner robots, whereas the other is yet to move with an outdated data. When a robot r gets activated and finds itself as a boundary robot on $\mathcal{CH}_r$, it changes its color to BOUNDARY, whereas the corner robots change their color to CORNER and the interior robots remains with the color OFF. If r gets activated with the color CORNER, it follows the above movement strategy (Case 5.1) to move to its target point with the color MOVE. The BOUNDARY and OFF-colored robots do not move until they become corner robots. In the next cycle, when r is activated with the color MOVE and finds at least one of its neighbouring robots r_{Nbr_1} and r_{Nbr_2} with the color CORNER, as shown in Fig. 6.20, it does not change its current color or move anywhere. Otherwise, it switches its color to MOVE-C without any movement. This is to let the neighbouring robots know that it has executed a movement. In case of r getting activated with color MOVE-C and seeing a neighbouring robot with MOVE, as depicted in Fig. 6.21, it does not change its current color, as the neighbouring robot might be in its move phase and r allows it to change its color to MOVE-C. When the neighbouring robots do not have the color MOVE, r changes its color to CORNER and follows the same strategy as discussed above. In total, this strategy requires 5 colors in the ASYNC setting.

6.5.2 Analysis of the Algorithm

First, we show that the movements of the robots described in the above algorithm, FAULT_MV$_{LC}^{f=1}$ are free from collision. We then show that the algorithm achieves the goal of making all the robots corner of a convex hull.

Lemma 6.5.1. *The movements of the robots are collision-free.*

Proof. Let us consider two robots, r and r'. As long as one of them is a corner and the other is a boundary or an interior robot, it is immediate that they cannot collide. Now, we assume that both are corner robots. If they are not neighbours of each other, at least two other robots, r'' and r''', exist on the boundary of the convex hull such that r and r' lie on the different half-planes delimited by the line passing through r'' and r'''. It is pretty immediate to see that they cannot collide as $\Delta r\, p_{mid_1} p_{mid_2}$ and $\Delta r'\, p'_{mid_1} p'_{mid_2}$ are disjoint triangles and are separated by the line passing through r'' and r''', even in case of asynchronous activation.

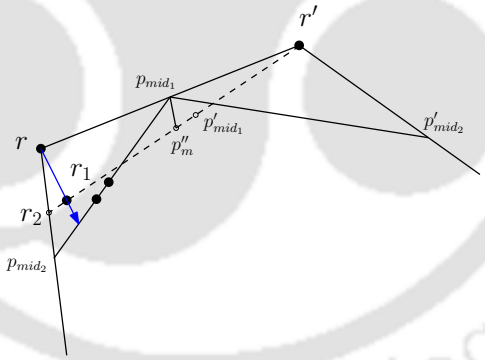


Figure 6.22: r' takes the snapshot when r is at r_1

Let us now discuss the case when r and r' are neighbours of each other. If both of them are with color CORNER and the look phase of r' has started before the move phase of r, the positions of r and r' in the snapshot of r match with that of r', as shown in Fig. 6.19. In this situation, the movement of r and r' are restricted in the triangles $\Delta r\, p_{mid_1} p_{mid_2}$ and $\Delta r'\, p'_{mid_1} p'_{mid_2}$ respectively. These two triangles are disjoint except at one vertex. r' might execute its LCM cycle multiple times due to asynchronous activation, but r has an outdated position of r' (because r did not start its move phase). In this situation, if r and

r' become non-neighbours after the movement of r', they do not meet a collision by the previous argument. Otherwise, r' does not change the current color MOVE until r starts its movement with the color MOVE. So, it is not possible for r' to execute multiple movements while r is still having the outdated position of r'. Let us now consider the scenario where r' executes its look phase while r is in motion. Let r' capture the snapshot when r is at r_1. As shown in Fig. 6.22, p'_{mid_1} and p'_{mid_2} are the midpoints of $\overline{r'r_1}$ and $\overline{r'r'_{Nbr_2}}$ respectively, where r'_{Nbr_2} is the neighbor of r' other that r. Let r_2 be the point of intersection of $\overline{rp_{mid_2}}$ and $\overleftrightarrow{r'r_1}$. Further, we denote the midpoint of $\overline{r'r_2}$ by p''_m. Geometrically, it implies that $\overline{p_{mid_1}p''_m} \parallel \overline{rr_2}$, i.e., $\overline{p_{mid_1}p''_m} \parallel \overline{rp_{mid_2}}$. If p''_m lies inside the triangle $\Delta r p_{mid_1} p_{mid_2}$, then $\overline{p_{mid_1}p''_m}$ cannot be parallel to $\overline{rr_2}$, contradicting the previous statement. Thus, p''_m must lie outside the triangle $\Delta r p_{mid_1} p_{mid_2}$. Since, p'_{mid_1} lies on the line segment $\overline{r'p''_m}$, it also lies outside the triangle $\Delta r p_{mid_1} p_{mid_2}$. So, the two triangles $\Delta r p_{mid_1} p_{mid_2}$ and $\Delta r' p'_{mid_1} p'_{mid_2}$ are disjoint and hence the two robots do not collide. In the case where r' being an interior robot, finds itself lying on the center of a regular polygon $\mathcal{CH}_{r'}$, r moves to the midpoint of line segment $\overline{p_{mid_1}p_{mid_2}}$ and r' to the midpoint of $\overline{r'p_{mid}}$. These two target points can not be the same, as two lines $\overleftrightarrow{p_{mid_1}p_{mid_2}}$ and $\overleftrightarrow{r'p_{mid}}$ can only intersect at p_{mid}. □

Lemma 6.5.2. *A corner robot r remains a corner robot after its movement, irrespective of its neighbours being in the look, compute or move phase under ASYNC activation.*

Proof. Let r_{Nbr_1} and r_{Nbr_2} be the two neighbours of the corner robot r on $\mathcal{CH}_r$ before the movement, as shown in Fig 6.23. Without loss of generality, let us consider that r_{Nbr_1} is in the move phase and r_{Nbr_2} is stationary when r gets activated. Let $t_{r_{Nbr_1}}$ be the target point of r_{Nbr_1}. We consider the cases when the robot r sees the neighbouring robot r_{Nbr_1} at r'_1 and r''_1. Let us also assume that r_k is an interior robot that lies between r and r''_1. If r sees the robot r_{Nbr_1} on r''_1 during its movement, r remains stationary as $r_{Nbr_1}.color$ = MOVE.

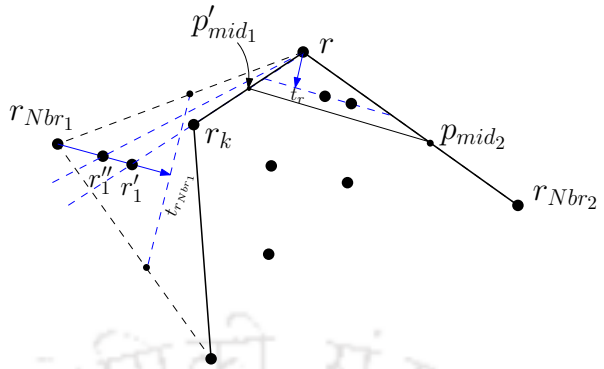


Figure 6.23: r cannot see r' at r'_2 as r_i obstructs the line of sight

If r takes the snapshot when r_{Nbr_1} is at r'_1, r cannot see r_{Nbr_1}, because of the obstruction by the robot r_k, as shown in Fig. 6.23. If r takes a snapshot at this time, it considers r_k as one of its neighbours in $\mathcal{CH}_r$. So, r considers the triangle $\Delta r\, p'_{mid_1} p_{mid_2}$ to find its destination, where p'_{mid_1} is the midpoint of $\overline{rr_k}$. If $\Delta r\, p'_{mid_1} p_{mid_2}$ does not contain any interior robot, r moves to the midpoint of $\overline{p'_{mid_1} p_{mid_2}}$. After the movement of r, it still remains a corner robot, as there are no other robots on the line segment $\overline{p'_{mid_1} p_{mid_2}}$ and all the interior robot maintain the status quo. If $\Delta r\, p'_{mid_1} p_{mid_2}$ contains at least one robot, r remains a corner robot of the outer most convex hull even after moving to t_r (the movement is indicated by the arrow from r to t_r). We have described the calculation of the destination point of r in Section 6.5.1. □

Lemma 6.5.3. *The total number of the corner and boundary robots is monotonic increasing.*

Proof. From Lemma 6.5.2, we can say that the total number of the corner and boundary robots can never decrease as a corner robot remains corner. Moreover, a boundary robot does not move until it becomes corner. The number increases when a corner robot r moves to its target after finding an interior robot r_i lying inside the triangle $\Delta r\, p_{mid_1} p_{mid_2}$. □

Theorem 6.5.4. *All the robots eventually becomes corner robots of a convex hull even if at most one robot becomes faulty.*

Proof. The fault may occur at any time. If the faulty robot r is a corner robot, it does not move from that time. The robots r_{Nbr_1} and r_{Nbr_2} eventually become corner robots because of the movements of the neighbours (other than the robot r) of r_{Nbr_1} and r_{Nbr_2}.

If r is a boundary robot, it becomes a corner due to the movement of its neighbouring robot. Finally if the faulty robot is an interior robot, it eventually becomes a corner robot from Lemma 6.5.3, as the total number of corner robots monotonically increases with time. Once a robot becomes corner, it remains as a corner robot from thereon, which leads us to our goal. □

6.6 Discussion

We have proved in Section 6.3 that if all the non-faulty robots are positioned on the vertices of a regular polygon and the faulty robot is placed at the center of the polygon, the problem becomes impossible to solve. If there is exactly one faulty robot and an adversary places the faulty robot very close to the center of the polygon, our propose algorithm FAULT_MV$_{LC}^{f=1}$ poorly performs in terms of running time, although the algorithm is simpler and uses only 2 colors in FSYNC setting. In that case, if the number of non-faulty robots lying on the boundary of the convex hull is very high, the contraction of the convex hull towards the faulty robot can be very slow, because of which the purpose of making all the robots a corner of some convex hull suffers. This algorithm works better if the interior robots are placed near the boundary of the convex hull $\mathcal{CH}$.

6.7 Conclusion

In this chapter, we studied the problem of mutual visibility of a swarm of autonomous mobile robots. We assume that the robots do not have any agreement on coordinate axes. Moreover, the robots are vulnerable to mobility faults. We devise a technique to achieve mutual visibility among the non-faulty robots and propose an algorithm that can tolerate any number of faulty robots under the FSYNC setting. So far the fault-tolerant mutual visibility is studied under the one-axis agreement. We approach the problem with robots having no consensus about the coordinate axes and no lights on them. Also, none of the robots know about N. Under this model, we prove that the problem is not solvable for any initial configuration. Our proposed algorithm needs $O(N^2)$ rounds in the FSYNC setting.

However, the algorithm fails to achieve mutual visibility when the innermost layer has a single faulty robot. As an immediate future work, we want to overcome this shortcoming. We propose another algorithm that tolerates a single faulty robot and requires 2 colors in the FSYNC setting and 5 colors in the ASYNC setting. This algorithm is far simpler than the former one and can be applied in the presence of a smaller number of robots.

Many future possibilities are still open. It is open to study the problem with robots that are susceptible to both mobility and light failure. One can further investigate the problem under the ASYNC setting where robots have no axes agreement. Along with that, the problem can be examined using a lesser number of colors. It will also be interesting to study fault-tolerant mutual visibility problems with fat robots.

Chapter 7

Uniform Partitioning of a Bounded Region

7.1 Introduction

The *Uniform Partitioning* of a bounded region is a problem that is inspired by our daily life. For example, if a number of people are asked to paint a wall, the natural strategy is to divide the region into equal parts and assign each person to a distinct part of the wall for painting. Keeping the same motivation in mind, autonomous mobile robots can be more useful in much more critical situations like cleaning spillage of liquid radioactive waste in a laboratory. In such hazardous situations, robots are the safest options for us. Uniform partitioning is equally applicable to another scenario where a city needs to be well-covered with networks. A group of autonomous drones, each of which is equipped with necessary instruments, can be deployed to serve the purpose, where drones should position themselves in such a way that each of them covers a part of the whole city of the same area as others.

In this chapter, we initiate the study of distributed uniform partitioning of a bounded region using opaque luminous mobile robots. The activation schedule of the robots is ASYNC. The primary motive is to use a swarm of mobile robots with assumptions as few as possible. The problem aims to arrange the robots in such a way that the region gets di-

vided into equal partitions, and each partition contains exactly one robot. We retain our focus on this objective only, whereas the problem could even be extended to a version where the robots have sufficient memory or some extra ability to store the coordinates of the partitions. The problem gets challenging due to the oblivious nature of the robots. The obstructed visibility of the robots adds to the challenge even more because it is not always possible to count the number of robots present in the region. We also assume that the robots do not have any knowledge about the total number of robots. Although mutual visibility algorithms can be used to make any three robots non-collinear and robots can count the total number of robots, the obliviousness of the robots disables them to keep the information beyond the current LCM cycle. From the application point of view, we assume that the robots can detect the boundary of the region. We propose algorithms for the robots that consider the region to be a standard geometric shape, such as a rectangle, square, or circle.

7.1.1 Contributions

Our contributions in this chapter are listed below.

- We propose Algorithm RECTANGLE_PARTITION when $\Re$ is a rectangular region. The algorithm requires 2 colors, which is optimal with respect to the number of colors. The algorithm runs in $O(N)$ epochs, where N is the total number of robots deployed in $\Re$.
- We propose Algorithm SQUARE_PARTITION when $\Re$ is a square region, which runs in $O(N)$ epochs and uses 5 colors.
- We propose Algorithm CIRCLE_PARTITION when $\Re$ is a circle, which runs in $O(N^2)$ epochs and uses 8 colors. All of our algorithms are collision-free.
- To the best of our knowledge, this is the first work towards the problem of distributed uniform partitioning of a bounded region $\Re$ using a swarm of N ASYNC oblivious opaque mobile robots.

7.2 Model and Preliminaries

Robots: A set of N disoriented and opaque luminous mobile robots $\{r_1, r_2, \cdots r_N\}$ is deployed at distinct points within a bounded region. This region can be thought of as a subset of the Euclidean plane. The robots can detect the boundary of the region. All the robots operate in LCM cycles and are activated under a fair ASYNC scheduler. A robot is visible to itself but might be unable to see all the robots inside the region due to obstructed visibility. Moreover, robots do not know N and have no agreement on their coordinate axes.

Region: We are given with a bounded region, denoted by $\Re$. The *interior* of the region $\Re$, denoted by $Int(\Re)$ is defined to be the part of $\Re$ without the boundary. In this chapter, we consider the region to be a standard geometric region such as a rectangle, square or circle. A robot lying on $Int(\Re)$ is called an *interior robot*. A robot is a *boundary robot* when it lies on the boundary of $\Re$ but not on the corners. When $\Re$ has corners, robots lying on them are called *corner robots*. The boundary of $\Re$ is identifiable by the robots from any point in the region, which enables them to identify whether the region is a rectangle, square or circle. For any two points A and B in the region, $\overline{AB}$ denotes the line segment joining A and B. $\overleftrightarrow{AB}$ is the line passing through the two points. The length of a line segment $\overline{AB}$ is represented by $len(\overline{AB})$. We denote the distance between two points A and B by $d(A,B)$. A similar notation $d(p,L)$ is used to represent the shortest distance between a point p and a line L.

Problem 5 (Distributed Uniform Partitioning): N oblivious, opaque, luminous point robots are deployed at arbitrary distinct points on a bounded region $\Re$. The robots neither have a global agreement on the coordinate axis nor knowledge of N. Each of them operates in *Look-Compute-Move* activation cycles and has a persistent light attached to it that can assume a color from a predefined color set. The objective is to divide the region $\Re$ into N uniform partitions using N mobile robots such that each partition contains exactly one robot.

Partition Types: Depending on the positions and distribution of the robots over the region $\Re$, robots decide the type of partitioning in a distributed manner. All the partitions should

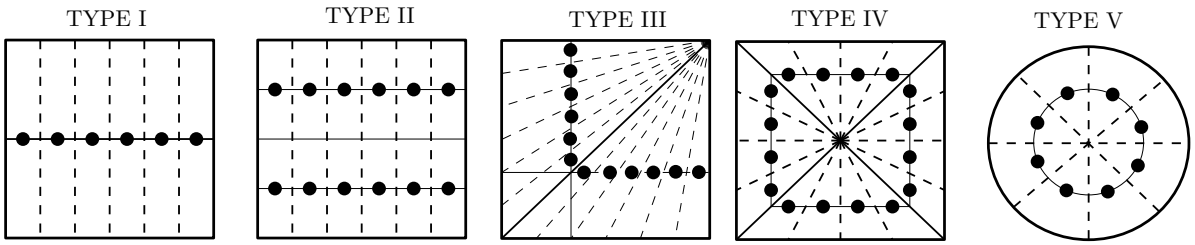


Figure 7.1: Illustrating different types of partitioning

have the same area. We identify four types of partitioning for the rectangular and square regions, as shown in Fig. 7.1. When the region $\Re$ is a rectangle, robots follow either Type I or Type II partitioning, whereas, in the case of a square region, robots terminate in one of the four types (Type I, II, III and IV). If $\mathscr{P}_1, \mathscr{P}_2, \cdots, \mathscr{P}_N$ are the sub-regions of equal area (i.e., partitions of $\Re$), then $\bigcap_{i=1}^{N} \mathscr{P}_i$ is a point and we call it a *vertex* of Type III and IV partitioning. In the case of a circular region, Type V partitioning is followed.

Region	**Color**	**Specification**
Rectangle	`OFF`	Initial color
	`FINISH`	Used by robots at termination for Type I and II partitions
Square	`OFF`	Initial color
	`MONITOR`	Used when the robot moves to the apex point to count the number of robots in case of robots are only on one or two opposite sides of $\Re$
	`FINISH`	Used by robots at termination for Type I and II partitions
	`FINISH1`	Used by robots at termination for Type III partition
	`FINISH2`	Used by robots at termination for Type IV partition
Circle	`OFF`	Initial color of the robots
	`HEAD`	Used for the head of an eligible cluster
	`TAIL`	Used for the tail of an eligible clusters
	`MID`	Used for the middle robots of an eligible cluster
	`MOVE-H`	Used for the movement of the head towards the other cluster when the head moves half of the excess distance
	`HALF`	Used after the color `MOVE-H`
	`FULL`	Used for the movement of the head towards the other cluster when the head moves the excess distance completely
	`FINISH`	Used by robots at termination

Table 7.1: The list of colors for different regions with their specification

7.3 Algorithm for a Rectangular Region

This section proposes the algorithm RECTANGLE_PARTITION when the region $\Re$ is a rectangle. Initially, robots are deployed over distinct points on the rectangle $\Re$ with color OFF. Robots will follow either Type I or Type II partitioning, which our algorithm ensures. The algorithm, in this case, uses 2 colors which are described in the Table 7.1. Our strategy is to bring the robots to the rectangle's boundary first. Then, each robot will occupy a point of a partition of the rectangle. We define some notations that will be used to explain the algorithm. We must note that the variables are defined with respect to the local coordinate system of a robot r. We mentions the notations and conventions used throughout this chapter in Table 7.2. A point visible to two different robots can be perceived differently, depending on their coordinate system.

Notations: For any robot r, we define the following notations.

- S_r is the nearest longest side of the rectangle $\Re$ to r or it denotes the specific longest side if r lies along one of them.
- S_r^{opp} denotes the side opposite to S_r in the rectangle.
- L_r is the line perpendicular to S_r that passes through r.
- p_r is the point of intersection of the two lines L_r and S_r.
- The two endpoints of a side S of $\Re$ is denoted by e_S^1 and e_S^2.
- For two opposite sides S and S' of $\Re$ and a constant $0 < k < 1$, kS is the line segment of length $len(S)$ parallel to the side S that satisfies $d(kS, S) = kd(S, S')$ and intersects $\Re$ at two points.
- For any two constants k and k' with $0 < k \neq k' < 1$ and for a side S of $\Re$, $[kS, k'S]$ is the rectangular region bounded by kS, $k'S$ and two adjacent sides of S. We denote the interior of the region $[kS, k'S]$ by $(kS, k'S)$. We also denote $(kS, k'S) \cup k'S$ by $(kS, k'S]$.
- $\mathcal{CH}_r$ is the local convex hull of all the visible robots of r.

Variables	Specifications
$\Re$	The given region
$Int(\Re)$	The interior of the region $\Re$
S_r	Nearest longest side to r when $\Re$ is a rectangle or the nearest side to r when $\Re$ is a square
SS_r	One of the shortest side of the rectangular region $\Re$
S^{opp}	The opposite side of $\Re$ of the side S
$len(S)$	The length of the side S
$\mathcal{V}_r$	The set of all visible robot to r
L_r	The line perpendicular to S_r and passes through r
p_r	The point of intersection of L_r and S_r
t_r	Target point of r
e_S^1, e_S^2	The two endpoints of the side S
$d(a,b)$	The Euclidean distance between the points a and b
$d(a,L)$	The Euclidean distance between a and the line segment L
kS	The line segment parallel to S, insects $\Re$ at exactly two points and satisfies $len(kS) = len(S)$ and $d(kS,S) = kd(S,S^{opp})$ for $0 < k < 1$
$[kS, k'S]$	The sub-region of $\Re$ bounded by kS, $k'S$ and two adjacent sides of S for $0 < k \neq k' < 1$
$(kS, k'S)$	The interior of the region $[kS, k'S]$
$(kS, k'S]$	The union of the region $(kS, k'S)$ and the line segment $k'S$
$\mathcal{CH}_r$	The convex hull of the visible robot to r
r_{Nbr}	The neighbour of r on $\mathcal{CH}_r$
c_r^1	Total number of robots on $[S_r, \frac{1}{8}S_r]$ including r
c_r^2	Total number of robots on $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$
C	The center of the square region $\Re$
S_r^L, S_r^R	Two sides of $\Re$ other than S_r and S_r^{opp} for the robot r
ΔS	The side triangle $\Delta C e_S^1 e_S^1$
D_r, D_r^{opp}	The two diagonals of $\Re$ for the robot r
$\|S\|$	The number of robot on the side triangle ΔS
Max_r	Set of all sides with the maximum number of robots lying on their respective side triangles
e_P	The common corner of S_r and S_r^L
e_Q	The common corner of S_r and S_r^R
e_R	The common corner of S_r^{opp} and S_r^R
e_X	The common corner of S_r^{opp} and S_r^L
O	The center of the circular region $\Re$
$\overset{\frown}{AB}$	The segment of the circumference starting from the point A to B
$alen(AB)$	The arc length of $\overset{\frown}{AB}$
rad	The radius of the circular region $\Re$
s	One of the arc length of $\Re$ after dividing the region into N equal parts
Cl	The cluster of r

Table 7.2: The list of notations and conventions used throughout this chapter

7.3.1 Description of the Algorithm

After activation with $r.color$ = OFF, the target of r is to reach one of the two longest sides of $\Re$ and waits till all other interior robots, all the visible corner robots, and the robots on the two shortest sides with color OFF reach on one of the two longest sides of $\Re$. Before moving to one of the longest sides of $\Re$, r needs to choose a side of $\Re$ as its target. Let us denote this target side by S_r. If r is already situated on one of the two longest sides of $\Re$, then it selects that side as S_r. If r is a corner robot, r chooses the longest side incident with it as S_r. If r is an interior robot or a boundary robot on one of the two shortest sides, it chooses the nearest longest side as S_r. In case of r being equidistant from both the longest sides, r chooses any one of them as S_r.

Strategy MOVE_TO_LONGESTSIDES: When r gets activated with $r.color$ = OFF, r finds tthehe target side S_r in the current LCM cycle and calculates the point of intersection p_r of S_r and L_r. Now, r needs to consider its current position in the region provided the point p_r is visible to r. It maintains the status quo if the point p_r is not visible to r. When p_r is visible, we can differentiate the following four cases based on the different positions of r and propose different strategies for r. (i) If r is an interior robot with p_r empty, r simply moves to p_r with its current color OFF. (ii) If r is a boundary robot on one of the two shortest sides of $\Re$ and p_r (which, in this case, is one of the corners of $\Re$) has a robot on it, r waits with no change in its current color till p_r gets empty. (iii) r is a boundary robot on one of the two shortest sides of $\Re$, and p_r is empty. (iv) r is either an interior robot with another robot on p_r or a corner robot. In the last two cases, (iii) and (iv), r needs to find a target point t_r on S_r in such a way that it does not encounter any collision. In this process, r finds a set $\mathcal{V}_r$ that consists of all visible robots not lying on L_r. There can be two sub-cases.

- $\mathcal{V}_r$ **is empty:** In this case, r selects a target point t_r on the side S_r as shown in Fig. 7.2, such that $d(p_r, t_r) = \frac{1}{2}\max\{d(e^1_{S_r}, p_r), d(e^2_{S_r}, p_r)\}$.

- $\mathcal{V}_r$ **is non-empty:** As shown in Fig. 7.3, r in this case, chooses the target point t_r that satisfies $d(p_r, t_r) = \frac{1}{4}\min_{r' \in \mathcal{V}_r}\{d(r', L_r)\}$.

Finally, r moves to the point t_r without changing the current color OFF. If r is already posi-

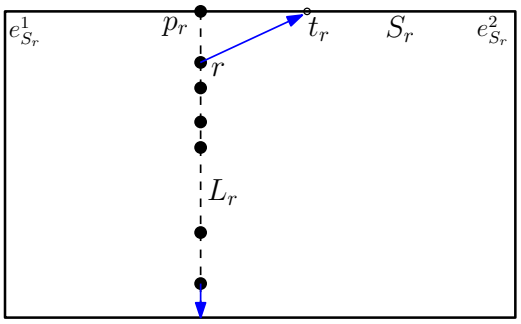


Figure 7.2: The movement of r to t_r when all robots are on a line

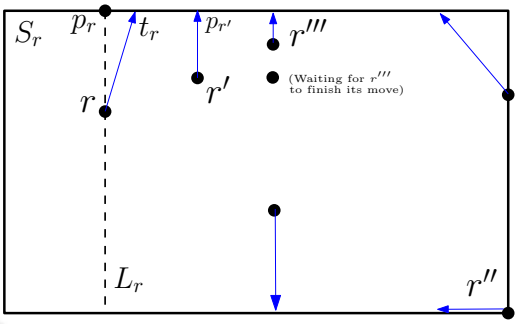


Figure 7.3: r moves to t_r that lies on the nearest longest side S_r

tioned on one of the longest sides of $\Re$, it does not change its current position or color until all the OFF-colored robots not lying on the longest sides reach on their nearest longest side.

Now, we need the following definitions to categorize the robots.

Definition 7.3.1. (Terminal Robot): Let L be a line segment with two endpoints e^1_L and e^2_L. A non-corner robot r lying on L is called a terminal robot on L if at least one of line segments $\overline{re^1_L}$ and $\overline{re^2_L}$ is not occupied by any other robot.

Definition 7.3.2. (Monitor Robot): Let S_r be the longest side of $\Re$ where r is situated in the current LCM cycle. r is called a monitor robot if it satisfies all the following conditions. (i) r is a terminal robot on S_r. (ii) $(S_r, \frac{7}{8}S_r)$ has no robot and all the robots on $(S_r^{opp}, \frac{1}{8}S_r^{opp}]$ are with color FINISH. (iii) There is no corner robot visible to r. (iv) There is no robot on both the shortest sides of $\Re$.

After all the visible robots lying either on S_r or on S_r^{opp} (where S_r is one of the longest sides of $\Re$), the robot r finds whether it is a monitor robot or not. The strategy is to move the monitor robots at a particular distance (inside the region) from the longest sides of $\Re$ so that they can count the number of total robots present in the region and decide the partition type of $\Re$. This is important because neither the robots possess any memory to store the value of N nor are they transparent to see each other. It does nothing if r does not qualify as a monitor robot. When r is a monitor robot on S_r, it needs to move to a point a_r on $\frac{1}{8}S_r$, referred to as *apex point*, from where it can count the number of robots which is necessary to decide the next action of r. The robot r moves to its apex point with color FINISH. At this stage, our target is to gather all the robots on one of the longest sides of $\Re$ if

the number of robots lying on the two longest sides differs. In this case, our objective is to relocate all the robots from the side with a smaller number to the side with a larger number of robots. After all the robots have positioned themselves on one of the longer sides of $\Re$, monitoring robots place themselves again at apex points with the color FINISH to count N and then move to their final positions of the partitioning on $\frac{1}{2}S_r$ without changing their color. Other robots decide their final positions based on these FINISH-colored robots on $\frac{1}{2}S_r$. We refer to this kind of partitioning as Type I partitioning.

On the other hand, if the number of robots on both of the longest sides is the same, we adopt a different strategy to partition the region. Here, the robots on the apex points first place themselves in their respective final positions on $\frac{1}{4}S_r$ without changing their color. Other robots decide their final positions based on the FINISH-colored robots. We refer to this as Type II partitioning.

In the following, we describe in detail the action of a monitor robot r lying on S_r.

Strategy MONITORROBOT_MOVEMENT_RECTANGLE: If each of S_r and S_r^{opp} has exactly one robot on it, r simply moves to the midpoint of the line segment $\frac{1}{4}S_r$ from S_r, after changing its color to FINISH. If S_r^{opp} has exactly one robot, but S_r has multiple robots on it, r can understand this scenario by confirming the existence of its neighbouring robot on S_r. In this case, r waits until the robot on S_r^{opp} moves to S_r. If r is the only robot on S_r, but S_r^{opp} has multiple robots, r decides its target side as S_r^{opp} and moves to a point on it with the color OFF, following a strategy similar to the strategy MOVE_TO_LONGESTSIDES. Observe that in all of the above scenarios, r does not need to compute and move to its apex point on $\frac{1}{8}S_r$. We now focus on the case where both S_r and S_r^{opp} have multiple robots, and the monitor robot r moves to the apex point a_r.

Movement to the Apex Point a_r: When r finds no robot on S_r^{opp}, it selects the point of intersection of L_r and $\frac{1}{8}S_r$ as its apex point a_r. as shown in Fig 7.4. When there is at least one robot on S_r^{opp}, r first selects the line $L = \overleftrightarrow{r r_{Nbr}}$, where r_{Nbr} is the neighbour of r on the convex hull $\mathcal{CH}_r$ lying on the side $(S_r^{opp}, \frac{1}{8}S_r^{opp}]$. Then, r chooses the intersection point of L and $\frac{1}{8}S_r$ as the apex point a_r (refer to Fig 7.5). The calculation of the apex point depends on the positions of the robots on S_r^{opp} to confirm that r can see all other robots in $\Re$ and does not become an obstruction between the other two robots after its movement, as depicted

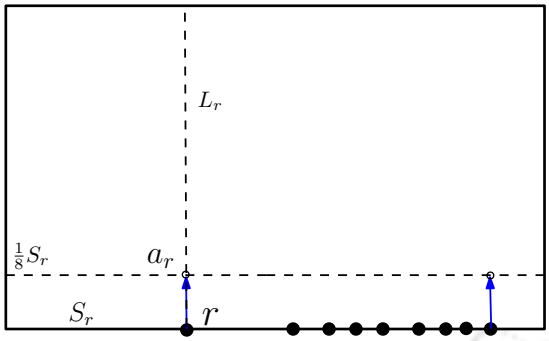


Figure 7.4: If S_r^{opp} has no robot, a_r is the intersection point of L_r and $\frac{1}{8}S_r$

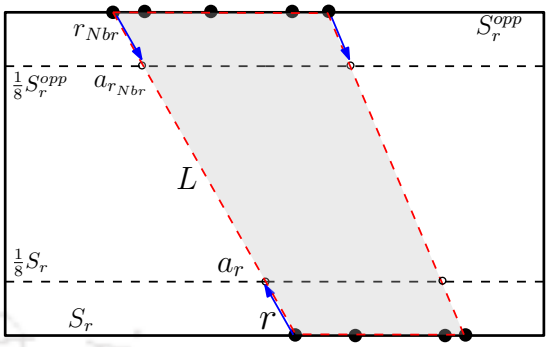


Figure 7.5: r moves to a_r considering r_{Nbr} on $\mathcal{CH}_r$, which is not on S_r

in Fig 7.6. Finally, r moves to a_r after changing its color from OFF to FINISH.

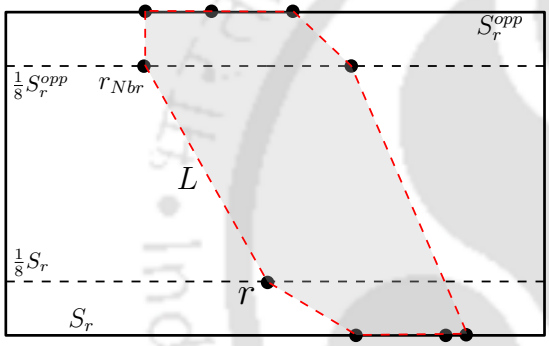


Figure 7.6: r remains a vertex of $\mathcal{CH}_r$ after the movement to apex point

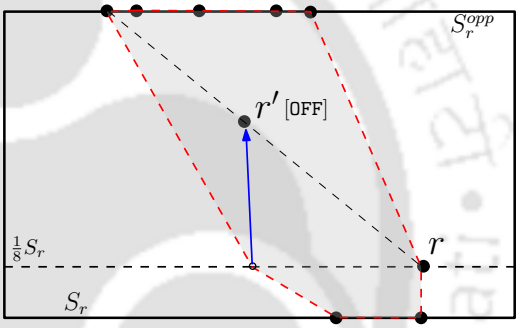


Figure 7.7: Movement of the OFF-colored robot r' creates obstruction for r

We now identify two cases based on the position of r.

Case 1 (r lies on $\frac{1}{8}S_r$): In this case, if r gets activated and finds itself lying on $\frac{1}{8}S_r$, then $r.color$ must be FINISH, as r must have moved to its apex point from the side S_r in its previous LCM cycle. Now, r identifies the nearest longest side of $\Re$ as S_r. It is possible for r to encounter an OFF-colored robot r' on $Int(\Re)$, which might occur if r' is in its move phase to allocate itself on one of the longest sides of $\Re$ (either from $\frac{1}{8}S_r$ or from $\frac{1}{8}S_r^{opp}$). Fig. 7.7 depicts one such situation. This might occur due to the asynchronous activation of the robots. Here r does not change its position and color. Otherwise (when no OFF-colored robot is visible in $Int(\Re)$), r can identify the other FINISH-colored robot r_i within $[S_r, \frac{1}{8}S_r]$, if exists (which might have been coming from S_r to its apex point). Moreover, it can also identify the robot with color FINISH on $[\frac{1}{8}S_r, \frac{1}{2}S_r]$ (this is possible when r_i is moving to its

final position either on $\frac{1}{4}S_r$ for Type II partitioning or on $\frac{1}{4}S_r$ for Type I partitioning and it is observed by r). Similarly, r can easily identify the robots with color FINISH which are nearest to S_r^{opp} by observing the region $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$. If r finds all the FINISH-colored robots on $\frac{1}{8}S_r$ and $\frac{1}{8}S_r^{opp}$, it calculates c_r^1 and c_r^2 where c_r^1 is the total number of the robots on the band $[S_r, \frac{1}{8}S_r]$ including itself. c_r^2 is the total number of robots on the band $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$. We identify the following four sub-cases for the robot r lying on $\frac{1}{8}S_r$ based on c_r^1 and c_r^2.

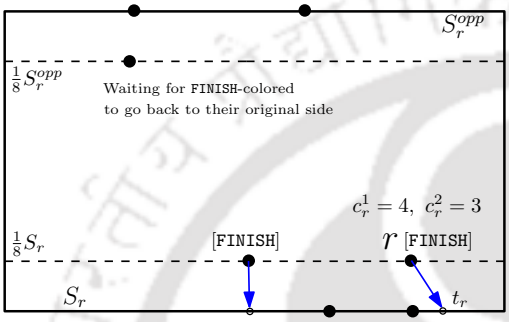


Figure 7.8: The FINISH-colored robot r moves back to the side S_r from its apex point on $\frac{1}{8}S_r$, when $c_r^1 > c_r^2 > 0$

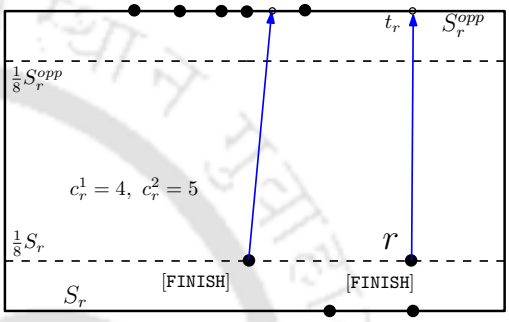


Figure 7.9: r moves to t_r on S_r^{opp} when $0 < c_r^1 < c_r^2$ with no FINISH-colored robot on $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$

- **Case 1.1** ($0 < c_r^2 < c_r^1$)**:** It means that there is a lesser number of robots near S_r^{opp} than that of S_r, as shown in Fig. 7.8. In this case, our aim is to relocate all the robots lying on $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$ to S_r. r also moves back to S_r in this case. If the point of intersection p_r of L_r and S_r is visible to r and contains no robot, r moves to p_r with $r.color =$ OFF. Otherwise, it chooses a target point t_r on S_r such that $d(p_r, t_r) = \frac{1}{4}\min_{r' \in \mathcal{V}_r} d(r', L_r)$, where $\mathcal{V}_r$ is the set of all visible robots not lying on the line L_r. Then, r moves to t_r with current color FINISH.

- **Case 1.2** ($0 < c_r^1 < c_r^2$)**:** It means that there are fewer robots near S_r than that of S_r^{opp} (Fig 7.9). In this case, r needs to move to S_r^{opp}. Before the movement to S_r^{opp}, it first checks whether there is any FINISH-colored robot in $(S_r^{opp}, \frac{1}{8}S_r^{opp}]$. If such a robot r_1 exists, then in ASYNC settings, the movement of r_1 might lead to a miscount of the number of the robots for r, as shown in Fig 7.10. In this situation, r waits until these robots allocate themselves on S_r^{opp}. Whenever r does not find any FINISH-colored robots in $(S_r^{opp}, \frac{1}{4}S_r^{opp}]$, it decides to move on S_r^{opp} by finding the target point t_r on

S_r^{opp}. It needs to consider p_r^{opp}, which is the point of intersection of L_r and S_r^{opp}. Using the same strategy described in Case 1.1 (instead of S_r, r considers S_r^{opp}), r finally moves on t_r after changing its current color to OFF.

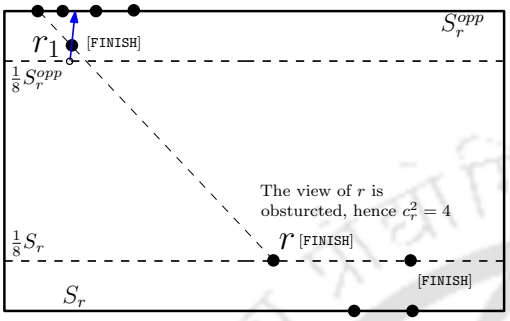


Figure 7.10: r miscalculated c_r^2 if a robot lies on $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$ moving to its target point on S_r^{opp}

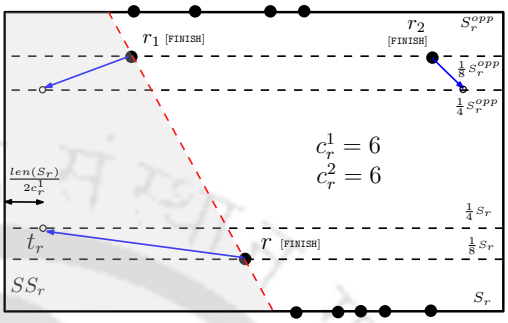


Figure 7.11: r moves to t_r on $\frac{1}{4}S_r$ because of same number of robot around S_r and S_r^{opp}

- **Case 1.3** ($0 < c_r^1 = c_r^2$)**:** In this case, we aim to make the robots follow the Type II partitioning. r first identifies the shortest side of $\Re$, denoted as SS_r, which either lies on or intersects the half plane delimited by the line $\overleftrightarrow{rr_1}$ where no other robots of $\Re$ are present. Here r_1 is the neighbour of r on $\mathcal{CH}_r$ whose nearest longest side is S_r^{opp}. Now, it computes a target point t_r on $\frac{1}{4}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$, as shown in Fig. 7.11. Finally, r moves to t_r from its current position with the current color FINISH.

- **Case 1.4** ($0 = c_r^2 < c_r^1$)**:** This case arises when there is no robot lying on S_r^{opp} and the robot r has moved from S_r to a point on $\frac{1}{8}S_r$, as shown in Fig 7.12. In this case, we want the robots to follow the Type I partitioning. So, r first identifies SS_r, which lies on the half plane delimited by L_r where no other robot in $\Re$ resides. It then calculates a target point t_r on $\frac{1}{2}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$. It then moves to t_r with the current color FINISH.

If r is activated on $\frac{1}{8}S_r$ with its color FINISH and sees one FINISH-colored robot r' on $(\frac{1}{8}S_r, \frac{7}{8}S_r)$, it moves back to S_r with color FINISH by following similar movement strategy as described in Case 1.1. This is because the position of r' (which is in its move phase

towards the final point) might mislead to the wrong calculation of c_r^1 and c_r^2. If r sees a FINISH-colored robot on $[S_r, \frac{1}{8}S_r) \cup [S_r^{opp}, \frac{1}{8}S_r^{opp})$, it does nothing.

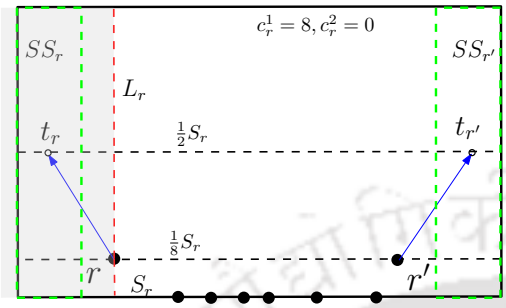


Figure 7.12: r moves to $\frac{1}{2}S_r$ from its apex point on $\frac{1}{8}S_r$ by following the Type I partitioning. The green box shows a partition of the region

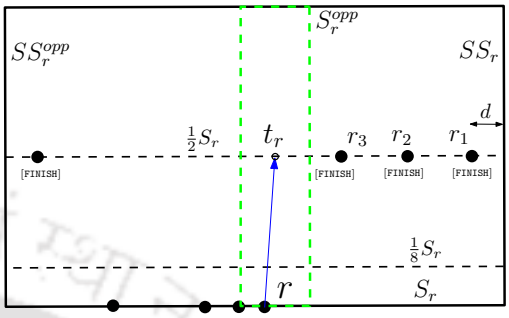


Figure 7.13: r is on S_r with color OFF and utilizes the position of the FINISH colored robots on $\frac{1}{2}S_r$ to reach its final position in a partition

Case 2 (r lies on S_r): If r gets activated with color FINISH on S_r, it understands that it has moved back to S_r from its apex point in its previous LCM cycle. So, it changes its color to OFF. The positions of the FINISH-colored robots in $Int(\Re)$ help the OFF-colored robots on S_r to decide their final positions inside $\Re$. If r is a non-terminal robot on S_r, r waits till it becomes terminal on S_r. If r becomes a terminal robot with color OFF and all the robots on $Int(\Re)$ are with color FINISH, we list down the following three sub-cases based on the different positions of FINISH-colored robots, where r decides to move to a target point t_r.

- Case 2.1: r finds a FINISH-colored robot on $\frac{1}{4}S_r$ and some robots on $[S_r^{opp}, \frac{1}{4}S_r^{opp}])$
- Case 2.2: r finds a FINISH-colored robot on $\frac{1}{4}S_r^{opp}$, but not on $[S_r, \frac{1}{4}S_r]$
- Case 2.3: r finds a FINISH-colored robot on $\frac{1}{2}S_r$

In the above cases, r first identifies the sides SS_r and SS_r^{opp}. SS_r is the shortest side of $\Re$ for which the intersection point of S_r and SS_r is visible to r. SS_r^{opp} is the opposite side of SS_r. Afterwards, r sets $L = \frac{1}{2}S_r$ when Case 2.3 occurs, as depicted in Fig 7.13. Otherwise (for Case 2.1 and Case 2.2), it sets $L = \frac{1}{4}S_r$, as shown in Fig. 7.14. r now considers a set $\mathcal{F}_r$, that consists of all the FINISH-colored robots lying on $\frac{1}{4}S_r$, $\frac{1}{4}S_r^{opp}$, and $\frac{1}{2}S_r$. It now calculates $d = \min_{r' \in \mathcal{F}_r}\{\min\{d(r', SS_r), d(r', SS_r^{opp})\}\}$. Let $r_1, r_2, ...r_k$ be a sequence of robots

of maximum length on L starting from the robot r_1 with $d(r_1, SS_r) = d$ till the robot r_k such that two consecutive robots in the sequence are exactly $2d$ distance apart from each other. When no robot is on L, we consider $k = 0$. Finally, r moves to t_r on L such that $d(t_r, SS_r) = (2k+1)d$ after changing its color to FINISH from OFF. In all other situations, it is possible that a FINISH-colored robot is in its move phase towards its target point or it is on its apex point. In those cases, r maintains the status quo.

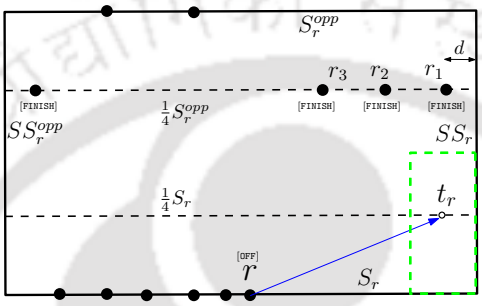


Figure 7.14: r moves to $\frac{1}{4}S_r$ from S_r by taking help of the FINISH-colored robots on $\frac{1}{4}S_r^{opp}$

When r gets activated with the color FINISH and finds itself on either $\frac{1}{4}S_r$ or $\frac{1}{2}S_r$, it terminates.

7.3.2 Analysis of the Algorithm

Here, we analyse the algorithm RECTANGLE_PARTITION. We prove the correctness and the time complexity of this algorithm. We also show that the movement of the robots is free from collision. For our convenience, we refer to the rectangular region $\Re$ as $ABCD$ with two longest sides $\overline{AB}$ and $\overline{CD}$ throughout the analysis of RECTANGLE_PARTITION algorithm.

Lemma 7.3.3. *All the robots not lying on one of the two longest sides of $\Re$, move to one of the longest sides of $\Re$ without any collision.*

Proof. Let $\overline{EF}$ divide the rectangle into two halves, as shown in Fig. 7.15. The robots lying inside the rectangle $ABFE$ choose the side $\overline{AB}$ as their target and move to some points on it. Similarly, the robots inside the rectangle $CDEF$ move to some point of the side $\overline{CD}$. The robots on $\overline{EF}$ choose any of the two sides $\overline{AB}$ and $\overline{CD}$ as the target side and move to it. Let us consider a robot r on $\overline{EF}$ that chooses the side $\overline{AB}$ as the target side, but there are robots

on the line segment $\overline{rp_r}$ which are nearer to $\overline{AB}$. In this case, r waits until all these robots reach some point on $\overline{AB}$. So, eventually, r gets the chance to move to a point on $\overline{AB}$.

Now we choose another robot r' lying inside $ABFE$, but not on L_r. We show that r and r' do not collide in their way to AB. For both r and r', if the two points p_r and $p_{r'}$ remain empty, they follow the path $\overline{rp_r}$ and $\overline{r'p_{r'}}$ to move to $\overline{AB}$. Since, $\overline{rp_r}||\overline{r'p_{r'}}$, r and r' cannot collide in this movement. Let us assume that p_r and $p_{r'}$ are non-empty and r' is the nearest robot in $\mathcal{V}_r$. So, r calculates a point t_r on $\overline{AB}$ such that $d(p_r,t_r) \leq \frac{1}{4}d(r',L_r) < \frac{1}{2}d(r',L_r)$.

Algorithm 17: RECTANGLE_PARTITION

```
1  if r.color = OFF then
2      Follow RECTANGLE_TARGET_SIDE() to calculate S_r
3      if r is not a boundary robot on S_r then
4          Follow TARGETPOINT_ON_SIDES() to find a target point
5          Move to the target point with the current color OFF
6      else // r lies on one of the longest sides of ℜ
7          if r is a monitor robot on S_r then
8              if Each of S_r and S_r^opp contains only one robot then
9                  Change r.color to FINISH
10                 Move to the midpoint of (1/4)S_r
11             else if S_r^opp has exactly one robot, but S_r has more than one robot then
12                 No change in color and position
13             else if S_r has exactly one robot, but S_r^opp has more than one robot then
14                 Set S_r^opp as the target side and follow TARGETPOINT_ON_SIDES() to
                    find the target point
15                 Move to the target point with the current color
16             else
17                 Follow MOVE_TO_APEXPOINT() and calculate the apex point a_r
18                 Change r.color to FINISH from OFF
19                 Move to the point a_r
20         else if r is a terminal robot on S_r with a FINISH-colored robot on (1/2)S_r or (1/4)S_r
            or (1/4)S_r^opp then
21             Follow MOVE_TO_FINALPOINT() to compute the target point
22             Change r.color to FINISH
23             Move to the target point
24         else
25             No change in color and position
```

else if $r.color = FINISH$ *and* r *lies on* $\frac{1}{8}S_r$ **then**
  if *there is an* OFF*-colored robot visible in* $Int(\Re)$ **then**
    r does not move and change its current color
  else
    if r *finds all the* FINISH*-colored robot on* $\frac{1}{8}S_r$ *and* $\frac{1}{8}S_r^{opp}$ **then**
      $c_r^1 \longleftarrow$ Number of robots on $[S_r, \frac{1}{8}S_r]$ including itself
      $c_r^2 \longleftarrow$ Number of robots on $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$
      if $0 < c_r^2 < c_r^1$ **then**
        S_r is the target side and follow TARGETPOINT_ON_SIDES() to find a target point
        Move to the target point with the color FINISH
      else if $0 < c_r^1 < c_r^2$ **then**
        if *there is a* FINISH*-colored robot on* $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$ **then**
          r does not change its color or position
        else
          S_r^{opp} is target side and follow TARGETPOINT_ON_SIDES() to find the target point
          Change its color to OFF and move to the target point
      else // $0 < c_r^1 = c_r^2$ or $0 = c_r^2 < c_r^1$
        Follow MOVE_TO_FINALPOINT() to find a target point
        Move to the target point with color FINISH
    else if *a* FINISH*-colored robot on* $(\frac{1}{8}S_r, \frac{7}{8}S_r)$ **then**
      S_r is the target side and follow TARGETPOINT_ON_SIDES() to find a target point
      Move to the target point with the color FINISH
    else
      No change in color and position
else if $r.color = FINISH$ *and* r *lies on* S_r **then**
  Change $r.color$ to OFF without any movement
else // $r.color =$ FINSH and r lies on $\frac{1}{2}S_r$ or $\frac{1}{4}S_r$
  r terminates.

Even if r' calculates its target point $t_{r'}$ on the line segment $\overline{AB}$, we have $d(p_{r'}, t_{r'}) \leq \frac{1}{4}d(r', L_r)$. Thus, $d(t_r, t_{r'}) \geq d(p_r, p_{r'}) - d(p_r, t_r) - d(p_{r'}, t_{r'}) \geq d(r', L_r) - \frac{1}{4}d(r', L_r) - \frac{1}{4}d(r', L_r) > 0$. Moreover, the line L passing through the midpoint of $\overline{t_r t_{r'}}$ separates the two paths $\overline{r\,t_r}$ and $\overline{r'\,t_{r'}}$, as depicted in Fig 7.15. So, the two robots r and r' cannot collide. In

the case of r and r' lying on the same line, they move sequentially to the side $\overline{AB}$, because of which they cannot meet a collision. □

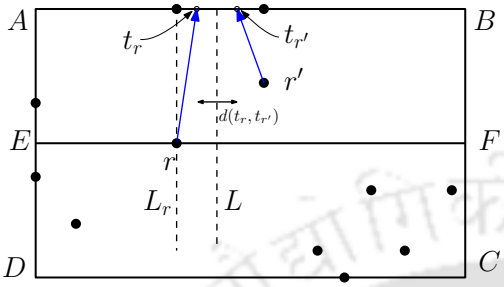


Figure 7.15: Collision free movement of the robots r and r', even in simultaneous execution

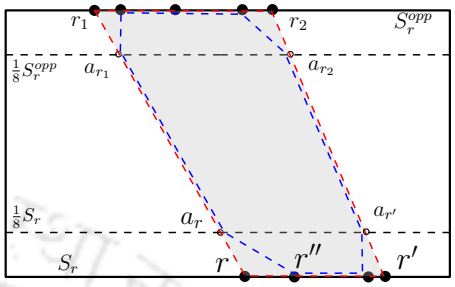


Figure 7.16: Montior robots move to their respective apex points to see all the robots present in the region

Algorithm 18: RECTANGLE_TARGET_SIDE()

```
if r is an interior robot then
    Choose one of the nearest longest side of ℜ as S_r
else if r is a corner robot then
    Choose the incident longest side of ℜ as S_r
else if r is a boundary robot on one of the shortest sides of ℜ then
    Choose one of the nearest longest sides of ℜ as S_r
else
    Choose the side where it is currently situated as S_r
```

Lemma 7.3.4. *A monitor robot r lying on S_r selects its apex point a_r on $\frac{1}{8}S_r$ so that all the robots on $\Re$ are visible to it after its movement.*

Proof. We want to take advantage of the properties of the convex hull. Let us consider a convex hull $\mathcal{CH}^G$ of N points where all the points lie on the boundary of $\mathcal{CH}^G$. If a point v and its two neighbours are three vertices of $\mathcal{CH}^G$, then no two points on $\mathcal{CH}^G$ are collinear with v. By the definition of a monitor robot, r must be a terminal robot on S_r. Let us denote the convex hull of all the robots of $\Re$ by $\mathcal{CH}^G$. Let us further assume that r', r_1 and r_2 are the other monitor robots in $\Re$, where r' lies on S_r and both r_1 and r_2 lie on S_r^{opp}, (Fig. 7.16). Observe that all of r, r', r_1 and r_2 are vertices of $\mathcal{CH}^G$ (shaded region with red

Algorithm 19: TARGETPOINT_ON_SIDES()

$\mathcal{V}_r \longleftarrow$ The set of all robots not lying on L_r
if *the target side of* r *is* S_r **then**
  if p_r *is not visible to* r **then**
    No change in color and position
  else
    if *(*r *is a boundary robot not lying on one of the sides* S_r *and* S_r^{opp} *of* $\Re$ *with no robot on* p_r*)* $\vee$ *(*r *is an interior robot with another robot on* p_r*)* $\vee$ *(*r *is a corner robot)* **then**
      if $\mathcal{V}_r$ *is empty* **then**
        Choose the point t_r as the target point on the side S_r such that $d(p_r, t_r) = \frac{1}{2}\max\{d(e^1_{S_r}, p_r), d(e^2_{S_r}, p_r)\}$
      else
        Choose the target point t_r on S_r such that $d(p_r, t_r) = \frac{1}{4}\min_{r' \in \mathcal{V}_r}\{d(r', L_r)\}$
    else if r *is an interior robot with no robot on* p_r **then**
      r chooses the point p_r as the target point
    else
      r does not move or change its current color
else
  $p_r^{opp} \longleftarrow$ The point of intersection L_r and S_r^{opp}
  if p_r^{opp} *has no robots on it* **then**
    Choose the point p_r^{opp} as the target point
  else
    Choose a point t_r on S_r^{opp} as the target point such that $d(p_r^{opp}, t_r) = \frac{1}{4}\min_{r' \in \mathcal{V}_r} d(r', L_r)$

Algorithm 20: MOVE_TO_APEXPOINT()

if S_r^{opp} *has some robots on it* **then**
  $r_{Nbr} \longleftarrow$ The neighbour of r on $\mathcal{CH}_r$ lying on $(S_r^{opp}, \frac{1}{8}S_r^{opp}]$
  Choose the point of intersection of $\frac{1}{8}S_r$ and the line $\overleftrightarrow{r r_{Nbr}}$ as the target point a_r
else
  Choose the point of intersection of $\frac{1}{8}S_r$ and L_r as the target point a_r

boundary), and r_1 is one of the neighbours of r. After activation, r chooses the point of intersection of the line $\overleftrightarrow{r r_1}$ and $\frac{1}{8}S_r$ as its apex point a_r. Since r follows a path to a_r on the line $\overleftrightarrow{r r_1}$, it remains a vertex of $\mathcal{CH}^G$ after the movement. If the other neighbour (lying on

Algorithm 21: MOVE_TO_FINALPOINT()

if $r.color = FINISH$ **then**
  if $0 < c_r^1 = c_r^2$ **then**
    $r_{Nbr} \longleftarrow$ The neighbour of r on $\mathcal{CH}_r$ whose nearest longest side is S_r^{opp}
    $SS_r \longleftarrow$ The shortest side of $\Re$ which either lies on or intersects the half plane delimited by the line $r r_{Nbr}$ where no other robot in $\Re$ is present
    Compute the target point t_r on $\frac{1}{4}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$
  else // $0 = c_r^2 < c_r^1$
    $SS_r \longleftarrow$ The shortest side of $\Re$ which lies on the half plane delimited by L_r where no other robot in $\Re$ resides
    Calculates the target point t_r on $\frac{1}{2}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$
else
  $SS_r \longleftarrow$ The side of $\Re$ for which the intersection point of S_r and SS_r is visible to r
  $SS_r^{opp} \longleftarrow$ The side of $\Re$ opposite to SS_r
  if *r finds a FINISH-colored robot on $\frac{1}{4}S_r$ but no robot on $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$* **then**
    r does not change its color and position
  else
    if *there is a FINISH-colored robot on $\frac{1}{2}S_r$* **then**
      Choose $L = \frac{1}{2}S_r$
    else // (a FINISH-colored robot on $\frac{1}{4}S_r$ and some robots on $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$) or (a FINISH-colored robot on $\frac{1}{4}S_r^{opp}$, but not on $[S_r, \frac{1}{4}S_r]$)
      Choose $L = \frac{1}{4}S_r$
    $\mathcal{F}_r \longleftarrow$ Set of all FINISH-colored robots
    Calculate $d = \min_{r' \in \mathcal{F}_r}\{\min\{d(r', SS_r), d(r', SS_r^{opp})\}\}$
    $r_1, r_2, \cdots, r_k$ is the sequence of robots of maximum length on L starting from r_1 with $d(r_1, SS_r) = d$ till r_k such that two consecutive robots in the sequence are exactly at $2d$ distance apart from each other
    Choose the target point t_r on L such that $d(t_r, SS_r) = (2k+1)d$

S_r) of r is r'', then r'' becomes a vertex of the convex hull $\mathcal{CH}^G$ as r has moved to a_r. If the robot r_1 stays on S_r^{opp}, it remains a vertex of $\mathcal{CH}^G$. In simultaneous movement with r, r_1 also follows a path on $\overleftrightarrow{r r_1}$, leading r_1 to remain a vertex of $\mathcal{CH}^G$. Similarly, if r' and r_2 move to their respective apex points, they remain as vertices of $\mathcal{CH}^G$ (shaded region with blue boundary). So, no two robots become collinear with r after moving to its apex point.

When all the robots lie on one side S_r (r_1 and r_2 do not exist), a_r is the point of inter-

section of $\frac{1}{8}S_r$ and L_r. If another monitor robot r' moves simultaneously with r to its apex point $a_{r'}$ on $\frac{1}{8}S_r$, all other robots lie between L_r and $L_{r'}$ and on S_r. Hence, r can see all the robots on S_r after moving to the apex point a_r. □

Lemma 7.3.5. *A robot r moves to its apex point a_r on $\frac{1}{8}S_r$ without any collision.*

Proof. When all the robots are positioned on one longest side, say CD of $\Re$, two monitor robots lie on CD. In this case, if r and r' be the monitor robots, they choose their respective apex points a_r and $a_{r'}$ on $\frac{1}{8}CD$ (as it is the same as $\frac{1}{8}S_r$ and $\frac{1}{8}S_{r'}$) such that $\overline{ra_r} \perp CD$ and $\overline{r'a_{r'}} \perp CD$. This implies $\overline{ra_r} \parallel \overline{r'a_{r'}}$, which means that the two robots cannot collide.

When both the longest sides have some robots, there can be at most four monitor robots. Without loss of generality, we assume that r_1 and r_2 are the two monitor robots on S_r^{opp} along with r and r' being the monitor robots on S_r. As depicted in Fig 7.16, r_1 is a neighbour of r on the convex hull $\mathcal{CH}^G$. r cannot meet a collision with r_1 and r_2 even in case of simultaneous movement because a_r lies on $\frac{1}{8}S_r$ whereas a_{r_1} and a_{r_2} lie on $\frac{1}{8}S_r^{opp}$ (which is same as $\frac{7}{8}S_r$). It is also not possible for r to collide with r', as r and r' follow the paths $\overline{rr_1}$ and $\overline{r'r_2}$ which are two different sides of the convex hull $\mathcal{CH}^G$, as described in Lemma 7.3.4. □

Remark 7.3.6. When neither there is a `OFF`-colored robot in $Int(\Re)$ nor a `FINISH`-colored robot on $(\frac{1}{8}S_r, \frac{7}{8}S_r)$, a `FINISH`-colored robot lying on $\frac{1}{8}S_r$ accurately calculates c_r^1 and c_r^2, where c_r^1 is the total number of the robots on S_r and on the band $[S_r, \frac{1}{2}S_r]$ including itself. c_r^2 is the total number of robots on S_r^{opp} and on the band $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$.

Lemma 7.3.7. *If the longest side AB of $\Re$ contains less number of robots than the other longest side CD, all the robots on AB move to CD without collision.*

Proof. Let r be a robot on AB which means $S_r = AB$ and $CD = S_r^{opp}$. Let us assume that r is a terminal robot on S_r. If there exists a `OFF`-colored robot r' in $Int(\Re)$, either r' is moving towards one of the longest sides from its apex point, or it is still in its initial position. In both situations, r' eventually allocates itself on one of the longest sides of $\Re$, by Lemma 7.3.3. So r eventually becomes a monitor robot. It then moves to its apex point a_r and checks whether there is any `FINISH`-colored robot r'' in $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$. When r'' finds $c_{r''}^1 > c_{r''}^2$, it

moves back to S_r^{opp}. After this, r accurately computes c_r^1 and c_r^2 by Remark 7.3.6. By the assumption of this Lemma 7.3.7, $c_r^1 < c_r^2$, so it finds p_r^{opp}. r moves to p_r^{opp}, if it is empty. If not, r finds a point t_r on S_r^{opp} such that $d(t_r, p_r^{opp}) = 1/4 \min_{r' \in \mathcal{V}_r} d(r', L_r)$ and moves to the point t_r. By similar arguments presented in Lemma 7.3.3, this type of movement is collision-free even if there is another robot simultaneously moving towards S_r^{opp}. If r is not a terminal robot on S_r, it eventually becomes a terminal robot and thus, every robot on AB moves to some point on CD. □

Remark 7.3.8. Eventually, either one of the longest sides contains all of the robots of $\Re$, or both the longest sides contain an equal number of robots.

Lemma 7.3.9. *For a* `FINISH`*-colored robot r lying on $\frac{1}{8}S_r$ with $0 < c_r^1 = c_r^2$ or $0 = c_r^2 < c_r^1$, when there is no robot on $(\frac{1}{8}S_r, \frac{7}{8}S_r)$, it terminates to a unique partition of $\Re$ and the movement is collision-free.*

Proof. We first prove the lemma when r satisfies $0 < c_1^r = c_r^2$. In this situation, r first checks whether any `OFF`-colored robot exists in $Int(\Re)$. If such a robot r' exists, it must be in the move phase towards its target position. Since r lies on $\frac{1}{8}S_r$ in the current LCM cycle, it must have been qualified for the monitor robot when it was situated on S_r and did not find `OFF`-colored robots in $Int(\Re)$. Thus, in the current LCM cycle of r, the robot r' must be in the compute or move phase, aiming towards one of the longest sides of $\Re$ from its apex point $a_{r'}$. Therefore, r eventually finds no `OFF`-colored robots in $Int(\Re)$. After this, according to our strategy, one of the shortest sides is selected as SS_r. There can be another robot r'' exists that satisfies all the conditions similar to those of r. Then, if it is activated simultaneously with r, it also chooses $SS_{r''}$. Let's consider r_1 as the neighbour of r on the convex hull $\mathcal{CH}_r$. If the shortest side AD lies within or intersects the half-plane defined by the line $\overleftrightarrow{rr_1}$, where no other robot of $\Re$ resides, then r assigns $SS_r = AD$, and r'' assigns $SS_{r''} = BC$. Otherwise, $SS_r = BC$ and $SS_{r''} = AD$. Thus, the selection of SS_r and $SS_{r''}$ are different. Then, r chooses its destination point on $\frac{1}{4}S_r$ at $\frac{len(S_r)}{2c_r^1}$ distance away from SS_r. Similarly r'' chooses its destination point on $\frac{1}{4}S_r$ (as it is same as $\frac{1}{4}S_{r''}$) at $\frac{len(S_r)}{2c_r^1}$ distance away from $SS_{r''}$. These two destination points are $len(S_r) - \frac{len(S_r)}{c_r^1}$ distance apart from each

other, and the movement is free from any collision. So r and r'' both move to separate partitions, each with an area $\frac{len(SS_r)}{2} \times \frac{len(S_r)}{c_r^1} = \frac{len(AC)\cdot len(AB)}{2c_r^1}$.

We now prove the lemma when $0 = c_r^2 < c_r^1$. Similar to the previous arguments, r must eventually see no OFF-colored robot on $Int(\Re)$. If another robot r'' satisfies conditions similar to those of r, they find SS_r and $SS_{r''}$ individually. According to our algorithm, r chooses its destination point on $\frac{1}{2}S_r$ at $\frac{len(S_r)}{2c_r^1}$ distance away from SS_r and r'' chooses its destination point on $\frac{1}{2}S_r$ at $\frac{len(S_r)}{2c_r^1}$ distance away from $SS_{r''}$. By a similar argument, it can be proved that r and r'' reach different partitions, each with the area $len(SS_r) \times \frac{len(S_r)}{c_r^1}$.

Now, we prove that the movement of r is free from the collision. Without loss of generality, we assume that r'' lies on the right half plane delimited by L_r. Then SS_r must lie on the left half plane delimited on L_r. Since $d(t_r, SS_r) < d(t_{r''}, SS_r)$ for $t_r, t_{r''}$ lying on $L = \frac{1}{2}S_r$ (or $\frac{1}{4}S_r$), $t_{r''}$ lies on the right of t_r on L. If t_{mid} and r_{mid} are the midpoints of the line segments $\overline{t_r t_{r''}}$ and $\overline{r r''}$, respectively, the line $\overleftrightarrow{t_{mid} r_{mid}}$ separates the two paths $\overline{r t_r}$ and $\overline{r'' t_{r''}}$. Hence, the movement of r is free from collision even if r'' simultaneously moves with r. □

Lemma 7.3.10. *A OFF-colored robot r, which lies on S_r and satisfies one of the following conditions, terminates to a unique partition of $\Re$, and the movement is collision-free.*
(i) r finds a FINISH-colored robot on $\frac{1}{4}S_r$ and some robots on $[S_r^{opp}, \frac{1}{4}S_r^{opp}]$.
(ii) r finds a FINISH-colored robot on $\frac{1}{4}S_r^{opp}$, but not on $[S_r, \frac{1}{4}S_r]$.
(iii) r finds a FINISH-colored robot on $\frac{1}{2}S_r$.

Proof. If r satisfies (i), it first identifies the side SS_r, as described in Case 2, and r follows the Type II partitioning. Another robot r' on S_r might exist that satisfies conditions similar to those of r. It also chooses $SS_{r'}$ if it is simultaneously activated with r. If the point of intersection of S_r and AD is visible to r, it selects AD as SS_r and r' selects BC as $SS_{r'}$. Otherwise, $SS_r = BC$ and $SS_{r'} = AD$. Thus SS_r and $SS_{r'}$ are different. r sets $L = \frac{1}{4}S_r$ and computes the number k, d, its target point t_r on L such that $d(t_r, SS_r) = (2k+1)d$, as explained in Case 2. Similarly, r' can find its target point $t_{r'}$ on L such that $d(t_{r'}, SS_{r'}) = (2k'+1)d$, where k' is the length of the sequence of robots on L considered by the robot r' with respect to $SS_{r'}$. Observe that $d(t_r, t_{r'}) = len(S_r) - 2(k+k'+1)d > 0$, as $len(S_r) \geq 2(k+k')d + 4d$ and $d \neq 0$. Hence, r terminates at a unique partition of $\Re$. When r satisfies (ii) and (iii), by similar

arguments as above, we can prove that it terminates at a unique partition of $\Re$.

Using analogous reasoning as presented in the proof of collision freeness in Lemma 7.3.9, we can demonstrate that the movement of r is also free from collisions. □

Theorem 7.3.11. *Our algorithm* RECTANGLE_PARTITION *solves the uniform partitioning without collision for the rectangular region in $O(N)$ epochs, where N is the total number of robots in $\Re$.*

Proof. By Remark 7.3.8, eventually, either all the robots lie on one of the longest sides of $\Re$, or both of the longest sides have an equal number of robots. On both accounts, a robot r must qualify as a monitor robot, which later moves to its apex point and finds either $0 < c_r^1 = c_r^2$ or $0 = c_r^2 < c_r^1$. By Lemma 7.3.9, it is ensured that r terminates at a unique partition of $\Re$ (it lies either on $\frac{1}{2}S_r$ for Type I partitioning or on $\frac{1}{4}S_r$ for Type II partitioning). After termination of r, if there is an `OFF`-colored robot r' on one of the longest sides, the position of r helps it find its destination point either on $\frac{1}{2}S_r$ or on $\frac{1}{4}S_r$. Lemma 7.3.10 ensures that r' terminates at a unique partition of $\Re$.

In the worst case, initially, all robots lie on a line inside $\Re$. It takes $O(N)$ epochs for all the robots to reach the boundary, as the movement of the robots will be sequential. Monitor robots on the sides of $\Re$ move to their apex point in one epoch. Moreover, if the two longest sides of $\Re$ have different numbers of robots, the robots move from the longest side with fewer robots (say AB) to the other longest side (CD) of $\Re$. A monitor robot r on AB gets activated and moves to its apex points in one epoch. Since the number of robots on CD is more, r needs to wait till all the robots on $[S_r^{opp}, \frac{1}{8}S_r^{opp}]$ move back to CD, which requires four epochs, as each of the two `FINISH`-colored robots lying on $\frac{1}{8}CD$ might take one epoch to reach CD and one epoch to change its color to `OFF`. It might also need to wait for one epoch if there is another robot on $(\frac{1}{8}S_r, \frac{7}{8}S_r)$ moving towards CD. When no such robot exists on $(\frac{1}{8}S_r, \frac{7}{8}S_r)$, r calculates c_r^1 and c_r^2 (by Remark 7.3.6) and decides to move to CD which takes one epoch. So, it needs seven epochs for r to move to CD from AB. In the worst case, it is possible that the robots on AB sequentially execute the movement to CD. So, this process takes $O(N)$ epochs to complete. So, all the robots reposition themselves on exactly one longest side or get distributed equally among the two longest sides of $\Re$

takes $O(N)$ epochs. When all the robots are on one longest side CD of $\Re$, there can be two monitor robots r and r'. They reach their respective apex points in one epoch. In the worst case, r' gets activated first and moves to its final position on $\frac{1}{2}CD$. While r' is executing its movement, r gets activated and finds r' on $(\frac{1}{8}CD, \frac{7}{8}CD)$ (as $S_r = CD$). So, it decides to move back to CD with color FINISH, which needs one epoch. When it reaches CD, it needs another epoch to change its color to OFF. When r is again activated and finds r' on $\frac{1}{2}S_r$, it calculates and moves to its final position on $\frac{1}{2}S_r$ in one epoch. Thus, r needs at most four epochs to reach its final point. In the worst case, the other OFF-colored robot on CD moves to their final point one by one, requiring $O(N)$ epochs. On the other hand, if the robots are distributed equally on both the longest sides, by similar arguments as above, we can prove that the robots need $O(N)$ epochs to terminate. So, overall, the algorithm requires $O(N)$ epochs to achieve uniform partitioning when the region is rectangular.

Lemma 7.3.3, 7.3.5, 7.3.7, 7.3.9 and 7.3.10 ensure that the movements of the robots are collision-free. □

7.4 Algorithm for a Square Region

The region $\Re$ is considered to be square in this section. Our proposed algorithm SQUARE_PARTITION uses 5 colors described in the Table 7.1 with their specification.

7.4.1 Description of the Algorithm

From any initial deployment of the robots inside $\Re$, our target is to move the interior and corner robots to the boundary. A robot r, after activating with color OFF, determines whether it is a corner, boundary or interior robot. In case of r being a corner or an interior robot, it finds a target side S_r and moves to a point on it with maintaining its color OFF. Then r waits for other interior robots and the visible corner robots to move to one of the sides of $\Re$.

If r is a corner robot, it chooses any of the incident sides as S_r. In the case of r being an interior robot on $\Re$, it selects one of the nearest sides from its current position as S_r.

The strategy is the same as the rectangle. Since there is no shortest side in a square, the difference here is when r is a boundary robot in $\Re$, it does not move. In all other cases, a target point t_r is calculated on S_r, and the robot r moves to it with color OFF by following a movement strategy similar to the strategy MOVE_TO_LONGESTSIDES.

After this, the outline of the strategy for the square region has similarities with the algorithm RECTANGLE_PARTITION. In rectangular regions, we ask the monitor robots to move at a particular distance from the longest sides and make them compare the number of robots near the two longest sides of the region. If the number is not the same, we bring the robots from the side with fewer robots to the side with a larger number of robots. In the case of the square region, we follow a similar strategy, but here, the robots can be situated on any side of $\Re$. If all the robots are distributed among the two sides S_r and S_r^{opp}, the robot r follows the Type I or the Type II partitioning. For other cases, r follows Type III and Type IV partitioning. We define some new notations and conventions to describe the algorithm for the square regions.

- C is the center of the square $\Re$.
- The triangle $\Delta C e_S^1 e_S^2$, denoted by ΔS, is called the *side triangle* of the side S where e_S^1 and e_S^2 are the endpoints of the side S.
- When we say that r lies on ΔS, we mean that r lies either on the side S or strictly within the triangle $\Delta C e_S^1 e_S^2$, but not on $\overline{Ce_S^1}$ or $\overline{Ce_S^2}$.
- $|S|$ denotes the number of robots lying on the side triangle ΔS of S.
- For the robot r, S_r^L and S_r^R represent the sides of $\Re$ other than S_r and S_r^{opp}.
- S_r is the nearest side of the square $\Re$ for the robot r.
- D_r and D_r^{opp} are the two diagonals of $\Re$ for the robot r.
- Max_r represents a set of all sides with the maximum number of robots lying on their corresponding side triangle.

Max_r helps r to decide which side triangles contain the maximum number of robots. For example, $Max_r = \{S_r, S_r^{opp}\}$ if $|S_r| = |S_r^{opp}| > |S_r^L|, |S_r^R|$. Here, the sides S_r and S_r^{opp} have the maximum number of robots. Our aim is to move all the robots from the sides not in Max_r to the sides in Max_r. When Max_r contains all the sides of $\Re$, all the four sides have $\frac{N}{4}$ robots.

Now, we redefine the monitor robot for the square region. A robot r on S_r is called a *monitor robot* (refer to Fig. 7.17) if all the four conditions are satisfied. (i) r is a terminal robot on S_r. (ii) $\left((S_r, \frac{1}{8}S_r) \cap \Delta S_r\right) \cup \left((\frac{1}{8}S_r, \frac{7}{8}S_r) \cap (\frac{1}{8}S_r^L, \frac{7}{8}S_r^L)\right)$ has no robots (shown as the shaded region). (iii) All the robots on $(S_r^{opp}, \frac{1}{8}S_r^{opp}] \cup (S_r^L, \frac{1}{8}S_r^L] \cup (S_r^R, \frac{1}{8}S_r^R]$ are with color MONITOR. (iv) There is no visible corner robot.

We differentiate the rest of the algorithm into three major cases for a terminal robot r lying on the side S_r.

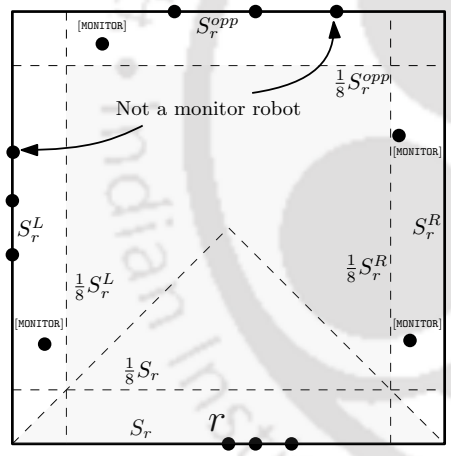


Figure 7.17: r qualifies to be a monitor robot

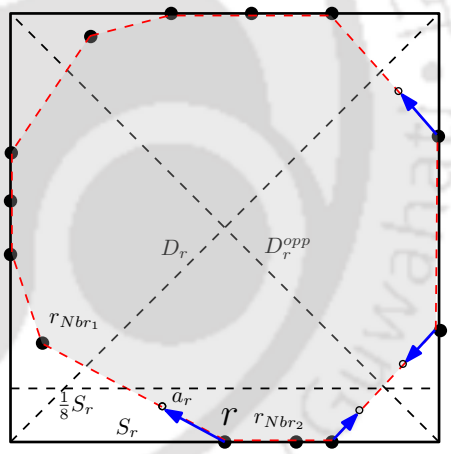


Figure 7.18: r moves to its apex point a_r considering the convex hull $\mathcal{CH}_r$

Case 1 (r is a monitor robot and $Int(\Re)$ has no robot with color FINISH or FINISH1 or FINISH2): Computing Max_r is possible when r moves to a point in $Int(\Re)$, referred to as *apex point*, to calculate the number of robots on each side. The apex point must be chosen so that r gets to see all the robots in $\Re$, and it does not obstruct the visibility of any other robot after the movement. Also, we want r to lie within ΔS_r so that it can choose S_r easily. If S_r has no other robot, it does not need to move to an apex point. Rather, it can find out Max_r without moving from S_r and directly follow the movement strategy for the MONITOR-

colored robots lying on $(S_r, \frac{1}{8}S_r]$, which is described later in this case. We now discuss the movement strategy of a monitor robot r lying on S_r to its apex points where $|S_r| \geq 2$.

Strategy MONITORROBOT_MOVEMENT_SQUARE: If the robot r does not find any robot on ΔS_r^{opp} nor ΔS_r^L nor ΔS_r^R, it chooses the point of intersection of L_r and $\frac{1}{8}S_r$ as its apex point a_r and moves to it with color `MONITOR`. Otherwise, r finds the neighbour r_{Nbr_1} on the convex hull $\mathcal{CH}_r$, which does not lie on the side S_r. r also finds the neighbour r_{Nbr_2} on $\mathcal{CH}_r$ other than r_{Nbr_1} which lies on S_r, as shown in Fig 7.18. It calculates the apex point a_r on the line segment $\overline{rr_{Nbr_1}}$ such that $d(r,a_r) = \frac{1}{2}\min\{d(r,D_r), d(r,D_r^{opp}), d(r,\frac{1}{4}S_r), d(r,r_{Nbr_2})\}$. The point a_r is chosen on $\overline{rr_{Nbr_1}}$ to ensure that r does not become an obstruction for other monitor robots after its movement. Moreover, it also ensures that r does not cross D_r and D_r^{opp}, allowing r to accurately determine S_r (so that it remains the nearest side to r) after moving to the apex point. We also want r to lie on or below the line segment $\frac{1}{8}S_r$, as our aim is to terminate r either on $\frac{1}{2}S_r$, $\frac{1}{4}S_r$, $\frac{1}{3}S_r$ or $\frac{1}{6}S_r$. Additionally, we do not cross the line $L_{r_{Nbr_2}}$ after the movement to ensure that all other robots on S_r must be on one side of the half-plane delimited by L_r. Finally, r changes its current color to `MONITOR` and moves to a_r.

Next, we explain how a `MONITOR`-colored robot r lying on $(S_r, \frac{1}{8}S_r]$, decides its destination based on Max_r. It moves from its apex point either to one of the sides or to the final position.

Movement of a `MONITOR`-colored Robot r lying on $(S_r, \frac{1}{8}S_r]$: If r sees any `OFF`-colored robot in $Int(\Re)$, r does not change its color or position. Otherwise, before calculating Max_r, r checks whether there is any visible `FINISH` or `FINISH1` or `FINISH2`-colored robot. These three colors are used when a robot moves to its final position. Such a robot might lead to the miscalculation of Max_r, as it can become an obstruction for r. We explain the rest of the algorithm using a proper example and figures for better comprehension. We consider a square with corners e_P (common corner of S_r and S_r^L), e_Q (common corner of S_r and S_r^R), e_R (common corner of S_r^{opp} and S_r^R) and e_X (common corner of S_r^{opp} and S_r^L), as depicted in Fig. 7.19. We now identify four sub-cases.

- **Case 1.1 (r sees a `FINISH`-colored robot):** r waits till all `FINISH`-colored robots reach $\frac{1}{2}S_r$ or $\frac{1}{4}S_r$ or $\frac{3}{4}S_r$. The robot r now moves to its final position on $L = \frac{1}{2}S_r$ or $\frac{1}{4}S_r$

with color FINISH by following the same movement strategy as an OFF-colored robot, described in Case 2 of RECTANGLE_PARTITION algorithm (Section 7.3). The only difference is the choice of SS_r, which, in this case, belongs to $\{S_r^L, S_r^R\}$ (as there is no shortest side in a square).

- **Case 1.2 (r sees a FINISH1-colored robot):** r waits till all FINISH1-colored robots reach $\frac{1}{3}S_r$ or $\frac{1}{3}S$ where $S \in \{S_r^L, S_r^R\}$. Let c be number of FINISH1-colored robot on $\frac{1}{3}S_r$. Note that $c \leq 1$ in this case. Let us also assume that c' is the number of robots lying on ΔS_r without the color FINISH1. Without loss of generality, let us assume $S = S_r^L$. r first chooses the corner e_R as the vertex of Type III partitioning and then calculates two points A and B on S_r which are $\frac{len(S_r)}{c+c'}$ distance away from e_P and e_Q, respectively, as shown in Fig. 7.19. If S^{opp} lies on the half plane delimited by L_r where other robots on ΔS_r lie, r chooses the triangle $\mathcal{T} = \Delta A e_P e_R$. Otherwise, it chooses $\mathcal{T} = \Delta B e_Q e_R$. r moves to the centroid of the triangle $\mathcal{T}$ after changing its current color to FINISH1 from FINISH.

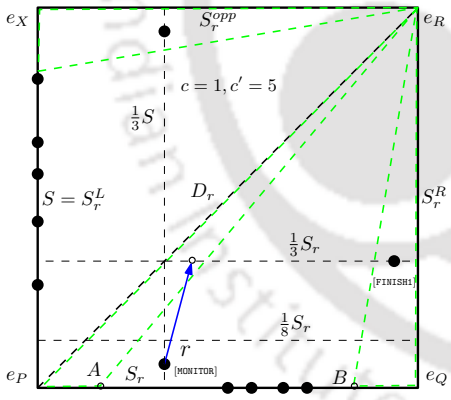


Figure 7.19: The movement of r for Type III partitioning when $r.color$ = MONITOR

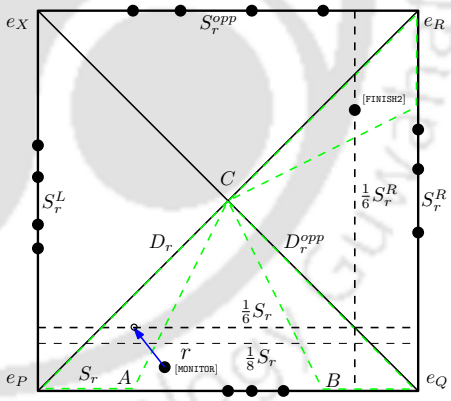


Figure 7.20: The movement of r for Type IV partitioning when $r.color$ = MONITOR

- **Case 1.3 (r sees a FINISH2-colored robot):** r waits till all the FINISH2-colored robots lie on $\frac{1}{6}S$ where $S \in \{S_r, S_r^{opp}, S_r^L, S_r^R\}$. Afterwards, r computes the two points A and B on S_r which are $\frac{len(S_r)}{|S_r|}$ distance away from e_P and e_Q, respectively as shown in Fig. 7.20. If e_P and all the other robots on ΔS_r lie on the different half plane delimited by L_r, the triangle $\mathcal{T}$ is chosen that satisfies $\mathcal{T} = \Delta A e_P C$. Otherwise, it chooses $\mathcal{T} =$

ΔBe_QC. Then r moves to the centroid of the triangle $\mathcal{T}$ after changing its current color to FINISH2 from FINISH.

- **Case 1.4 (r does not see any FINISH or FINISH1 or FINISH2-colored robot):** In this case, r considers Max_r to understand the type of partitioning of the region $\Re$. We identify ten sub-cases, out of which the first four sub-cases discuss the process of choosing the type of partitioning and the movement of r to the final position in its respective partition.

  - **Case 1.4.1 ($Max_r = \{S_r\}$ and ΔS contains no robot for all $S \in \{S_r^{opp}, S_r^L, S_r^R\}$):** Similar to Case 1.4 in the algorithm RECTANGLE_PARTITION (in Section 7.3), r moves to the final position on $\frac{1}{2}S_r$ with color FINISH by considering SS_r from $\{S_r^L, S_r^R\}$.
  - **Case 1.4.2 ($Max_r = \{S_r, S_r^{opp}\}$ and no robots on ΔS for all $S \in \{S_r^L, S_r^R\}$):** Similar to Case 1.3 in the algorithm RECTANGLE_PARTITION (in Section 7.3), r moves to the final position on $\frac{1}{4}S_r$ with color FINISH by considering SS_r from $\{S_r^L, S_r^R\}$.
  - **Case 1.4.3 ($Max_r = \{S_r, S\}$ and both ΔS_r^{opp} and ΔS^{opp} contain no robot where $S \in \{S_r^L, S_r^R\}$):** Without loss of generality, let us assume that $S = S_r^L$. Similar to Case 1.2 of SQUARE_PARTITION algorithm, r calculates two points A and B on S_r, as illustrated in Fig 7.19. It selects the triangle $\mathcal{T}$ and moves to the centroid of $\mathcal{T}$ after changing its color to FINISH1 from MONITOR.
  - **Case 1.4.4 ($Max_r = \{S_r, S_r^{opp}, S_r^L, S_r^R\}$):** Similar to Case 1.3 of SQUARE_PARTITION algorithm, r finds the triangle $\mathcal{T}$ and moves to the centroid of the triangle $\mathcal{T}$, as depicted in Fig 7.20 after changing its current color to FINISH2 from MONITOR.

  For the remaining six sub-cases, r chooses a target side before its movement. Our aim is to gather all the robots to one of the four sides of $\Re$. If, due to symmetry, it is not possible, robots should be gathered on two sides of $\Re$. When Case 1.4.1, Case 1.4.2, Case 1.4.3 and Case 1.4.4 do not hold, the following cases may occur for different Max_r. r decides a side as its target for its movement based on Max_r.

– **Case 1.4.5** ($Max_r = \{S_r, S_r^{opp}, S\}$ **where** $S \in \{S_r^L, S_r^R\}$)**:** Without loss of generality, if $S = S_r^L$, the side S_r^R has the minimum number of robots. We target to move r to the side S_r^L, which is opposite to the side having minimum number of robots. So, r sets the side S as its target side.

– **Case 1.4.6** ($Max_r = \{S\}$ **or** $\{S, S'\}$**, where** $S, S' \in \{S_r^L, S_r^R\}$ **with** $S \neq S'$)**:** In this case, either one or two adjacent sides to S_r contain maximum number of robots. So r targets to move on one of them accordingly. The target side is chosen to be the side S.

– **Case 1.4.7** ($Max_r = \{S_r^L, S_r^R, S_r^{opp}\}$ **or** $\{S_r^{opp}\}$)**:** The target side is the side S_r^{opp}.

– **Case 1.4.8** ($Max_r = \{S, S_r^{opp}\}$**, where** $S \in \{S_r^L, S_r^R\}$)**:** The target side for r is S.

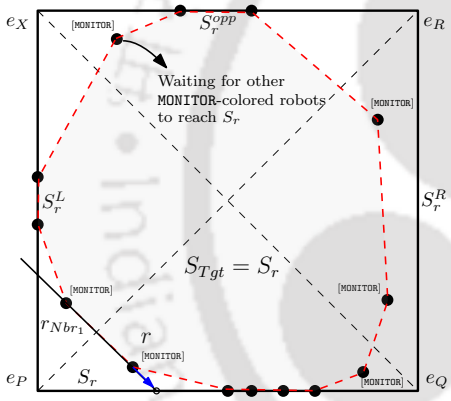


Figure 7.21: r moves back to S_r from its apex point a_r considering the convex hull $\mathcal{CH}_r$

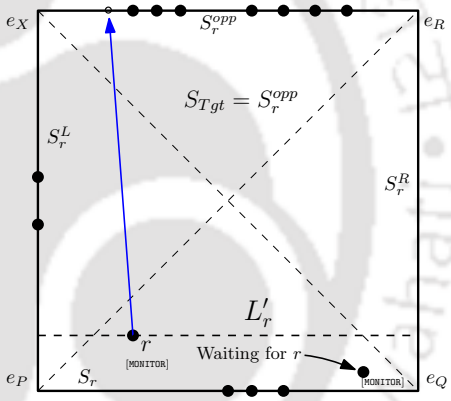


Figure 7.22: The movement of r to S_{Tgt} when the half plane delimited by L_r' has no MONITOR-colored robot

– **Case 1.4.9** ($Max_r = \{S_r\}$ **or** $\{S_r, S_r^L, S_r^R\}$)**:** In this case, r chooses S_r its target side.

– **Case 1.4.10** ($Max_r = \{S, S_r\}$ **where** $S \in \{S_r^{opp}, S_r^L, S_r^R\}$)**:** S_r is the target side of r.

Let S_{Tgt} be the target side of r and L_r' be the line passing through r and parallel to S_{Tgt}. If $S_{Tgt} = S_r$, then r considers the line $\overleftrightarrow{r r_{Nbr_1}}$ (ref. Fig 7.21) where r_{Nbr_1} is the neighbour of r on the convex hull $\mathcal{CH}_r$ not lying on ΔS_r. It moves to the point of intersection t_r of S_r and $\overleftrightarrow{r r_{Nbr_1}}$ with the current color MONITOR. If r gets activated again with color MONITOR on S_r, it changes its color to OFF with no movement. When $S_{Tgt} \neq S_r$, r waits until there is no MONITOR-colored robot lying on the half plane delimited by L_r' that contains S_{Tgt}, as

shown in Fig. 7.22. r moves with color `OFF` to the target side S_{Tgt}, by following the similar movement strategy as MOVEMENT_TO_LONGESTSIDE. The above cases can interchangeably occur in different LCM cycles of a robot r.

Case 2 ($Int(\Re)$ has `FINISH`-colored robots): r understands that the partitioning of $\Re$ would be of Type II after seeing a `FINISH`-colored robot. r figures out its final position by looking at the positions of the robots with color `FINISH`. r follows the movement strategy similar to Case 2 in RECTANGLE_PARTITION algorithm (Section 7.3) to move to the final position on $\frac{1}{2}S_r$ (for Type I) or on $\frac{1}{4}S_r$ (for Type II) with color `FINISH`.

Case 3 ($Int(\Re)$ has `FINISH1`-colored robots): r understands the partitioning would be of Type III after seeing `FINISH1`-colored robots. r figures out its final position by looking at the positions of the robots with color `FINISH1`.

- $\frac{1}{3}S_r^L$ **has** `FINISH1`**-colored robot:** r chooses the side $S = S_r^L$.
- $\frac{1}{3}S_r^R$ **has a** `FINISH1`**-colored robot:** r chooses S_r^R as S.
- **A robot lying on the side triangle of** ΔS_r^L**:** r chooses the side S_r^L as S.
- **A robot lying on the side triangle of** ΔS_r^R**:** The side S_r^R is chosen as S.

The robot r waits till all the `FINISH1`-colored robots lie either on $\frac{1}{3}S_r$ or on $\frac{1}{3}S$. Without loss of generality, we assume $S = S_r^L$. At this point, r understands that the partitioning is of Type III. It chooses the point of intersection of S_r^{opp} and S^{opp} (i.e., the corner e_R in this scenario) as the vertex of Type III partitioning. D_r is the diagonal of the region $\Re$ passing through the vertex of Type III partitioning and D_r^{opp} is the other diagonal of $\Re$. Now it computes the length of the base bl of each triangular partition depending on the positions of the `FINISH1`-colored robots, as shown in Fig. 7.23. So, there could be two sub-cases.

- **Case 3.1 (There is a** `FINISH1`**-colored robot on** $\frac{1}{3}S_r$**):** Let r' be a terminal robot on $\frac{1}{3}S_r$ with color `FINISH1`. Let bl_1 be the length of the base of the triangle whose one side is D_r and the centroid is r'. bl_2 is the length of the base of the triangle whose one side is S^{opp} and the centroid is at r'. So, r computes bl such that $bl = \min\{bl_1, bl_2\}$.

- **Case 3.2 (There is no FINISH1-colored robot on $\frac{1}{3}S_r$):** In this case FINISH1-colored robot (say r') must be on $\frac{1}{3}S$. Like the previous case, bl is computed by finding bl_1 and bl_2. The definition of bl_1 is the same as in the previous case (Case 3.1). We calculate bl_2 by considering the triangle with one side of S_r^{opp} and the centroid at r'.

Let u_1 be the point of intersection of $\frac{1}{3}S_r$ and the diagonal D_r. u_2 is the point of intersection of $\frac{1}{3}S_r$ and S^{opp}. If the side S^{opp} lies in the half plane delimited by $\overleftrightarrow{re_R}$, where the other robots on S_r reside, r finds the number of FINISH1-colored robots c on $\frac{1}{3}S_r$ starting from the robot $bl/3$ distance apart from u_1 towards u_2 such that two consecutive robots are at $2bl/3$ distance away from each other, as depicted in Fig. 7.23. Then, it changes its color to FINISH1 and moves to the point t_r on $\frac{1}{3}S_r$ such that $t_r = Centroid(\Delta e_Y e_R e_Z)$, where e_Y and e_Z are the points on S_r satisfying $d(e_Y, e_P) = c \cdot bl$ and $d(e_Z, e_P) = (c+1) \cdot bl$. Otherwise, c is calculated as the number of robots on $\frac{1}{3}S_r$ starting from the robot $bl/3$ distance apart from u_2 towards u_1 to a robot such that two consecutive robots are at $2bl/3$ distance away from each other. r moves to t_r on $\frac{1}{3}S_r$ such that $t_r = Centroid(\Delta e_Y e_R e_Z)$, where e_Y and e_Z are the points on S_r satisfying $d(e_Y, e_Q) = c \cdot bl$ and $d(e_Z, e_Q) = (c+1) \cdot bl$ with color FINISH1.

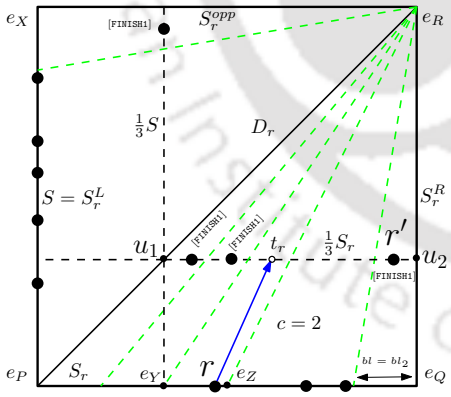


Figure 7.23: r's movement for Type III partitioning for $r.color$ = OFF

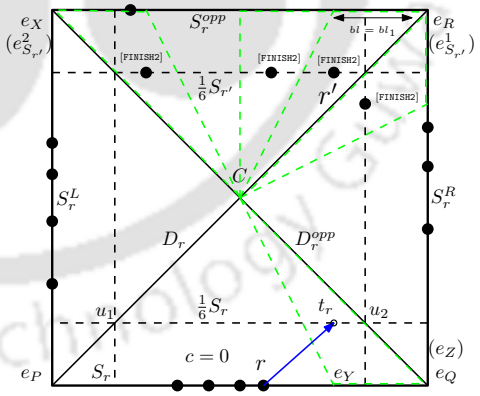


Figure 7.24: r's movement for Type IV partitioning for $r.color$ = OFF

Case 4 ($Int(\Re)$ has FINISH2-colored robots): In this case, r understands that the partitioning would be of Type IV after seeing FINISH2-colored robots. r waits till any robot with color FINISH2 lying on the side triangle of a side S reaches $\frac{1}{6}S$, where $S \in \{S_r, S_r^{opp}, S_r^L, S_r^R\}$. It now needs to calculate the base length bl of the triangular partition. Let r' be a FINISH2-

colored terminal robot on $\frac{1}{6}S_{r'}$, in Fig. 7.24. Let us also consider that bl_1 is the length of the base of the triangle whose centroid is r', and the two vertices are C and $e^1_{S_{r'}}$. bl_2 is the length of the base of the triangle whose centroid is r' and the two vertices are C and $e^2_{S_{r'}}$. Now, r calculates the base length $bl = \min\{bl_1, bl_2\}$. D_r and D_r^{opp} are the two diagonals of passing through e_P and e_Q respectively. Let us further assume u_1 (and u_2) is the point of intersection of $\frac{1}{6}S_r$ and D_r (and D_r^{opp}). If the line segment $\overline{Ce_Q}$ lies on the half plane delimited by the line $\overleftrightarrow{rC}$, where other robots on S_r reside, r finds the number of FINISH2-colored robots c on $\frac{1}{6}S_r$ starting from the robot $bl/3$ distance away from u_1 towards u_2 such that two consecutive robots are $2bl/3$ distance apart from each other. Finally, r changes its color to FINISH2 and moves to the point t_r such that $t_r = Centroid(\Delta Ce_Y e_Z)$, where e_Y and e_Z are the points on S_r satisfying $d(e_P, e_Y) = c \cdot bl$ and $d(e_P, e_Z) = (c+1) \cdot bl$. Otherwise, if the line segment $\overline{Ce_Q}$ lies on the other half plane delimited by the line $\overleftrightarrow{rC}$, where other robots on S_r reside, r finds the number of FINISH2-colored robots c on $\frac{1}{6}S_r$ starting from the robot $bl/3$ distance away from u_2 towards u_1 such that two consecutive robots are $2bl/3$ distance apart from each other. Finally, r changes its color to FINISH2 and moves to the point t_r such that $t_r = Centroid(\Delta Ce_Z e_Y)$, where e_Z and e_Y are the points on S_r satisfying $d(e_Q, e_Z) = c \cdot bl$ and $d(e_Q, e_Y) = (c+1) \cdot bl$.

7.4.2 Analysis of the Algorithm

In this subsection, we analyse the algorithm SQUARE_PARTITION. The movement of the robots is free from collision. The following lemmas and theorems provide the correctness and time complexity of the algorithm.

Lemma 7.4.1. *All interior and corner robots move to the boundary without collision.*

Proof. The proof of this lemma follows from Lemma 7.3.3. □

Lemma 7.4.2. *A monitor robot r lying on S_r selects its apex point a_r on $(S_r, \frac{1}{8}S_r] \cap \Delta S_r$ so that all the robots on $\Re$ are visible to it after its movement and moves to it without collision.*

Proof. When r qualifies as a monitor robot on S_r, it finds the apex point a_r such that $d(r, a_r) < d(r, D_r)$ and $d(r, a_r) < d(r, D_r^{opp})$. So, a_r cannot lie on D_r or $D_r opp$ or outside

ΔS_r. Also, $d(r, a_r) \leq d(r, \frac{1}{8}S_r)$. Thus a_r must be on $(S_r, \frac{1}{8}S_r]$. Hence a_r lies on $(S_r, \frac{1}{8}S_r] \cap \Delta S_r$. The point a_r is chosen on the line segment $r r_{Nbr_1}$ where r_{Nbr_1} is the neighbour of r on $\mathcal{CH}_r$ which does not lie on S_r. By the similar argument as presented in Lemma 7.3.4, we can conclude that r remains a vertex of $\mathcal{CH}^G$ (which is the convex hull of all the robots in $\Re$) after its movement which enables it to see all the robots in $\Re$ after reaching to its apex point.

Observe that even if r and r_{Nbr_1} move simultaneously on $\overline{r r_{Nbr_1}}$, they do not collide as

Algorithm 22: SQUARE_PARTITION

```
1  C ⟵ The center of ℜ
2  e_P ⟵ The common corner of S_r and S_r^L,
3  e_Q ⟵ The common corner of S_r and S_r^R
4  e_R ⟵ The common corner of S_r^opp and S_r^R
5  e_X ⟵ The common corner of S_r^opp and S_r^L
6  if r.color = OFF then
7      Choose one of the nearest sides of ℜ as S_r
8      if r is not a boundary robot on S_r then
9          Follow TARGETPOINT_ON_SIDES() to find the target point
10         Move to the target point on S_r with color OFF
11     else
12         if r is a monitor robot and Int(ℜ) has no FINISH or FINISH1 or
             FINISH2-colored robot then
13             Follow APEX_POINT_INSQUARE() to find the target point t_r
14             Change r.color to MONITOR
15             Move to the point t_r
16         else if r is a terminal robot and Int(ℜ) has FINISH-colored robots then
17             Follow MOVE_TO_FINALPOINT() to find a target point
18             Change r.color to FINISH
19             Move to the target point
20         else if r is a terminal robot and Int(ℜ) has FINISH1-colored robots then
21             Follow PARTITIONTYPE_III to find a target point
22             Change r.color to FINISH1
23             Move to the target point
24         else if r is a terminal robot and Int(ℜ) has FINISH2-colored robots then
25             Follow PARTITIONTYPE_IV to find a target point
26             Change r.color to FINISH2
27             Move to the target point
28         else
29             No change in color and position
```

```
else if r.color = MONITOR then
    if r lies on S_r then
        Change r.color to OFF
    else
        if r finds any OFF-colored robot in Int(ℜ) then
            r does not change its color and position
        else
            Follow MonitorToFinal_inSqaure() to find the target point and move
             to it
else // r.color = FINISH or FINISH1 or FINISH2
    r terminates
```

their respective apex points are separated by either D_r or D_r^{opp}. Additionally, another monitor robot lying on a different side triangle other than ΔS_r cannot collide with r as its movement to its apex point is restricted within its respective side triangle. If r' is another monitor robot on S_r moving simultaneously with r, they move to their respective apex points by following two different sides of the convex hull $\mathcal{CH}^G$ similar to Lemma 7.3.5, resulting in no collision. □

Remark 7.4.3. When r finds itself on $(S_r, \frac{1}{8}S_r]$ with color MONITOR and does not see any OFF or FINISH or FINISH1 or FINISH2 robot in $Int(\Re)$, it accurately calculates Max_r.

Remark 7.4.4. Let r be the first robot to set its color to FINISH. Then, all the robots are positioned on one side of $\Re$ or the two opposite sides of $\Re$ before the movement of r with color FINISH, according to our strategy described in Case 1.4.1 and 1.4.2.

Remark 7.4.5. Let r be the first robot to set its color to FINISH1. Then, all the robots are positioned on the two adjacent sides of $\Re$ before the movement of r with color FINISH1, according to our strategy described in Case 1.4.3.

Remark 7.4.6. Let r be the first robot to set its color to FINISH2. Then, all the robots are distributed equally on the four sides of $\Re$ before the movement of r with color FINISH2, according to our strategy described in Case 1.4.4.

Lemma 7.4.7. *All robots form either a one-side configuration (where all robots gather on one side), or a two-side configuration (where robots gather on either two adjacent or two*

Algorithm 23: APEX_POINT_INSQAURE()

if *r does not find any robot on ΔS_r^{opp} nor ΔS_r^L nor ΔS_r^R* **then**
  r chooses the point of intersection of L_r and $\frac{1}{8}S_r$ as the apex point a_r
else
  $r_{Nbr_1} \longleftarrow$ The neighbor of r on $\mathcal{CH}_r$ that does not lie on S_r
  $r_{Nbr_2} \longleftarrow$ Another neighbor of r on $\mathcal{CH}_r$ that lies on S_r
  Compute the target point t_r on $\overline{rr_{Nbr_1}}$ such that
  $d(r,t_r) = \frac{1}{2}\min\{d(r,D_r), d(r,D_r^{opp}), d(r,\frac{1}{4}S_r), d(r,r_{Nbr_2})\}$

Algorithm 24: MONITORTOFINAL_INSQAURE()

if *r finds `FINISH`-colored robots and all of them lying on $\frac{1}{2}S_r$, or $\frac{1}{4}S_r$ or $\frac{3}{4}S_r$* **then**
  Follow MOVE_TO_FINALPOINT() to find a target point and set $r.color$ = `FINISH`
else if *r finds `FINISH1`-colored robots and all of them lying on $\frac{1}{3}S_r$ or $\frac{1}{3}S$ where $S \in \{S_r^L, S_r^R\}$* **then**
  Set $k = \frac{1}{3}$ and follow PARTITIONTYPE_III() to get the target point
  Change $r.color$ to `FINISH1`
else if *r sees `FINISH2`-colored robots and all of them lying on $\frac{1}{6}S_r$ or on $\frac{1}{6}S$ where $S \in \{S_r^{opp}, S_r^L S_r^R\}$* **then**
  Set $k = \frac{1}{6}$ and follow PARTITIONTYPE_IV() to get the target point
  Change $r.color$ to `FINISH2`
else if *(r finds a `FINISH`-colored robot, but not on $\frac{1}{2}S_r$, $\frac{1}{4}S_r$ or $\frac{3}{4}S_r$) $\vee$ (r finds a `FINISH1`-colored robot, but not on $\frac{1}{3}S_r$ or $\frac{1}{3}S$ where $S \in \{S_r^L, S_r^R\}$) $\vee$ (r finds a `FINISH2`-colored robot, but not on $\frac{1}{6}S_r$ or on $\frac{1}{6}S$ where $S \in \{S_r^{opp}, S_r^L S_r^R\}$)* **then**
  No change in color or position
else
  Follow FINALPOSITION_WITHMax_r()

opposite sides), or a four-side configuration (where robots are equally distributed on four sides).

Proof. After reaching all the robots on the boundary, a robot r will move to the apex point to calculate Max_r. r first checks whether there is any `OFF`-colored robot in $Int(\Re)$. If yes, it means that the `OFF`-colored robot is in its move phase towards one of the sides from its apex point. So r eventually finds no robot with color `OFF` in $Int(\Re)$. We assume that there is neither a `FINISH`, nor `FINISH1` nor a `FINISH2` robot in $Int(\Re)$. If such a robot exists, then the lemma holds from Remarks 7.4.4, 7.4.5 and 7.4.6. If Max_r has only one element (side),

r moves towards the side with maximum robots (follows from Case 1.4.6 and Case 1.4.9), which leads to gathering all robots on one side. If Max_r contains two sides, then r moves to one of those two sides in Max_r (follows from Case 1.4.6, 1.4.8, and 1.4.10), leading all the robots lying on two sides of $\Re$. If Max_r has three sides, our target is to move r to the side opposite to the side having minimum robots (follows from Case 1.4.5, 1.4.7, and 1.4.9) and eventually Max_r contains exactly one side, leading to the gathering of all robots on one side. Otherwise, Max_r contains all the sides. □

Remark 7.4.8. A `MONITOR`-colored robot r with no robot with color `OFF`, or `FINISH`, or `FINISH1`, or `FINISH2` eventually satisfies one of the following conditions.

(i) $Max_r = \{S_r\}$ and ΔS contains no robot where $S \in \{S_r^{opp}, S_r^L, S_r^R\}$ (Case 1.4.1, Section 7.4)

(ii) $Max_r = \{S_r, S_r^{opp}\}$ and no robots on ΔS for $S \in \{S_r^L, S_r^R\}$ (Case 1.4.2, Section 7.4)

(iii) $Max_r = \{S_r, S\}$ and both ΔS_r^{opp} and ΔS^{opp} contain no robot where $S \in \{S_r^L, S_r^R\}$ (Case 1.4.3, Section 7.4)

(iv) $Max_r = \{S_r, S_r^{opp}, S_r^L, S_r^R\}$ (Case 1.4.4, Section 7.4).

Lemma 7.4.9. *The robot r moving from the apex point to one of the sides of $\Re$ does not collide while moving.*

Algorithm 25: FINALPOSITION_WITHMax_r()

1 $Max_r \longleftarrow$ The set of all sides with maximum number of robots lying on their corresponding side triangle
2 **if** $Max_r = \{S_r\}$ *and* ΔS *has no robots for all* $S \in \{S_r^{opp}, S_r^L, S_r^R\}$ **then**
3 Follow PARTITIONTYPE_I_II() with $k = \frac{1}{2}$ to find the target point and change $r.color$ to `FINISH`
4 **else if** $Max_r = \{S_r, S_r^{opp}\}$ *and no robots on* ΔS *for all* $S \in \{S_r^L, S_r^R\}$ **then**
5 Follow PARTITIONTYPE_I_II() with $k = \frac{1}{4}$ to find the target point and change $r.color$ to `FINISH`
6 **else if** $Max_r = \{S_r, S\}$ *and both* ΔS_r^{opp} *and* ΔS^{opp} *contain no robot where* $S \in \{S_r^L, S_r^R\}$ **then**
7 Follow PARTITIONTYPE_III() with $k = \frac{1}{3}$ to find the target point and change $r.color$ to `FINISH1`
8 **else if** $Max_r = \{S_r, S_r^{opp}, S_r^L, S_r^R\}$ **then**
9 Follow PARTITIONTYPE_IV() with $k = \frac{1}{6}$ to find the target point and change $r.color$ to `FINISH2`

else
  if ($Max_r = \{S_r, S_r^{opp}, S\}$ *where* $S \in \{S_r^L, S_r^R\}$) $\vee$ ($Max_r = \{S\}$ *or* $\{S, S'\}$, *where* $S, S' \in \{S_r^L, S_r^R\}$ *with* $S \neq S'$) $\vee$ ($Max_r = \{S, S_r^{opp}\}$, *where* $S \in \{S_r^L, S_r^R\}$)) **then**
    Set S as the target side S_{Tgt}
  else if ($Max_r = \{S_r\}$ *or* $\{S_r, S_r^L, S_r^R\}$) $\vee$ ($Max_r = \{S, S_r\}$ *where* $S \in \{S_r^{opp}, S_r^L, S_r^R\}$) **then**
    Set S_r as the target side S_{Tgt}
  else // $Max_r = \{S_r^L, S_r^R, S_r^{opp}\}$ or $\{S_r^{opp}\}$)
    Set S_r^{opp} as the target side S_{Tgt}
  if $S_{Tgt} = S_r$ **then**
    $r_{Nbr_1} \longleftarrow$ The neighbour of r on $\mathcal{CH}_r$ not lying on S_r
    Select the point of intersection of $\overleftrightarrow{r r_{Nbr_1}}$ and S_r as the target point t_r
  else
    $L_r' \longleftarrow$ The line parallel to S_{Tgt} passing through r
    if *there is a* `MONITOR`*-colored robot lying on the half plane delimited by* L_r' *that contains* S_{Tgt} **then**
      r does not change its color or position
    else
      $L_r^{\perp} \longleftarrow$ The line passing through r and perpendicular to S_{Tgt}
      $p_{r_{Tgt}} \longleftarrow$ The point of intersection of $L_r^{\perp}$ and S_{Tgt}
      if $p_{r_{Tgt}}$ *has no robots on it* **then**
        Choose the point $p_{r_{Tgt}}$ as the target point
      else
        Choose a point t_r on S_{Tgt} as the target point such that $d(p_{r_{Tgt}}, t_r) = \frac{1}{4} \min_{r' \in \mathcal{V}_r} d(r', L_r^{\perp})$
      Change $r.color$ to `OFF`

Proof. Let S_{Tgt} be the target of r, calculated based on Max_r. If $S_{Tgt} = S_r$, r considers $\overleftrightarrow{r r_{Nbr_1}}$ to move back to S_r where r_{Nbr_1} is the neighbour of r on $\mathcal{CH}_r$, not lying on ΔS_r. By the similar argument presented in Lemma 7.4.2, we can argue that the movement of the robot r is free from collision. When $S_{Tgt} \neq S_r$, the movement of r is similar to the strategy MOVEMENT_TO_LONGESTSIDE which is collision-free by Lemma 7.3.3. □

Lemma 7.4.10. *A* `MONITOR`*-colored robot* r *on* $(S_r, \frac{1}{8}S_r]$ *satisfying one of the conditions in Remark 7.4.8 terminates to a unique partition of* $\Re$ *and the movement is collision-free.*

Algorithm 26: PARTITIONTYPE_I_II()

1 $c_r^1 \longleftarrow$ The number of robots on ΔS_r
2 **if** $k = \frac{1}{2}$ **then**
3 $SS_r \longleftarrow$ The side which lies on the half plane delimited by L_r where no other robot resides
4 Calculate the target point t_r on $\frac{1}{2}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$
5 **else**
6 $r_{Nbr} \longleftarrow$ The neighbour of r on $\mathcal{CH}_r$ whose nearest side is S_r^{opp}
7 $SS_r \longleftarrow$ The side which either lies on or intersects the half plane delimited by the line rr_{Nbr} where no other robot is present
8 Compute the target point t_r on $\frac{1}{4}S_r$ such that $d(t_r, SS_r) = \frac{len(S_r)}{2c_r^1}$

Proof. If r satisfies (i), it follows Type I partitioning, whereas if r satisfies (ii), it follows Type II partitioning. The movement strategy for both cases is similar to the algorithm RECTANGULAR_PARTITION (in Section 7.3). So, the statement of the above lemma holds from the Lemma 7.3.9. If r satisfies (iii), it follows Type III partitioning. It first calculates two points A and B on S_r, which are $\frac{len(S_r)}{c'}$ distance apart from e_P and e_Q respectively, where c' is the number of robots on ΔS_r including itself and e_P and e_Q are the common endpoints of S_r, S and S_r, S^{opp}, respectively. It then chooses two triangles $\mathcal{T}_1 = \Delta Ae_Pe_R$ and $\mathcal{T}_2 = \Delta Be_Qe_R$ where e_R is the vertex of the Type III partitioning. If another robot r' satisfies the conditions similar to those of r, it also chooses the above-mentioned two triangles. The choice of the apex point for r (as described in the strategy MONITORROBOT_MOVEMENT_SQUARE ensures that S^{opp} must lie on either the same or different half-planes delimited by L_r where other robots of ΔS_r lie. If S^{opp} lies on the half plane delimited by L_r where other robots on ΔS_r lie, r and r' choose the centroid of $\mathcal{T}_1$ and $\mathcal{T}_2$ as their final positions, respectively. Otherwise, r and r' choose the centroid of $\mathcal{T}_2$ and $\mathcal{T}_1$ as their final positions, respectively. So, r and r' select different triangles for their termination and their two final positions are $\frac{2}{3}len(S_r) - \frac{2len(S_r)}{3c'}$ distance apart from each other, and the movement is free from any collision. Thus, r and r' both move to distinct partitions, each having e_R as the vertex and with an area of $\frac{(len(S_r))^2}{2c'}$. If r satisfies (iv), it follows Type IV partitioning. It calculates A and B on S_r and chooses the centroid of one of the triangles ΔAe_PC and Be_QC as its final point. Based on similar arguments presented above, r moves to a unique partition without

collision. □

Lemma 7.4.11. *A* `MONITOR`*-colored robot r on $(S_r, \frac{1}{8}S_r]$ seeing either a* `FINISH` *or* `FINISH1` *or* `FINISH2`*-colored robot, terminates to a unique partition of $\Re$ and the movement is collision-free.*

Proof. If r sees a `FINISH`-colored robot r'' in $Int(\Re)$, it waits till r'' reach either on $\frac{1}{2}S_r$ or on $\frac{1}{4}S_r$ or on $\frac{3}{4}S_r$. Since r'' is in its move phase to its final position, r eventually finds all the `FINISH`-colored robots lying on $\frac{1}{2}S_r$ or $\frac{1}{4}S_r$ or $\frac{3}{4}S_r$. Then it moves to its final position in a partition by following a similar strategy as an `OFF`-colored robot, described in Case 2 of the algorithm RECTANGLE_PARTITION (in Section 7.3). In this case, the statement of the above lemma follows from the similar arguments in Lemma 7.3.10.

If r sees a `FINISH1`-colored robot in $Int(\Re)$, it waits for all `FINISH1`-colored robots to reach on either $\frac{1}{3}S_r$ or $\frac{1}{3}S$ where $S \in \{S_r^L, S_r^R\}$ and eventually sees all those robots reach their

Algorithm 27: PARTITIONTYPE_III()

```
1  if k = 1/3 and r.color = MONITOR then
2      c ⟵ The number of all FINISH1-colored robots on (1/3)S_r
3      c' ⟵ The number of robots lying on ΔS_r without the color FINISH1
4      if S = S_r^L then
5          A, B ⟵ The two points on S_r which are len(S_r)/(c+c') distance away from e_P and
             e_Q, respectively
6          if S^opp lies on the half plane delimited by L_r where other robots on ΔS_r lie
             then
7              Choose the triangle T = ΔAe_P e_R
8          else
9              Choose T = ΔBe_Q e_R
10     else
11         A, B ⟵ The two points on S_r which are len(S_r)/(c+c') distance away from e_Q and
             e_P, respectively
12         if S^opp lies on the half plane delimited by L_r where other robots on ΔS_r lie
             then
13             Choose the triangle T = ΔAe_Q e_X
14         else
15             Choose T = ΔBe_P e_X
16     Set the centroid of the triangle T as the target point
```

else // $k = \frac{1}{3}$ and $r.color =$ OFF
  if *($\frac{1}{3}S_r^L$ has a FINISH1-colored robot) $\vee$ (a robot lying on ΔS_r^L)* **then**
    Choose $S = S_r^L$
  else
    Choose $S = S_r^R$
  if *there is a FINISH1-colored robots not lying on $\frac{1}{3}S_r$ or on $\frac{1}{3}S$* **then**
    No change in color or position
  else
    Choose the point of intersection of S_r^{opp} and S^{opp} as the vertex v_r
    $D_r \longleftarrow$ Diagonal of $\Re$ passing through v_r
    if *there is a FINISH1-colored robot on $\frac{1}{3}S_r$* **then**
      $r' \longleftarrow$ Terminal FINISH1-colored robot on $\frac{1}{3}S_r$
      $bl_1, bl_2 \longleftarrow$ The length of the base of the triangle whose one side is D_r (resp. S^{opp}) and the centroid is at r'
    else
      $r' \longleftarrow$ Terminal FINISH1-colored robot on $\frac{1}{3}S$
      $bl_1, bl_2 \longleftarrow$ The length of the base of the triangle whose one side is D_r (resp. S_r^{opp}) and the centroid is at r'
    $bl = \min\{bl_1, bl_2\}$
    $u_1 \longleftarrow$ The point of intersection of $\frac{1}{3}S_r$ and the diagonal D_r
    $u_2 \longleftarrow$ The point of intersection $\frac{1}{3}S_r$ and the diagonal S^{opp}
    $e_P \longleftarrow$ The endpoint of D_r other than v_r
    $e_Q \longleftarrow$ The endpoint of S_r other than e_P
    if *S^{opp} lies in the half plane delimited by $\overleftrightarrow{r v_r}$, where the other robots of S_r reside* **then**
      $c \longleftarrow$ The number of FINISH1-colored robots on $\frac{1}{3}S_r$ starting from the robot $\frac{bl}{3}$ distance apart from u_1 towards u_2 such that two consecutive robots are $\frac{2bl}{3}$ distance away from each other
      $e_Y, e_Z \longleftarrow$ The points of S_r satisfying $d(e_Y, e_P) = c \cdot bl$ and $d(e_Z, e_P) = (c+1) \cdot bl$
    else
      $c \longleftarrow$ The number of FINISH1-colored robots on $\frac{1}{3}S_r$ starting from the robot $\frac{bl}{3}$ distance apart from u_2 towards u_1 such that two consecutive robots are $\frac{2bl}{3}$ distance away from each other
      $e_Y, e_Z \longleftarrow$ The points of S_r satisfying $d(e_Y, e_Q) = c \cdot bl$ and $d(e_Z, e_Q) = (c+1) \cdot bl$
    Calculate the target point t_r such that it is the centroid of $\Delta e_Y e_Z v_r$

final positions. Since the FINISH1-colored robot on $\frac{1}{3}S_r$ may reach to another side triangle than ΔS_r while executing its movement to the final position, we separately count c' (which is the number of the robot on ΔS_r without color FINISH1) and c (which is the number of FINISH1-colored robots on $\frac{1}{3}S_r$). Similar to Lemma 7.4.10, r selects A, B which are $\frac{len(S_r)}{c+c'}$ distance apart from e_P and e_Q, respectively and the centroid of one of the two triangles ΔAe_Pe_R and ΔBe_Qe_R as its final position. By Lemma 7.4.10, this final position of r lies in a unique partition of $\Re$ and the movement is collision-free. Similarly, we can prove that if r sees a FINISH2-colored robot, it terminates to a unique partition without collision. □

Lemma 7.4.12. *An OFF-colored robot r on S_r terminates to a unique partition of $\Re$, and the movement is collision-free.*

Proof. By Remark 7.4.8, there exists a terminal robot r'' on one of the sides, which moves to its apex point on $(S_r, \frac{1}{8}S_r]$ with color MONITOR and satisfies one of the four conditions as described in the remark. By Lemma 7.4.10 and 7.4.11, r'' reaches to a unique partition of $\Re$ with color either FINISH or FINISH1 or FINISH2. Let us assume that r is a terminal robot of S_r. If $r''.color$ = FINISH, r follows a similar strategy as presented in Case 2 of RECTANGLE_PARTITION. By Lemma 7.3.10, r terminates to a unique partition without collision.

Let $r''.color$ = FINISH1. So, r understands that it needs to follow Type III partitioning. In this case, r must have seen some robots on ΔS ($S \neq S_r$) and calculates the distance bl, the two points e_Y and e_Z and the number c, as described in Case 3 (Section 7.4). It then selects two points e_Y and e_Z and moves to the centroid of the triangle $\Delta e_Y e_R e_Z$. If r' is another terminal robot on S_r, which simultaneously gets active with r, then it selects the centroid of the different triangle than that of r, as described in Case 3 (in Section 7.4). The choice of the triangles ensures that the two robot r and r' terminate at t_r and $t_{r'}$. Let u_1 be the point of intersection of $\frac{1}{3}S_r$ and D_r and u_2 be the point of intersection of $\frac{1}{3}S_r$ and S^{opp}. If we consider that k (and k') is the number of FINISH1-colored robots on $\frac{1}{3}S_r$ starting from the robot $\frac{bl}{3}$ distance apart from u_1 (u_2) towards u_2 (u_1) such that two consecutive robots are $\frac{2bl}{3}$ distance apart from each other, where $d(t_r, t_{r'}) = \frac{2}{3}len(S_r) - \frac{2}{3}(k+k'+1)bl > 0$ as $len(S_r) \geq (k+k')bl + 2bl$. Hence, r and r' terminate at distinct partitions of the region $\Re$. The movement of the two robots is free from collision, by the similar arguments presented

in Lemma 7.3.9.

For $r''.color$ = FINISH2, we similarly prove that r terminates at a unique partition. □

Theorem 7.4.13. *Algorithm* SQUARE_PARTITION *solves uniform partitioning for the square region in $O(N)$ epochs without collision.*

Proof. By Lemma 7.4.1, starting from any initial configuration, all the interior and corner robots become boundary robots in $\Re$. After this, all the robots eventually form either a one-side, two-side, or four-side configuration by Lemma 7.4.7. Thereafter, monitor robots terminate to distinct partitions of $\Re$ by Lemma 7.4.10, 7.4.11 and all the other robots terminate to distinct partitions by Lemma 7.3.10. Hence, SQUARE_PARTITION solves uniform partitioning for the square region.

In the worst case, all robots may lie on a straight line on $Int(\Re)$, in which it takes $O(N)$ epochs for all robots to reach the boundary. After moving to the boundary, it takes one epoch for monitor robots to move to apex points. In the worst case, there may be 8 MONITOR-colored robots in $Int(\Re)$, each lying on a different line parallel to S_r and r is the farthest robot from S_r^{opp}. If $Max_r = \{S_r^{opp}\}$, r needs to wait for seven epochs to reach S_r^{opp} because the other seven robots move to S_r^{opp} one by one. Then r needs one more epoch to reach S_r^{opp}. Thus, r takes nine ($O(1)$) epochs so that the whole configuration of the robots becomes a one-side configuration. Hence, reaching a one-side configuration takes $O(N)$ epoch. Similarly, we can prove that if Max_r contains more than one element, we can reach a two-side or a four-side configuration. After this, if r is again a monitor robot, it needs one epoch to reach its apex point, waits for another seven epochs for other seven MONITOR-colored robots to reach their respective final positions, and needs one more epoch to reach its final position in a partition. Thus, r terminates in nine epochs. After seeing r at its final position, all the other OFF-colored robots on the sides position themselves at their respective final positions one by one in $O(N)$ epochs, as each of them needs $O(1)$ epoch to reach their final position. Hence, SQUARE_PARTITION solves the uniform partitioning problem in $O(N)$ epochs.

The algorithm is free from collision by Lemma 7.4.1, 7.4.2, 7.4.9, 7.4.10, 7.4.11, and 7.4.12. □

Algorithm 28: PARTITIONTYPE_IV()

if $k = \frac{1}{6}$ *and* $r.color =$ `MONITOR` **then**
  $A, B \longleftarrow$ The two points on S_r which are $\frac{len(S_r)}{|S_r|}$ distance away from e_P and e_Q, respectively
  if e_P *and all the other robots on* ΔS_r *lie on the different half plane delimited by* L_r **then**
    Choose the triangle $\mathcal{T} = \Delta A e_P C$
  else
    Choose $\mathcal{T} = \Delta B e_Q C$
  r chooses the centroid of $\mathcal{T}$ as the target
else // $k = \frac{1}{6}$ and $r.color =$ `OFF`
  if *there is a* `FINISH2`*-colored robot not lying on* $\frac{1}{6}S$ *for* $S \in \{S_r, S_r^{opp}, S_r^L, S_r^R\}$ **then**
    No change in color and position
  else
    $r' \longleftarrow$ A `FINISH2`-colored terminal robot on $\frac{1}{6}S_{r'}$
    $e^1_{S_{r'}}, e^2_{S_{r'}} \longleftarrow$ Two endpoints of $S_{r'}$
    $D_r, D_r^{opp} \longleftarrow$ Two diagonals of $\Re$ passing thorugh e_P and e_Q respectively
    $bl_1 \longleftarrow$ Length of the base of the triangle whose centroid is r' and two vertices are C and $e^1_{S_{r'}}$
    $bl_2 \longleftarrow$ Length of the base of the triangle whose centroid is r' and two vertices are C and $e^2_{S_{r'}}$
    $bl = \min\{bl_1, bl_2\}$
    $u_1 \longleftarrow$ The point of intersection of $\frac{1}{6}S_r$ and D_r
    $u_2 \longleftarrow$ The point of intersection of $\frac{1}{6}S_r$ and D_r^{opp}
    if $\overline{Ce_Q}$ *lies on the half plane delimited by* $\overleftrightarrow{rC}$ *where other robots of* S_r *reside* **then**
      $c \longleftarrow$ The number of `FINISH2`-colored robots on $\frac{1}{6}S_r$ starting from the robot $bl/3$ distance away from u_1 towards u_2 such that two consecutive robots are $2bl/3$ distance apart from each other
      $e_Y, e_Z \longleftarrow$ The points on S_r satisfying $d(e_P, e_Y) = c \cdot bl$ and $d(e_P, e_Z) = (c+1) \cdot bl$
      Choose the centroid of $\Delta C e_Y e_Z$ as the target point
    else
      $c \longleftarrow$ The number of `FINISH2`-colored robots on $\frac{1}{6}S_r$ starting from the robot $bl/3$ distance away from u_2 towards u_1 such that two consecutive robots are $2bl/3$ distance apart from each other
      $e_Y, e_Z \longleftarrow$ The points on S_r satisfying $d(e_Q, e_Z) = c \cdot bl$ and $d(e_Q, e_Y) = (c+1) \cdot bl$
      Choose the centroid of $\Delta C e_Y e_Z$ as the target point

7.5 Algorithm for a Circular Region

This section considers the region $\Re$ as a circle of radius rad. We follow the Type V partitioning for circular regions. Our target in this process is to form a regular N-gon inscribed

in the region using the robots. In this section, we discuss two techniques to achieve the target.

We can use an algorithm for the widely popular *Uniform Circle Formation* (UCF) problem, presented by Feletti et al [32]. They proposed a $O(\log(N))$ algorithm for the ASYNC setting. The algorithm relies on the mutual visibility algorithm, proposed by Sharma et al [68], where the robots first position themselves on the vertices of a convex hull. Taking advantage of the mutual visibility, robots then place themselves on the smallest enclosing circle, which later gets translated to a uniform circle configuration using a subroutine, named as `Uniform Transformation` that runs in $O(\log(N))$ epochs. The mutual visibility algorithm in [68] requires 47 colors to achieve the goal in $O(1)$ epochs. Regarding the uniform partitioning problem in a circular region, we can achieve the same goal of repositioning the robots on the boundary of $\Re$ using no colors in $O(1)$ epochs. From the application-oriented point of view, we can take advantage of the bounded region where the robots can detect the boundary of the region. Robots can be placed on the boundary using the movement strategy MOVE_TO_BOUNDARY (which is described later in this section), maintaining the initial color `OFF` throughout the strategy. The detailed steps of the technique `Uniform Transformation` are not repeated here to keep the section concise.

7.5.1 Description of the Algorithm

Initially, all robots are with color `OFF`. Before going into the details of the algorithm, we define the following notations that we will use in the future to comprehend the algorithm better.

Notations: For a robot r,

- O is the center of $\Re$. p_r and p_r^{opp} are the two diametrically opposite points of intersection of the line $\overleftrightarrow{rO}$ and the boundary of $\Re$, out of which p_r is the nearest to r.

- $\widehat{AB}$ represents the segment of the circumference of $\Re$, starting from the point A to B such that the segment has no robot except on the endpoints. In the case of $\widehat{AB}$

having no robots in both clockwise and counter-clockwise directions, $\overset{\frown}{AB}$ is chosen to be the arc with the shortest arc length.

- $alen(AB)$ is the arc length of $\overset{\frown}{AB}$.

Strategy MOVE_TO_BOUNDARY: If r is a boundary robot and there is at least one OFF-colored interior robot visible to it, r does not move or change its current color. If r is an interior robot and lies on the center O, it waits till all other interior robots with color OFF reach the boundary of $\Re$. When there are no other interior robots left, r does not change its color and moves to any target point t_r on the boundary that does not contain a robot. Otherwise (r is an interior robot not lying on O), r considers the point p_r and checks whether the point p_r is visible or not. If p_r is visible and no robot lies on it, r simply moves to p_r with current color OFF. On the other hand, r computes the set $\mathcal{V}_r$, that consists of all visible robots to r not lying on the line $\overleftrightarrow{rO}$. We identify two sub-cases.

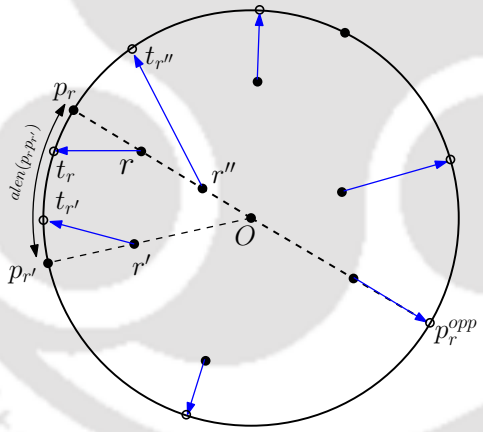


Figure 7.25: The target point t_r of r depends on the distance of r from the point p_r

- $\mathcal{V}_r$ **is non-empty:** r calculates $d_r = \frac{1}{4}\min\{alen(p_r p_{r'}) |\ r' \in \mathcal{V}_r$ and $\overset{\frown}{p_r p_{r'}}$ is defined$\}$, as shown in Fig. 7.25.

- $\mathcal{V}_r$ **is empty:** It happens when all the robots lie on one line passing through O. Here, r calculates $d_r = \frac{1}{4} alen(p_r p_r^{opp})$.

Finally, r finds the target point t_r such that $alen(t_r p_r) = \frac{d_r}{rad} d(r, p_r)$ and moves to t_r with the current color OFF, as illustrated in Fig 7.25.

Observe that the above strategy does not require any change in color to displace the robots on the boundary from the interior of $\Re$. From now on, we can use the `Uniform Transformation` sub-problem proposed in [32] to achieve a configuration of the robots such that $alen(r_i r_j) = \frac{2\pi \cdot rad}{N}$ for any two neighbouring robots r_i and r_j in $O(\log(N))$ epochs under ASYNC setting. The above technique uses more than 17 colours to achieve a uniform circular configuration.

Although the number of colors required in `Uniform Transformation` is a constant, if we turn our focus on reducing the number of colors, we can propose another technique to achieve the same goal of repositioning the robots in such a way that the neighbouring robots are $\frac{2\pi \cdot rad}{N}$ distance apart from each other. Observe that when all robots are on the boundary of $\Re$, they all can see each other, as no three robots are collinear. We will turn this to our advantage. We now define a *cluster* that will be useful for the rest of the algorithm.

Definition 7.5.1. (Cluster) A sequence of robots $Cl = \{r_1, r_2, \cdots, r_k\}$ $(k \geq 1)$ lying on the boundary of $\Re$, is called a cluster if the following conditions hold. (i) r_i and r_{i+1} are consecutive robots on the boundary. (ii) $alen(r_i r_{i+1}) = \frac{2\pi \cdot rad}{N}$. (iii) $alen(r_1 r_1^{Nbr}), alen(r_k r_k^{Nbr}) \neq \frac{2\pi \cdot rad}{N}$, where r_1^{Nbr} and r_k^{Nbr} are the neighbors of r_1 and r_k respectively such that $r_1^{Nbr}, r_k^{Nbr} \notin Cl$.

For convenience, we denote $(2\pi \cdot rad)/N$ by s. Any robot can detect its own cluster, as it is able to see all the robots on the boundary. A robot r is called the *head* of a cluster Cl, if $alen(r r^{Nbr_1}) = s$ and $alen(r r^{Nbr_2}) > s$ for two neighbours r^{Nbr_1}, r^{Nbr_2} of r such that $r^{Nbr_1} \in Cl$ and $r^{Nbr_2} \notin Cl$. The robot r is called *tail* of the cluster Cl, when $alen(r r^{Nbr_1}) = s$ for $r^{Nbr_1} \in Cl$ and $alen(r r^{Nbr_2}) < s$ for $r^{Nbr_2} \notin Cl$. It is possible that the head or tail of a cluster does not exist. Moreover, a cluster can be made of a single robot r. In such cases, if $alen(r r^{Nbr_1}) > s$ and $alen(r r^{Nbr_2}) < s$, r itself is called the head and tail of the cluster. An example is given in Fig. 7.26. Note that a cluster having no head and tail contains all the robots in $\Re$.

Definition 7.5.2. (Eligible Cluster) A cluster is eligible for movement if it satisfies the following two conditions. (i) The cluster has both head and tail robots. (ii) All the robots in that cluster are with color `OFF`.

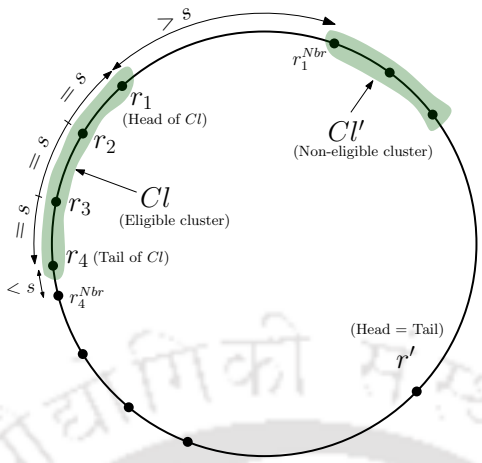


Figure 7.26: Illustrating eligible clusters with head and tail

In other words, clusters that only have heads or tails are not eligible for movement. In an eligible cluster, the head initiates the movement along the perimeter of $\Re$. By movement of a cluster $Cl = \{r_1, r_2, \cdots, r_k\}$, we mean that all the robots in Cl move in the direction of the head sequentially. Note that the robots do not have identifiers. We use $r_1, r_2, \cdots, r_k$ for better comprehension. The movement of a cluster is explained below.

Movement of an Cluster Cl: Let us consider that r_1 is the head, r_k is the tail of Cl and r_1' is neighbor of r_1, not in Cl. Let Cl' be the cluster of r_1', as shown in Fig. 7.27. When r_1 gets activated with color `OFF` with all other robots in Cl having color `OFF`, and finds Cl as an eligible cluster, it changes its color to `HEAD` and waits till all robots in its cluster change their color either to `MID` or `TAIL`. When r_k finds r_1 in its own cluster Cl with color `HEAD` and determines that all the other robots in Cl lies between r_1 and r_k on the boundary of $\Re$, it sets its color to `TAIL`. All other robots in Cl change their color to `MID` after seeing r_1 with the color `HEAD`. After this, r_1 determines the eligibility of Cl'. If any robot lying on the boundary sees another interior robot with color other than `OFF`, it waits as the robots are in their move phase toward their target points through the interior of $\Re$. Otherwise, we have two cases.

Case 1 (Cl' is eligible and $r_1'.color$ = `OFF` or `HEAD`): In this case, our strategy is to move each head of Cl and Cl' a distance $\frac{1}{2}(alen(r_1 r_1') - s)$ towards each other so that the distance between them becomes s after movement. All other robots of the respective clusters sequentially move in the direction of the head's movement. The process is as follows and

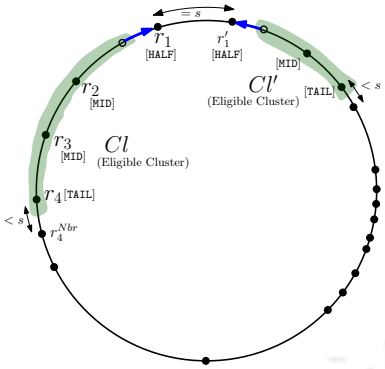


Figure 7.27: Movement of the heads of the eligible clusters

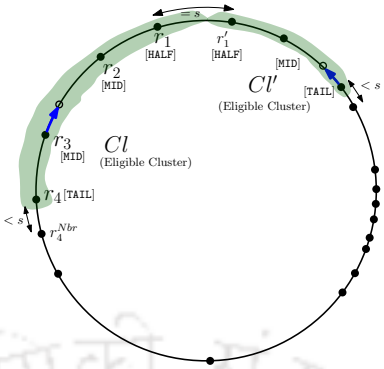


Figure 7.28: Movements of the robots with color MID & TAIL

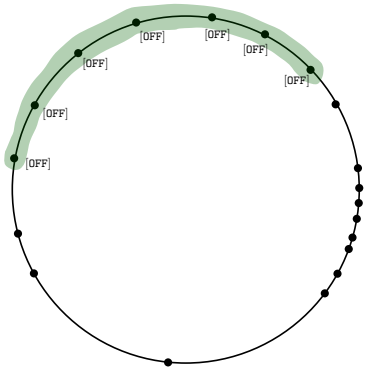


Figure 7.29: Two clusters are merged with each other after the movement

illustrated in Fig. 7.27. The head r_1 changes its color to MOVE-H and moves towards r'_1 to a point t_{r_1} such that $alen(t_{r_1}r_1) = \frac{1}{2}(alen(r_1r'_1) - s)$. After getting activated with color MOVE-H, it changes its color to HALF. It waits till the tail of the cluster (if it exists) r_k sets its color to OFF. At this moment, r_k and all MID-colored robots do not change their position or color. After seeing r_1 on the boundary of $\Re$ with the color HALF and $alen(r_1, r_2) > s$, r_2 starts moving towards r_1 without changing its current color to a point t_{r_2} on the boundary such that $alen(t_{r_2}r_2) = alen(r_1r_2) - s$. Other robots in Cl except r_2 remain in place at this time. Similarly, for any robot r_i $(3 \leq i \leq k)$, if r_{i-1} lies on the boundary of $\Re$ with $alen(r_{i-1}r_i) > s$ and $alen(r_{i-1}r_{i-2}) = s$, r_i moves to t_{r_i} towards r_{i-1} with its current color such that $alen(t_{r_i}r_i) = alen(r_{i-1}r_i) - s$, as shown in Fig. 7.28. After the movement, r_k finds the cluster Cl with its head r_1 having color HALF. It changes its color to OFF. The MID-colored robots in Cl change their color to OFF after seeing the tail with color OFF. The head r_1 changes its current color to OFF from HALF as illustrated Fig. 7.29, when both of its neighbours are s arc-length away from it and all the robots in its cluster are with color OFF.

Case 2 (Cl' is eligible and $r'_1.color$ = HALF) or (Cl' is non-eligible): Here, we move the head r_1 with color FULL towards r'_1 to a point on the boundary t_{r_1} such that $alen(t_{r_1}r_1) = (alen(r_1r'_1) - s)$, as shown in Fig. 7.30. The rest of the strategy is the same as the previous case (Case 1) for other robots in the cluster. Finally, Cl gets merged with Cl', as shown in Fig. 7.31. When r_1 is with color FULL and finds all the robots in Cl with color OFF, it sets its color to OFF.

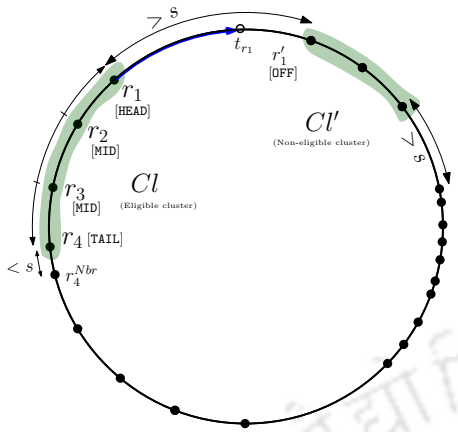


Figure 7.30: Cl is an eligible cluster and Cl' is the non-eligible cluster

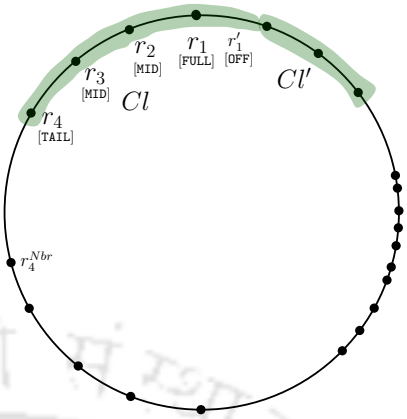


Figure 7.31: The two clusters got merged after the movement of Cl

When any robot r finds itself with color OFF and all robots are in one cluster, i.e., every two consecutive robots on the boundary are at s distance apart from each other, it moves to the point t_r on the line segment $\overline{rO}$ such that $d(r,t_r) = \frac{1}{2}d(r,O)$ with color FINISH. Even if a boundary robot r sees a FINISH-colored robot in $Int(\Re)$, it follows the same strategy. In all other cases, r maintains status-quo.

7.5.2 Analysis of the Algorithm

In this subsection, we discuss the correctness and the time complexity of the above-mentioned algorithm for circular regions. We also prove that the robots do not meet collision during any movement.

Lemma 7.5.3. *Any interior robot in $\Re$ with color OFF moves to its boundary without collision.*

Proof. Let us consider two robots, r and r', lying in $Int(\Re)$. We can identify two cases here. **Case 1 (r' lies on $\overleftrightarrow{rO}$):** In this case, we investigate two sub-cases. When $p_r = p_{r'}$, it signifies that r and r' lie on $\overline{Op_r}$. Without loss of generality, let us assume that r is farther from p_r than r'. So, $d(p_r,r') < d(p_r,r)$. We further assume that p_r has a robot on it. If both of them get activated simultaneously, they find each other in the same radius. Both of them calculate d_r and $d_{r'}$ respectively, such that $d_r = d_{r'}$. If they choose t_r and $t_{r'}$ as their respective target points on the boundary, where $alen(t_r p_r) = \frac{d_r}{rad}d(r,p_r) > \frac{d_{r'}}{rad}d(r',p_{r'}) =$

$alen(t_{r'}p_{r'}) = alen(t_{r'}, p_r)$. We also have $d(r, p_r) > d(r', p_r)$. Both of the above inequalities imply that the line passing through r_{mid} (the midpoint of $\overline{rr'}$) and $t_{r_{mid}}$ (the midpoint of $\widehat{t_r t_{r'}}$) separates the paths of r and r' towards their respective target points. In case of $p_r \neq p_{r'}$, if we assume that $\mathcal{V}_r$ is non-empty with $r'' \in \mathcal{V}_r$, then the line $\overleftrightarrow{r''O}$ separates the two target points t_r and $t_{r''}$. If $\mathcal{V}_r$ is empty, the line $\overleftrightarrow{p_{r_{mid}}O}$ separates $\overline{rt_r}$ and $\overline{r't_{r'}}$ where $p_{r_{mid}}$ is the midpoint of $\widehat{p_r p_{r'}}$. So, the movement paths of the two robots r and r' do not cross each other. Hence, the movements of r and r' are free from collision.

Case 2 (r' does not lie on $\overleftrightarrow{rO}$): In this case, $p_r \neq p_{r'}$. By the similar argument presented above, the movements of the robots are free from collision. □

Lemma 7.5.4. *There always exists an eligible cluster or all the robots are in the same cluster.*

Proof. If all the robots are in the same cluster, the statement of the lemma follows trivially.

Algorithm 29: CIRCLE_PARTITION

```
1  if r.color = OFF then
2      if r is an interior robot then
3          Follow MOVETOBOUNDARY_INCIRCLE() to find a target point on the
             boundary
4          Move to the target point with the color OFF
5      else if r is a boundary robot with at least one OFF-colored robot in Int(ℜ) then
6          r does not change its color or position
7      else // r is a boundary robot with no OFF-colored robot in Int(ℜ)
8          Cl ⟵ The cluster of r
9          if Cl is eligible then
10             if r is the head of Cl then
11                 Change r.color to HEAD
12             else if r is the tail of Cl then
13                 Change r.color to TAIL
14             else
15                 Change r.color to MID
16         else
17             r does not change its position or color
18 else if r.color = MOVE-H then
19     Change r.color to HALF
```

else if $r.color = HEAD$ **then**
  $r' \longleftarrow$ Head of the neighbouring cluster Cl' and the neighbour of r
  if *($r'.color = HALF$ and Cl' is eligible) $\vee$ (Cl' is non-eligible)* **then**
    Change $r.color$ to FULL
    Move to a point t_r on $\overset{\frown}{rr'}$ such that $alen(r\,t_r) = alen(r\,r') - s$
  else // $r'.color =$ `OFF or HEAD with` Cl' `being eligible`
    Change $r.color$ to `MOVE-H`
    Move to a point t_r on $\overset{\frown}{rr'}$ such that $alen(r\,t_r) = \frac{1}{2}(alen(r\,r') - s)$
else if $r.color = MID$ **then**
  $r_{Nbr_1}, r_{Nbr_2} \longleftarrow$ Neighbours of r in Cl with r_{Nbr_1} nearer to the head of Cl than r_{Nbr_2}
  if $alen(r\,r_{Nbr_1}) = alen(r\,r_{Nbr_2}) = s$ **then**
    r does not change its color or position
  else
    r moves to a point t_r on $\overset{\frown}{rr_{Nbr_1}}$ such that $alen(r\,t_r) = alen(r\,r_{Nbr_1}) - s$
else if $r.color = TAIL$ **then**
  $r_{Nbr_1} \longleftarrow$ Neighbour of r in Cl
  if $alen(r\,r_{Nbr_1}) = s$ **then**
    r does not change its color or position
  else
    Move to a point t_r on $\overset{\frown}{rr_{Nbr_1}}$ such that $alen(r\,t_r) = alen(r\,r_{Nbr_1}) - s$ with color `TAIL`
else if $r.color = HALF$ *or* $FULL$ **then**
  $r_{Nbr_1}, r_{Nbr_2} \longleftarrow$ The two neighbours of r
  if $alen(r\,r_{Nbr_1}) = alen(r\,r_{Nbr_2}) = s$ **then**
    Change $r.color$ to `OFF`
  else
    r does not change its color or position
else // $r.color =$ `FINSH and` r `lies on` $\frac{1}{2}S_r$ `or` $\frac{1}{4}S_r$
  r terminates.

Let us assume that there are at least two clusters. If all the clusters are non-eligible, then all the arc-length between two consecutive clusters is always either greater or lesser than s, which is a contradiction to the fact that the sum of the distances between two consecutive robots on the boundary is $2\pi \cdot rad = s \cdot N$. □

Lemma 7.5.5. *After all the robots in a cluster complete their movement, they remain as a*

cluster.

Proof. For an eligible cluster $Cl = \{r_1, r_2, \cdots r_k\}$, the head r_1 executes its movement towards another cluster. After that, $alen(r_1 r_2) > s$, since r_2 is a member of Cl. When r_1 reaches its final position and changes its color to either `HALF` or `FULL`, r_2 moves to a point t_{r_2} towards r_1 such that $alen(r_1 t_{r_2}) = s$. All other robots in Cl sequentially execute the same process which leads to the completion of one movement of the whole cluster. Hence the statement follows. □

Algorithm 30: MOVETOBOUNDARY_INCIRCLE()

```
1  if r lies on O and there is at least interior robot other than r then
2      r does not change its color or position
3  else if r lies in Int(ℜ), but not on O then
4      p_r ⟵ The point of intersection of the line r⟷O and the boundary of ℜ
5      p_r^opp ⟵ The point diametrically opposite to p_r on the boundary of ℜ
6      if p_r does not have any robot on it then
7          r chooses p_r as its target point
8      else
9          V_r ⟵ The set of all visible robots to r not lying on the line r⟷O
10         if V_r is non-empty then
11             Calculate d_r = (1/4) min{alen(p_r p_r') | r' ∈ V_▽ and arc(p_r p_r') is defined}
12         else
13             Calculate d_r = (1/4) alen(p_r p_r^opp)
14         r finds a target point on the boundary t_r such that alen(t_r p_r) = (d_r / rad) d(r, p_r)
```

Lemma 7.5.6. *Let k ($< N$) be the length of an eligible cluster Cl. The length of Cl gets increased at least by one without collision in $O(k)$ epochs.*

Proof. First the head of an eligible cluster having all robots with color `OFF`, changes its color to `HEAD`. Upon seeing this, all other robots in the cluster change their color to either `MID` or `TAIL`. This operation takes total 2 epochs. For a cluster $Cl = \{r_1, r_2, \cdots r_k\}$ with r_1 as head and r_k as tail, r_1 moves first in the direction of the another cluster. Then `MID`-colored robots move sequentially (first r_2 moves in the direction of r_1, then r_3 and at last the tail r_k). This step takes $O(k)$ epochs. If the cluster Cl moves towards another cluster Cl', the heads of

the two clusters become s arc-length apart in just one epoch (either both the heads move towards each other or one head moves towards the other). After the movement of the whole cluster, the tail of that cluster changes its color to `OFF`, then the `MID`-colored robots change their color to `OFF` and finally the head changes its color to `OFF` when both of its neighbors are s arc-length apart from it and all other robots in its cluster are with color `OFF`. This process takes 3 epochs. So, the movement of a cluster of length k takes $O(k)$ epochs. After the movement, both the neighbor of r_1 are s distance apart from each other. Therefore the length of the cluster gets increased at least by one. Hence the proof. □

Theorem 7.5.7. *Our algorithm solves uniform partitioning for the circular region in $O(N^2)$ epochs.*

Proof. The robots lying on $Int(\Re)$, moves to the boundary. The process of moving all the interior robots to the boundary takes $O(1)$ epochs. In the worst case, it is possible that the length of an eligible clusters gets increased only by one in every movement of the cluster. Lemma 7.5.6 proves that the movement of an eligible cluster takes $O(k)$ epochs, where k is the length of the cluster. So, it takes $\sum_{k=1}^{N-1} O(k) \approx O(N^2)$ epochs for all robots to become one cluster. Finally, all robots moves to their final positions in one epochs. Thus, our algorithm takes overall $O(N^2)$ epochs to partition the region. □

7.6 Conclusion

We studied the distributed version of uniform partitioning of a bounded region using mobile robots. The problem becomes interesting and challenging due to the robot model that is considered in this chapter. The robots are opaque and do not have enough memory to store the past information except for a persistent memory in form of a light. We solved this problem when the region is either a rectangle, a square or a circle. The reason behind this, is the application oriented point of view, as the regions, we deal with in our daily life, are known geometric figures. We believe that the problem can be extended to other convex regions under the same model. Also, the problem can even be studied when some of the robots become faulty.

Chapter 8

Conclusion and Future Works

In this thesis, we have studied different problems under the classical LCM Robot model. We have considered luminous robots throughout the thesis to explore all the problems. The primary reason behind using this model is its scalability and feasibility in solving problems in the distributed setup. Our goal has been to strike a balance between the capabilities of the robots and the solvability of specific problems. Many problems, such as pattern formation, exploration, gathering, etc., have been studied under this robot model. In Chapter 4, we have addressed a variation of graph exploration that requires the robots to settle on the vertices of a port-labelled graph such that the set of occupied vertices forms an MIS of the graph. The graph has some specific vertices called Doors through which robots enter the graph and fill out an MIS of the graph. Finding the MIS of a connected graph is a natural extension of the literature on filling problems.

In Chapter 5, we have considered a fundamental problem in the field of swarm robotics, which is called the mutual visibility problem. We have introduced the concept of inaccuracy in the movement of the robots, which makes the process of making any three robots non-collinear more difficult. The problem is classically considered with a swarm of luminous robots. The two major components of the problem are the obliviousness and the opaqueness of the robots. We have achieved the optimal number of colors under the SSYNC setting for the problem. The algorithm for the ASYNC setting uses 3 colors to achieve the goal. We have also studied the robustness of the model considering the mobil-

ity failure of the robots along with inaccuracy. In this case, we have presented an algorithm that solves the problem using a constant number of colors. All the algorithms under this model need $O(N)$ epochs to achieve mutual visibility.

In Chapter 6, we have reconsidered the problem of mutual visibility and investigated it from the aspect of fault tolerance using the robots having no axis agreement. We have shown that the problem is not solvable for any initial configuration of the robots when the robots do not agree on any coordinate system or orientation. More specifically, we have proved the impossibility in the case of a symmetric initial configuration. Besides this initial configuration, we have proposed a collision-free algorithm that tolerates any number of faulty robots and requires $O(N^2)$ rounds in the FSYNC setting. The algorithm uses 21 colors to achieve mutual visibility. We have further explored a simpler algorithm that tolerates a single faulty robot using only 2 colors in the FSYNC setting and 5 colors in the ASYNC setting.

Chapter 7 deals with a relatively new problem, called the uniform partitioning problem, which has been explored with ASYNC opaque luminous robots. The problem has a wide range of applications in the real world. Our target is partitioning a given bounded region into equal sub-regions where every sub-region contains exactly one robot. We have considered three standard geometric shapes as regions: rectangle, square and circle. The algorithm proposed for the rectangular region is optimal with respect to the number of colors, whereas the algorithm for the square region requires 5 colors to achieve the goal. These two algorithms run in $O(N)$ epochs. Further, we have studied a trade-off between the number of colors and the time complexity for the circular region.

8.0.1 Future Plan

We believe that the thesis opens up many interesting directions. The solution for the MIS Filling problem for the multiple Doors is limited to the SSYNC setting. We want to extend it to the most general ASYNC setting. The model can also be applied to different graph problems, such as dominating sets, vertex cover, etc.

Secondly, the problem of mutual visibility still needs new solutions for more realistic

fat robots. We want to introduce inaccuracy in the fat robot model to solve the mutual visibility problem. Our target is not just restricted to the mutual visibility problem but also to explore the pattern formation problem. Moreover, the mutual visibility under the ASYNC setting for the robots having no axis agreement is still open which is the weakest among all in the point robot model.

Third, we want to extend the uniform partitioning for any bounded convex regions. We believe that the problem of uniform partitioning can even be extended to faulty robot models, where it would be interesting to see the feasibility of the problem under such conditions. We also want to see whether the problem can be solved using robots without lights.

Bibliography

[1] José Joaquín Acevedo, Begoña C Arrue, Ivan Maza, and Anibal Ollero. A distributed algorithm for area partitioning in grid-shape and vector-shape configurations with multiple aerial robots. *Journal of Intelligent & Robotic Systems*, 84:543–557, 2016.

[2] Ranendu Adhikary, Kaustav Bose, Manash Kumar Kundu, and Buddhadeb Sau. Mutual visibility by asynchronous robots on infinite grid. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS '18*, pages 83–101. Springer, 2018.

[3] Aisha Aljohani, Pavan Poudel, and Gokarna Sharma. Fault-tolerant complete visibility for asynchronous robots with lights under one-axis agreement. In *International Workshop on Algorithms and Computation, WALCOM '18*, pages 169–182. Springer, 2018.

[4] Aisha Aljohani and Gokarna Sharma. Complete visibility for mobile agents with lights tolerating a faulty agent. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW '17*, pages 834–843. IEEE, 2017.

[5] John Augustine and William K. Moses. Dispersion of mobile robots: A study of memory-time trade-offs. In *Proceedings of the 19th International Conference on Distributed Computing and Networking, ICDCN '18*, New York, NY, USA, 2018. Association for Computing Machinery.

[6] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Deployment of asynchronous robotic sensors in unknown orthogonal environments. In Sándor P. Fekete,

editor, *Algorithmic Aspects of Wireless Sensor Networks ALGOSENSORS '08*, pages 125–140, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[7] Eduardo Mesa Barrameda, Shantanu Das, and Nicola Santoro. Uniform dispersal of asynchronous finite-state mobile robots in presence of holes. In Paola Flocchini, Jie Gao, Evangelos Kranakis, and Friedhelm Meyer auf der Heide, editors, *Algorithms for Sensor Systems, ALGOSENSORS '14*, pages 228–243, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[8] Lali Barriere, Paola Flocchini, Eduardo Mesa-Barrameda, and Nicola Santoro. Uniform scattering of autonomous mobile robots in a grid. *International Journal of Foundations of Computer Science*, 22(03):679–697, 2011.

[9] Subhash Bhagat, Paulomi Dey, and Rajarshi Ray. Collision-free linear time mutual visibility for asynchronous fat robots. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN '24*, page 84–93, New York, NY, USA, 2024. Association for Computing Machinery.

[10] Subhash Bhagat and Krishnendu Mukhopadhyaya. Mutual visibility by robots with persistent memory. In *International Workshop on Frontiers in Algorithmics, FAW '19*. Springer, 2019.

[11] Subhash Bhagat and Krishnendu Mukhopadyaya. Fault-tolerant gathering of semi-synchronous robots. In *Proceedings of the 18th International Conference on Distributed Computing and Networking, ICDCN '17*, New York, NY, USA, 2017. Association for Computing Machinery.

[12] Adri Bhattacharya, Barun Gorain, and Partha Sarathi Mandal. Treasure hunt in graph using pebbles. In Stéphane Devismes, Franck Petit, Karine Altisen, Giuseppe Antonio Di Luna, and Antonio Fernandez Anta, editors, *Stabilization, Safety, and Security of Distributed Systems, SSS '22*, pages 99–113, Cham, 2022. Springer International Publishing.

[13] Adri Bhattacharya, Barun Gorain, and Partha Sarathi Mandal. Pebble guided treasure hunt in plane. In David Mohaisen and Thomas Wies, editors, *Networked Systems, NETYS '23*, pages 141–156, Cham, 2023. Springer Nature Switzerland.

[14] Adri Bhattacharya, Giuseppe F. Italiano, and Partha Sarathi Mandal. Black hole search in dynamic cactus graph. In Ryuhei Uehara, Katsuhisa Yamanaka, and Hsu-Chun Yen, editors, *WALCOM: Algorithms and Computation*, pages 288–303, Singapore, 2024. Springer Nature Singapore.

[15] Adri Bhattacharya, Giuseppe F. Italiano, and Partha Sarathi Mandal. Black hole search in dynamic tori. *CoRR*, abs/2402.04746, 2024.

[16] Mainak Biswas, Saif Rahaman, Moumita Mondal, and Sruti Gan Chaudhuri. Multiple uniform circle formation by fat robots under limited visibility. In *Proceedings of the 24th International Conference on Distributed Computing and Networking, ICDCN '23*, page 311–317, New York, NY, USA, 2023. Association for Computing Machinery.

[17] Kaustav Bose, Abhinav Chakraborty, and Krishnendu Mukhopadhyaya. Mutual visibility by fat robots with slim omnidirectional camera. *Journal of Parallel and Distributed Computing*, 180:104716, 2023.

[18] Prabhat Kumar Chand, Anisur Rahaman Molla, and Sumathi Sivasubramaniam. Run for cover: Dominating set via mobile agents. In Konstantinos Georgiou and Evangelos Kranakis, editors, *Algorithmics of Wireless Networks, ALGOWIN '23*, pages 133–150, Cham, 2023. Springer Nature Switzerland.

[19] Yufeng Chen, Hongqiang Wang, E. Farrell Helbling, Noah T. Jafferis, Raphael Zufferey, Aaron Ong, Kevin Ma, Nicholas Gravish, Pakpong Chirarattananon, Mirko Kovac, and Robert J. Wood. A biologically inspired, flapping-wing, hybrid aerial-aquatic microrobot. *Science Robotics*, 2(11):eaao5619, 2017.

[20] Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. Arbitrary pattern formation on infinite regular tessellation graphs. *Theoretical Computer Science*, 942:1–20, 2023.

[21] Serafino Cicerone, Alessia Di Fonso, Gabriele Di Stefano, and Alfredo Navarra. The geodesic mutual visibility problem: Oblivious robots on grids and trees. *Pervasive and Mobile Computing*, 95:101842, 2023.

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

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

[24] Deepanwita Das and Srabani Mukhopadhyaya. Distributed painting by a swarm of robots with unlimited sensing capabilities and its simulation. *ArXiv*, abs/1311.4952, 2013.

[25] Deepanwita Das and Srabani Mukhopadhyaya. Distributed algorithm for painting by a swarm of randomly deployed robots under limited visibility model. *International Journal of Advanced Robotic Systems*, 15(5):1729881418804508, 2018.

[26] Deepanwita Das, Srabani Mukhopadhyaya, and Debashis Nandi. Swarm-based painting of an area cluttered with obstacles. *International Journal of Parallel, Emergent and Distributed Systems*, 36(4):359–379, 2021.

[27] Shantanu Das, Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Masafumi Yamashita. The power of lights: Synchronizing asynchronous robots using visible bits. In *2012 IEEE 32nd International Conference on Distributed Computing Systems, ICDCS'12*, pages 506–515, 2012.

[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] Giuseppe Antonio Di Luna, Paola Flocchini, Federico Poloni, Nicola Santoro, and Giovanni Viglietta. The mutual visibility problem for oblivious robots. In *CCCG*, 2014.

[30] Yotam Elor and Alfred M. Bruckstein. Uniform multi-agent deployment on a ring. *Theoretical Computer Science*, 412(8):783–795, 2011.

[31] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. Uniform circle formation for fully, semi-, and asynchronous opaque robots with lights. *Applied Sciences*, 13(13), 2023.

[32] Caterina Feletti, Carlo Mereghetti, and Beatrice Palano. O(log{n})-Time Uniform Circle Formation for Asynchronous Opaque Luminous Robots. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, *27th International Conference on Principles of Distributed Systems, OPODIS '23*, volume 286 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:21, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[33] Paola Flocchini, Giuseppe Prencipe, Nicola Santoro, and Peter Widmayer. Hard tasks for weak robots: The role of common knowledge in pattern formation by autonomous mobile robots. In *Algorithms and Computation*, pages 93–102, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

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

[35] Attila Hideg and Tamás Lukovszki. Uniform dispersal of robots with minimum visibility range. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS '17*, pages 155–167. Springer, 2017.

[36] Attila Hideg and Tamás Lukovszki. Asynchronous filling by myopic luminous robots. In *Algorithms for Sensor Systems: 16th International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS '20, Pisa, Italy, September 9–10, 2020, Revised Selected Papers*, page 108–123, Berlin, Heidelberg, 2020. Springer-Verlag.

[37] Attila Hideg, Tamás Lukovszki, and Bertalan Forstner. Filling arbitrary connected areas by silent robots with minimum visibility range. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics, ALGOSENSORS '18*, pages 193–205. Springer, 2018.

[38] Tien-Ruey Hsiang, Esther M Arkin, Michael A Bender, Sándor P Fekete, and Joseph SB Mitchell. Algorithms for rapidly dispersing robot swarms in unknown environments. In *Algorithmic Foundations of Robotics V*, pages 77–93. Springer, 2004.

[39] Sayaka Kamei and Sébastien Tixeuil. An Asynchronous Maximum Independent Set Algorithm By Myopic Luminous Robots On Grids. *The Computer Journal*, 11 2022. bxac158.

[40] Tanvir Kaur and Kaushik Mondal. Distance-2-dispersion: Dispersion with further constraints. In *Networked Systems: 11th International Conference, NETYS 2023, Benguerir, Morocco, May 22–24, 2023, Proceedings*, page 157–173, Berlin, Heidelberg, 2023. Springer-Verlag.

[41] Ajay D. Kshemkalyani and Faizan Ali. Efficient dispersion of mobile robots on graphs. In *Proceedings of the 20th International Conference on Distributed Computing and Networking, ICDCN '19*, page 218–227, New York, NY, USA, 2019. Association for Computing Machinery.

[42] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Fast dispersion of mobile robots on arbitrary graphs. In Falko Dressler and Christian Scheideler, editors, *Algorithms for Sensor Systems, ALGOSENSORS '19*, pages 23–40, Cham, 2019. Springer International Publishing.

[43] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots in the global communication model. *ICDCN '20*, New York, NY, USA, 2020. Association for Computing Machinery.

[44] Ajay D. Kshemkalyani, Anisur Rahaman Molla, and Gokarna Sharma. Dispersion of mobile robots on grids. In M. Sohel Rahman, Kunihiko Sadakane, and Wing-Kin

Sung, editors, *WALCOM: Algorithms and Computation*, pages 183–197, Cham, 2020. Springer International Publishing.

[45] HL Mando. Parking robot parkie. `https://www.hlmando.com/en/solution/robot/parkingrobot_test.do`, 2022.

[46] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses. Efficient dispersion on an anonymous ring in the presence of weak byzantine robots. In Cristina M. Pinotti, Alfredo Navarra, and Amitabha Bagchi, editors, *Algorithms for Sensor Systems, ALGOSENSORS '20*, pages 154–169, Cham, 2020. Springer International Publishing.

[47] Anisur Rahaman Molla, Kaushik Mondal, and William K. Moses. Byzantine dispersion on graphs. In *2021 IEEE International Parallel and Distributed Processing Symposium, IPDPS '21*, pages 942–951, 2021.

[48] Brati Mondal, Pritam Goswami, Avisek Sharma, and Buddhadeb Sau. Arbitrary pattern formation on a continuous circle by oblivious robot swarm. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN '24*, page 94–103, New York, NY, USA, 2024. Association for Computing Machinery.

[49] Debasish Pattanayak, John Augustine, and Partha Sarathi Mandal. Randomized gathering of asynchronous mobile robots. *Theoretical Computer Science*, 858:64–80, 2021.

[50] Debasish Pattanayak, Subhash Bhagat, Sruti Gan Chaudhuri, and Anisur Rahaman Molla. Maximal independent set via mobile agents. In *Proceedings of the 25th International Conference on Distributed Computing and Networking, ICDCN '24*, page 74–83, New York, NY, USA, 2024. Association for Computing Machinery.

[51] Debasish Pattanayak, Klaus-Tycho Foerster, Partha Sarathi Mandal, and Stefan Schmid. Conic formation in presence of faulty robots. In Cristina M. Pinotti, Alfredo Navarra, and Amitabha Bagchi, editors, *Algorithms for Sensor Systems, ALGOSENSORS '20*, pages 170–185, Cham, 2020. Springer International Publishing.

[52] Debasish Pattanayak, Kaushik Mondal, Ramesh H., and Partha Sarathi Mandal. Gathering of mobile robots with weak multiplicity detection in presence of crash-faults. *Journal of Parallel and Distributed Computing*, 123:145–155, 2019.

[53] Marco Pavone, Alessandro Arsie, Emilio Frazzoli, and Francesco Bullo. Distributed algorithms for environment partitioning in mobile robotic networks. *IEEE Transactions on Automatic Control*, 56(8):1834–1848, 2011.

[54] Pavan Poudel, Aisha Aljohani, and Gokarna Sharma. Fault-tolerant complete visibility for asynchronous robots with lights under one-axis agreement. *Theoretical Computer Science*, 850:116–134, 2021.

[55] Pavan Poudel and Gokarna Sharma. Fast uniform scattering on a grid for asynchronous oblivious robots. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems, SSS '20*, pages 211–228, Cham, 2020. Springer International Publishing.

[56] 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 '19*, page 238–247, New York, NY, USA, 2019. Association for Computing Machinery.

[57] Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, and Partha Sarathi Mandal. Distributed uniform partitioning of a region using opaque async luminous mobile robots. In *Proceedings of the 25th International Conference on Distributed Computing and Networking*, ICDCN '24, page 55–64, New York, NY, USA, 2024. Association for Computing Machinery.

[58] Subhajit Pramanick, Saswata Jana, Adri Bhattacharya, and Partha Sarathi Mandal. Mutual visibility of luminous robots despite angular inaccuracy. *Theoretical Computer Science*, 1011:114723, 2024.

[59] Subhajit Pramanick and Partha Sarathi Mandal. Fault-tolerant mutual visibility for async mobile robots with local coordinate. In *2023 IEEE Guwahati Subsection Conference (GCON)*, pages 1–6, 2023.

[60] Subhajit Pramanick, Sai Vamshi Samala, Debasish Pattanayak, and Partha Sarathi Mandal. Distributed algorithms for filling mis vertices of an arbitrary graph by myopic luminous robots. *Theoretical Computer Science*, 978:114187, 2023.

[61] Subhajit Pramanick, Gurram Joseph Spourgeon, Kritika Raj, and Partha Sarathi Mandal. Asynchronous line formation in presence of faulty robots. NSysS '22, page 75–82, New York, NY, USA, 2022. Association for Computing Machinery.

[62] Michael Rubenstein, Alejandro Cornejo, and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 2014.

[63] Dibakar Saha, Debasish Pattanayak, and Partha Sarathi Mandal. Surveillance of uneven surface with self-organizing unmanned aerial vehicles. *IEEE Transactions on Mobile Computing*, 21(4):1449–1462, 2022.

[64] Gokarna Sharma. Mutual visibility for robots with lights tolerating light faults. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW '18*, pages 829–836. IEEE, 2018.

[65] 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 '18*, New York, NY, USA, 2018. Association for Computing Machinery.

[66] Gokarna Sharma, Costas Busch, and Supratik Mukhopadhyay. Mutual visibility with an optimal number of colors. In *International Symposium on Algorithms and Experiments for Wireless Sensor Networks, ALGOSENSORS '15*, pages 196–210. Springer, 2015.

[67] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry Trahan. Optimal randomized complete visibility on a grid for asynchronous robots with lights. *International Journal of Networking and Computing*, 11(1):50–77, 2021.

[68] Gokarna Sharma, Ramachandran Vaidyanathan, and Jerry L. Trahan. Constant-time complete visibility for robots with lights: The asynchronous case. *Algorithms*, 14(2), 2021.

[69] Gokarna Sharma, Ramachandran Vaidyanathan, Jerry L Trahan, Costas Busch, and Suresh Rai. Complete visibility for robots with lights in o(1) time. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems, SSS '16*, pages 327–345. Springer, 2016.

[70] Takahiro Shintaku, Yuichi Sudo, Hirotsugu Kakugawa, and Toshimitsu Masuzawa. Efficient dispersion of mobile agents without global knowledge. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems, SSS '20*, pages 280–294, Cham, 2020. Springer International Publishing.

[71] Samia Souissi, Xavier Défago, and Masafumi Yamashita. Gathering asynchronous mobile robots with inaccurate compasses. In Mariam Momenzadeh Alexander A. Shvartsman, editor, *Principles of Distributed Systems, OPODIS '06*, pages 333–349, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[72] Ramachandran Vaidyanathan, Costas Busch, Jerry L. Trahan, Gokarna Sharma, and Suresh Rai. Logarithmic-time complete visibility for robots with lights. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS '15*, pages 375–384, 2015.

[73] Ramachandran Vaidyanathan, Gokarna Sharma, and Jerry L. Trahan. On fast pattern formation by autonomous robots. In Taisuke Izumi and Petr Kuznetsov, editors, *Stabilization, Safety, and Security of Distributed Systems, SSS '18*, pages 203–220, Cham, 2018. Springer International Publishing.

[74] Masafumi Yamashita, Samia Souissi, and Xavier Défago. Gathering two stateless mobile robots using very inaccurate compasses in finite time. In Alan F. T. Winfield and Jason Redi, editors, *Proceedings of the 1st International Conference on Robot Communication and Coordination, ROBOCOMM '07, Athens, Greece, October 15-17, 2007*,

volume 318 of *ACM International Conference Proceeding Series*, page 48. ICST/ACM, 2007.

[75] Moju Zhao, Tomoki Anzai, Fan Shi, Xiangyu Chen, Kei Okada, and Masayuki Inaba. Design, modeling, and control of an aerial robot dragon: A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation. *IEEE Robotics and Automation Letters*, 3(2):1176–1183, 2018.

Research Articles from the Thesis Work

Papers published in International Journals

- Subhajit Pramanick, Sai V. Samala, Debasish Pattanayak, and Partha Sarathi Mandal. **Distributed algorithms for filling MIS vertices of an arbitrary graph by myopic luminous robots,** *Theoretical Computer Science (Elsevier)*, Volume 978, November 2023.

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya and Partha Sarathi Mandal. **Mutual visibility of luminous robots despite angular inaccuracy**, *Theoretical Computer Science (Elsevier)*, Volume 1011 , October 2024.

Papers published in International Conferences

- Subhajit Pramanick, Sai V. Samala, Debasish Pattanayak and Partha Sarathi Mandal. Filling MIS Vertices of a Graph by Myopic Luminous Robots, in Proc. of *19th International Conference on Distributed Computing and Internet Technology (ICDCIT 2023)*, Lecture Notes in Computer Science (LNCS-13776), (Springer-Verlag), Bhubaneswar, India, January 18-20, 2023. (Best Paper Award).

- Subhajit Pramanick and Partha Sarathi Mandal.
  Fault-tolerant Mutual Visibility for ASYNC Mobile Robots with Local Coordinate, In Proc. of *IEEE GCON 2023, (IEEE Xplore)*, Guwahati, India, June 23-25, 2023, 10.1109/GCON58516.2023.10183479.

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya and Partha Sarathi Mandal. Mutual Visibility with ASYNC Luminous Robots having Inaccurate Movements. in Proc. of *International Symposium on Algorithmics of Wireless Networks (ALGOWIN formerly ALGOSENSORS)*, LNCS 14061 (Springer-Verlag), Amsterdam, Netherlands, September 4-8, 2023, https://doi.org/10.1007/978-3-031-48882-5_4

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya and Partha Sarathi Mandal. Distributed Uniform Partitioning of a Region using Opaque ASYNC Luminous Mobile Robots, in Proc. of *25th International Conference on Distributed Computing and Networking (ICDCN 2024)*, (ACM), Chennai, India, January 4-7, 2024, https://doi.org/10.1145/3631461.3631555

Manuscripts Submitted/ Under Preparation

- Subhajit Pramanick, Saswata Jana and Partha Sarathi Mandal. **Fault-Tolerant mutual visibility without any axis agreement** [Submitted to the *Theoretical Computer Science (Elsevier)* after first revision], September 2024.

- Subhajit Pramanick, Saswata Jana, Adri Bhattacharya and Partha Sarathi Mandal. **Uniform Partitioning of a Bounded Region using Opaque ASYNC Luminous Mobile Robots**, [Submitted to the *Theoretical Computer Science (Elsevier)*], May 2024.

Other Publication

- Subhajit Pramanick, Gurram J. Spourgeon, Kritika Raj, and Partha Sarathi Mandal. Asynchronous Line Formation in Presence of Faulty Robots, in Proc. *9th International Conference on Networking, Systems and Security (NSysS 2022)*. ACM, Cox's Bazar, Bangladesh, pp. 60-64, December 20-22, 2022, https://doi.org/10.1145/3569551.3569553.