Sensor Deployment by Robot in an Unknown Orthogonal Environment

by

Najmeh Taleb

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science
in

Computer Science
Carleton University
Ottawa, Ontario, Canada
January 2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87840-8

Our file Notre référence
ISBN: 978-0-494-87840-8

Abstract

Sensor deployment is one the fundamental problems in distributed computing. In this problem, sensors are deployed in an unknown environment to collect information. This problem, has been studied extensively in the literature. In *static sensor networks*, sensors are usually deployed by a human. But this method is not applicable for dangerous or inaccessible environment. In *mobile sensor networks*, sensors move and scatter in the environment. Mobile sensors are very expensive and this method is not efficient when sensors are in danger because of the environmental conditions. A new method to solve this problem is by using a mobile robot to deploy static sensors. There are few algorithms in the literature that provide sensors deployment by the robot. However, these algorithms do not achieve full coverage since in some cases the robot gets stuck in a dead end while some parts of the environment are still uncovered. In some other cases, the robot leaves uncovered areas near the boundaries. Moreover, in all the excising algorithms there are critical areas which are not covered by the grid and lead to uncovered areas in the environment. There is only one algorithm in the literature that solves the dead end problem. However, this algorithm does not achieve full coverage because of the critical areas and uncovered areas near the boundaries. In this thesis first we propose two algorithms, A1 and A2, to solve the boundary problem. We show that these algorithms guarantee full coverage in any orthogonal environment without critical areas. Then we proposed algorithm B to work in an orthogonal environment containing critical areas.

Acknowledgments

No words can express my gratitude to my supervisor Professor Nicola Santoro for his significant contribution that led to the ultimate achievement of my thesis. I was deeply impressed with his comprehensive knowledge, insightful guidance and extraordinary project management during the course of my thesis, which will be a great help to my career development in the near future. I am also especially grateful to my thesis co-supervisors, Dr. Wei Shi for her guidance, critical reading of thesis and her valuable comments . In addition, I would like to thank my friend Eduardo for his knowledgeable suggestions that greatly enhanced my enjoyment of this process, and significantly improved the end result. I must thank my parents, for their support to achieve the milestones in my life and for their endless love. Also I would like to thanks my lovely sisters Zahra and Ferdos for their understanding and consistent encouragement. Finally, I would like to thank my lovely roommates Zienab and Elnaz for their infinite support.

Table of Contents

List of Figures

List of Acronyms

Acronyms	Definition
WSNs	Wireless Sensor Networks
WSANs	Wireless Sensor and Actor Networks
WSRNs	Wireless Sensor and Robot Networks
SANETs	Sensor Actuator Networks
MSNs	Mobile Sensor Networks
ROI	Region Of Interest
R-sensor	Regular Sensor
B-sensor	Boundary Sensor
E-sensor	Entrance Sensor

Chapter 1

Introduction

1.1 Motivation

Wireless sensor networks (WSNs) consist of a large number of wireless sensor nodes which have limited sensing, computational and communicating capabilities. Sensor nodes are densely deployed and cooperatively work to achieve a common task. Each sensor in the network senses the environment, processes the information and then sends information to the base stations. The base stations are one or more components of the WSNs that act as gateways between sensor nodes and the end user. Sensor networks are widely applied to many areas, including Military Applications, Environmental Applications, Health Applications, Habitat Monitoring, and Home Applications [1, 2, 23].

Wireless sensor and actor networks (WSANs), also called sensor *actuator networks (SANETs)* [11] and *wireless sensor and robot networks (WSRNs)* [19], are an extension of wireless sensor networks. WSANs are composed of sensor nodes and actor (actuator or robot) nodes which cooperatively perform a common task. Sensor nodes are low-cost, low-power devices with limited sensing, computational, and communication capabilities. Actor nodes are resource-rich nodes equipped with more

powerful processors, longer-range radio transceivers, and longer battery lives. In addition, actors may have the capability to perform movement. Typically the number of sensor nodes is much larger than the number of actor nodes. Sensor nodes collect data from the environment, send it to actor nodes, and the actor nodes make a decision and perform appropriate actions. In WSANs, the area where sensors and actors are distributed is defined as the *sensor-actor field* [3,23]. In the literature, the sensor-actor field is also referred to as *target area* [3], *Region of Interest (ROI)* [19], *terrain* [1], *monitoring area* [14], and *sensor field* [2]. In this study, we use the term ROI to indicate the area we are working on.

Sensor deployment is one of the fundamental problems in WSNs. In this problem, the goal is to deploy sensors in the environment in such a way that the entire ROI is fully covered without any *sensing holes*. Sensing holes are the uncovered areas of a ROI which are not under the sensing range of any sensor. In order to cover the entire ROI, either a human or robot deploys static sensors, and the number of static sensors should be sufficiently large. However, if the ROI is unknown, determining the minimal number is difficult [8–10]. Many deployment algorithms were proposed to increase the coverage of the sensor field. In the literature, existing deployment algorithms are classified into three categories depending on the type of network: 1) stationary sensor networks; 2) mobile sensor networks; 3) sensor and mobile robot networks. In *static sensor networks*, static sensors are deployed in a static configuration by a human. In *mobile sensor networks (MSN)*, the sensors have the capability of movement and are responsible for computing and moving to appropriate locations. It is expensive to support the mobility of the network. So, for dangerous ROIs where sensors may die because of the environmental conditions, using mobile sensors is not efficient. An alternative method is deploying static sensors by a robot. The robot carries the static sensors and moves in the monitoring region to deploy them at proper positions [12–14]. The robot can also have other commissions, such as patrolling to

detect created sensing holes due to sensor failure.

Sensor deployment by a robot is very effective for deploying static sensors in dangerous, or inaccessible ROIs. An example of a dangerous ROI is a building where a reactor is located. Assume a situation when an earthquake causes serious damages to the reactor, such that radioactive material is leaking. To control the situation, we need to collect a lot of information from the ROI. One way to gather information is to deploy sensors around it. However, because of the radioactive radiation, the ROI is very dangerous for humans to enter. In this situation a robot can deploy sensors properly. Another example of a dangerous ROI is when a forest is in an uncontrollable fire which progresses very fast. In addition to dangerous ROIs, new planets are another instance in which the ROI is inaccessible during space discoveries. To collect information from these types of new ROI, sensors should be deployed on the surface of planets by a robot.

In literature, there are few algorithms that cover sensor deployment by robots. However, non of these algorithms solve this problem properly. Several algorithms do not guarantee full coverage since the robot leaves uncovered areas close to the boundaries. In other cases, the robot is stuck by boundaries and predeployed sensors, so it cannot get out of this situation to cover other uncovered parts of the ROI. Moreover, some algorithms do not terminate and the robot does not stop moving when full coverage is achieved.

In addition to the above problems, all of the existing algorithms have another serious problem. Although in reality most of the ROIs have arbitrary shapes, in these algorithms the robot is only capable of deploying sensors in a rectangle, square or circle. However, most real life ROIs have arbitrary boundaries, such as cities with streets and buildings with many internal rooms and corridors. Figure 1 shows such a ROI with arbitrary boundaries; however, using the existing algorithms, a robot will not be capable of deploying sensors in such a ROI. Hence, in this research we seek

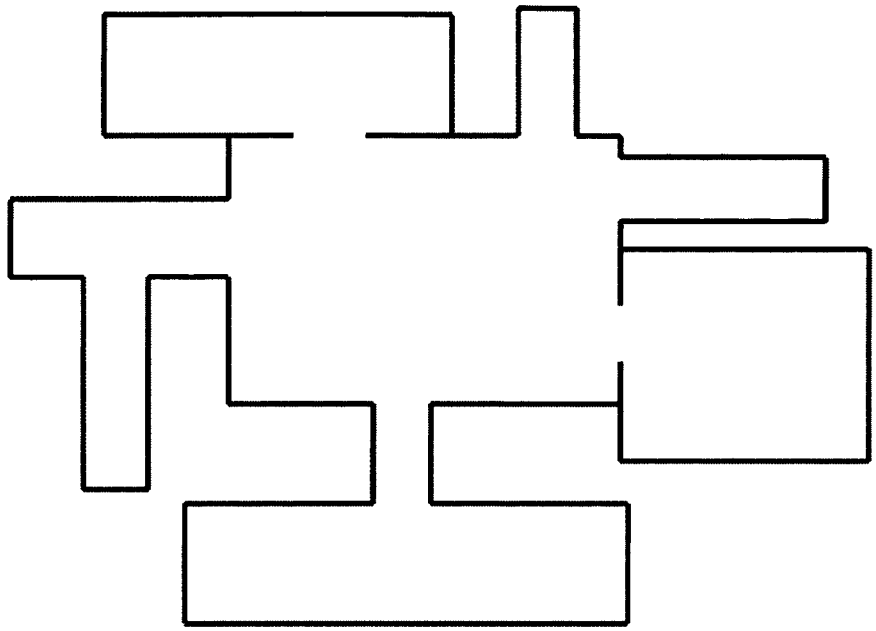


Figure 1: A ROI with arbitrary shape/boundaries.

solutions for using a single robot to deploy sensors in a ROI with arbitrary boundaries.

1.2 Contributions

We propose an algorithm on sensor deployment by a robot in an unknown but bounded ROI. The ROI is an obstacle-free orthogonal polygon. We assume a single robot scenario in which the robot has neither a map of the ROI nor a GPS. The robot is equipped with a compass and can detect nearby boundaries and sensors within sensing range. Sensors are able to communicate with neighboring sensors and the robot within their communication radius.

In the existing algorithms, the robot mostly deploys sensors on the vertices of a gird such as triangular or square grid. In this situation, if some part of the ROI is

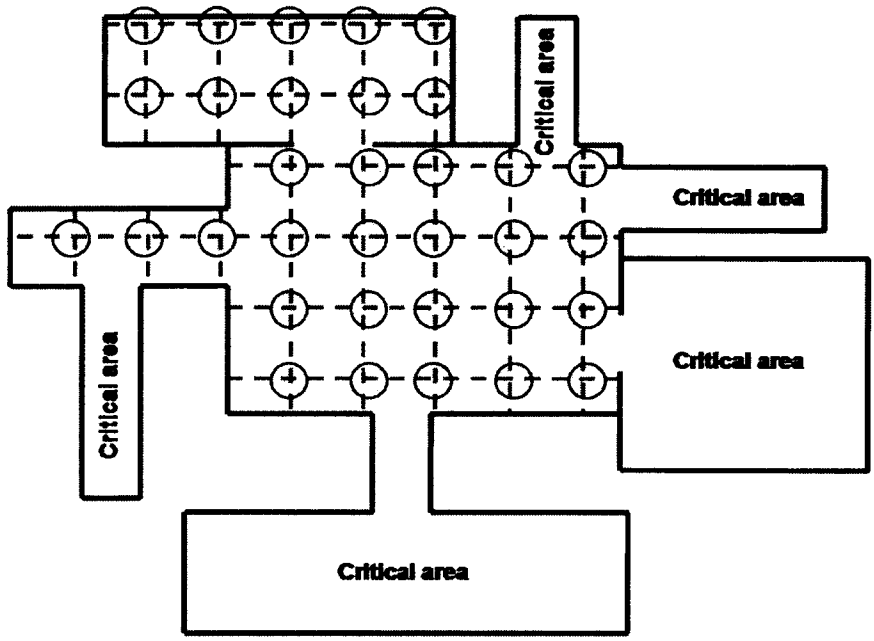


Figure 2: Critical areas which are not covered by the grid, are not covered by the sensors neither.

not covered by the grid, it will not be covered by the sensors, creating what we call *critical areas.* A critical area could be caused by a narrow corridor that connects different parts of a building, or the entrance of a room. Figure 2 illustrates a ROI which is an orthogonal polygon containing critical areas. In this Figure, when the robot enters the ROI, it creates a virtual square grid based on its local coordinate system and starts deploying sensors on the vertices of this grid. However, as it is shown in this Figure, the critical areas are not covered by the sensors.

Our main goal in this thesis is to develop a sensor deployment algorithm by the robot which guarantees full coverage for an orthogonal ROI containing critical areas. As we just mentioned, none of the existing algorithms achieve full coverage even when there are no critical areas in the ROI. Therefore, first we propose two algorithms to guarantee full coverage in an orthogonal ROI without any critical areas. Then, we propose a solution to solve the deployment problem in the orthogonal ROI containing

critical areas.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 includes background information about the subject area including wireless sensor network and sensor deployment problem. It also includes the models and assumptions which are used in this study. Related work is discussed in preparation for algorithms which are proposed in the following chapters. In Chapter 3, two algorithms are presented to solve the sensor deployment problem in a ROI without critical areas. In Chapter 4, an algorithm is proposed to solve sensor deployment problem for a ROI containing critical areas. Finally, conclusions and future work are discussed in Chapter 5.

Chapter 2

Definitions, Model, and Related Work

2.1 Definitions and Terminology

Sensor deployment is a fundamental problem in distributed computing using mobile entities. This problem has been studied extensively in the literature. In the following section we introduce some of the definitions and terminologies which are frequently used in sensor deployment problems in general. These concepts are as follows: *Movement pattern* [12, 13], *Deployment policy* [12, 13], *Order rule* [10], *Snake-like Movement policy (Serpentine deployment mechanism)* [12, 13], *Spiral Movement Policy* [14], *Cardinal directions* [14], *Boundary handling rule* [12], *dead end recovery rule* [19], *Termination* [14, 18, 19, 26].

2.1.1 Movement pattern and Deployment policy

To deploy sensors in the ROI, the robot can move randomly and deploy sensors in arbitrary positions. However, in this case there is no guarantee that the ROI is fully covered and there may exist sensing holes in some parts of it. To have complete coverage, the robot can use one of the following ways: *square deployment*, *triangle deployment* or *hexagon deployment* [12]. When the robot enters the ROI, it moves on a virtual grid, and starts deploying sensors on the vertices of this grid. For this purpose,

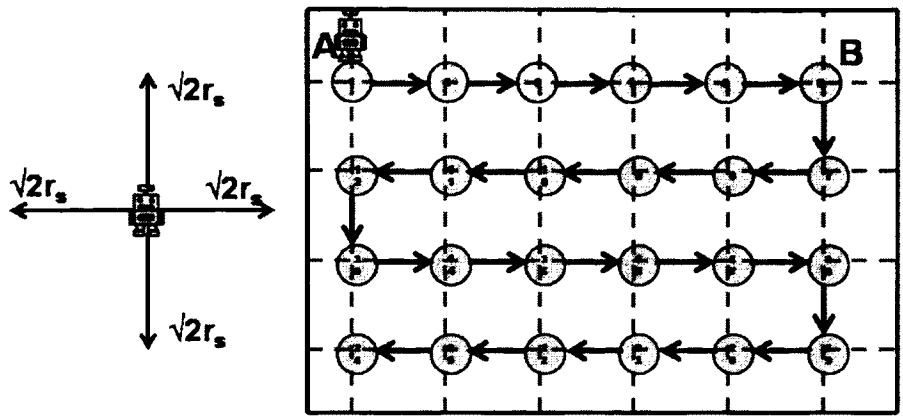


Figure 3: Square Deployment

it needs an appropriate pattern movement to move around the grid. Moreover, it also needs to follow deployment policy in order to deploy sensors on the grid.

In square deployment, sensors are deployed on the vertices of a square grid with square's length of $\sqrt{2}r_s$, where r_s is the sensing range of the sensors. To move on a square grid, the robot moves in four directions (West, East, North, and South). An order rule indicates the priority of each direction that will be selected as the next movement direction. So, deployment policy in a square grid is moving in one of four directions and deploying a sensor for every $\sqrt{2}r_s$. Figure 3 shows a square grid example which is covered using this movement pattern. In this figure, the circles denote sensors and the arrows illustrate the pattern movement of the robot. The robot enters the ROI from location A and starts deploying sensors based on the order rule which is east, west, north and south in this example. After placing the first sensor, the robot moves to the east direction and drops the second sensor. It keeps deploying sensors in the east direction until it gets to location B where east, west and north directions are bounded. The robot then moves south for a distance of $\sqrt{2}r_s$ and deploys the next sensor in a new row. It keeps deploying sensors following the

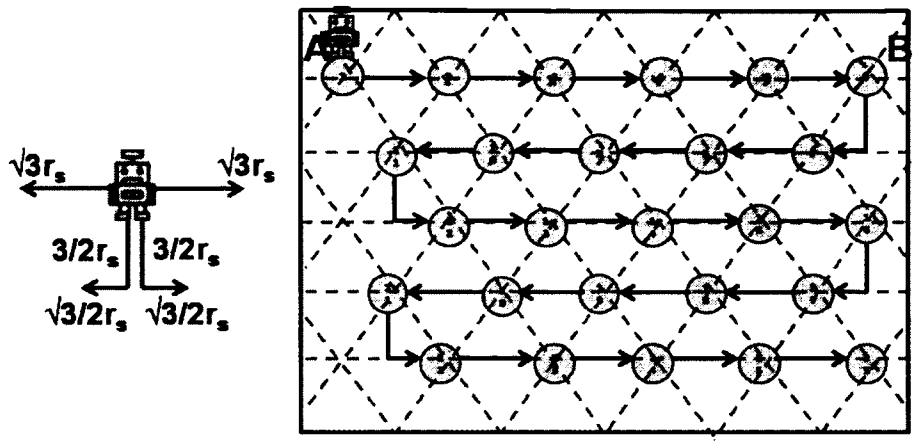


Figure 4: Triangle Deployment

order rule.

In triangle deployment, sensors are deployed on the vertices of a triangle grid with a length of $\sqrt{3}r_s$. To deploy sensors, the robot uses a snake-like deployment policy. Based on this policy, the robot moves in the east or west direction and deploys a sensor for every $\sqrt{3}r_s$. Whenever it encounters a left or right boundary, it locates the sensors on the next row. For this purpose, it moves south for a distance of $3/2r_s$ and then moves west for a distance of $\sqrt{3}/2r_s$ and deploys a sensor if it encounters right boundary. If it does not encounter a right boundary, it begins to move in the east direction. Figure 4 shows a triangle deployment using snake-like deployment policy. In this figure, the robot starts deploying sensors starting at location A. After deploying the first sensor, it moves east and drops the second sensor at a distance of $\sqrt{3}r_s$. It keeps deploying sensors until it encounters the right boundary at location B. In this case the robot moves south for a distance of $3/2r_s$ and then moves west for a distance of $\sqrt{3}/2r_s$ and deploys the next sensor in a new row. The robot keeps deploying sensors for every $\sqrt{3}r_s$.

In hexagon deployment, the ROI is divided to into hexagons with lengths of $\sqrt{3}r_s$

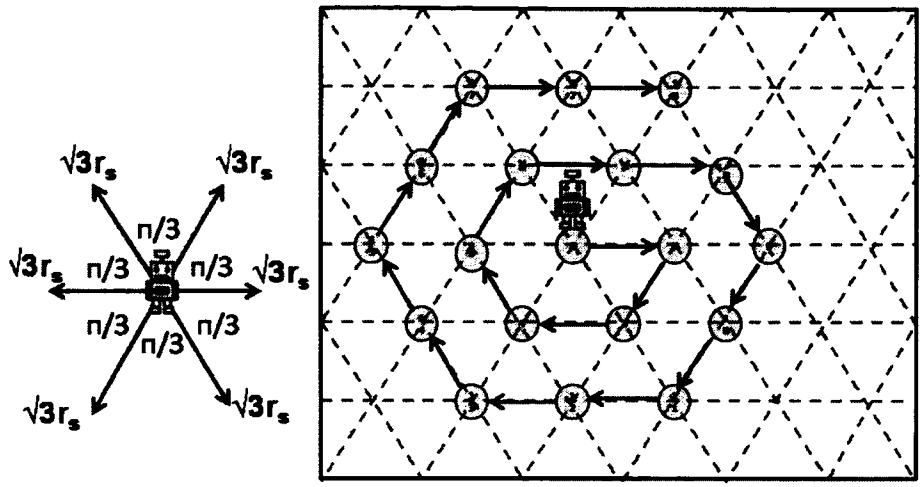


Figure 5: Hexagon Deployment

[3]. To place sensors on the vertices of these hexagons, the robot follows a spiral movement policy, that is moving in six cardinal directions, and deploys sensors for every $\sqrt{3}r_s$ [14]. Figure 5 shows a hexagon deployment by using a simple spiral movement policy.

2.1.2 Boundary handling Rule

When the robot approaches the boundary, if it does not deploy sensors close enough to the boundaries, it will leave uncovered spots creating sensing holes. For example, in a square grid, the robot is supposed to deploy a sensor for every $\sqrt{2}r_s$, so if the distance between the robot and the boundary is less than $\sqrt{2}r_s$ it will move south or north to deploy sensors on the next row. The distance that remains may result in a sensing hole. This problem is called a *boundary problem.* To overcome this, the robot needs to follow a boundary handling rule to deploy sensors near the boundaries when it is needed. In Figure 6(a) and 6(b) the unshaded points are the sensing holes in a square and triangle grid respectively.

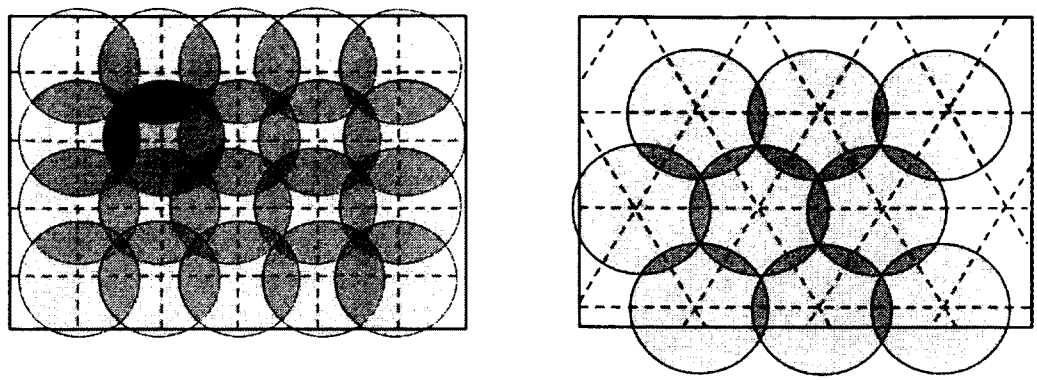


Figure 6: Unshaded points show sensing holes in (a) square grid, and (b) triangular grid.

2.1.3 Dead end recovery Rule

When the robot is deploying sensors, it may be stuck by early deployed sensors, obstacles or boundaries. In this case, there is no *open direction* for the robot to move and it is stuck in a *dead end.* The open direction is actually a direction that, by moving toward it, the robot reaches an empty neighboring vertex. In order for a robot to get out of a dead end, one of the most commonly used methods is back tracking to a location which has open directions. For back tracking, the robot either needs to remember the traversed path or needs to ask sensors for guidance. In order to remember the traversed path, each robot needs an unbounded memory. This is because in most of the sensor deployment cases the size of the ROI in unknown. Obviously, such an assumption is very expensive and unrealistic.

The other method for a robot to back track, is to ask the sensors about the backward path. This solution requires both the sensors and the robot to have communication capability. In this case, sensors can help the robot find its path and do back tracking. Figure 7 shows a situation in which the robot is stuck in a dead end after deploying sensor 18 at location A. In this case, the robot can back track to a location such as B or C, which is adjacent to an uncovered vertex, in order to resume

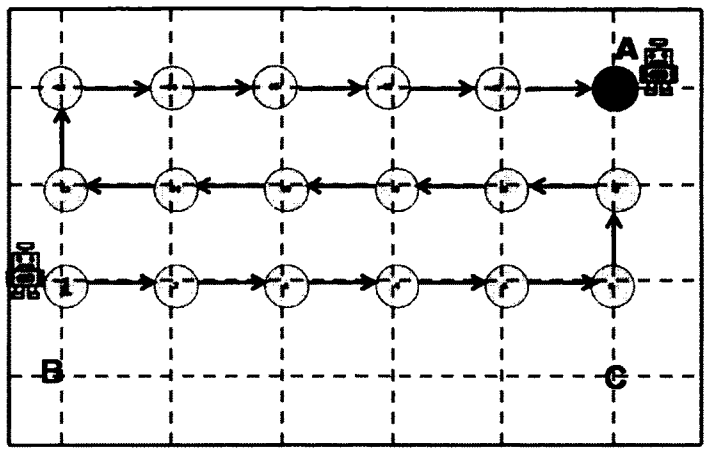


Figure 7: The robot is stuck in a dead end at location A. It can back track to either location B or C.

its movement.

2.1.4 Termination

In sensor deployment, there are two different scenarios. In the first scenario it is supposed that there is no sensor failure in ROI and the deployed sensors work properly. This ROI is called a *failure-free* ROI [19]. In the second scenario, the sensors may fail and cause a sensing hole in the ROI; this ROI is called a *failure-prone* ROI [19]. In the first scenario, the proposed algorithms are only deployment algorithms. In this case, a robot should be able to determine when the ROI is fully covered and terminates executing the algorithm accordingly in order to save energy. However, in the second scenario, a robot usually keeps moving around and executing maintenance algorithms. In the second case, after deploying all sensors, the robot patrols in the ROI. Once a sensing hole is detected due to sensor failure it fixes or replaces the malfunctioning sensor. In this case the algorithm does not necessarily need to terminate. Otherwise, clear termination of the algorithm should be assumed for the failure-free ROI.

2.2 Model and Assumptions

In the following sections we introduce our model and assumptions which are applied for all the proposed algorithms in this work.

2.2.1 Robot, Sensors and ROI

Let R denote a single robot which has neither a map of the ROI nor GPS. It is equipped with a compass. It can detect and measure the distance to the boundaries and the deployed sensors within distance $2r_s$ where r_s is the sensing radius of the sensors. The robot is able to communicate with sensors within its communication radius which is $2r_s$. The ROI is a connected bounded orthogonal region of unknown shape. A simple polygon is called orthogonal if all its edges are parallel to either the x axis or the y axis [27]. Moreover, we assume there are no obstacles in the ROI. The ROI may or may not contain *critical areas.* In the following subsection, we provide a definition for critical areas. We call an orthogonal ROI simple, if does not contain any critical areas, and complex orthogonal otherwise. The task of R is to deploy a set of sensors, denoted as S on the ROI in order to fully cover it. Each sensor ($\forall s \in S$) has limited computation capability. The sensing range of each sensor s is r_s, and if it is disconnected by a boundary, the disconnected area cannot be sensed by this sensor. Sensor s has no visibility, but is able to communicate with another neighboring sensor $s' \in S$ and the robot within distance $2r_s$. Both R and S can communicate with each other (sensor to robot, robot to sensor, sensor to sensor) within $S_c = R_c = 2r_s$ where S_c and R_c are communication radius of sensors and robot respectively. Moreover, we assume that all sensors work properly and there are no sensor failures in the ROI.

We assume R enters the ROI from an arbitrary location. By using a compass R computes a fixed square grid according to its local coordinate system such that each cell of the grid is a square with the length of $\sqrt{2}r_s$. It can also move in one of the

four directions: West, East, North, and South on the vertices of the grid denoted as v_i. If the boundary of the ROI overlaps on a grid line, R is able to move along the boundary and deploy sensors on the boundary if needed.

2.2.2 Critical Area

Since the robot does not have any map of the ROI, critical areas are locally detected from the current location of the robot. In this study the ROI is a connected orthogonal region with unknown boundaries. Let P denote the polygon that represents the boundary of the ROI. Let x and y be two points on P such that the segment $\overline{xy}$ (a straight line which ends on x and y on each side) is entirely inside P. This segment defines two polygons P_1^{xy} and P_2^{xy}. The first one starts from x and includes every vertex of P between x and y in counter-clockwise order, while the second one is obtained in the same way but starting from y instead. Note that, since both polygons contain the segment $\overline{xy}$ on their boundary's, then they divide the ROI into two disjoint polygons.

Consider a circle C centered at a vertex v (assume that the robot is located at this vertex) of grid G with radius $2r_s$, and let C_i $(1 \leq i \leq 4)$ be one the four quadrants of C defined by North-South and East-West directions. Critical areas are detected for each of these quadrants separately. Consider any C_i which is intersected by P. Let x and y be two points on P (not necessarily vertices) such that the length of the segment $\overline{xy}$ is less than $2r_s$, within P and completely visible from v. In Figure 8, v is the current vertex and the circle represents C with radius $2r_s$. Considering two points x and y on P, the line segment $\overline{xy}$ divides P into two polygons P_1 $(Xx_1x_2x_3x_4Y)$ and P_2 $(Yy_1zy_2y_3y_4y_5y_6y_7y_8y_9y_{10}y_{11}y_{12}y_{13}y_{14}y_{15}y_{16}y_{17}X)$. Now, consider the polygon P_j^{xy}, $j = 1$ or $j = 2$, that does not contain v. We say P_j^{xy} contains critical area/s, if there is a vertex z, $z \neq x$ and $z \neq y$, (z belongs to P_j^{xy}) such that $\overline{zv}$ is not entirely inside P_j^{xy} (provided that $\overline{zv}$ does not intersect P_j^{xy}), or the length of it is greater than $2r_s$.

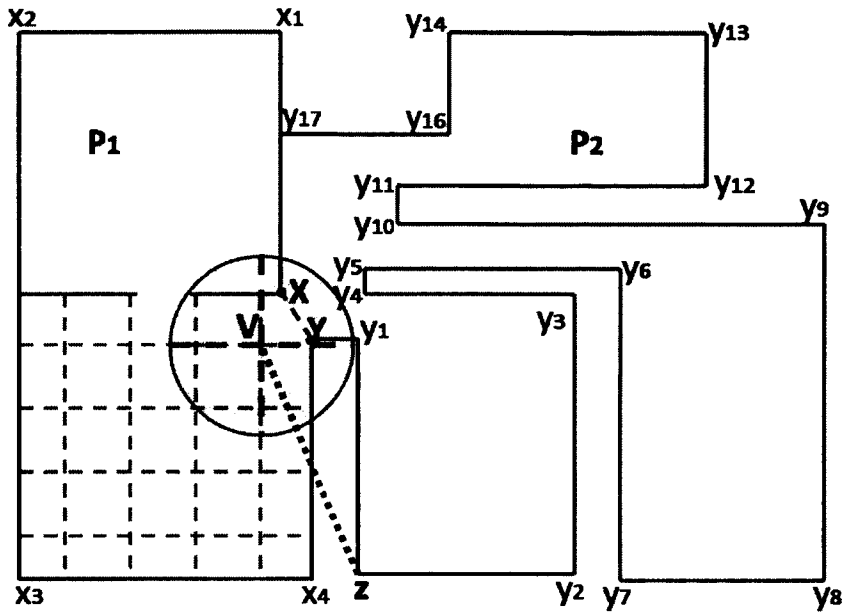


Figure 8: Polygon P_2 contains critical area since line segment $\overline{zv}$ is located outside of its boundaries.

This notion is formally stated in the following definition.

Definition 1 *Let v be a vertex of the grid G, and C_i be one of the four quadrants of C centered at v. Consider two points x and y on P determining a segment $\overline{xy}$ visible from v and entirely contained in the interior of P and C_i. We say that the polygon P_j^{xy}, $j = 1$ or $j = 2$, is a critical area if and only if there is a vertex z on P_j^{xy} that satisfies any of the following conditions:*

- *The segment $\overline{zv}$ is not totally inside P (provided that $\overline{zv}$ does not intersect P).*
- *The length of $\overline{zv}$ is greater than $2r_s$.*

In Figure 8, polygon P_2 is a critical area since line segment $\overline{zv}$ is located outside of P_2.

According to the number of critical areas, a ROI can be either simple or complex. A *Simple ROI* does not have any critical areas. Since the robot does not have any

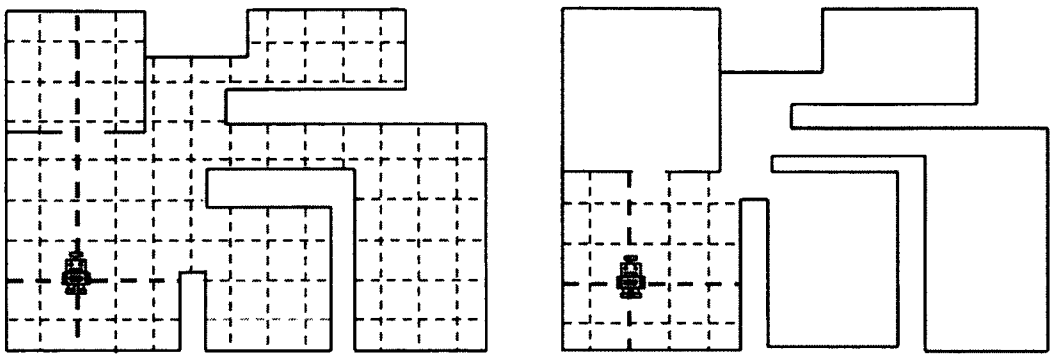


Figure 9: (a) A Simple ROI. (b) A Complex ROI.

map of the ROI, the critical areas are detected locally from each grid point. Hence, when we say the ROI is simple, the robot cannot detect a critical area from any grid point. Figure 29 (a) illustrates a simple ROI. As it is shown in this Figure all parts of the ROI are covered by the grid. The bold dashed lines in this figure show the starting point of the robot. On the other hand, a *Complex ROI* contains critical areas. Figure 29 (b) illustrates a complex ROI. As it is shown in this figure the critical areas are not covered by the grid.

2.3 Related Work

The sensor deployment problem is to deploy sensor nodes, each with limited sensing range, into the environment by a process that guarantees full coverage. In the literature, existing deployment algorithms are classified into three categories: human deploying static sensors, self deploying mobile sensor, and mobile robot deploying sensors [12–14]. In this study, we classify them into the four following categories: Sensor deployment in Static (Stationary) Sensor Networks, Sensor deployment in Mobile Sensor Network (MSN), Sensor deployment in Hybrid Sensor Network, and Sensor deployment in Wireless Sensor and Robot Network (WSRN). In the following

sections, we briefly discus the first three categories. Then we discuss all the proposed algorithms in the last category in detail.

2.3.1 Sensor Deployment in Static Sensor Networks

Several random deployment algorithms have been proposed for the deployment of stationary sensors. The random deployment is simple and easy to implement. However, to ensure full coverage, its number of deployed sensors is much larger than the actual required number of sensors. In random deployment of static sensors, some areas may have high density of sensors while others have low density. So, this method is not efficient since areas with high density increase hardware costs, computation time, and communication overheads whereas areas with low density may raise the problem of sensing holes [12–14].

Since in random deploying, sensors usually obtain an undesired coverage ratio, the static sensor networks must be deployed according to a predefined shape decided by the different optimal algorithms [26]. In this case sensors are deployed in a static configuration such that every point in the environment is covered at every instant of time. For complete static coverage of an environment, the number of sensors should be larger than a critical size. Determining the critical number is difficult or impossible if the environment is unknown a priori [7–10].

2.3.2 Sensor deployment in Mobile Sensor Network

In a sensor self-deployment approach [19] a mobile WSN is considered as a network that is wholly composed of mobile sensors. By this approach, sensors can place themselves by intelligently changing their geographic location without others' help [21], and they can cooperatively adjust their location to achieve full coverage. However, each mobile sensor requires additional hardware cost that supports the mobility, and

considerable energy consumption is required for each mobile sensor that moves from one location to another [12].

The key advantage of employing mobile sensors is that sensors can dynamically move to cover uncovered areas for increasing coverage ratio. The sensors may move from a densely deployed area toward a sparse area [26].

The existing works that have studied sensor self-deployment are categorized as the following according to [21]: 1) virtual force (vector-based) approach, 2) Voronoi-based approach, 3) load-balancing approach, 4)stochastic approach, 5) point-coverage approach, 6)incremental approach, 7) maximum-flow approach, and 8)genetic algorithm approach.

2.3.3 Sensor Deployment in Hybrid Sensor Networks

A hybrid sensor network is a sensor network that is mixed with static and mobile sensors [26]. After an initial phase of random deployment of stationary sensors, mobile sensors are coordinated to compute for their target locations according to the information regarding the holes in the monitoring area and then move to the target locations to heal the existing coverage holes. However, hardware costs cannot be reduced in the areas that were densely deployed with stationary sensors [12].

2.3.4 Sensor Deployment in Wireless Sensor and Robot Networks

Wireless sensor and robot (also called actuator or actor) networks (WSRN) are an integration of wireless sensor networks and multi-robot systems. They consist of networked sensor and robot nodes that communicate via wireless links to perform distributed sensing tasks [19]. Robots explore the environment and deploy stationary sensors to the target location from time to time. The robot deployment can achieve

full coverage, increase the sensing effectiveness of stationary sensors, and guarantee full coverage and connectivity [12–14].

Few works have studied sensor deployment in WSRN. We categorize the existing algorithms as the following: 1) Least Recently Visited approach, 2)Snake-Like Deployment approach, 3) Spiral Deployment approach, 4) Back-Tracking Deployment approach, 5)Robot Deployment in concave region, 6)FOCUSED Coverage, and 7)Flying Robot.

Least Recently Visited approach

LRV (Least Recently Visited) is a deployment and coverage maintenance algorithm [5–10]. In this algorithm the robot neither has a GPS, nor a map of the environment. It is equipped with a compass, wireless communication and can carry sensor nodes as payload. Each sensor is equipped with a small processor and a radio of limited range to communicate with the robot; however, this algorithm does not use inter-node communication, which means sensors do not communicate with each other. Moreover, this algorithm uses square deployment method that implements in bounded and unknown ROIs with irregular obstacles. LRV has been applied by both single and multiple robots.

In LRV, the task of each sensor is telling the robot which direction to explore next. Each node is equipped with a 2-bit compass. It also maintains a state and a counter for each direction. The state can be either OPEN or EXPLORED depending on whether the direction was explored by the robot previously or not. The counter counts the number of times that a direction is traversed by the robot. All OPEN directions are recommended first in order from south to west and then EXPLORED directions with the least counter.

In this algorithm, the robot executes four behaviors: ObstacleAvoidance, At-Beacon, DeployBeacon, and SearchBeacon. When the robot enters an empty ROI it

places first sensor and activates AtBeacon behavior since it is in communication range of a sensor. As soon as a sensor is placed, it starts emitting messages that contain suggested directions for the robot. AtBeacon behavior allows the robot to analyze received messages to get suggested directions. Then it checks the suggested direction for obstacles and if it was bounded the sensor will be informed by the robot to not offer this direction anymore and the robot asks for another direction. Before moving to open suggested direction the robot sends a notification message to sensor to increment the counter of that direction. Moreover, the status of that direction will be changed from OPEN to EXPLORED by the sensor. Then the robot activates SearchBeacon behavior and moves to the suggested direction. At this state, the robot moves for a predefined distance (less than or equal to twice the sensing range) and then waits for a specific time. If it does not receive any message after a certain timeout, it switches to DeployBeacon state and places a new sensor. This new deployed sensor is the current sensor of the robot and it starts emitting messages. If at SearchBeacon state the robot encounters a small obstacle, ObstacleAvoidance behavior is activated and the robot avoids an obstacle. However, if it faces a large obstacle DeployBeacon is activated and the robot deploys a new sensor near the obstacle. LRV is also a maintenance algorithm. Hence, when all sensors are deployed in the ROI the robot keeps moving around based on suggested directions to detect created sensing holes due to sensor failure and fixing them by replacing new sensors.

Figure 10 is an illustration of the LRV algorithm. A robot starts from location A and travels in four geographic directions which are ordered as South, East, North, and West. In this figure, the arrowed thick line indicate robot trajectory; numbers around a node imply local weight of the four directions. Crosses are used to mark directions that are locally known (from robot's information) to be obstructed. In figure 10(a) the robot drops a sensor at location A, which then suggests it to move to the South. Following the suggestion, the robot moves to location B, drops a new sensor there,

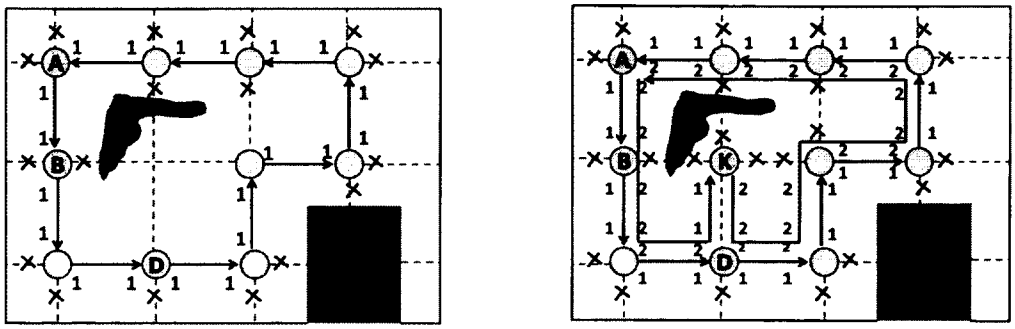


Figure 10: LRV algorithm (a) The robot deploys sensors based on suggested directions. (b) The robot keeps moving to detect and cover existing holes.

and takes it as its current sensor. It keeps traveling according to previously deployed sensors' recommendation and dropping sensors at proper positions, and finally returns to the starting point A. Since LRV is a maintenance algorithm hence the robot should keep moving around to cover created sensing holes due to sensor failures. As figure 10(b) shows, after reaching the starting point A there is no OPEN direction and the robot moves to south and increments its weight to two. When the robot arrives at location, D north direction is OPEN and it moves to the north and drops a sensor at location K. The robot keeps traveling according to suggestions and finally returns to starting point A.

In the literature, several papers [5–10] use LRV to solve the sensor deployment problem. Although these works follow the same idea, however they differ in some details. In [6] and [5] multiple robots deploy sensors while in other works a single robot deploys sensors in the ROI. In most works it is supposed that unlimited number of sensors are available to the robot, but in [6] limited number of sensors are available. In [10], [9] and [7] there is a probability of failure for deployed sensors, while in other works a failure free ROI is supposed. Most of the papers that use LRV, mainly focus on the sensor deployment problem. However, CED (Coverage, Exploration and

Deployment algorithm) [7] is a coverage, exploration and deployment algorithm. In this study the robot explores an unknown dynamic ROI and deploys sensors to create a communication network into ROI. EEL (Efficient Exploration without Localization) [8] is another coverage and exploration problem in unknown ROI that uses LRV method. In this work, the sensors are dropped off by robot as signposts to aid exploration.

LRV algorithm requires many unnecessary movements to fully explore ROI and construct a full coverage. Moreover, these extra movements lead to an extremely large number of messages sent from the robots [18, 19]. Since the next movement of the robot is guided by only one sensor, full coverage is achieved after a long time and it also requires more sensors due to a big overlapping area. In addition, it causes the problem of coverage holes or overlapping in the sensing range when the robot encounters obstacles. Meanwhile, it is not clear how the robot handles the irregular obstacles. This algorithm is not efficient in power consumption because during robot deployment, all deployed sensors stay in an active state in order to participate in guiding tasks [14].

Snake-Like Deployment approach

There are two algorithms OFRD (Obstacle-Free Robot Deployment) [13] and ORRD (Obstacle-Resistant Robot Deployment) [12] that present this SLD (Snake-Like Deployment) approach [21] to solve the deployment problem. In these algorithms, a single robot deploys sensors in an unknown and bounded ROI with unpredictable regular and irregular obstacles; however, the boundaries of a given region are known by the robot. The robot carries limited number of sensors and is equipped with a compass to remember the direction of last deployed sensor. There is neither inter-node communication nor any communication between sensors and robot. The robot deploys sensors on the vertices of a triangle grid.

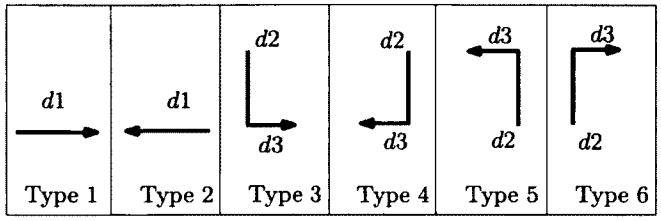


Table (a)

State	Prefer Direction 1	Prefer Dircetion 2
East	Type 1	Type 4
West	Type 2	Type 3

Table (b)

Figure 11: (a) six legal pattern movements of the robot. (b) robot's preferred direction for movement.

In SLD the robot has a deployment policy to place sensors every $\sqrt{3}r_s$. In addition to the deployment policy, a serpentine movement policy [12] guides the robot's movement. For this purpose, the robot follows six legal patterns [12, 13] for basic movements as shown in table (a) of Figure 11. When the robot moves towards the east or west direction, it uses types 1 and 2 and when it encounters a boundary or obstacle, it uses types 4 and 5 to deploy sensors on the next row. In this case, the robot will move toward the south for a distance of $3/2r_s$ and then moves toward the east or west for a distance of $\sqrt{3}/2r_s$. Using types 5 and 6, the robot moves toward the north direction when it tries to overcome the obstacle.

In the serpentine movement, the robot stays in either the East or West states that represent the direction in which the robot is moving toward. In a simple serpentine movement, the robot has two legal patterns of basic movement and chooses one pattern as its movement policy according to the priority as shown in table (b) of Figure 11. Figure 12(a) shows simple serpentine deployment by using these patterns. However, using simple serpentine movement may result in sensing holes when the robot encounters the obstacles or boundary as are shown in this figure. For this purpose the robot applies the obstacle-handling rule and the boundary-handling rule to overcome the obstacle and boundary problems. According to the obstacle-handling rule, before the robot applies the simple serpentine movement, it first checks to see

if a sensing hole exists or not. In general, if the robot stays in the East state, the six movement types have the following priorities:

$Type2 > Type5 > Type6 > Type1 > Type3 > Type4$

Otherwise, the six movement types have the following priorities:

$Type1 > Type6 > Type5 > Type2 > Type4 > Type3$

The boundary problem arises due to created small holes near the boundary. To solve this problem, the sensors are placed near the boundary. Deploying a sensor near the boundary depends on the distance between the robot and boundary. If this distance is d, then when the robot is moving towards either the east or the west and encounters a boundary, the boundary rule is as the following: "if $(\sqrt{3}/2)r_s < d < (\sqrt{3})r_s$, the robot will deploy a sensor near the boundary. Otherwise, the robot does not need to deploy a sensor near the boundary, and it simply returns to the last deployed sensor [12]." When the robot is moving towards the north or south direction and encounters a boundary, the boundary rule is as the following: "if $(1/2)r_s < d < (\sqrt{3}/2)r_s$, the robot should deploy a sensor near the boundary or obstacle. Otherwise, the robot does not need to deploy any sensor and it simply returns to last deployed sensor [12]." Figure 12(b) illustrates sensor deployment using serpentine movement policy, boundary handling rule, and obstacle handling rule to deploy sensors in the same ROI as figure 12(a). As it is obvious, there are no sensing holes near the boundaries and obstacles.

In [12] a boundary handling rule is presented to solve the boundary problem while there is no such rule in [13]. However, both SLD algorithms do not guarantee full coverage because the robot sticks in dead ends due to obstacles or early dropped sensors, since the algorithms do not support any method to get out of dead ends [18,21]. Moreover, it is not clear under what conditions the algorithms terminate [21] and neither of these algorithms support multiple robots [18,19].

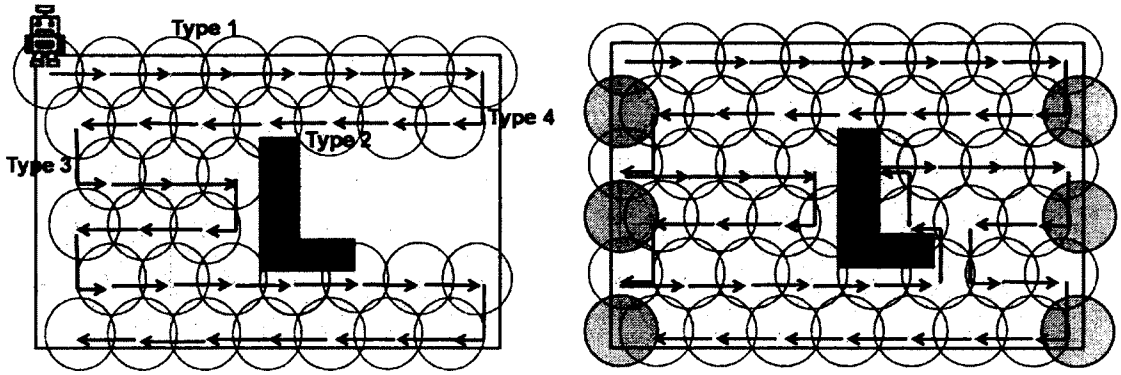


Figure 12: SLD algorithm (a) Sensor placement using simple serpentine movement policies. (b) Sensor deployment using simple serpentine movement policies, obstacle rule, and boundary rule.

Spiral Deployment approach

OFPE (Obstacle-Free and Power-Efficient deployment algorithm) [14] is a single robot algorithm that the robot is equipped with both GPS and compass. Moreover, a radio system is embedded in the robot to communicate with deployed sensors. It is also assumed that the ROI is unknown and there are obstacles in the field that the robot is able to discover them as it moves closer to them.

In this algorithm, the robot follows a node placement policy and a spiral movement policy to deploy sensors. Based on node deployment policy the robot moves and deploys a sensor for every $(\sqrt{3})r_s$. To achieve spiral movement the robot moves in one of six cardinal directions. A simple sensor deployment by using spiral movement policy is illustrated in Figure 13.

To overcome the obstacle problem, the robot applies an obstacle surrounding movement policy. For this purpose, the robot uses two states namely, the steady and obstacle states. In the steady state, the robot deploys sensors using the spiral movement and sensor deployment policies. It switches from steady to obstacle state when it encounters an obstacle and moves to deploy sensors according to the obstacle

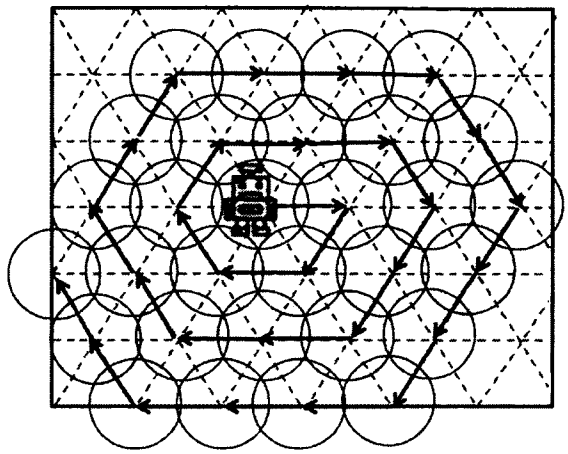


Figure 13: Simple sensor deployment using spiral movement policy.

surrounding movement policy. According to this policy, the robot always moves around the obstacle in a clockwise direction until all around the obstacle is covered by sensors. Sensors that have been deployed by the robot in obstacle state are considered virtual obstacles and when the robot encounters with them, it switches from steady to obstacle and treats them as virtual obstacles. If the robot does not deploy any sensor for a period of time, this means that each location was monitored by at least one sensor and the robot terminates the deployment process.

This algorithm does not guarantee full coverage because there is no dead end recovery policy, so the algorithm is very likely to be stuck at dead ends due to obstacles or early dropped sensors [18, 19]. The boundary problem is not addressed in this algorithm and it does not support multiple robots either.

Back-Tracking Deployment approach

BTD (Back-Tracking Deployment) algorithm [19] is a sensor deployment algorithm that presents a dead end recovery policy to provide a back tracking method for situations that the robot is stuck due to early deployed sensors or obstacles. In this algorithm, a number of robots which are equipped with GPS are scattered randomly

in an unknown bounded ROI. They are preloaded with static sensors and are able to detect obstacles and boundaries.

In single robot scenario each deployed sensor keeps three pieces of information sequence number, color and back pointer. The color of a sensor is either white or black. A sensor is white if it has an empty neighbor and black otherwise. The sequence number is assigned to sensors by dropping robots. By exchanging this sequence numbers via hello message, the sensors know their neighbors. The back pointer points to the location of the last white deployed sensor. Each sensor periodically sends a Hello message containing its sequence number, color and back pointer. On the other hand, the robot moves based on pre-ordered directions (West, East, North, and South) and deploys a sensor for every $\sqrt{2}r_s$ on a square grid. Whenever the robot is stuck due to early deployed sensors or obstacles it backs track. For this purpose, it takes the back pointer stored in its current sensor as the destination and sends a message to all adjacent sensors to ask their back pointers. Then, it moves to the next neighboring sensor with the lowest sequence number whose back pointer location is the same as the robot's destination. This is a short cut method for back tracking. Figure 14 shows BTD algorithm in a single robot scenario. In these figures, sensor 18 denotes the location where there is a dead end and white sensors (7 and 8) denote the locations where there are empty neighbors. In figure 14(a), the robot is stuck after deploying sensor 18. In figure 14(b) the robot back tracks to a white sensor (number 8) and resumes deploying sensors.

In a multi-robot scenario each robot has a unique ID and the sensors need to store the ID of dropping robots in addition to other information. So, sensors can be distinguished by the value pair (robot ID, sequence number). In multi robot version of this algorithm, each robot follows the BTD algorithm as if it was the only robot in the ROI. In a dead end situation, if the robot cannot find a back pointer on its current sensor, it will select a back pointer with the largest sequence number stored

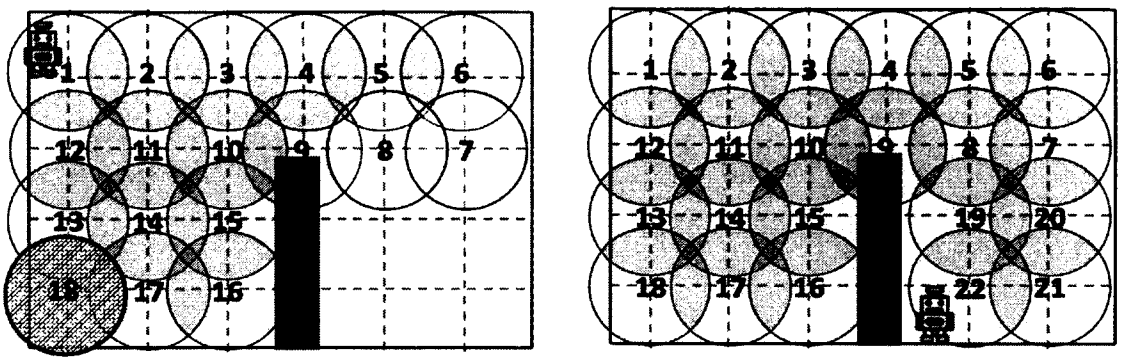


Figure 14: BTD algorithm (a) the robot is stuck at dead end. (b) The robot does back tracking and resumes deploying sensors to complete coverage.

in the neighboring sensors and back tracks to the destination.

BTD is a maintenance algorithm, however in the presence of sensor failures it may lose its ability to terminate in some situations [19]. Moreover, this algorithm supports neither an obstacle handling rule nor a boundary handling rule to overcome the obstacle or boundary problem. The other problem which is not addressed in this algorithm is load the balancing problem. This problem arises when multiple robots are placing sensors and some of them stop moving early during the deployment because of dead end and back pointer elimination by another robot, while others continue to explore almost the entire ROI [19].

Robot Deployment in concave region

In the proposed algorithm in [26] it is assumed that the ROI is unknown and bounded with no obstacles. A single robot which is not equipped with GPS deploys sensors on the monitoring region. However, an instrument is attached to the robot that can evaluate the distance to the boundary.

In this new approach the ROI is divided to vertical and horizontal paths and sensors are deployed in three phases: left phase, bottom phase, and regular phase.

The distance between sensors on the vertical path is $(\sqrt{3})r_s$ and the adjacent sensors on the horizontal path are separated by the distance of $3r_s$. In the left phase, the sensors are deployed along the left boundary. In the bottom phase, the robot deploys sensors using moving vectors, and the distance to the right boundary is evaluated. After the first two phases, the horizontal path, vertical path, and the beginning of the regular phase are known. In the regular phase the robot follows two moving vectors: MV-R2L and MV-L2R. The robot deploys sensor by MV-R2L and MV-L2R in turn and when the final sensor is deployed the robot reaches the Exit.

This algorithm does not guarantee full coverage because dead end recovery rule is overlooked.

FOCUSED Coverage

The algorithm in [18] addresses the FOCUSED coverage (F-coverage) problem. In F-coverage, sensors surround a point of interest (POI) and maximize coverage radius. This algorithm applied for an unknown bounded ROI that the location of POI is given to the robots. Robots are aware of their own locations and are able to detect nearby obstacles. They have limited capacity and load sensors from different fixed locations (called base points) which are located out of the ROI.

In this algorithm, a robot deploys sensors on a hexagon grid layer by layer. The early deployed sensors save the status of neighboring sensors and adjacent vertices in the grid. They report the empty vertices to sensors which are located on outer layers of the gird. The robots enter the ROI and move toward POI. When they get in touch with a deployed sensor, the sensor recommends the best empty location. So, a robot moves to drop a sensor at the recommended location. After finishing the current task, the robot returns to one of the base points to reload sensors and then re-enter the ROI to cover other empty locations.

Flying Robot

In [16] and [17] a flying robot (helicopter) deploys sensors on a desired network topology like star, grid or random. The algorithm has three phases. First an initial deployment is executed and then the connection topology of the deployed network is measured and compared to the desired topology. If there are some gaps in the network, the next step is repairing the connectivity. So, the new deployment locations that repair the desired topology will be computed and the new sensors will be emplaced in these locations.

2.3.5 WSRN algorithms and critical areas

As we discussed earlier, a complex ROI contains critical areas. In this section we briefly discuss why WSRN algorithms such as LRV, SLD, spiral movement algorithm, and BTD cannot cover critical areas.

In LRV, the robot is equipped with two 180 field-of-view planar laser range finders positioned back-to-back (equivalent to a 2-D omnidirectional laser range finder) [10]. So, we can assume the robot has visibility to detect critical areas. However, according to the algorithm the robot only moves on the vertices of a square grid based on the received message of nearby sensors. So, the robot never gets in the critical areas since they are out of the defined grid, and senors never suggest them.

In spiral movement algorithm, the ultrasonic or laser sensor equipped on the robot is able to detect the distance between the farthest point of an obstacle boundary and the robot [14]. We assume that the robot has visibility to detect critical areas. However, the robot moves in six cardinal directions to deploy sensors on a hexagon grid. Moreover, the robot gets stuck in dead end since there is no dead end recovery rule. So, if the robot detects critical areas it cannot get in them, and even if it enters with arbitrary movement, it will be stuck in a dead end.

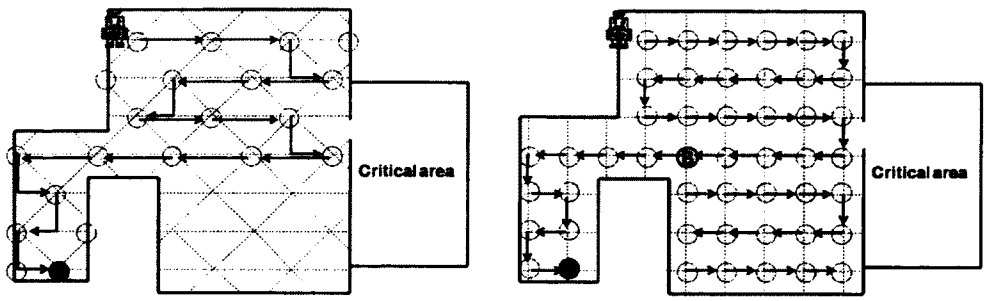


Figure 15: (a) SLD and critical areas (b) BTD and critical areas

In SLD, there is no discussion on robot's visibility, however we assume the robot has visibility. According to SLD algorithm [13] the robot only walks on the triangular grid based on six types of pattern movement. So, it never has the chance to gets in the critical areas which are out of the grid. Figure 15 (a) shows an example of SLD in a ROI containing critical areas. In this figure the circles denote deployed sensors. The black one denotes the location where the robot gets stuck in a dead end. As it is shown in this figure the critical area is not covered.

In BTD, Robots are able to detect obstacles and boundaries of the ROI by attached scanning laser range-finder [19]. So, we assume that the robot has sufficient visibility to detect critical areas. In this algorithm the robot moves to four directions (West, East, North and South) to deploy sensors on the vertices of a square grid. Hence, it does not have any plan to enter the critical areas. However, we assume when the robot detects critical areas it does an arbitrary movement and enters them. But, arbitrary deployments may cause sensing holes in the entrance of critical areas, and result in a new grid which has conflict with the old one. Figure 15 (b) shows an example of BTD in ROI containing critical areas. In this figure the circles denote deployed sensors, and the black sensor denotes the location where the robot gets stuck. However, the robot backs track to location of sensor B and resumes deploying other sensors, but when the algorithm terminates the critical area is not covered yet.

In conclusion, none of the existing algorithms in the sensor deployment by the robot achieve full coverage even in a simple ROI without any critical area, let alone deploying sensors inside critical areas.

Chapter 3

Sensor Deployment in a Simple Orthogonal ROI

3.1 Introduction

As we discussed in the previous chapter, all the existing algorithms for sensor deployment by the robot do not achieve full coverage, even in a simple ROI. In this thesis, we use BTD [19] algorithm as the base for our solution algorithms. We chose BTD because it is the only algorithm that has a back tracking method for the situations when the robot gets stuck in a dead ends. However, in BTD algorithm the boundary problem is intentionally ignored. In this chapter we propose two algorithms, A1 and A2, to solve the boundary problem of BTD. Both algorithms work in the same way, but they only differ in their boundary handing rule. In algorithm A1 a straightforward boundary handling rule is proposed to solve the boundary problem. Using this rule many extra sensors are deployed. So, in algorithm A2 the boundary handling rule is improved to decrease the number of extra sensors. In the following sections, first we briefly explain what a BTD algorithm is and then discuss algorithms A1 and A2 in detail.

3.2 BTD (Back Tracking Deployment) Algorithm

BTD algorithm [19] with single robot scenario in a failure free environment is used as the basis of our work. In BTD the robot is equipped with GPS and carries all static sensors as preloaded. It deploys sensors in an unknown bounded environment with the obstacles. In addition, the robot is able to detect the boundaries/obstacles.

When the robot enters the ROI, it computes a unique virtual square grid of edge length $\sqrt{2}r_s$. If a grid point is not occupied by any sensor, then it is considered as an empty point. All grid points are initially empty and the robot explores the ROI and drops a sensor at each empty point. The sensor placed at the grid point where a robot is currently located is called the current sensor of this robot.

In this algorithm each sensor keeps three pieces of information: ID, color, and back pointer. The sensor ID is assigned by the robot and indicates the order in which the sensors were placed. Each sensor updates its color and back pointer dynamically as it communicates with other sensors. A sensor is white if it has an empty neighbor and black otherwise. The back pointer points to the last white sensor that this robot deployed. Sensors periodically broadcast a Hello message containing their ID, color and back pointer.

The color of each sensor indicates whether there are empty neighbors around it or not. A white sensor has empty neighbors; so, when the robot gets stuck in a dead end it can back track to a white sensor to resume deployment. If there is no white sensor the entire ROI is covered and the algorithm terminates. According to the received Hello messages, the sensors update their color. If a sensor does not receive a message from one direction it assumes that the direction is open, unless the sensor is informed of the existing boundary or obstacle on that direction at the time of deploying by the robot.

Moreover, the sensors use the information of the received Hello messages to update

their back pointer. A sensor may receive several messages from its neighbors. Among the received messages, the sensor only considers the received message from the closest neighbor whose ID is smaller than sensor's. To find the closest neighbor, the sensor computes the subtraction of its ID from the neighbor's, and the closest neighbor has the least subtraction. The sensor considers the ID of the closest neighbor as its back pointer if it is white, and the back pointer of the closest neighbor otherwise.

In this algorithm, the order rule (directional order) is West, East, North, South, and the deployment policy is emplacing a sensor for every $\sqrt{2}r_s$ on the vertices of a square grid. For forward moving, the robot moves to grid points in open directions, based on the order rule until a dead end is reached. A direction is closed if it is obstructed by an obstacle, a boundary, or an early deployed sensor, and open otherwise. A dead end occurs at a grid point in which all four directions are obstructed, so the robot can no longer move forward.

In a dead end situation, the robot will back track to the nearest white sensor along its backward path, and resume exploration of the ROI and sensor dropping from there, or stop moving if no such a sensor exists. For back-tracking, the robot finds its destination by taking the back pointer stored on its current sensor. Then, the robot moves to the next adjacent sensor with the lowest Id whose back pointer is the same as the robot's destination. The robot keeps moving in this way until it reaches the back pointer which is a white sensor, and the robot can resume deploying sensors from there. Figure 16 illustrates deployed sensors in a simple ROI using the BTD algorithm. The bold circle denotes the first deployed sensor and the highlighted band shows the uncovered areas close to the boundaries.

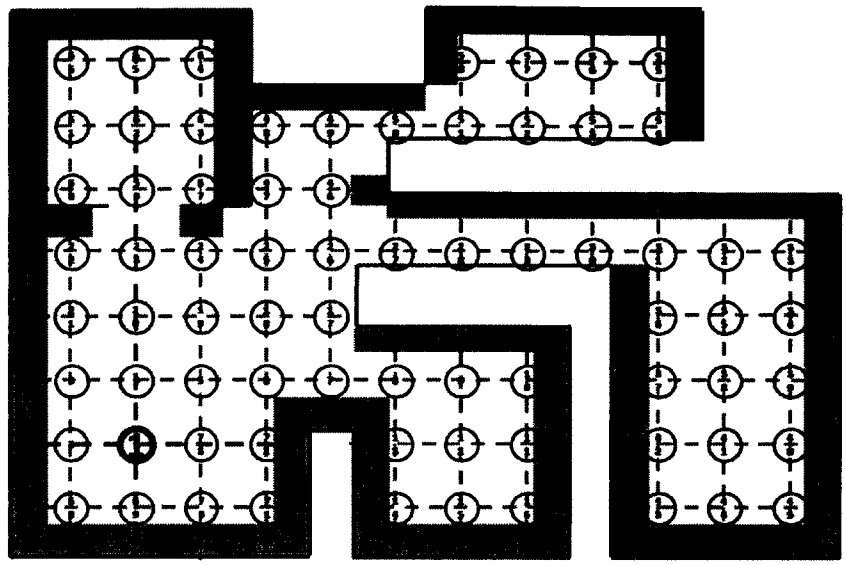


Figure 16: BTD in a Simple ROI. The highlighted band shows the uncovered areas close to boundaries.

3.3 Algorithms A1 and A2

As we discussed, BTD algorithm does not guarantee full coverage since it does not have any rule to overcome the boundary problems. As it is shown in Figure 16, when algorithm terminates there are still uncovered areas close to the boundaries. So, we propose algorithms A1 and A2 to solve the boundary problem of BTD algorithm. Unlike BTD, in these algorithms the robot does not have a GPS. It is equipped with a compass and has the visibility to detect boundaries and predeployed sensors within distance $2r_c$. It deploys preloaded sensors in an unknown but bounded ROI which is an arbitrary orthogonal two dimensional space. However, we assume the ROI is simple, and there is no critical area in it. The robot follows *Regular Movement* to deploy preloaded sensors on the verities of the grid. Regular Movement is moving for every $\sqrt{2}r_s$ in one of four directions with a priority of West, East, North and South. So, we say a vertex v_i belongs to the square grid only if it is accessible by Regular

Movement from the starting point.

In algorithms A1 and A2, beyond *Regular Sensors (R-sensor)*, which are deployed on the vertices of the square grid, there are also *Boundary sensors (B-sensor)*. Boundary sensors (B-sensor) are located near the boundaries to cover uncovered areas, but not on any vertices of the square grid. All sensors keep some pieces of information including color, Id, back pointer, state, and their locations. Color, Id and back pointer are the same as in BTD, but state of each sensor can be Regular or Boundary, depending on the location that a sensor is dropped. Moreover, each sensor is informed of its location in time of deploying by the robot. R-sensors periodically send a Hello message containing their information. They also update their color and back pointer according to the received information from other R-sensors. On the other hand, B-sensors do not participate in the algorithm. They only send a Hello message to neighboring sensors to keep the network connected, but they do not update their color and back pointer. In the following sections, we first study the boundary problem and then propose our algorithms to solve this problem.

3.3.1 Boundary Problem

The boundary problem arises when uncovered areas are created near the boundaries. The boundary problem occurs when the robot movement is obstructed by the boundary, and using Regular Movement the robot cannot deploy the sensors near them. In BTD algorithm boundary problem is intentionally ignored, and it is stated that "boundary effects are ignored as they have no impact on the algorithm design in any essential way except that a robot may need to drop an extra sensor (if necessary) when its movement is obstructed by the ROI boundary" [19]. Figure 17(a) illustrates a ROI which is covered using BTD algorithm. Light gray circles denote R-sensors with sensing range of r_s. As it is obvious in this figure, there are uncovered areas (unshaded area) near the boundaries. Let B denote the boundary, and d the distance

between the current sensor and a point b on B ($b \in B$). In a square grid, if the distance (d) between the current sensor and the boundary ($b \in B$) in one of four directions North, South, East or West is less than $\sqrt{2}r_s$ and greater than $\sqrt{2}/2r_s$ then *boundary holes* will be created near the boundaries.

Definition 2 *Let d be the distance from a vertex on the grid to a boundary in either one of the four directions: North, South, East, West. If* $\sqrt{2}/2r_s < d < \sqrt{2}r_s$, *we observe that there will still be an uncovered area left after a robot has deployed regular sensors. We call such uncovered area a boundary hole.*

A simple solution to solve this problem is to deploy B-sensor(s) whenever a boundary hole is detected. If a boundary hole is detected in more than one direction, the robot deploys B-sensor(s) based on an order rule. According to the order rule directions have a priority which is West, East, North, and South. For example, if two boundary holes are detected in north and east directions, first the boundary hole in east is covered. Right after deploying a B-sensor, the robot should return to the last deployed R-sensor to resume it's Regular Movement. In Figure 17(b) the robot follows BTD algorithm and deploys B-sensors whenever a boundary hole is detected. In this figure, light gray sensors denote R-sensors and the dark gray ones denote B-sensors. The robot drops sensor 1 and then checks all four directions to detect the boundary hole. A boundary hole is detected in North, so the robot deploys B-sensor 2 to cover it. Then it returns to R-sensor 1 to continue its Regular Movement. After deploying R-sensor 3 and B-sensor 4, two boundary holes are detected in north and east at location of R-sensor 5. So, the robot drops B-sensors 6 and 7 near east and north boundaries respectively to cover these holes. The robot keeps deploying until the entire ROI is covered by sensors. However, as it is shown in this figure, there are still uncovered areas on the corners. Figure 17(c) illustrates another example using the previously mentioned rule to cover boundary holes. However, an uncovered area

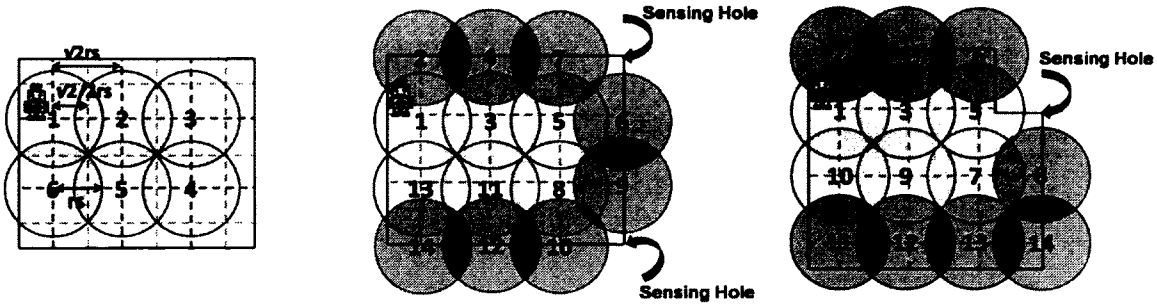


Figure 17: (a) Deployment using BTD algorithm. (b) and (c) Deployment using BTD algorithm and a simple boundary handling rule in two different ROI. There are uncovered corner holes in the ROI.

is still left on the corner. The created holes on the corners are named *corner holes.*

Definition 3 *Let d be the distance from a vertex on the grid to a boundary in either one of the four directions: Northeast, Southeast, Northwest, Southwest. If $r_s < d < 2r_s$, we observe that there will still be an uncovered area left after a robot has deployed regular sensors. We call such uncovered area a corner hole.*

So, to have a proper boundary handling rule both boundary and corner holes should be considered.

3.3.2 Algorithm A1

In Figures 17(b) and 17(c), corner holes are uncovered because the robot only checks four directions: North, South, West, and East. The most straightforward solution to cover corner holes is to check boundaries in Northeast, Northwest, Southeast, and Southwest directions as well. In this case, if $r_s < d < 2r_s$ there exist a corner hole that needs to be covered. As a result, the robot starts checking boundary problem from north in a clockwise direction, and keep checking all the eight directions. The proposed boundary handling rule to cover all the boundary/corner holes is as follows:

- Check the boundary problem for all eight directions, starting from north in a clockwise direction.
- Whenever a boundary or corner hole is detected push the location of a B-sensor, which can cover detected hole, to a *B-sensors list.*
- After checking all eight directions, pop the locations of each B-sensor from the B-sensors list, deploy it, and then remove it from the list.
- When the B-sensor list is empty, return to the last deployed R-sensor.

Algorithm A1 follows BTD algorithm to deploys R-sensors on the vertices of a square grid, and back tracking when the robot is stuck in a dead end. Moreover, it applies to the proposed boundary handling rule to detect and cover boundary and corner holes. In this algorithm, after deploying each R-sensor the robot checks the boundary problem for the deployed R-sensor starting from north in clockwise direction. If a boundary or corner hole is detected, the robot pushes the location of a B-sensor to a B-sensor list. The robot starts deploying B-sensors when checking all the eight directions is done. The robot pops the location of a B-sensor from the B-sensor list, deploys it, and then removes it from the B-sensors list. When the B-sensor list is empty all the B-sensors are deployed, and the robot returns to the last deployed R-sensors to resumes Regular Movement. In the following section we show some examples using algorithm A1 to cover the ROI.

Figure 18 illustrates an example of deployment using the proposed boundary handling rule to solve the boundary problem. In Figure 18(a), after deploying R-sensor 1, boundary problems are detected in both north and northeast directions. The robot drops B-sensors 2 and 3 to solve these problems respectively. After deploying R-sensor 4 and B-sensors 5 and 6 there is no hole to be covered by B-sensor 3 and it is actually an extra sensor. However, at the time of deploying B-sensor 3, the robot could

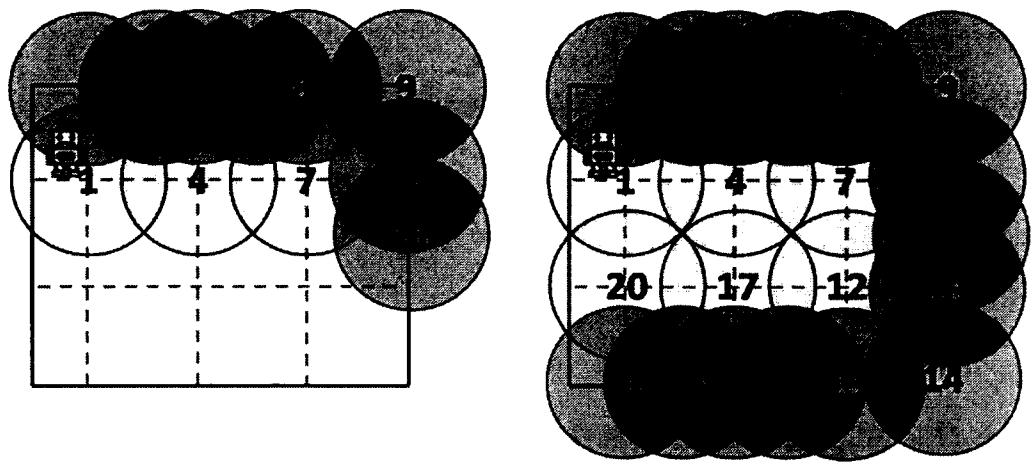


Figure 18: (a) At location of R-sensor 7 four B-sensors 8, 9, 10 and 11 are dropped while B-sensors 8 and 11 are extra. (b) Many extra B-sensors are deployed in the ROI after achieving full coverage.

not predict deploying B-sensor 5. On the other hand, after deploying R-sensor 7 the robot detects a corner hole in northeast and drops B-sensor 9 which is really needed and is not an extra B-sensor. At this location, another corner hole is detected in the southeast and B-sensor 11 is dropped to cover it. However, in Figure 18(b) after deploying R-sensor 12 and B-sensor 13, there is no hole to be covered by B-sensor 11 and this B-sensor is extra again. In fact the robot dropped B-sensor 11 because it could not predict deploying sensors 12 and 13 at time of deploying R-sensor 7. Both these two figures illustrate the fact that extra B-sensors are introduced after achieving full coverage of the ROI.

Figure 19 illustrates another example of deployment following the proposed boundary handling rule. Actually this example shows a worse situation in which there is no corner hole in the ROI, while applying this boundary handling rule leads to deploying many extra B-sensors in the ROI because of detecting fake corner holes. Fake

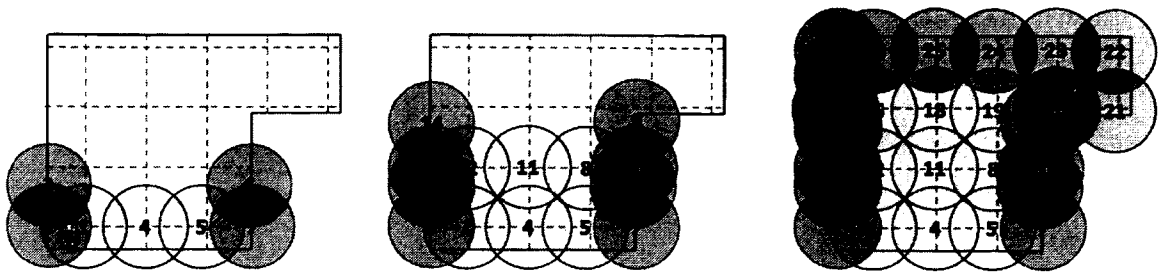


Figure 19: (a) The robot deploys B-sensors 3 and 6 to cover detected corner holes. (b) The robot deploys B-sensors 9 and 14 to cover corner holes. (c) Full coverage is achieved while many extra B-sensors are deployed.

corner holes are covered by R-sensors and B-sensors which will be deployed in future. In Figure 19(a) the robot drops R-sensor 1 and B-sensors 2 and 3 near the west and northwest boundaries respectively. In Figure 19(b), at location of R-sensor 8 the robot detects a corner hole in the northeast and drops B-sensor 9 to cover it. However, this B-sensor is extra because, after deploying R-sensor 20, the corner hole will be covered by it. As shown in figure 19(c), to achieve full coverage, many extra B-sensors are deployed in the ROI.

The proposed boundary handling rule covers all detected boundary and corner holes. However, it is not an efficient rule because of deploying many extra B-sensors. The problem of this rule is that the robot cannot distinguish fake corner holes from the real ones, and has to drop a B-sensor whenever a corner hole is detected.

3.3.3 Algorithm A2

Improved Boundary Handling Rule

As discussed, the problem of algorithm A1 is that it deploys many extra B-sensors since the robot cannot distinguish fake corner holes from the real ones. In this section we propose an improved boundary handling rule to solve the boundary problem, without deploying extra B-sensors.

In a square grid, the grid points are indicated as the vertices of created squares. For each grid point we define eight neighboring grid points which are located in North, South, West, East, Northeast, Northwest, Southeast, and Southwest. The adjacent grid points to the East, West, North, and South of the current grid point are called it's *Main Neighbors*. The grid points to the Northeast, Northwest, Southeast, and Southwest of the current grid point are called it's *Corner Neighbors*. Similarly, the sensors located on the above-mentioned grid points are called *Main Neighbors (main sensor)* or *Corner Neighbors (corner sensor)*, respectively. Figure 20 represents a sensor and its main and corner neighbors. In this figure a sensor C denotes the current sensor, while sensors 1, 2, 3, and 4 denote main neighbors and sensors 5, 6, 7, and 8 denote corner neighbors of the current sensor.

As the deployment policy of algorithm BTD stated, from the location of the current sensor, the robot can only deploy main neighbors of the current sensor by using Regular Movement, while the corner neighbors cannot be deployed from this location by Regular Movement. For example, in Figure 20, using Regular Movement the robot can only deploy main neighbors 1, 2, 3 and 4 from the location of current sensor, while corner neighbors 5, 6, 7 and 8 cannot be deployed from this location. Moreover, each corner neighbor of the current sensor has two *associate neighbors*. The associate neighbors of the northeast corner neighbor are north and east main neighbors of the current sensor. The associate neighbors of northwest, southeast and southwest corner neighbors are north and west, south and east, and south and west main neighbors respectively. Each corner neighbor can be deployed from the location of one of its associate neighbor. For example in Figure 20 the associate neighbors of corner neighbor 5 are sensors 1 and 2, and sensor 5 can be deployed from the location of one them. Moreover, as it is shown in this figure, all main and corner neighbors of the current sensor are located on vertices of a big square which could be divided to four sub squares. Each sub square, having the current sensor and one corner neighbor

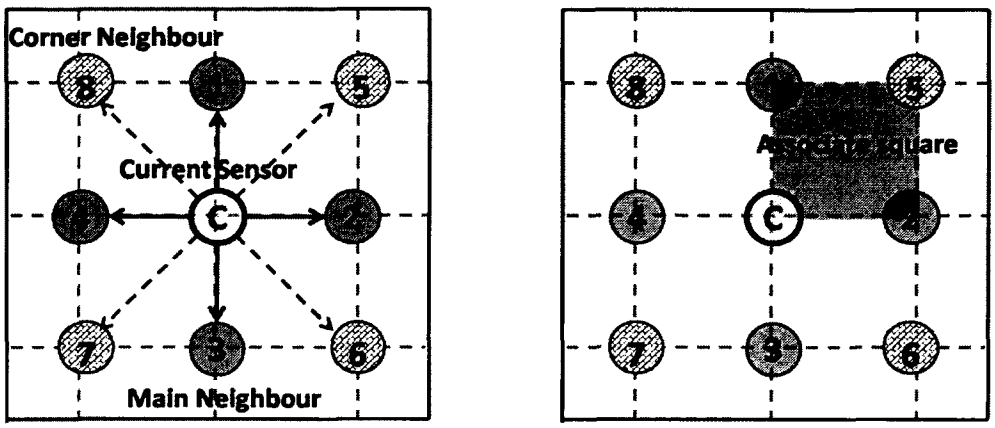


Figure 20: (a) Sensor C denotes the current sensor. Sensors 1, 2, 3, and 4 denote main neighbors of the current sensor, and sensors 5, 6, 7, and 8 denote corner neighbors. (b) Associate square of northeast sensor (sensor 5) is highlighted.

on its vertices is considered as *associate square* of that corner neighbor. For example, the associate square of northeast corner neighbor is the one containing the current sensor and northeast corner sensor on its vertices. In Figure 20(b) the associate square of northeast corner sensor is highlighted.

According to the distance between the current sensor and the boundary, boundary and corner holes may be created when main and corner neighbors are not deployed because of the existing boundaries. However, the created corner holes may be fake since they might be covered after deploying other R-sensors or B-sensors. So, to avoid deploying extra B-sensors, the robot should distinguish real corner holes from the fake ones. Absence of a corner sensor may create a corner hole in its associate square. However, in a simple ROI, if one of the associate neighbors of corner sensor drops a B-sensor in the associate square, the created corner hole will be covered. For example, in Figure 19(b) from the location of R-sensor 1 a corner hole is detected in northwest, since the northwest corner sensor cannot be deployed because of the

existing boundary, and the distance between the R-sensor 5 and the boundary in northwest is $r_s < d < 2r_s$. However, since its associate neighbor in north is accessible, the robot does not need to deploy a B-sensor to cover this fake corner hole. As it is shown in this figure, this corner hole will be covered after deploying R-sensor 9 (its associate sensor) and B-sensor 10.

The improved boundary handling rule works as the following. After deploying each R-sensor, the robot starts checking boundary problem from north in a clockwise direction. Whenever a boundary hole is detected, the robot computes the location of a B-sensor, which can be dropped to cover it, and stores that location in a B-sensors list. After detecting all boundary holes, the robot has a B-sensor list containing the location of all B-sensors. So, it pops their locations one by one from the B-sensor list, and deploys them in the appropriate locations. When the B-sensor list is empty, it means that all boundary holes are covered, and the robot can return to the last deployed R-sensor to do Regular Movement. In conclusion, we propose the following boundary handling rule to overcome the boundary problem:

- Check the boundary problem for all eight directions, starting from north in a clockwise direction.
- A boundary hole is detected if d in North, East, South, or West is $\sqrt{2}/2r_s < d < \sqrt{2}r_s$.
- A corner hole is detected if d in Northeast, Southeast, Southwest, or Northwest is $r_s < d < 2r_s$, and neither of associate sensors drop a B-sensor in associate square.
- Whenever a boundary or corner hole is detected push the location of a B-sensor, which can cover detected hole, to a B-sensors list.

- After checking all eight directions, pop the locations of B-sensors from the B-sensors list and deploy them one by one.
- When B-sensor list is empty, returns to the last deployed R-sensor.

Summary of Algorithm A2

In this algorithm, the robot applies BTD algorithm for deploying R-sensors on the vertices of a square grid and back tracking whenever it is stuck in a dead end. After deploying each R-sensor, the robot executes improved boundary handling rule. Based on this rule, the robot checks all eight directions starting from north in a clockwise direction, trying to detect boundary problems. Whenever a boundary hole is detected, the robot pushes the location of a B-sensor to B-sensor list. After checking all directions, the robot starts deploying B-sensors according to the stored locations in B-sensors list. After deploying each B-sensor, the robot removes its location from the B-sensors list. As long as there is a boundary problem around the current sensor, the robot does not deploy next R-sensor before the boundary problem is solved. So, we can say deploying a B-sensor has a higher priority than deploying a R-sensor. When the B-sensor list is empty, the robot returns to last deployed R-sensor to do Regular Movement and deploy next R-sensor based on the order rule described in BTD. Figure 21 illustrates the same ROI as Figure 16 covered by using algorithm A2. As it is shown in this Figure, the highlighted band of Figure 16 which was not covered by algorithm BTD, is now covered in this algorithm by B-sensors (illustrated as dark gray circles). The algorithm pseudo which is based on these descriptions is as follows.

Algorithm A2
Main-Direction (North, East, South, West)
Corner-Direction (Northeast, Southeast, Southwest, Northwest)
Wake up
Locate first R-sensor
while *there are uncovered areas* **do**
  if *a boundary is detected* **then**
    Check-Boundary-Handling-Rule ()
  end
  if *Dead-End* **then**
    Back-Track()
  end
  else
    Do regular movement and drop a R-sensor.
  end
end

Check Boundary-Handling-Rule ()
for *8 cardinal Directions starting from North* **do**
  if *in Main-Direction* $\sqrt{2}/2rs < d < \sqrt{2}$ **then**
    push the location (according to Robot's local coordinate system) of B-sensor in B-sensors List.
  end
  if *in Corner-Direction* $rs < d < 2rs$, **and** *no one of associate sensors drop (already dropped or will drop) a B-sensor in associate square* **then**
    push the location (according to Robot's local coordinate system) of B-sensor in B-sensors List (if the robot cannot see the associate sensors, it assumes they do not drop a B-sensor in associate square).
  end
end
while *B-sensors List is not empty* **do**
  Drop a B-sensor.
end
Return to last deployed R-sensor.

Back-Track ()
if *back-pointer of the current sensor is not NULL* **then**
  back track to back-pointer of the current sensor
end
else
  Terminate the algorithm
end

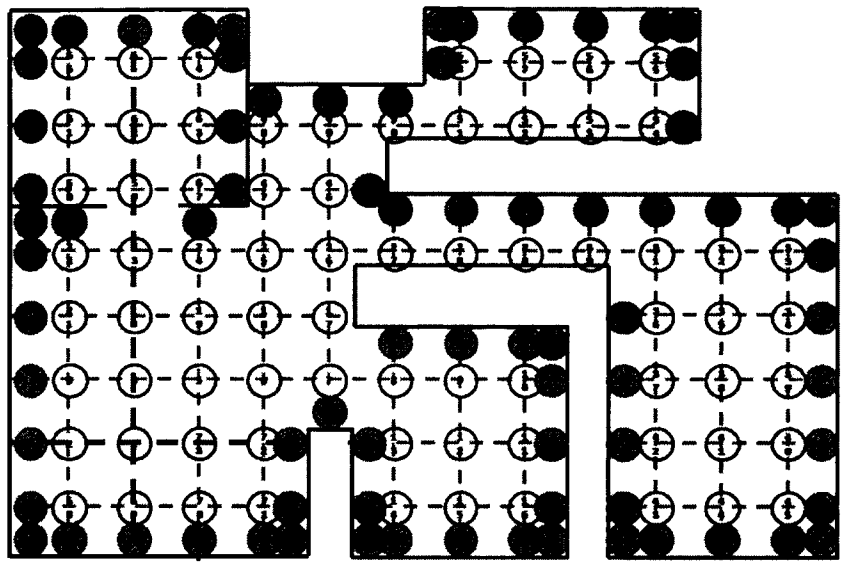
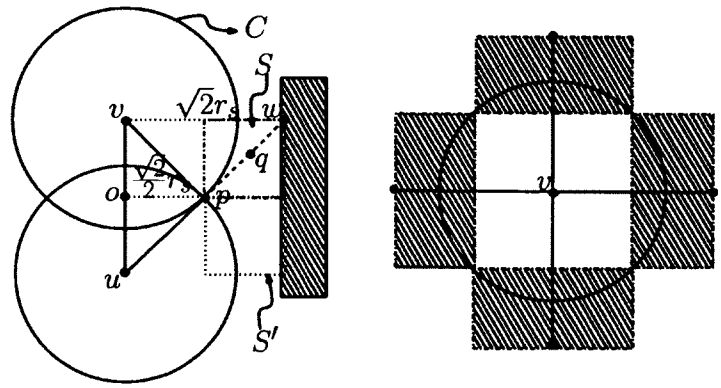


Figure 21: Algorithm A2 in a simple ROI

3.4 Correctness of Algorithms A1 and A2

As we mentioned before, algorithm BTD cannot cover critical areas. To solve that problem algorithms A1 and A2 deploys B-sensors in any area that could potentially cause a hole. In order to study the correctness of our algorithms it is important to characterize such types of regions, and show how the robot could locally find them. After that, we will prove that, using our algorithms, the robot is able to detect any potential hole and handle it.

First we denote such region as *Potential boundary holes.* This happen when, a vertex of the grid is at distance d, $\frac{\sqrt{2}}{2}r_s < d < \sqrt{2}r_s$, in any of four directions North, South, East or West. To see more clearly what these areas are, consider two vertical adjacent regular sensors u and v (figure 22 (a)). Since R-sensors are placed always at distance $\sqrt{2}r_s$ from each other, we know that $d(\overline{uv}) = \sqrt{2}r_s$. Let o be the middle point of $\overline{uv}$ and p be a point, on the bisector of $\overline{uv}$, which is at distance r_s from both u and v. Note that $d(\overline{op}) = \frac{\sqrt{2}}{2}r_s$. Now, consider the point q on $\overline{pw}$ where w is at distance



(a) (b)

Figure 22: (a) The black rectangle shows an obstacle in east direction of R-sensor v. (b) Black squares show eight squares around R-sensor v that have boundary hole's potential.

$\sqrt{2}r_s$ from v in the eastern direction. This point cannot be covered by u because it is at distance greater than r_s. Furthermore, since the segment $\overline{uw}$ is tangent in p to the sensing circle of v, then q is at distance greater than r_s from v too. This implies that q cannot be covered neither by u nor v. Thus, if there is no R-sensor on w, q will not be covered. This situation could happen only if the boundary of the ROI intersects the upper side of the square S. In other words, if the robot sees the boundary of the ROI at distance d, $\frac{\sqrt{2}}{2}r_s < d < \sqrt{2}r_s$, in the eastern direction. Hence, in that case the square S is a potential boundary hold for v. In a similar way, the square S' could be a potential boundary hole for u. In fact, performing this analysis in the four directions (North, South, East or West) we could find eight of such squares (figure 22 (b)) depending on the shape of the ROI.

The second type of potential uncovered region is called *Potential corner holes.* They appear when the grid point is at distance d, $r_s < d < 2r_s$ in any of four directions, Northeast, Southeast, Northwest or Southwest. As shown in Figure 23 (a), the point q cannot be covered by any of the R-sensors u, t nor v. This is because it is at distance greater than r_s from v and on the tangent in p of the sensing circle

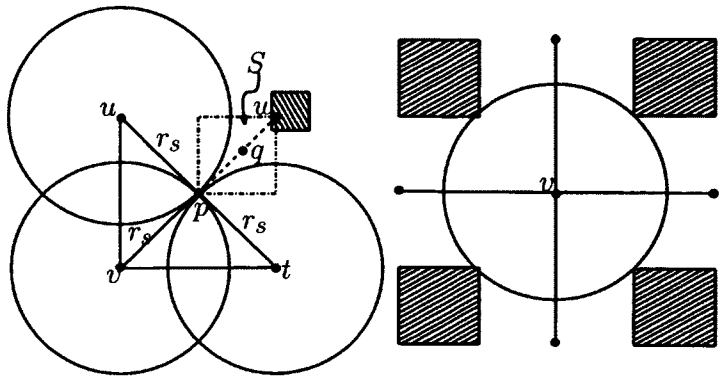


Figure 23: (a) The black square shows an obstacle in Northeast direction of R-sensor v. (b) The black squares show four squares around R-sensor v that have corner hole's potential.

of u and t. Thus, if there is no R-sensor at point w then q will not be covered. This situation could happen only if the boundary of the ROI intersects the segment $\overline{pw}$, in other words, if the robot see the boundary of the ROI at distance d, $r_s < d < 2r_s$, in the northeastern direction. Hence, in that case, the square S is a potential corner hole for v. As we did for boundary holes, performing a similar analysis in the four directions (Northeast, Southeast, Northwest or Southwest) we could find four possible potential corner holes for each vertex of the grid (Figure 23 (b)).

Now we are able to start proving that our algorithms completely cover the ROI.

Lemma 4 *The proposed boundary handling rules alow the robot to detect every boundary and corner hole.*

Proof. Since algorithms A1 and A2 do not need to deal with critical areas, they only need to find and cover every region that is not covered by R-sensor. In other word, every potential hole must be checked and covered if it is necessary. Note that, since there are no critical areas, any uncovered area is either a corner or a boundary hole. According to the rules of the algorithm, every time the robot reaches a vertex of the grid, it checks all the potential main and corner hole. In addition, we know

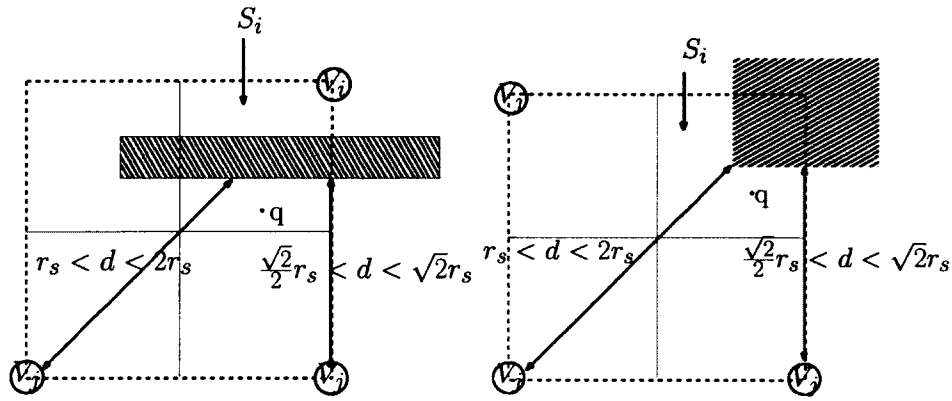


Figure 24: (a) The boundary intersects cell S such that v_i and q are located in two disconnected areas. In this case q is not covered by v_i. (b) v_i is not located because of the boundary.

that every vertex of the grid is visited (proved in [19]). Therefore, every uncovered region will be detected by the robot using the boundary handling rules of algorithms A1 and A2. ■

Now we need to show that the robot deals effectively with such region and that, after termination, the entire ROI is covered. In other word we prove that our algorithms are correct. We will start proving the correctness of algorithm A2. Note that, in the same ROI, the set of B-sensors deployed in algorithm A2 is a subset of the set of B-sensors deployed by algorithm A1. Thus, if algorithm A2 completely covers the ROI then algorithm A1 will guaranty full-coverage as well.

Theorem 5 *Algorithm A2 totally covers any simple ROI.*

Proof.

Let G be the grid defined by the robot and let G' be the same grid as G extended over the entire ROI such that every square in G' contains at least one vertex of G. Since there is no critical areas in the ROI G' will contain the entire ROI.

According to the rules of the algorithm, we know that the robot acts only based on local information and decides whether to place B-sensor or not, by considering

the situation in a specific cell of G'. So, if we prove that the rules of the algorithm guaranties that any cell S of G' is always covered, then we prove that algorithm $A2$ covers the entire ROI.

First, note that since the diagonal of S is $2r_s$, if the boundary of the ROI does not intersect S, then the cell will contain a R-sensor on each vertex covering it completely. This is because if we divide S in four equal squares S_i with edge $\frac{\sqrt{2}}{2}r_s$, and containing a vertex v_i of S at one corner, then every S_i will be totally covered by the R-sensor placed at v_i, where v_i is a vertex of G'. However, this is not necessarily the case when S is intersected by the boundary, because some sensing areas could become disconnected (Figure 24(a)) or even worst, some vertices of S could lie outside the ROI (Figure 24(b)) so that no R-sensor would be placed on it. In algorithm $A2$, such cases are handled by the robot when it is at the remaining vertices of S by placing B-sensors on the uncovered area.

To prove the lemma, let assume that the cell S is intersected by P where P is the boundary which is an orthogonal polygon. By contradiction, let assume that there is a point q which is still uncovered after termination of algorithm $A2$. In this case q is located in one of the squares S_i where v_i is not accessible. By accessible we mean that v_i is not located because of the boundary (Figure 24(b)), or v_i is located but S_i is disconnected by the boundary such that q and v_i are located in two disconnected areas (Figure 24(a)), otherwise q was covered by v_i. So, we could assume that the region of S_i containing v_i is disconnected from the region of S_i containing q. Since we only need to take into account the region containing q, we could assume that there are only two disconnected regions of S_i. Note that, to disconnect S_i in two different pieces, the boundary must intersect its border at least two times on two different edges of S_i, otherwise it will remain connected. In this case, the area containing q intersects at least one of the line segments joining v_i to other three vertices of S. In particular, it will intersect the segment $\overline{v_i v_j}$. However, since this intersection occurs

inside S_i, the robot will see the boundary along this segment at distance between $\frac{\sqrt{2}}{2}r_s$ and $\sqrt{2}r_s$ if v_i and v_j are adjacent, or between distances r_s and $2r_s$ if they are opposed. But, according to the boundary handing rules, it will place a B-sensor on the segment $\overline{v_i v_j}$ next to the boundary. So, the B-sensor will be inside the region in S_i containing q, and cover it which is a contradiction. Hence, algorithm $A2$ guaranties a full coverage providing there are no critical area. Or in other word, algorithm A2 completely covers any simple ROI. ■

3.5 Complexity of Algorithm A2

In this section we compute the Number of Deployed Sensors ($N_{sensors}$), Number of Robot's Movement (R_{mov}), and Number of Transmitted Messages (R_{msg}) of algorithm A2. Then we compare our results with what is obtained from BTD algorithm.

Number of deployed sensors: There are several factors to be analyzed when studying the complexity of this type of algorithms. The number of deployed sensors is one of the principal complexity measures to be checked. This is because the cost of network is strongly related to the number of deployed sensors. However, in this case, the number of sensors depends not only on the algorithm but also on the starting point of the robot. That is because the robot defines a virtual grid according to its local coordinate system, and any further action will rely on this grid. So, the question is: Can the robot computes a different grid such that the number of deployed sensor reduce before starting sensor deployment? The following lemma addresses this question.

Lemma 6 *The robot cannot distinguish among different grids without global map.*

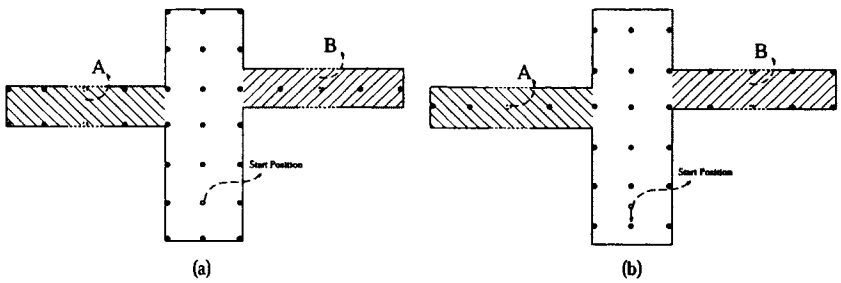


Figure 25: (a) The robot stays at its starting point. (b) The robot changes its starting point.

Proof. To proof this lemma, let assume the ROI is like the ROI showed in Figure 25. Since the robot has limited visibility, it cannot distinguish which one of the two corridors, A or B, is longer. According to robot's view, the only two possible actions

are either computing the grid from its start position (Figure 25 (a)) or moving toward the corner to be at distance r_s from it (Figure 25 (b)). Note that in this case the boundary would be at distance $\frac{\sqrt{2}}{2}r_s$, so that no B-sensor would be needed. Further more, in first case(the grid is computed from the start position Figure 25 (a)), corridor B needs one row of sensors while corridor A needs two rows. In the second case (Figure 25 (b)) corridor B needs tow rows of sensors while corridor A needs one row. So, if the robot decides to stay at it's starting point corridor A could be very long that needs more sensors compare to the second case that the robot changes it's starting point. However, if the robot moves to the corner corridor B could be very long that needs more sensors compare to the first case. Hence, without a global map, the robot cannot make any decision about the optimal grid. ■

The principal implication of the above lemma is that the virtual grid computed by the robot is fixed and depends only on the starting position of the robot. Thus in both algorithms, BTD and A2, the robot moves trough the same virtual grid deploying a R-sensor on each vertex. However, since in BTD algorithm the boundary problem has not been studied, the amount of deployed sensors will be different. The difference will be precisely the amount of B-sensors deployed by algorithm A2. Therefore, to know the complexity of our algorithm, first we need to know the minimum number of sensors needed to solve the boundary problems, and then compare it with the number of deployed B-sensors by our algorithm.

Lemma 7 *The number of deployed sensors in algorithm A2 is minimal.*

Proof. Let consider the worse case scenario in which there is a boundary problem near all boundaries. In this situation, the uncovered area near the boundaries forms a band of width $\frac{\sqrt{2}}{2}r_s$ along the entire boundary like in Figure 26 (a). The minimal solution deploys B-sensors along the middle line of this band (dash line in the middle of band in Figure 26 (b)) at distance $\sqrt{\frac{7}{2}}r_s$ which is the maximum distance that

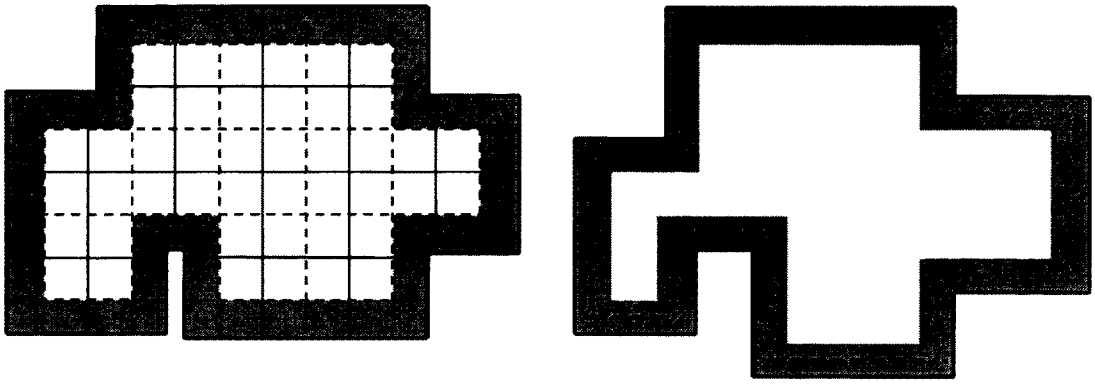


Figure 26: (a) Uncovered band around the ROI with width $\frac{\sqrt{2}}{2}r_s$. (b) In the minimal case, B-sensors are deployed in the middle of this band along dark lines.

provides no hole is created between two consecutive B-sensors. Our algorithm on the contrary, places B-sensors on the external border of the band at distance $\sqrt{2}r_s$ to guaranty that everything is covered. The following expression, shows the relation among both distances, where d_{Max} is the maximum distance between two consecutive B-sensors in minimal case:

$$d_{Max} = \sqrt{\frac{7}{2}}r_s = \frac{\sqrt{7}}{2}\sqrt{2}r_s < 1.5(\sqrt{2}r_s)$$

Thus, if the band was straight the number of B-sensors ($N_{B-sensors}$) deployed by our algorithm would be $N_{B-sensors} = 1.5N^{Min}_{B-Sensors}$ where $N^{Min}_{B-Sensors}$ is the minimum number of deployed B-sensors. However, since the ROI is orthogonal, the band is not straight. On the other hand, in minimal solution B-sensors are deployed in the middle of this band while in our algorithm they are deployed on the external border.

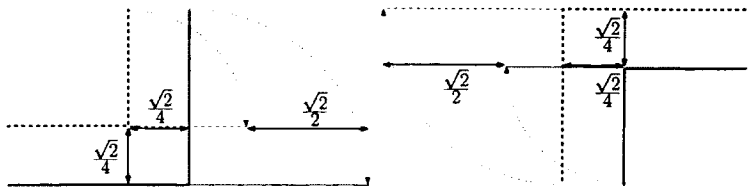


Figure 27: (a) Convex corner (b) Concave corner

It implies that the entire distance of the path along which B-sensors are deployed in our algorithm differs from the one used in the minimal case. Fortunately, this difference could be computed easily if we look at the effect that each type of corners (concave C_{conc} and convex C_{conv}) have on these paths. If we stretch a convex corner (figure 27 (a)) the distance to be covered by our algorithm increase in $\frac{\sqrt{2}}{2}r_s$ respect to the distance to be covered by the minimal solution. On the contrary, in concave corner (figure 27 (b)) the distance to be covered by the minimal solution increases in $\frac{\sqrt{2}}{2}r_s$ respect to the distance to be covered by our solution. Thus, the effect caused by one type of corner is nullified by the other type. Since the ROI is orthogonal, the number of convex corner is equal to the number of concave corner plus four. That implies that the distance to be covered by our algorithm is equal to the distance to be covered in the minimal solution plus $4\frac{\sqrt{2}}{2}r_s$ so that 2 more B-sensors are needed.

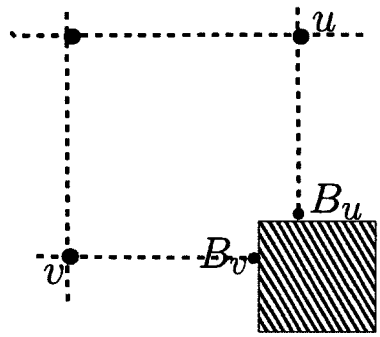


Figure 28: The robot drops two B-sensors on concave corner.

Another effect of concave corner in our algorithm is showed in figure 28. In this case, while being in the grid vertex v the robot cannot distinguish if a B-sensor (B_u) was already placed from u or not. So, the robot places an additional B-sensor on those cases. Hence, the final number of B-sensors needed by our algorithm ($N^{A_2}_{B-sensors}$) would be:

$$N^{A_2}_{B-sensors} = 1.5N^{Min}_{B-sensors} + 2 + C^{Total}_{conc} = O(N^{Min}_{B-sensors})$$

Note that since the B-sensors are placed from vertices of the grid, C_{conc} and C_{conv} are the corners of the polygon surrounding the virtual grid. In this way we avoid to count some small corners that are possible on the boundary of the ROI but which does not have any effect on the algorithm. Thus, summarizing, the total number of deployed sensor ($N_{sensors}$) of our algorithm would be:

$$N^{A_2}_{sensors} = N^{BTD}_{R-sensors} + O(N^{Min}_{B-sensors})$$

In the above equation $N_{R-sensor}$ and $N^{Min}_{B-sensors}$ are the number of deployed R-sensors and minimum number of deployed B-sensors respectively.

■

Number of Robot's Movement: As we saw, both algorithms BTD and A2 work on the same grid and behave in the same way except for boundary handling actions. It follows that the number of robot's movements in algorithm A2 is exactly the same as BTD plus the number of movements needed to deploy B-sensors. In [19] it is proved that each grid point is visited at most 4 times by the robot. On the other hand, B-sensors do not play any active role in the algorithm. That is, they do not have neither back pointer nor color. So, the robot only needs two additional movements for each B-sensor, one to drop it and one more to go back to last deployed R-sensor. After being deployed, B-sensors do not affect the robot movement any more. Hence the total number of robot movement (R_{mov}) in this algorithm is as the following:

$$R^{A2}_{mov} \leq 4N_{R-sensor} + 2N_{B-sensor} = O(R^{BTD}_{mov})$$

In the above equation $N_{R-sensor}$ and $N_{B-sensor}$ are the number of deployed R-sensors and B-sensors respectively.

Number of Transmitted Messages: Sensors communicate with each other

periodically by sending Hello messages; These messages can be used by the algorithm at no cost. So, we only need to consider the transmitted messages between the robot and sensors. The robot communicates with sensors only for back tracking. Since B-sensors do not perform any active role in the algorithm, there is no change compare to BTD. That means that both algorithms BTD and A2 need the same amount of robot's messages (R_{msg}). As we saw in robot's movement, the robot visits each R-sensor at most 4 times [19]; hence, the number of robot's message is bounded as follows:

$$R_{msg}^{A2} = 4(N_{R-sensor}) = R_{msg}^{BTD}$$

In the above equation $N_{R-sensor}$ is the number of deployed R-sensors.

Chapter 4

Sensor Deployment in a Complex Orthogonal ROI

4.1 Introduction

In the previous chapter, we proposed algorithm A1 to solve the boundary problem by introducing boundary handling rules. We then improve the boundary handling rule in algorithm A2, so that A2 uses less sensors than A1 in order to achieve full coverage. However, algorithms BTD, A1, and A2 can solve coverage problem only in a simple orthogonal ROI without critical areas. Figures 29 (a) and (b) illustrate algorithms BTD and A2 in a complex ROI respectively. Light and dark gray circles denote R-sensors and B-sensors respectively. The black sensor denotes the starting point of the robot. As it is shown, these algorithms do not achieve full coverage in a complex ROI because of the existing critical areas. In this chapter, we propose algorithm B that guarantees full coverage in a complex orthogonal ROI containing critical areas. Algorithm B is organized into four main steps. The first step is Detecting critical areas and Deploying E-sensors, while the second step is Detecting boundary problems and Deploying B-sensors. After detecting boundary problem and critical areas, the third step is Covering Critical areas. At this step the robot enters each critical area and

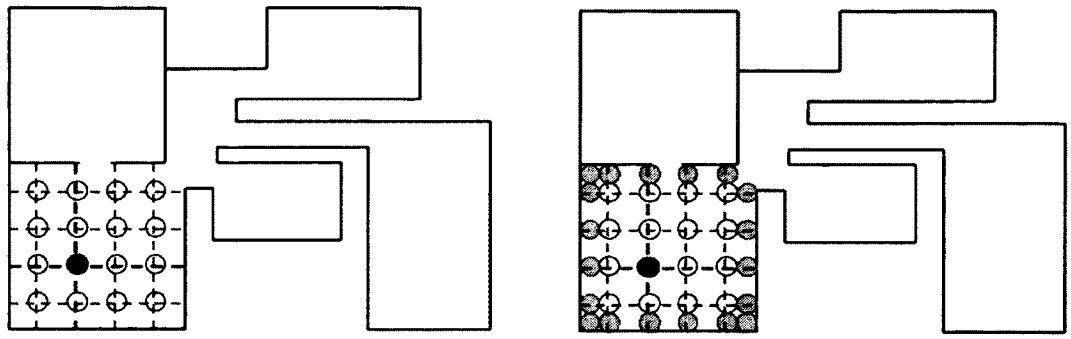


Figure 29: (a) BTD in a Complex ROI. (b) A2 in a Complex ROI.

starts applying algorithm B to cover it. Whenever all the detected critical areas are covered the last step is Regular Movement. In fact, algorithm B divides the ROI into sub-ROIs. We can consider the entire ROI and its starting point as the root of a tree. In this tree the children of a node are the detected critical areas for the sub-ROI defining that node. Hence, by using a DFS traversal algorithm B traverses the the entire tree, and covers each node of it. In the following section we explain each step of algorithm B in detail.

4.2 Step 1: Detecting critical areas and Deploying E-sensors

In algorithm A2, the color of each R-sensor is influenced by boundaries and neighboring sensors around it. In algorithm B, in addition to boundaries and neighboring sensors, the color of each R-sensor is also influenced by critical areas around it. An R-sensor is white as long as there are uncovered critical areas, and open directions around it. To this end, each R-sensor keeps counting the number of uncovered critical areas, and whenever one of them is covered, the counter is decremented by one. When all the critical areas are covered, the R-sensor can change it's color to black if there is

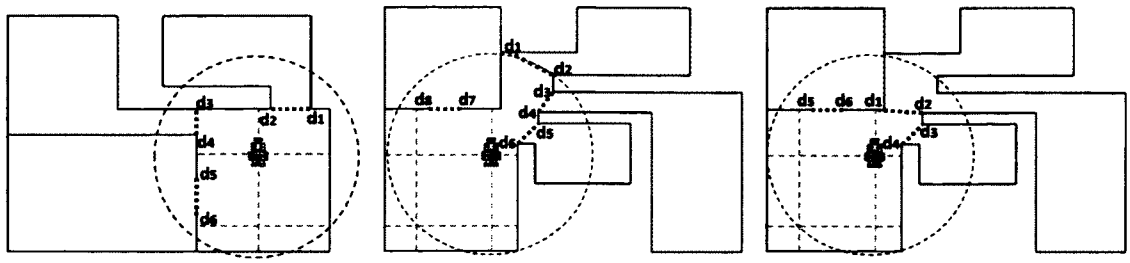


Figure 30: (a) Three detected critical areas are labeled by line segments $\overline{d_1d_2}$, $\overline{d_3d_4}$, and $\overline{d_5d_6}$. (b) and (c) four and five critical areas are detected respectively in the same ROI from the same location.

no open direction around it. Since R-sensors do not have visibility to detect critical areas, the robot informs the sensors of the existing nearby boundaries and number of uncovered critical areas around them. We consider an R-sensor as a *supportive sensor* when there are critical areas around it.

In order to detect critical areas locally by a robot, let us over see the entire ROI and study the map. In figure 30(a) we can easily see three critical areas around the current location of robot. In this figure the dash circle shows the visibility range of the robot, and the dash line segments ($\overline{d_1d_2}$, $\overline{d_3d_4}$, and $\overline{d_5d_6}$) are used to show the entrance of each critical area. Figures 30(b) and (c) illustrate two same ROIs in which four and three critical areas are detected respectively from a same location. As it is shown in these figures, it is important to divide the critical areas separately to count their number. Having a map, it is easy to figure out the critical areas, analyze their location and then make a decision on how to divide them properly. But, how to do it locally when the map of the ROI is absente is very complicated. To this end, we introduce a set of partitioning rules which help the robot to segment the local area in order to identify critical areas and divide them properly. So, in this step the robot performs two tasks: first it partitions the local areas to detect critical areas, and then deploys *Entrance sensors* in the entrance of detected critical areas.

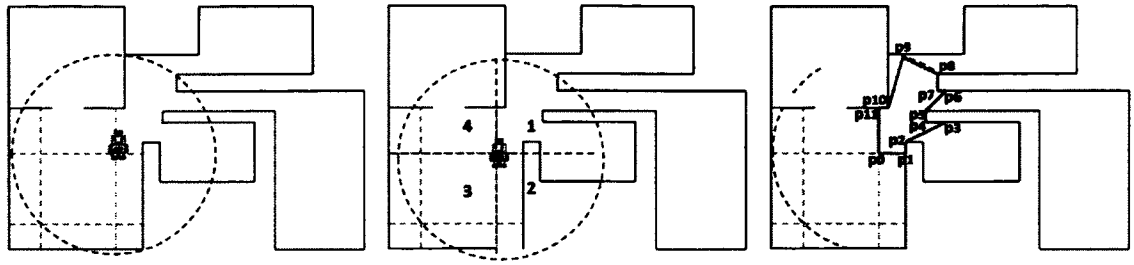


Figure 31: (a) Visibility circle of the robot. (b) Dividing the visibility circle to four quadrants 1, 2, 3, and 4. (c) Created visibility polygon ($p_0p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10}p_{11}$) for quadrant 4 which contains multiple critical areas.

4.2.1 Partitioning the local area

The robot can detect all the boundaries and predeployed sensors within distance $2r_s$ and 360 degrees. Using this visibility the robot can see a circle area with radius of $2r_s$ which is called *visibility circle* of the robot. In Figure 31(a) the highlighted circle illustrates the visibility circle of the robot. The first step of partitioning is dividing the visibility circle to four quadrants. Figure 31(b) shows the division of visibility circle to four quadrants. Each quadrant of the visibility circle may contain only a single critical area or multiple critical areas. For example, in Figure 31(b), quadrant 4 contains only a single critical area while quadrant 1 contains multiple critical areas. The second step of partitioning is to create a *visibility region (visibility polygon)* for each quadrant. "The visibility polygon or visibility region for a point p in the plane among non-transparent obstacles is the possibly unbounded polygonal region of all points of the plane visible from p" [24]. Figure 31(c) illustrates the visibility polygon ($p_0p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10}p_{11}$) for quadrant 1 which contains multiple critical areas.

After creating the visibility polygon, the next step is decomposing the created visibility polygon. There are different methods for decomposing a polygon [4, 15, 20, 22, 25]. In this thesis, we use the procedure presented in [4] which decompose a polygon to *star-shaped Polygons*. A star-shaped polygon is one in which the entire

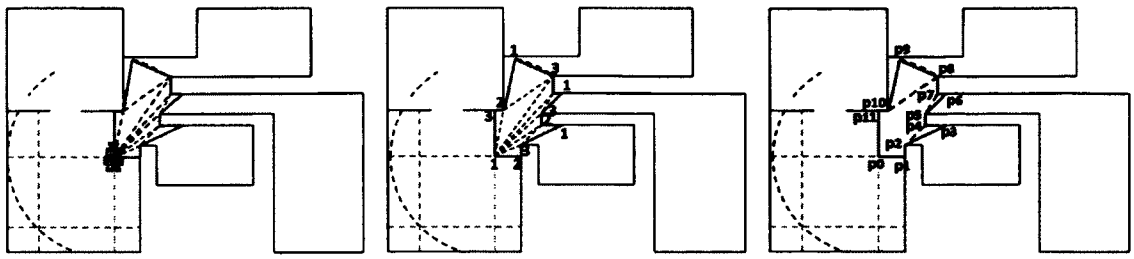


Figure 32: Decomposing procedure. (a) Triangulating created visibility polygon. (b) Coloring vertices of created triangles with colors 1, 2, and 3. (c) Color 1 is chosen to mix triangles and finally four star-shaped polygons ($p_0p_1p_2p_4p_5p_7p_8p_{10}p_{11}$, $p_2p_3p_4$, $p_5p_6p_7$, $p_8p_9p_{10}$) are created.

polygon is visible from at least one fixed point of the polygon [4]. The decomposing procedure has three steps: triangulation, coloring, and mixing. According to this procedure, first the robot obtains a triangulation of the visibility polygon. Figure 32(a) shows the triangulation of the visibility polygon ($p_0p_1p_2p_3p_4p_5p_6p_7p_8p_9p_{10}p_{11}$) of Figure 31(c). After triangulation, the vertices of the triangles are colored with colors 1, 2, and 3. As a rule, we color the current sensor by 1, and then other vertices are colored in a way that no vertices of a triangle have the same color. Figure 32(b) shows how the vertices of created triangles are colored. Then color 1 is chosen and adjacent triangles that have a common vertex with it are mixed together. After applying the decomposition procedure, several star-shaped polygons are obtained. Figure 32(c) shows four created star-shaped polygons ($p_0p_1p_2p_4p_5p_7p_8p_{10}p_{11}$, $p_2p_3p_4$, $p_5p_6p_7$, $p_8p_9p_{10}$).

Once the star-shaped polygons are created, the next step is to identify which ones represent critical area. The current sensor is located on one vertex, that belongs to one and only one star-shaped polygon, which is called a *main polygon.* In Figure 33(a) the polygon ($p_0p_1p_2p_4p_5p_7p_8p_{10}p_{11}$) denotes a main polygon. Among all the star-shaped polygons, only those which are separated by a *diagonal* whose ending points land on the boundaries of the ROI, are representatives of the critical areas. We call these

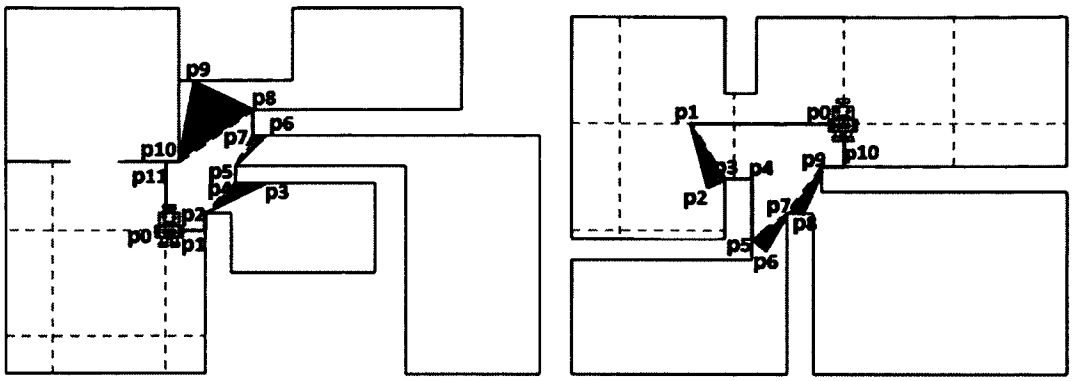


Figure 33: (a) The polygon ($p_0p_1p_2p_4p_5p_7p_8p_{10}p_{11}$) denotes the main polygon while the polygons ($p_2p_3p_4$, $p_5p_6p_7$, $p_8p_9p_{10}$) denote representative polygons. (b) The polygon ($p_0p_1p_2p_3p_4p_5p_7p_9p_{10}$) again is a main polygon. The polygons ($p_5p_6p_7$, $p_7p_8p_9$) denote representative polygons while the polygon ($p_1p_2p_3$) is nothing.

polygons *representative polygons*, and each diagonal which separates representative polygon from the main one is called *entrance edge* of a critical area. In fact, the diagonal is a line that joins two non-consecutive vertices of a polygon. In Figure 33(a), polygons ($p_2p_3p_4$, $p_5p_6p_7$, $p_8p_9p_{10}$) illustrate representative polygons, and the dashed line segments ($\overline{p_{10}p_8}, \overline{p_5p_7}, \overline{p_2p_4}$) denote the entrance edges. A representative polygon is actually the entrance of a critical area. In Figure 33(b) the polygon ($p_0p_1p_2p_3p_4p_5p_7p_9p_{10}$) again represents a main polygon, while the polygons ($p_5p_6p_7$, $p_7p_8p_9$) show representative polygons. In this figure a polygon ($p_1p_2p_3$) is created as well. However, this polygon is not considered as a representative polygon since the end points of it's entrance edge ($\overline{p_1p_3}$) does not land on the boundaries.

In conclusion, to divide a local region in order to identify and partition critical areas, the robot applies the following rules:

- Dividing the visibility circle to four quadrants (Figure 31 (b)).
- Creating the visibility polygon for each quadrant (Figure 31 (c)).
- Decomposing visibility polygons to star-shaped polygons using decomposition

procedure (Figures 32 (a), (b), and (c)).

- Recognizing representative polygons and entrance edges of the detected critical areas (Figure 33 (a)).

4.2.2 Deploying E-sensors

In algorithm B, in addition to R-sensors and B-sensors, *Entrance sensors (E-sensors)* need to be deployed in the entrance of detected critical areas. As a rule, the robot drops E-sensors in the middle of each entrance edge. At the end of decomposing procedure, the entrance edges of critical areas are obtained for the entire visibility circle of the robot. The robot computes the middle of each entrance edge (as a line segment) as the location of E-sensors. It stores all the calculated E-sensors locations into an *E-sensors list.* Then it starts deploying E-sensors on the computed locations. However, it is possible that a detected critical area maybe already covered by neighboring R-sensor(s). If this is the case, the robot will not deploy more E-sensor on the critical area. However, the robot replaces the location of the E-sensor which was supposed to be deployed with the location of the E-sensor which is already deployed in the detected critical area. At the end of this step, the robot informs the supportive sensor of the number of uncovered critical areas around it.

Since E-sensors have active role in algorithm B (they participate in back tracking) the robot assigns them an Id. As a rule, the E-sensors only consider the received information of their supportive sensor to update their back pointer. For this purpose, each E-sensor needs to keep the Id of its supportive sensor since it maybe receive Hello message from different neighboring sensors. Moreover, each E-sensor has a status which is covered if the critical area is covered and uncovered otherwise. After deploying E-sensors, they start sending Hello message to supportive and neighboring sensors. This message contains their location, Id, color, state, back pointer, status,

and the supportive sensor's Id. Moreover, the state of each E-sensor, is Entrance.

4.3 Step 2: Detecting boundary problems and Deploying B-sensors

After detecting critical areas the robot has three options: detecting and covering boundary problem, covering detected critical areas, or doing Regular Movement. The order of doing these actions is important, because it can affect the number of robot movements which is important in terms of energy consumption. Detecting and covering boundary problem has more priority to both Regular Movement and covering critical areas, because if the robot leaves boundary/corner holes, it may have to back track a long distance to cover them, leading to an unnecessary large number of robot's movement. So, detecting and covering boundary problem has the highest priority. However, deployed E-sensors may cover created boundary/corner holes. So, to avoid deploying extra B-sensors, the robot creates a virtual *coverage map* within its visibility circle. Using this coverage map, the robot is able to compute whether deployed E-sensors cover the existing boundary/corner holes or not.

4.3.1 Creating coverage map

The robot computes a coverage map for the entire visibility circle from its current location. First, it removes all the detected critical areas from the previous step. To this end, it assumes the visibility circle is cut by the entrance edges, and considers all of them as boundaries. Using E-sensors list, the robot places all the E-sensors on the map, and then places all the neighboring R-sensors of the current sensor. Note that some of the neighboring R-sensors maybe already deployed while some other will be deployed in future. However, the robot considers all of them on the map.

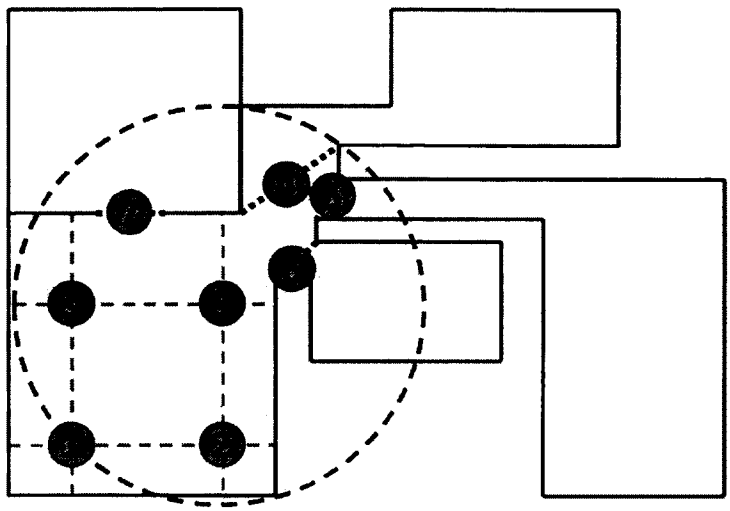


Figure 34: Created coverage map within the visibility circle of the robot located at current R-sensor 3. Sensors (1, 2, s) and sensors (4, 5, 6, 7) represent neighboring and entrance sensors respectively.

After deploying neighboring R-sensors and E-sensors on the map, the robot usees boundary handling rule of algorithm A2 to detect boundary problems. However, before deploying B-sensors the robot assures that the detected hole is not already covered by an E-sensor. Figure 34 illustrates a created coverage map within the visibility circle of the robot. In this figure, sensor 3 denotes the current sensor. Sensors 1 and 2 denote main and corner neighbors which are already deployed. Sensor s denotes another main neighbor which is not deployed yet, but will be deployed in future. Sensors 4, 5, 6, and 7 denote E-sensors which are deployed in the first step.

To make computation of coverage map simpler, we consider the *effective coverage square* of each sensor instead of its circle sensing range. "The effective coverage region is the average coverage region that the sensor contributes to the monitoring area" [12]. As shown in Figure 35(a), in square deployment the effective coverage area of a sensor is a square with length $\sqrt{2}r_s$. Figure 35(b) illustrates when sensors are deployed at distance $\sqrt{2}r_s$ from each other, and Figure 35(c) shows that the entire ROI is under

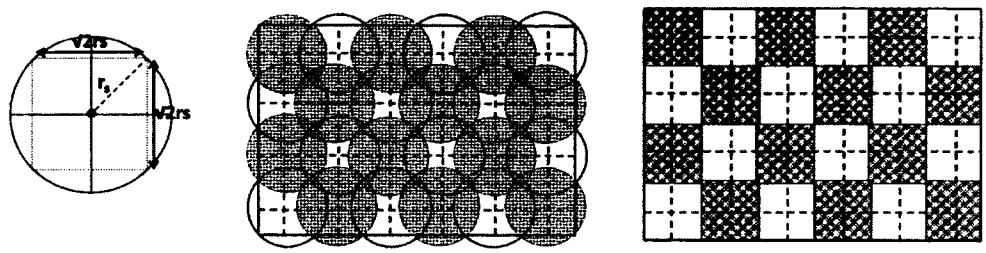


Figure 35: (a) Effective coverage square of each sensor is a square with length $\sqrt{2}r_s$. (b) Deployed R-sensors cover the ROI. (c) Deployed R-sensors, considering their effective coverage square.

shadow of effective coverage areas of the sensors. In this case, there are neither gaps nor overlaps between effective coverage areas of the sensors.

So, considering the effective coverage square of each sensor, the robot creates the coverage map. Figure 36 shows the created coverage map of Figure 34 using effective coverage square of each sensor. As it is shown in this Figure, the detected boundary hole in north is not covered by E-sensors, and the robot should drop a B-sensor to cover it. So, after creating the coverage map, the locations of all B-sensors are indicated. The robot removes the location of all E-sensors from the E-sensors list and, stores the location of all B-sensors in a *B-sensors list* in clockwise direction in the B-sensors list.

4.3.2 Deploying B-sensors

Using the coverage map, the robot has stored the location of all B-sensors in the B-sensors list. So, it drops the B-sensors on their locations one by one and removes them from the B-sensors list. When the B-sensors list is empty, there is no uncovered boundary/corner hole. So, the robot goes back to the last deployed R-sensor. At the end of this step, the robot removes the current coverage map from its memory to have enough space for next computations.

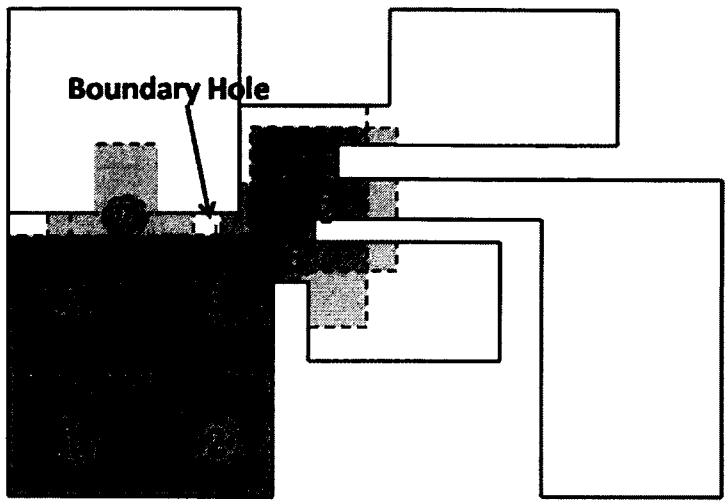


Figure 36: Created coverage map of Figure 34 using effective coverage square.

4.4 Step 3: Covering critical areas

After detecting and covering boundary problem, the robot has two options: covering detected critical areas, or doing Regular movement. Covering critical areas has higher priority than Regular Movement. To explain the reason, let us assume the situation is vice versa. In this case, the robot leaves the critical areas uncovered and does Regular Movement as long as it can. Assume a scenario in which the robot is stuck in a dead and has to back track a long distance to reach a white sensor, and the sensor is white just because of the small critical area around it. In another scenario, the robot leaves the detected critical area around its current R-sensor (sensor s_1) and deploys next R-sensor (sensor s_2). After deploying R-sensor s_2, the robot detects the previous critical area from its current position. Since the critical area is uncovered, the robot informs the R-sensor s_2 of its existence as well. In this case, both R-sensors s_1 and s_2 are white because of the same critical area. Then, after a while the robot is stuck in a dead end and R-sensors s_1 and s_2 are the only white sensors in the ROI. So, the robot back tracks to R-sensor s_2 to cover the critical area, which is the entrance of

a big area and the robot spends a lot of time to cover it. Again after a while, the robot gets stuck in a dead end, and R-sensor s_1 is the only left white sensor. So, the robot has to back track a long distance to reach R-sensor s_1. After back tracking, the robot figures out that this sensor is white because of a critical area which is already covered. In both of these scenarios, the robot has to back track a long distance to cover critical areas which are not actually covered on time. So covering critical area has more priority to Regular movement.

As long as there are uncovered critical areas around the currents sensor (supportive sensor), the robot does not do Regular Movement. For this purpose, the supportive sensor keeps the number of uncovered critical areas, and only the location of E-sensor that should be covered next. So, whenever the robot comes back to the supportive sensor and asks it where to go, the supportive sensor sends it to cover next critical area. After entering the critical area, the robot stops at the location of E-sensors and informs it to change it's status to covered. Then the robot goes through all four steps (Detecting critical areas and deploying E-sensor, Detecting boundary problem and Deploying B-sensor, Covering critical areas, and Regular Movement) one by one. The robot has to check boundary problems, and critical areas for the deployed E-sensor, because there may be some critical areas or boundary problems which are detectable only from this location. However, when the robot is detecting critical areas around an E-sensor, it may detect some critical areas which are located out of the current critical area. Since covering detected critical areas has more priority, the robot leaves the current critical area to cover the new detected critical area. In this case the robot never goes back to this critical area since there is an E-sensor with covered status in its entrance. To solve this problem, when the robot enters a critical area it only checks the area inside the current critical area. We can assume an invisible door in the entrance of the current critical area which will be closed when the robot gets in this region. Figure 37 illustrates the ROI and internal critical areas

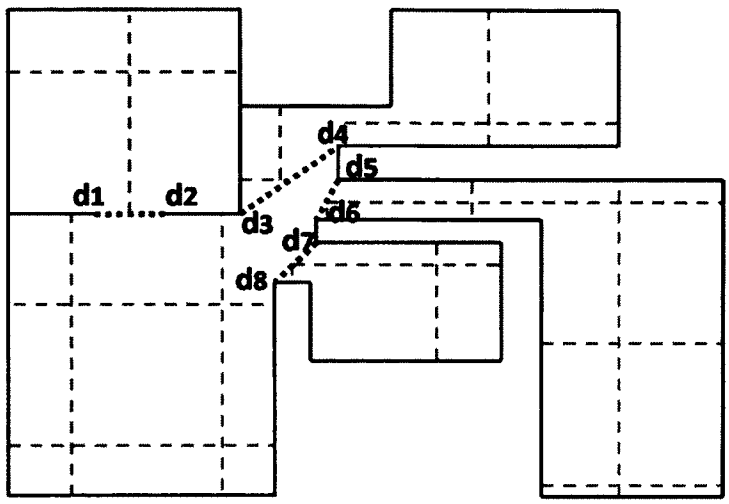


Figure 37: Invisible doors ($\overline{d_1d_2}$, $\overline{d_3d_4}$, $\overline{d_5d_6}$, and $\overline{d_7d_8}$), which are closed when the robot enters the critical areas.

which are separated by the invisible doors $\overline{d_1d_2}$, $\overline{d_3d_4}$, $\overline{d_5d_6}$, and $\overline{d_7d_8}$. These doors are closed when the robot enters the critical areas, which means that the robot does not look back at the door. To close the invisible door the robot only needs to know the entrance edge (because the invisible doors are actually the entrance edges) and on what side of this edge the new sub-ROI is defined. It could be easily done if we had the location of the E-sensor which is on the entrance edge of the new sub-ROI ,and a perpendicular vector of the entrance edge pointing to the new sub-ROI. In this situation the robot could ignore anything outside of the new sub-ROI.

So, briefly we can say that in this step the robot enters the critical area, stops at the location of E-sensor, closes the door, and starts applying algorithm B for the region inside the critical area.

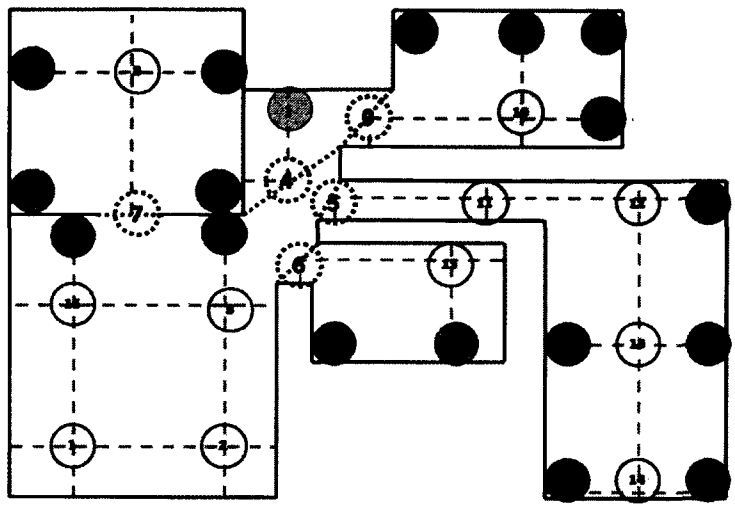


Figure 38: Full coverage is achieved by using algorithm B.

4.5 Step 4: Regular Movement

Whenever there is no boundary hole/uncovered critical area around the current sensor, the robot moves to distance $\sqrt{2}r_s$ based on order rule, which is West, East, North and South, and deploys an R-sensor. Whenever the robot is stuck in dead ends, it back tracks to the nearest white sensor in its backward path. A sensor is white because of the open direction or uncovered critical areas. Since covering critical areas has higher priority, the robot first checks to see if there is any uncovered critical areas around the sensor. If there is not any uncovered critical areas, the robot checks open directions and does Regular Movement to cover them. Figure 38 illustrates the same ROI of Figure 29 which was not covered by algorithm BTD and A2. However, as it is shown in this figure, it is fully covered using algorithm B. In this figure, light gray, dash, and dark gray circles denote R-sensors, E-sensors, and B-sensors respectively.

4.6 Algorithm B Summary

Algorithm B follows algorithm A2 for deploying R-sensors, detecting boundary problems, and back tracking when the robot is stuck in dead ends. This algorithm is organized in four main steps: Detecting critical areas and deploying E-sensors, Detecting boundary problems and deploying B-sensors, Covering critical areas, and Regular Movement.

After deploying each sensor (except B-sensors), first the robot starts detecting critical areas around it. According to the partitioning rules, the robot divides the visibility circle to four quadrants. Then it applies a decomposition procedure for created visibility polygons in each quadrant. Based on this procedure, the robot uses triangulation, coloring, and mixing to decompose visibility polygon to star-shaped polygons. Among all created star-shaped polygons, only those which are separated by the entrance edges are representative polygons. Since the E-sensors are supposed to be deployed in the middle of each entrance edge, the robot computes and stores the location of E-sensors in the E-sensors list. Then, it deploys E-sensors on their computed locations.

After detecting critical areas and deploying E-sensors, the robot creates a virtual coverage map within it's visibility circle to detect boundary problems. To create a coverage map, first the robot removes all detected critical areas and considers the entrance edges as part of the boundaries. Then it deploys all the neighboring R-sensors and E-sensors. Considering entrance edges as boundaries, the robot checks the boundary handling rule of algorithm A2. If detected boundary/corner hole is not covered by E-sensors, a B-sensor is deployed on the map to cover it. At the end of this step, the robot computes the locations of all B-sensors and stores them in a B-sensors list. Then it starts deploying B-sensors. When B-sensors list is empty the robot returns to last deployed R-sensor and removes the coverage map from its

memory to free up memory for next computations.

After detecting boundary problems and deploying B-sensors, to cover critical area, the robot asks the supportive sensor the location of an uncovered E-sensor. After entering a critical area, the robot informs the E-sensor to change it's status to covered. Then assuming a closed door in the entrance of the critical area the robot starts applying algorithm B for the region inside the critical area. During covering the critical areas, or even out of them the robot might get stuck by boundaries or predeployed sensors.

Whenever the robot is stuck in a dead end, it takes the back pointer of its current sensor. If the back pointer of the current sensor is Null, the ROI is fully covered and the algorithm terminates. If this is not the case, the robot back tracks to the last deployed white sensor in its backward path. A sensor is white because of the uncovered critical areas or open directions around it. The robot asks the white sensor where to go. If there are uncovered critical areas, the sensor sends the robot to cover them, otherwise the robot does Regular Movement to deploy next R-sensor in an open direction. The algorithm pseudo which is based on these descriptions is as follows.

Algorithm B
Main-Direction (North, East, South, West)
Corner-Direction (Northeast, Southeast, Southwest, Northwest)
Wake up
Locate first R-sensor
while *there are uncovered areas* **do**
  Decomposition-Procedure ()
  if *critical area(s) is detected* **then**
    Compute and store the location of all E-sensors within the visibility circle in the E-sensors list.
    Deploy E-sensors in the middle of detected entrance edges and mark the E-sensor as uncovered.
    Create-Coverage-Map ()
  end
  else
    Boundary-Handling-Rule ()
  end
  if *there is uncovered critical area* **then**
    Cover-critical-area()
  end
  if *Dead-End* **then**
    Back-Track (). **if** *there is uncovered critical area* **then**
      Cover-critical-area()
    end
    else
      Do Regular Movement.
    end
  end
  Regular Movement.
end

Sub-Procedures of Algorithm B

Decomposition-Procedure ()

Triangulate the visibility polygon.
Color the vertices of triangles using colors (1, 2, 3) such that color 1 is assigned to the current vertex.
Choose color 1 and merge the adjacent triangles that have a common vertex with it.

Create-Coverage-Map ()

Remove all the detected critical areas and consider their entrance edges as part of the boundary.
Drop all main and corner neighbouring R-sensors, and E-sensors.
Considering the coverage square of all deployed sensors,
Check-Boundary-Handling-Rule ().

Cover-critical-area()

Ask the current sensor the location of next uncovered E-sensor.
Move to the location of E-sensor and inform it to change its status to covered.
Applies algorithm B for the region inside the detected critical area.

Check Boundary-Handling-Rule ()

for *8 cardinal Directions starting from North* **do**
  if *in Main-Direction* $< \sqrt{2}/2rs < d < \sqrt{2} <$ **then**
    push the location of B-sensor in B-sensors List.
  end
  if *in Corner-Direction* $< rs < d < 2rs <$ ***and*** *no one of associate sensors drop a B-sensor in associate square* **then**
    push the location of B-sensor in B-sensors List (if the robot cannot see the associate sensors, it assumes they do not drop a B-sensor in associate square).
  end
end
while *B-sensors List is not empty* **do**
  Drop a B-sensor.
end
Return to last deployed R-sensor.

Back-Track ()

if *back-pointer of the current sensor is NULL* **then**
  Terminate algorithm.
end
else
  Back track to the back-pointer of the current sensor.
end

4.7 Correctness of Algorithm B

The effect of the algorithm B is to partition the ROI into disjoint pieces (that is what the "doors" do); each piece is a sub-ROI that the robot enters it, defines a new grid, and starts deploying sensors following algorithm B. In the following section first we prove that algorithm B totally covers a sub-ROI. So, every other sub-ROI can be covered in the same way as well. Then we will show that algorithm B covers the entire ROI.

We have already explained partitioning rules, but to avoid confusion we did not explain all mathematical details of creating visibility polygon and triangulation. However, to prove the correctness of the algorithm we need to know how partitioning rules work precisely. So, first we explain the details of partitioning rules and then start proving the correctness of algorithm B.

Partitioning rules in detail

Let $v = p_0,p_1,...,p_k$ the vertices in P_v ordered in counterclockwise where P_v is the visibility polygon created at current vertex v (note that P_v is the created visibility polygon for one of the quadrants of the visibility circle). For any point p_j, if the segment $\overline{vp_j}$ is a diagonal in P_v then the segment is added to the triangulation. If there is a point p_i such that the segment $\overline{vp_i}$ is not a diagonal in P_v, then since every vertex in P_v is visible from v, either $\overline{p_ip_{i+1}}$ or $\overline{p_{i-1}p_i}$ is collinear with v. If no such cases happen, the polygon is completely triangulated and every triangle contains v, so that no edge is reported as entrance to a critical area.

Before we continue the triangulation process, note that as a rule it is not possible for two consecutive edges in P_v to be collinear with v. If that case happens the vertices they have in common would be removed from P_v. Furthermore, if we consider three alternate edges (their extremes are consecutive vertices of P_v), since the ROI is

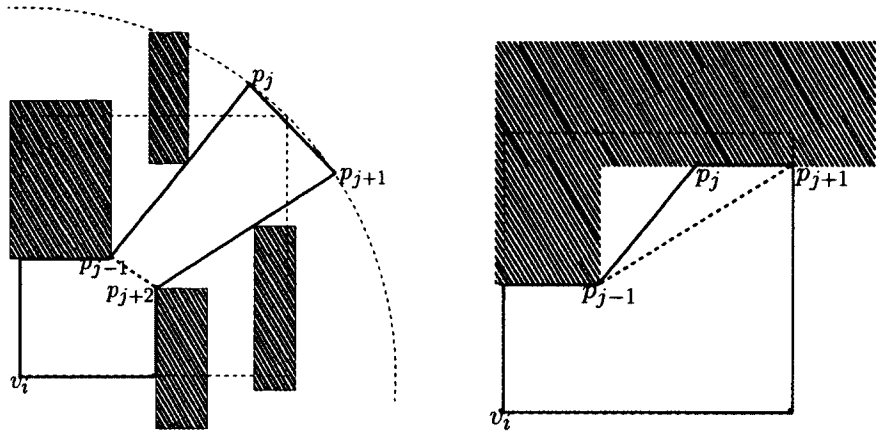


Figure 39: (a) The quadrilateral $\overline{p_{i-1}p_ip_{i+1}p_{i+2}}$ is cut from ROI, and edge $\overline{p_{i-1}p_{i+2}}$ is reported as an entrance edge. (b) The triangle $\Delta p_{i-1}p_ip_{i+1}$ is cut from ROI, and edge $\overline{p_{i-1}p_{i+1}}$ is reported as an entrance edge.

orthogonal, it is not possible that all of them become collinear with v. Thus let us consider what happens if two alternate edges $\overline{p_{i-1}p_i}$ and $\overline{p_{i+1}p_{i+2}}$ are collinear with v.

Let $\hat{P}$ be a set such that for any $p_i \in \hat{P}$, the segment $\overline{vp_i}$ is not a diagonal of P_v (either $\overline{p_{i-1}p_i}$ or $\overline{p_ip_{i+1}}$ is an edge in P_v collinear with v (Figure 39(a))). Since the ROI is orthogonal the only possible case is that $p_i, p_{i+1} \in \hat{P}$ and $p_{i-1}, p_{j+2} \notin \hat{P}$. In this case, the convex quadrilateral $\overline{p_{i-1}p_ip_{i+1}p_{i+2}}$ is cut from the rest by the edge $\overline{p_{i-1}p_{i+2}}$, and this edge is reported as an entrance to a critical area.

If there is an isolated collinear edge (none of the two prior and next edges are collinear (Figure 39(b))), then we take it's extreme p_i belonging to $\hat{P}$. The edge $\overline{p_{i-1}p_{i+1}}$ separates the triangle $\Delta p_{i-1}p_ip_{i+1}$ from the rest, so this edge is reported as an entrance to the critical area providing that both p_{i-1} and p_{i+1} belong to P.

The following lemma ensures that partitioning rules only report critical areas.

Lemma 8 *Partitioning rules only detect critical areas.*

Proof. In this lemma we show that the partitioning rules (Decomposition-Procedure ()) only report critical areas not a boundary or corner hole for example. The partitioning rules (Decomposition-Procedure ()) report an edges $\overline{xy}$ which is

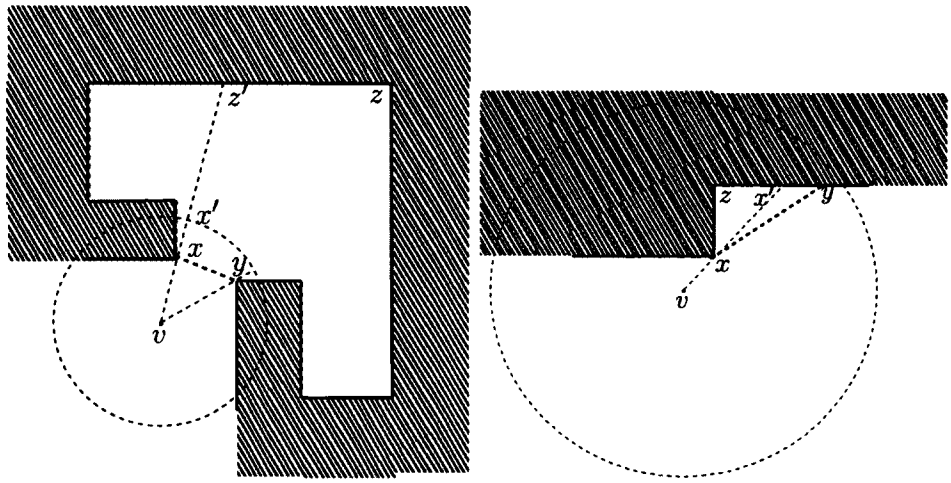


Figure 40: (a) Line segment $\overline{xx'}$ collinear with v, and x' does not belong to P. (b) Line segment $\overline{xx'}$ collinear with v, and x' belongs to P.

completely visible from v and totaly contained in P, where v is the current vertex where critical areas is reported, and P denotes orthogonal polygon of the ROI. Furthermore, both x and y lie on P. To prove the lemma we need to show that polygon P_j^{xy} separated by $\overline{xy}$ which does not contain v is a critical area. According to critical area definition, P_j^{xy} is a critical area if we can find a vertex z such that the segment $\overline{zv}$ is not totally contained inside P, or z is at distance greater than $2r_s$.

According to the partitioning rule (Decomposition-Procedure ()) edge $\overline{xy}$ is reported as an edge because at least one of the points x or y (let say x) is the extreme of an edge $\overline{xx'}$ in the visibility polygon P_v collinear with v (Figure 40(a)). Since the ROI is orthogonal, $\overline{xx'}$ cannot belong to P. Moreover, as it mentioned x belongs to P while two cases could happen for x'.

First let us assume x' does not belong to P, then it is at distance $2r_s$ (Figure 40(a)). In this case we could extend the ray starting from v and passing trough x' until it touches P at a point z' in P_j^{xy} farther than $2r_s$. If z' is one of the vertex of P_j^{xy} then z' would be z. But if z' is not a vertex of P_j^{xy}, we take the furthest vertex among the extremes of the edge in P_j^{xy} containing z'. So, we found a vertex (z) in

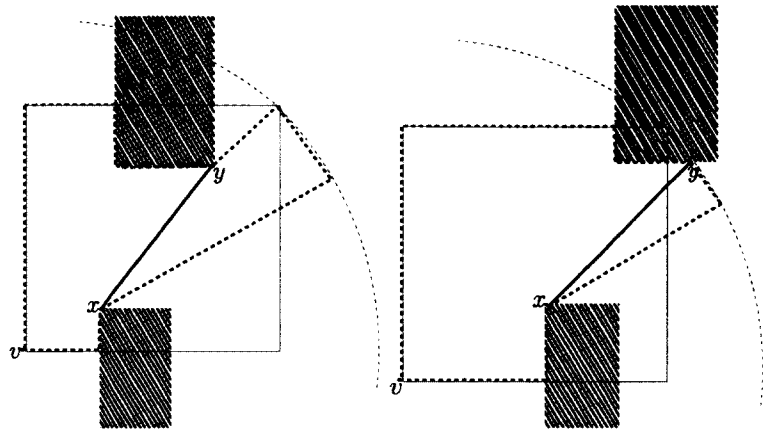


Figure 41: Line segment $\overline{xy}$, which is not collinear with v, is reported as the entrance edge of the critical area P_j^{xy}.

P_j^{xy} at distance greater than $2r_s$ which means P_j^{xy} is a critical area and reported edge $\overline{xy}$ is an entrance edge.

Now, let us assume that x' belongs to P (Figure 40(b)). In this case, since $\overline{xx'}$ does not belong to P, we can say that segment $\overline{xx'}$ defines a partition of the polygon P_j^{xy}. We show this partition by $P_j^{xx'}$ which does not contain neither the point y nor v. Note that in this polygon ($P_j^{xx'}$), only the vertices x and x' are visible from v. Thus, if we take a vertex z in $P_j^{xx'}$ distinct from x and x' the segment $\overline{vz}$ will not be totaly contained in P. Hence, since $P_j^{xx'}$ is a partition of P_j^{xy}, P_j^{xy} is a critical area and reported edge $\overline{xy}$ is an entrance edge. ■

In the following lemma we will prove that every critical area is detected.

Lemma 9 *Algorithm B detects every critical area.*

Proof. In order to prove the lemma, let us assume by contradiction that, after termination, there is a critical area P_j^{xy} from the point of view of a vertex v that is not detected. Let us assume without lost of generality that P_j^{xy} is minimum (there are no other segments like $\overline{x'y'}$, visible from v, defining a critical area $P_j^{x'y'}$, such that $P_j^{x'y'}$ contains P_j^{xy}). Note that in this case detecting a minimum critical area (using Decomposition-Procedure ()) means detecting the entrance edge of it, or visiting it

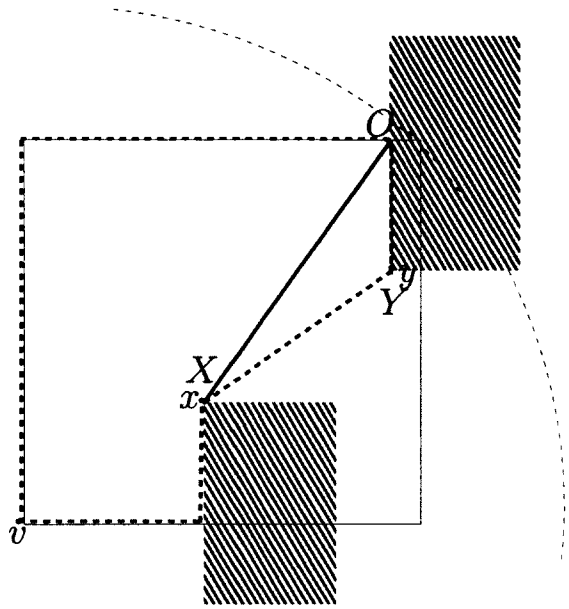


Figure 42: (a) $\overline{XO}$ is reported as an entrance edge of P_j^{xy}.

completely from one vertex of the grid. Since P_j^{xy} is minimum, two cases can happen for $\overline{xy}$. The first case is when $\overline{xy}$ is not collinear with v, and the second case is when it is collinear.

First we consider the case $\overline{xy}$ is not collinear with v. In this case according to the partitioning rules (Decomposition-Procedure ()) $\overline{xy}$ is reported as entrance edge which is a contradiction. This fact is illustrated in Figures 41.

Now we consider the case that the line segment $\overline{xy}$ is collinear with v. Since P_j^{xy} is minimum, just the edge $\overline{xy} \in P_j^{xy}$ is visible from v, and no other point inside P_j^{xy} could be seen from v. So, the line segment $\overline{xy}$ which is collinear with v is part of an edge in the visibility polygon P_v. Consider the edge $\overline{XY}$ of P_v containing $\overline{xy}$, such that X is between v and Y. Note that X could be equal to x and Y to y (Figure 42). According to the partitioning rules (Decomposition-Procedure ()), there is a vertex O in P_v such that the edge $\overline{XO}$ is reported as entrance edge to P_j^{xy} provided that O belongs to P which is contradiction.

Now let us assume that O does not belong to P . In this case there is a grid

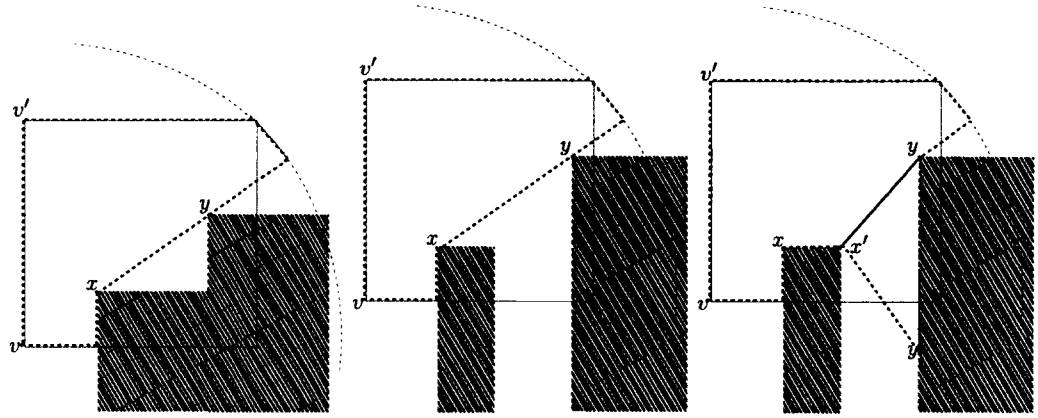


Figure 43: (a) P_j^{xy} is completely visible from v'. (b) P_j^{xy} is the minimum critical area from v. (c) $\overline{x'y}$ is reported as an entrance edge of $P_j^{x'y'}$

vertex v' between v and y such that the segment $\overline{xy}$ is also completely visible from v'. Otherwise, X would be connected with O on P. In this case P_j^{xy} might be completely visible from v' which is a contradiction (Figure 43(a)). So, let us consider the case that P_j^{xy} is not completely visible from v' (Figure 43(b)). If the segment $\overline{xy}$ does not define minimal critical area from v', then there is a segment $\overline{x'y'}$ such that from the point of view of v' the critical area $P_j^{x'y'}$ defined by this segment is minimal (Figure 43(c)). According to the partitioning rules (Decomposition-Procedure ()) an entrance edge will be reported for $P_j^{x'y'}$. Note that in this case some part of P_j^{xy} is completely visible from v', and for the other part which is not visible ($P_j^{x'y'}$) an entrance edge ($\overline{x'y}$) is reported which is a contradiction. Hence, every critical area is reported by algorithm B.

■

Lemma 10 *After removing all the detected critical areas from a sub-ROI, algorithm B fully covers it.*

Proof. Algorithm B follows algorithm A2 and in the case of critical areas it applies partitioning rules (Decomposition-Procedure ()) to the detect entrance edges.

After detecting each entrance edge it will drop an E-sensor in the middle of it, enters the critical area and performs algorithm B to deploy sensors inside it (Cover-critical-area()).

Let P' be the polygon obtained after removing every reported critical area from P. Note that we consider the entrance edges as edges of P'. Let G be the grid defined by the robot for P' and, let G' be the same grid as G extended over the entire sub-ROI such that every square in G' contains at least one vertex of G. So, if we prove that the rules of the algorithm guaranties that every cell S of G' is always covered, then we prove that algorithm B covers the entire sub-ROI. In the following section we prove that a cell of S is covered; So in the same way we can prove that any other cell S can be covered.

First, note that since the diagonal of S is $2r_s$, if the boundary of the sub-ROI does not intersect S, then the cell will contain a R-sensor on each vertex covering it completely. This is because if we divide S in four equal square S_i with edge $\frac{\sqrt{2}}{2}r_s$, and containing a vertex v_i of S at one corner, then every S_i will be totally covered by the sensor placed at v_i, where v_i is a vertex of G'. However, this is not necessarily the case when S is intersected by the boundary, because some sensing areas could become disconnected or even worst, some vertices of S could lie outside the sub-ROI so that no R-sensor would be placed on it. In algorithm $A2$, such cases are handled by the robot when it is at the remaining vertices of S by placing B-sensors on the uncovered area (Check Boundary-Handling-Rule ()).

To prove the lemma, let us assume that the cell S is intersected by P where P is the boundary. By contradiction, let us assume that there is a point q which is still uncovered after termination of algorithm B. In this case q is located in one of the squares S_i where v_i is not accessible. By accessible we mean that v_i is not located because of the boundary, or v_i is located but S_i is disconnected by the boundary such that q and v_i are located in two disconnected areas, otherwise q was covered

by v_i. So, we could assume that the region of S_i containing v_i is disconnected from the region of S_i containing q. Since we only need to take into account the region containing q, we could assume that there are only two disconnected regions of S_i. Note that to disconnect S_i in two different pieces, the boundary must intersect its border at least two times, otherwise it will remain connected. In this case the area containing q intersects at least one of the line segments joining v_i to other three vertices of S. In particular it will intersect the segment $\overline{v_i v_j}$. However, since this intersection occurs inside S_i, the robot will see the boundary along this segment at distance between $\frac{\sqrt{2}}{2} r_s$ and $\sqrt{2} r_s$ if v_i and v_j are adjacent or between distances r_s and $2r_s$ if they are opposed. But, according to the boundary handing rules (Check Boundary-Handling-Rule ()), it will place a B-sensor on the segment $\overline{v_i v_j}$ next to the boundary. So, the B-sensor will be inside the region in S_i containing q, and cover it which is a contradiction. However there are some cases that the robot does not drop a B-sensor inside S_i. That case only happens when there is already an E-sensors that fully covers S_i (Create-Coverage-Map ()). It means that q is already covered which is a contradiction. ■

Theorem 11 *Algorithm B completely covers any complex ROI.*

Proof. In order to prove the lemma let us first define a logically sub-division of the ROI forming a tree as it follows. The root of this tree will be the entire ROI containing the starting point. From this point the robot defines a grid, and according to lemma 9 it will detect all the critical areas (Decomposition-Procedure ()) for this grid. Critical areas are introduced by the entrance edges (Decomposition-Procedure ()) which divide the ROI into disconnected sub-ROIs. These sub-ROIs will be the children of the root. The starting point of each sub-ROI will be the middle point of the entrance edge. Applying the same idea for each sub-ROI we define the new nodes until we reach the leaves which are the sub-ROIs containing no critical areas from the

grid defined at their starting point. Now we need to show that following the rules of the algorithm the robot traverses the entire tree and covers the entire ROI.

According to the algorithm, at the beginning the robot is at the starting point (Locate first R-sensor) of the Root of the tree ($T[Root]$) and will move on the grid to cover the ROI. We know by lemma 9 that all the critical areas (which are the children of the root) will be detected (Decomposition-Procedure ()), and according to the algorithm each time a critical area is found the robot gets into it (Cover-critical-area()). When the robot gets into a child X (critical area), it moves to its start point and places an E-sensor that its back pointer will not be Null if and only if the tree $T[Root - X]$ (the sub tree containing the root but not X) is not totally covered. At this time the robot is at the start point of a disconnected sub-ROI. So, we have a smaller instance of the same problem in which the ROI is defined by the sub-Tree rooted at X. According to the algorithm the robot does the same until it reaches a leaf of the tree ($T[Root]$). Since the leaf does not have any critical area, according to lemma 10 it will be totally covered. At that moment the robot follows back pointer to the starting point which direct the robot to the leaf's parent Y (Back-Track ()). Then the robot will continue moving along the grid defined by the starting point of Y until it finds a new critical area which means there is an unvisited child of Y, or the entire sub-ROI is covered. If there is still an unvisited child, the robot will move into it, otherwise it will back track to Y's parent. Therefore the robot will never leave a node until all its children and it-self are covered. It implies the robot traverses the tree like a Depth-first search (DFS).

Now let us assume that the robot is at the moment in which it is getting into the last unvisited node which is a leaf. Note that, at that moment, any node which is not on the path from the Root to this leaf is totally covered. According to lemma 10 the robot will completely cover the leaf and back tracks if the back pointer of the starting point is not Null (Back-Track ()). Then it will continue acting the same way

(covering the node and back tracking) until it gets a node which after completely covering it the back pointer of its starting point is Null. At this moment two cases are possible either the node is the Root or not. Whatever the case every single node will be covered which means that the entire ROI is covered. ■

4.8 Complexity

In this section we compute the Number of Deployed Sensors ($N_{sensors}$), Number of Robot's Movement (R_{mov}), and Number of Transmitted Messages (R_{msg}) of algorithm B.

Number of deployed sensors: In order to analyze the number of deployed sensors in algorithm B, first note that the set of reported critical areas strongly depends on the virtual grid and on the order in which the robot visits the vertices of the grid. As we saw in Lemma 6, since the robot has limited visibility it cannot distinguish which grid is better. Also note that some reported critical areas lead to some parts of the ROI which are not accessible by the original grid while some others do not.

Taking this into account, first let us assume that every critical area which was reported leads to a new ROI that must be covered. In this case, each critical area is treated as a new ROI with its own entry point and virtual grid. Thus, at the end, the ROI is divided in k disconnected pieces E_i. Since algorithm B works the same as algorithm A2 when there are no critical areas, the number of deployed sensors in each E_i (N_{E_i}), $1 \leq i \leq k$, is:

$$N_{E_i}^{B} = N_{E_i}^{BTD} + O(N_{B-sensors}^{Min_{E_i}})$$

In the above equation $N_{B-sensors}^{Min_{E_i}}$ is the minimum number of deployed B-sensors in each E_i.

Note that since E-sensors are counted in the new ROI (E_i) they do not need to be counted in the old one. In other words, we do not need to count the E-sensors deployed in a ROI since they will be counted in the new one.

Now let us see what happens when a reported critical area f_c does not lead to a new ROI. In that case, for each reported $f_c \in F_c$ (F_c is total number of critical areas that do not lead to a new ROI) an E-sensor is deployed which could be unnecessary. Therefore, the total number of deployed sensors in algorithm B ($N^B_{sensors}$) will be:

$$N^B_{sensors} \leq \sum_{i=1}^{k} N^B_{E_i} + \|F_c\|$$

Number of Robot's Movement: In the case of robot's movements, since E-sensors have an active role in algorithm B (i.e. they are part of the backtracking procedure), they must be included in the analysis. Inside any pieces of the ROI (E_i) the amount of robot's movement is the same as in algorithm A2 ($4N_{R-sensor} + 2N_{B-sensor}$). Let for a moment not consider B-sensors since they do not affect the complexity of the algorithm. In this case, the amount of robot's movements would be at most two times the number of edges in the virtual grid. In algorithm B, since the robot moves between neighboring pieces of ROI, there is an additional edge per E-sensor, through which, according to the rules of the algorithm, the robot only moves at most twice. This is because, whenever the robot enters in a new piece E_i, it will never came back until E_i is completely covered. This implies that the amount of robot's movement is still bounded by two times the number of edges. Now the question is if in this case the number of edges could be greater than four times the number of sensors. If we look carefully to the characteristics of the entrance edges, we will notice that E-sensors can only have, at most, two neighbors on the new grid

that they define. This means that the number of edges will always be less than four times the number of sensors. Hence, the total number of robot's movement in this algorithm (R^B_{mov}) is as the following:

$$R^B_{mov} = 4(N_{R-sensor} + N_{E-sensor}) + 2N_{B-sensor}$$

In the above equation $N_{R-sensor}$, $N_{E-sensor}$, and $N_{B-sensor}$ are the number of deployed R-sensors, E-sensors, and B-sensors respectively.

Number of Transmitted Messages: In term of transmitted messages, as in algorithms BTD and A2, the sensors only communicate between themselves by means of Hello messages. Thus we only need to know how many messages are necessary for communication between the robot and sensors. In this way, algorithm B behaves in the same way as algorithm A2. That is, the robot only exchanges message with its current sensor when backtracking. Thus, following a similar analysis for robot's movement we could see that the total number of robot's messages (R^B_{msg}) is bounded by the following expression:

$$R^B_{msg} = O(N_{R-sensor} + N_{E-sensor})$$

In the above equation $N_{R-sensor}$ and $N_{E-sensor}$ are the number of deployed R-sensors and E-sensors respectively.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Sensor deployment is one the basic problems in WSNs. In this problem the sensors are scattered in the entire ROI to collect information. Many studies has been done to cover this problem. In static sensor networks the human deploys sensors in the ROI, but this method is not applicable for dangerous and inaccessible ROI. In mobile sensor networks, sensors have capability of movement. So, they move and scatter in the ROI. However, mobile sensors are more expansive than static sensors, and this method is not efficient for situations that sensors are in danger because of bad situation of the ROI. The existing problems motivated researchers to come up with solution for sensor deployment. In this solution one or more robots deploy static sensors in the ROI. However, few works have been done so far, and unfortunately non of the existing algorithms guarantee full coverage of an orthogonal region with arbitrary boundaries. In some algorithms the robot cannot cover areas near the boundaries and obstacles. While in some other cases the robot is unable to come out of a dead end when there are still uncovered areas in the ROI. Moreover, these algorithms can not be applied to an orthogonal ROI containing critical areas. Among proposed algorithms, BTD is the first one that has solved the dead end problem. However, in this algorithm

the boundary problem is intentionally ignored. Moreover, it does not fully cover the complex ROI.

So, firstly we proposed two algorithms A1 and A2 to solve the boundary problem of BTD by using boundary handling rule. In these two algorithms the robot deploys sensors in a simple orthogonal ROI without any critical area. These algorithms basically work in the same way and only differ in their boundary handling rule. In algorithm A1 the robot checks all eight directions to detect boundary and corner holes, and drops a B-sensor whenever a problem is detected. In this algorithm the robot drops extra B-sensors since it cannot distinguish fake and real corner holes. So, we improve the boundary handling rules in algorithm A2 that capable the robot to distinguish the real and fake corner holes. We proved that this algorithm guarantees full coverage in a simple orthogonal ROI. Moreover, it does not have any extra cost compare to BTD algorithm. In both algorithm the number of robot's movement, and transmitted messages between the robot and sensors is linear in number of deployed sensors. Algorithm A2 deploys the same number of R-sensors as BTD. Moreover, it uses some B-sensors to cover boundary and corner holes. There are no B-sensors in BTD algorithm since it does not cover boundary problem. So, we showed that the number of deployed B-sensors of our algorithm compared to minimal solution is linear.

Then we proposed algorithm B for sensor deployment by the robot that guarantees full coverage of an orthogonal complex ROI. This algorithm follows algorithm A2 to deploy R-sensors, handle boundary problem, and back track when the robot is stuck in a dead end. Moreover, it applies the partitioning rule to detect and handle critical areas. After entering each critical area, the robot executes algorithm B for the region inside the critical area. So, algorithm B is executing recursively until the entire ROI is covered. We proved that this algorithm achieves full coverage in any complex orthogonal ROI. The number of transmitted messages and robot's movement is linear

in the number of deployed sensors. The number of deployed sensors is the same as the number of deployed sensors in algorithm B plus some additional E-sensors; since the number of deployed E-sensors is linear, the total number of deployed sensors remains linear.

5.2 Future Work

In this work we are assuming a single robot is deploying sensors in an orthogonal ROI. Multi robots scenario can speed up deploying sensors process. In this case each robot should have a separate Id and each sensor in addition to remembering its Id needs to remember the Id of the robot which is deployed by it. As we discussed in previous chapters, the robot deploys sensors on the vertices of a square grid. However, deploying sensors on a triangular or hexagonal grids require less sensors [19]. Moreover, B-sensors can also be deployed like what we discussed in section (3.7) for minimal scenario.

Another potential future work is applying the proposed algorithms in a ROI with any arbitrary shapes, not necessarily orthogonal. Moreover, we can also consider obstacles in the ROI. These obstacles make a hole in the ROI that may confuse the robot when detecting critical areas. For example, the robot may detect a critical area, enters it, creates a new grid and starts deploying sensors on it. Since, this critical area is detected because of two close obstacles, the new defined grid to cover it will have conflict with the old grid. Moreover, in this work we are assuming there is no sensor failure in the ROI. However, after deploying sensors, they may fail and stop working because of different reasons. In this case a sensing hole creates in the ROI, and the robot should fix or replace the malfunction sensor.

References

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Commun. Mag.*, 40(8):102–114, 2002.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Comput. Netw.*, 38(4):393–422, 2002.

[3] I.F. Akyildiz and M.C. Vuran. *Wireless sensor networks.* John Wiley & Sons Inc, 2010.

[4] D. Avis and G. Toussaint. An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recognition*, 13(6):395–398, 1981.

[5] M. Batalin and G.S. Sukhatme. Multi-robot dynamic coverage of a planar bounded environment. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2002.

[6] M. Batalin and G.S. Sukhatme. Sensor coverage using mobile robots and stationary nodes. *Proc. Int. Society for Optics and Photonics Engineering (SPIE)*, 4868:269–276, 2002.

[7] M. Batalin and G.S. Sukhatme. Coverage, exploration and deployment by a mobile robot and communication network. *Proc. Int. Workshop on Information Processing in Sensor Networks*, pages 376–391, 2003.

[8] M. Batalin and G.S. Sukhatme. Efficient exploration without localization. *Proc. Int. Conf. on Robotic and Automation (ICRA)*, pages 2714–2719, 2003.

[9] M. Batalin and G.S. Sukhatme. The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment. *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 3478–3485, 2005.

[10] M. Batalin and G.S. Sukhatme. The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Transactions on Robotics*, 23(4):661–675, 2007.

[11] Z. Bojkovic and B. Bakmaz. A survey on wireless sensor networks deployment. *Wseas Transaction on Communications*, 7(12):1172–1181, 2008.

[12] C.Y. Chang, C.T. Chang, Y.C. Chen, and H.R. Chang. Obstacle-resistant deployment algorithms for wireless sensor networks. *IEEE Tran. On Vehicular Technology*, 58(6):2925–2941, 2009.

[13] C.Y. Chang, H.R. Chang, C.C. Hsieh, and C.T. Chang. OFRD: Obstacle-free robot deployment algorithms for wireless sensor networks. *Proc. IEEE Wireless Communications and Networking Conf. (WCNC)*, pages 4371–4376, 2007.

[14] C.Y. Chang, J.P. Sheu, Y.C. Chen, and S.W. Chang. An obstacle-free and power-efficient deployment algorithm for wireless sensor networks. *IEEE Tran. On Systems, Man, and Cybernetics-Part A: Systems and Humans*, 39(4):795–806, 2009.

[15] B. Chazelle and D. Dobkin. Decomposing a polygon into its convex parts. *Proc. 11th Symposium on Theory of Computing*, pages 38–48, 1979.

[16] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme. Deployment and connectivity repair of a sensor network with a flying robot. *Springer Tracts in Advanced Robotics*, 21(2006):333–343, 2006.

[17] P. Corke, S. Hrabar, R. Petersony, D. Rus, S. Saripalli, and G. Sukhatme. Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle. *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3602 – 3608, 2004.

[18] R. Falcon, X. Li, and A. Nayak. Carrier-based coverage augmentation in wireless sensor and robot network. *Proc. 7th IEEE Int. Workshop on Wireless Ad hoc and Sensor Networks (WWASN)*, 2010.

[19] G. Fletcher, X. Li, A. Nayak, and I. Stojmenovic. Back-tracking based sensor deployment by a robot team. *Proc. 7th IEEE Communications Society Conf. on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 1–9, 2010.

[20] J.M. Keil. Polygon decomposition. In *Handbook of Computational Geometry.* Elsevier Science Publishers, 2000.

[21] X. Li, A. Nayak, D. Simplot-Ryl, and I. Stojmenovic. Sensor placement in sensor and actuator networks. In *Handbook of Wireless Sensor and Actuator Networks:*

Algorithms and Protocols for Scalable Coordination and Data Communication. John Wiley, 2010.

[22] J.M. Lien and N.M. Amato. Approximate convex decomposition of polygons. *Proc. 20th Annual ACM Symp. Computat. Geom.(SoCG)*, pages 17–26, 2004.

[23] F. Martincic and L. Schwiebert. Introduction to wireless sensor networking. In *Handbook of Sensor Networks: Algorithms and Architectures.* John Wiley & Sons, 2005.

[24] F.P. Preparata and M. I. Shamos. *Computational Geometry.* Springer-Verlag, 1985.

[25] B. Schachter. Decomposition of polygons into convex sets. *Computers, IEEE Transactions*, C-27(11):1078–1082, 1978.

[26] L.C. Shiu. The robot deployment scheme for wireless sensor networks in the concave region. *Proc. IEEE Int. Conf. on Networking, Sensing and Control (ICNCS)*, pages 581–586, 2009.

[27] J. Urrutia. Art gallery and illumination problem. In *Handbook of computational geometry.* Elsevier Science Publishers, 1997.