Multi Robot Coverage Path Planning for Pursuit Evasion Problem in a Known Environment

Author

SAAD SHAIKH

2010-NUST-MSPhD-Mts-01

Supervisor

Lt. Col. Dr. Kunwar Faraz Ahmad Khan

DEPARTMENT OF MECHATRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

SEPTEMBER, 2014

Multi Robot Coverage Path Planning for Pursuit Evasion Problem in a Known Environment

Author

SAAD SHAIKH

2010-NUST-MSPhD-Mts-01

A thesis submitted in partial fulfillment of the requirements for the degree of

MS Mechatronics Engineering

Thesis Supervisor:

Lt. Col. Dr. Kunwar Faraz Ahmad Khan

Thesis Supervisor's Signature:_____

DEPARTMENT OF MECHATRONICS ENGINEERING

COLLEGE OF ELECTRICAL & MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY

ISLAMABAD

SEPTEMBER, 2014

Declaration

I certify that this research work titled "*Multi Robot Coverage Path Planning for Pursuit Evasion Problem in a Known Environment*" is my own work. The work has not been presented elsewhere for assessment. The material used from other sources has been properly acknowledged / referred.

Signature of Student
Saad Shaikh
2010-NUST-MSPhD-Mts-01

Language Correctness Certificate

This thesis has been read by an English expert and is free of typing, syntax, semantic, grammatical and spelling mistakes. Furthermore, the thesis is according to the format given by the university.

Signature of Student

Saad Shaikh

2010-NUST-MSPhD-Mts-01

Signature of Supervisor

Copyright Statement

Acknowledgements

I am thankful to my creator Allah Subhana-Watala to have guided me throughout this work at every step and for every new thought which You setup in my mind to improve it. Indeed I could have done nothing without Your priceless help and guidance. Whosoever helped me throughout the course of my thesis, whether my parents or any other individual was Your will, so indeed none be worthy of praise but You.

I am profusely thankful to my beloved parents who raised me when I was not capable of walking and continued to support me throughout in every department of life. It is because of their valued guidance and persistence that I have been able to achieve this milestone.

I would also like to express my gratitude to my supervisor Dr. Kunwar Faraz for his help throughout my thesis and also for the Mobile Robotics course he has taught me and my colleagues. I can safely say that I haven't learned any other engineering subject in such depth than the one he has taught us.

I would also like to pay special thanks to Mr. Athar Waqas (Advanced Engineering and Research Organization) for his tremendous support and cooperation. Without his help I wouldn’t have been able to complete this manuscript. I appreciate his patience and guidance throughout the whole thesis.

I would also like to thank Dr. Umar Shahbaz and Dr. Umar Izhar for being on my thesis guidance and evaluation committee and express my gratitude to Dr. Adnan Masood for taking time out from his commitments and being my external thesis supervisor.

Finally, I would like to express my gratitude to all the individuals who have rendered valuable assistance to my study, especially my siblings Mr. Haseeb Shaikh and Ms. Urooba Shaikh for their help in the compilation and finalization of this manuscript.

Dedicated to my special parents and adored siblings whose tremendous support and cooperation led me to this wonderful achievement.

Abstract

In the past decade, usage of multiple robots for various tasks has gained a lot of popularity. Robots are becoming a household item being used in multiple dimensions and areas of life. They are used for routine tasks in houses and offices to heavy-duty tasks in factories etc. But the application of robots gaining the most popularity in the current world is their use in security and surveillance. They are used for search and reconnaissance of large scale environments such as banks, shopping malls etc, as well as for inspection of various inaccessible areas such as sewers, tunnels and for guiding tourists safely through museums etc. This is mainly because of the advantages of robots over their human counterparts; reduced costs, time efficiency, lesser manpower requirement and reduced risk factor to name a few. This work targets the usage of robots for search and surveillance in a known environment map. That is, if it is required to secure an area or decontaminate an already 'contaminated' area with a known map, or to declare an area clear of any dangers/intruders. The term 'contaminated' refers to an area that has an intruder present in it. A lot of work has already been carried out and is ongoing in this field of robotics. Various techniques have been devised to complete the Pursuit Evasion Problem (as it is more commonly known) where a target/intruder is being pursued by a pursuer. The techniques vary in methods of employment of robots; some are targeted towards guidance of robots, some are related to single robots, while others are related to multiple robots and the degree of coordination required between them to secure an area. The concept of 'searcher' and 'blocker agents' is employed and an algorithm devised to achieve the abovementioned purpose and detect any intruders present in an area in a way that is time efficient and requires lesser number of agents than the techniques of uncoordinated search being used.

Keywords: *Search and Secure, Pursuit Evasion, Searcher and blocker agents, Coordination*

Table of Contents

Declaration ..... **i**
Language Correctness Certificate ..... **ii**
Copyright Statement ..... **iii**
Acknowledgements ..... **iv**
Abstract ..... **vi**
Table of Contents ..... **vii**
List of Figures ..... **ix**
List of Tables ..... **xi**
CHAPTER 1: INTRODUCTION **1**
1.1 Definitions and Terminologies 1
1.2 Background 2
1.2.1 Cyclic Strategies 3
1.2.2 Markers Based Strategies 3
1.2.3 Communication Based Decentralized Strategies 3
1.2.4 Cooperation Based Centralized Strategies 4
1.2.5 Partitioning Based Strategies 4
1.2.6 Contamination Based Strategies 4
1.3 Motivation 4
1.4 Scope 6
1.5 Thesis Outline 6
CHAPTER 2: OBJECTIVES AND LITERATURE REVIEW **7**
2.1 Literature Review 7
2.2 Search and Secure Algorithms Overview 7
2.2.1 Cyclic Strategies 8
2.2.2 Markers-based strategies 8
2.2.3 Communication-based decentralized strategies 9
2.2.4 Cooperation-Based Centralized Strategies 9
2.2.5 Partitioning Strategies 10
2.3 Developmental Stages of Partitioning-based Algorithms 11
2.3.1 Early Developments 11
2.3.2 Objective-based Algorithm Development 11
2.3.3 Search and Secure Using Static Robots 12
2.4 Search and Secure using Mobile Partitioning-based Robots 12
2.4.1 Segmentation and Merging 12
2.4.2 Comparison of Voronoi and Delaunay Triangulation 13
2.4.3 Creating Minimum Spanning Tree 13
2.4.4 Determining Robot Positions and Tree Components 13
2.4.5 Identifying Robot Paths 14
2.5 Problem Definition 14

2.6 Problem Solving Options 15
2.7 Thesis Objectives 15
CHAPTER 3: ANALYTICAL MODEL AND RESEARCH METHODOLOGY 16
3.1 General 16
3.2 Analytical Model and Research Methodology 16
3.2.1 Algorithm Overview and Description 16
3.2.2 Boundary and Obstacle Matrices 18
3.2.3 Visibility Graph of Corners 19
3.2.4 Triangulation 20
3.2.5 Merging Triangles into Convex Regions 25
3.2.6 Regions Connectivity Graph 27
3.2.7 Cycle Detection 28
3.2.8 Static Blocker Positions 31
3.2.9 Reduced Minimum Spanning Tree 34
3.2.10 RMST Root Optimization 35
3.2.11 Number of Branches and Search Order 36
3.2.12 Blocker Paths 36
3.2.13 Searcher Paths 38
CHAPTER 4: CASE STUDY FOR VALIDATION OF ALGORITHM 40
4.1 Case Study 40
4.1.1 Environment Map 40
4.1.2 Triangulation 40
4.1.3 Merging Triangles into Convex Regions 41
4.1.4 Cycle Detection and Blocker Positions 43
4.1.5 Reduced Minimum Spanning Tree 44
4.1.6 Root Optimization 44
4.1.7 Number of Branches and Search Order 45
4.1.8 Blocker Paths 45
4.1.9 Searcher Paths 47
4.1.10 Conclusion of the Case Study 49
CHAPTER 5: RESULTS 50
5.1 Environment Map 50
5.2 Triangulation and Triangles Graph 51
5.3 Merging Triangles into Convex Regions and Regions Graph 51
5.4 Cycle Detection and Blocker Positions 52
5.5 Reduced Minimum Spanning Tree and Root Optimization 53
5.6 Number of Branches and Search Order 53
5.7 Blocker Paths 53
5.8 Searcher Paths 54
CHAPTER 6: CONCLUSION 56
REFERENCES 57

List of Figures

Figure 1.1: Robotics market size 5
Figure 1.2: Global market outlook (KIRIA) 5
Figure 3.1: Flowchart of the algorithm 17
Figure 3.2: Example Case - Boundary and Obstacles 19
Figure 3.3: Example Case - Initial Non-Delaunay Triangulation 23
Figure 3.4: Circumcircle of a triangle 23
Figure 3.5: (a) Non-Delaunay Triangulation (b) Edge Flipped for Delaunay Triangulation 24
Figure 3.6: Example Case - Delaunay Triangulation 24
Figure 3.7: Example Case - Triangles Connectivity Graph (Voronoi Diagram) 25
Figure 3.8: Convex and Non-convex Regions 25
Figure 3.9: Example Case - Merged Regions 27
Figure 3.10: Example Case - Regions Graph 28
Figure 3.11: Results of Kruskal (yellow lines) and Prim's Algorithms (orange lines) 28
Figure 3.12: A XOR Example 29
Figure 3.13: Example Case - Regions Graph (Cycles Depiction) 31
Figure 3.14: Example Case - Regions Graph and Regions Numbering 32
Figure 3.15: Example Case - Vertex effecting most cycles - Blocker Position (green) 33
Figure 3.16: Example Case - Final Blocker Positions (green) 33
Figure 3.17: Example Case - RMST and Blocker Positions 34
Figure 3.18: Example Case - Root Finalization (cyan star) 35
Figure 3.19: Example Case - Blocker Paths 38
Figure 3.20: Example Case - Searcher Path (with region numbers) 39
Figure 3.21: Example Case - Searcher Path 39
Figure 4.1: Case Study - Environment Map 40
Figure 4.2: Case Study - Delaunay Triangulation and Triangles Graph (Parent Study) 41
Figure 4.3: Case Study - Delaunay Triangulation and Triangles Graph 41
Figure 4.4: Case Study - Convex regions and Regions Graph (Parent Study) 42
Figure 4.5: Case Study - Merging triangles into Convex Regions and Regions Graph 42
Figure 4.6: Case Study - Blocker Positions and RMST (Parent Study) 43
Figure 4.7: Case Study - Blocker Positions and RMST (with and without regions numbering) 43
Figure 4.8: Case Study - Root Node Selection (Parent Study) 44
Figure 4.9: Case Study - Root Optimization and Selection 45
Figure 4.10: Case Study - Blocker Paths (Parent Study) 46
Figure 4.11: Case Study - Blocker Paths (Shortest Paths) 46
Figure 4.12: Case Study - Searcher Paths (Parent Study) 47
Figure 4.13: Case Study - Searcher Path (with regions numbering) 48
Figure 4.14: Case Study - Searcher Path 48

Figure 5.1: Results - Environment Map 50
Figure 5.2: Results - Triangles Graph 51
Figure 5.3: Results - (a) Merged Regions (b) Unpathable-05 Regions 51
Figure 5.4: Results - Regions Graph (Voronoi Diagram) 52
Figure 5.5: Results - Blocker Positions (and remaining Regions Graph) 52
Figure 5.6: Results - RMST and Root Node Selection 53
Figure 5.7: Results - Blocker Paths (Shortest Paths) 54
Figure 5.8: Results - Searcher Paths 54
Figure 5.9: Results - (a) Seacher-1 Path (b) Searcher-2 Path 55

List of Tables

Table 4.1: Comparison of algorithms depicting time and cost efficiency 49
Table 6.1: Summary of results of the environment..... 56

CHAPTER 1: INTRODUCTION

The work presented in this dissertation primarily targets the "Search and Secure Algorithm" and having robots perform the searching task autonomously. For this purpose, the robots must be made capable of searching and securing the given environment of any intruders as well as ensuring that no intruders can sneak back into the already secured area undetected. This is accomplished through the Search and Surveillance Algorithm discussed in the upcoming chapters. The algorithm divides the given environment into areas where the robots can move, where there are obstacles and areas that need to be surveyed but are inaccessible to the robots (unpathable regions); and then calculates the number of robots/agents required and the paths/trajectories they have to follow in order to completely survey and secure the area.

1.1 Definitions and Terminologies

- **Obstacle:** Obstacle is a polygonal region that is totally inaccessible; both to robots as well as any sort of intruders.
- **Unpathable Region:** Unpathable regions are those regions that need to be searched and secured but they are physically inaccessible to the robots; though intruders can go into such regions.
- **Boundary:** Boundary is defined as the outer edges of the environment that contain all the obstacles, unpathables and the area to be secured. The boundary must be a closed polygon.
- **Decentralized:** Decentralized refers to a communication protocol between robots that is not centralized at a global center, rather it is a local communication protocol.
- **Uncoordinated/Non-Communicative**: Uncoordinated or non-communicative means there is no communication of any sort between the robots and they are completely independent in their tasks.
- **Blocker Robots or Blockers:** Blocker Robots or simply Blockers are those agents/UGVs that are used to "block" certain paths/regions; i.e. they secure certain regions in an environment in order to break loops/cycles.
- **Searcher Robots or Searchers:** Searcher Robots or simply Searchers are those agents/UGVs that are used to "search and secure" the environment by moving on their

specified paths. Searchers move from one region to another securing them visually through their cameras.

- **Convex Hull:** A convex hull refers to a polygon who's inner incident angles are never greater than 180 degrees. Consequently, each point of a convex hull is visible from any other point in it.
- **Circumcircle of Triangle:** The circumcircle of a triangle is that circle which touches all the vertices of that triangle.
- **Lexicographical Order:** Lexicographical Order refers to sorting a set of points/vertices in ascending order w.r.t. their X and Y-coordinates. A lexicographically minimum point is one that has minimum X and minimum Y coordinates in a set of points.
- **Graph:** A graph is a set of points joined together by edges between them showing their connectivity with each other. A graph may or may not have loops/cycles in it.
- **Tree:** A tree is a graph with no loops/cycles in it.
- **Cycles and Loops:** Cycles or loops are sets of points in a graph that are connected together through multiple paths and there exist back edges between them, such that a parent node of a vertex can be cycled back to by passing through another set of nodes.
- **Minimum Spanning Tree:** A Minimum Spanning Tree or MST is a tree that has all the nodes visited atleast once. The MST has no loops/cycles and if a graph is to be converted into an MST, its loops need to be broken through Blockers.
- **Root of a Tree:** A Root node of a tree is the start point of the tree i.e. all the tree branches are originated from the root.
- **Leaves of a Tree:** Leaves or leaf of a tree are those nodes that have no children. They are at the ends of a branch.
- **Branches:** Branches are series of atleast three connected nodes in a graph or tree.

1.2 Background

In the last decade, surveillance, patrolling, exploration and navigation algorithms/strategies have piqued the interest of the robotics world. This area has grown in leaps and bounds because of the variety of approaches that these algorithms discuss and comprise of. Different authors have come up with different numerical as well as analytical approaches to solve such complex problems.

Mobile robots are used in significant numbers in hazardous and dangerous industrial tasks such as aerospace, nuclear and mining industries. As the use of robots increases, so does their interaction with humans. In order to perform various tasks, mobile robots are used either in collaboration with humans or totally independently. So, research on path planning and environment coverage algorithms that identify a collision free path along with performing a certain task at hand is a fundamental requirement in the artificial intelligence industry these days. Moreover, these paths need to be optimized regarding time as well as space/length required to reach their termination points.

This section aims to present some of the background work related to different algorithmic strategies followed by researchers throughout the years for coverage path planning. Most of them are mixed strategies resulting from the merging of various analytical systems, hence enforcing the idea that it is not necessary to follow one particular strategy to solve a problem.

1.2.1 Cyclic Strategies

Cyclic strategies involve identifying the whole area to be explored and obstacles present in it as a set of vertices. Such strategies are usually used for Patrolling Algorithms where robots/agents have to "patrol" an area continuously.

1.2.2 Markers Based Strategies

Markers-based strategies are multi-robot, decentralized, non-communicative, local search techniques, which employ deployment of tags to identify areas which have already been explored by one of the robots in the team. These strategies are examples of dynamic path identification since there is no knowledge of the map/environment prior to the start of exploration.

1.2.3 Communication Based Decentralized Strategies

These techniques are based on communication between robots/agents. As the technique is not centralized, the communication between the robots is local and not controlled by any global control system.

1.2.4 Cooperation Based Centralized Strategies

As is perceivable from the name of this set of strategies, they employ the use of a central monitoring and control unit, which helps all robots/agents navigate and makes changes to their paths according to various requirements are runtime.

1.2.5 Partitioning Based Strategies

Partitioning based strategies are based on resolving a **known environment** into segments that are independent from each other and are assigned to different robots for exploration and monitoring. This particular set of strategies is discussed in most detail in the later sections of this paper.

1.2.6 Contamination Based Strategies

These are a type of algorithmic technique that assumes that all segments of a partition of a map/environment are "contaminated". Contamination can refer to any type of problem according to the task at hand e.g. a bomb in a bomb-disposal problem etc. All these segments are then "de-contaminated" by different robots/agents keeping in view that no de-contaminated segment gets contaminated again.

1.3 Motivation

As the security threat in the world is increasing day by day, so is the utilization and need of multiple mobile robots for searching and surveillance tasks in harsh and dangerous environments. Similarly, the requirement of robots in the industry has gone up significantly. Such problem have been widely addressed through mobile robots because their cost has significantly decreased with a remarkable increase in their capabilities. These advancements have led research towards utilizing robots rather than humans for accomplishing various tasks because:

- The tasks are monotonous and very dull.
- The tasks involve a certain element of danger for people, such as bomb disposal, hostile encounters etc.
- The tasks can be accomplished by robots more efficiently and accurately rather than by humans.
- Robots can be cost effective.
- Use of robots can result in reduced manpower requirement.

Because of all these reasons, the requirement of not only industrial robots, but service robots as well has gone up notably. According to the International Federation of Robotics (IFR), significantly growing commercial activities related to personal and service robots have been identified [1]. Similarly, the Korean Institute for Robot Industry Advancement has signified an expansion of 40% every year in the service robot sector [2]. This can be seen in the following figures:

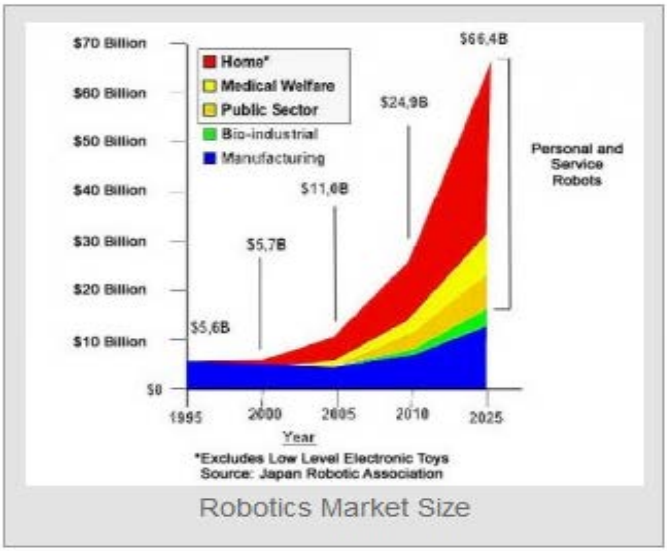

Figure 1.1: Robotics market size

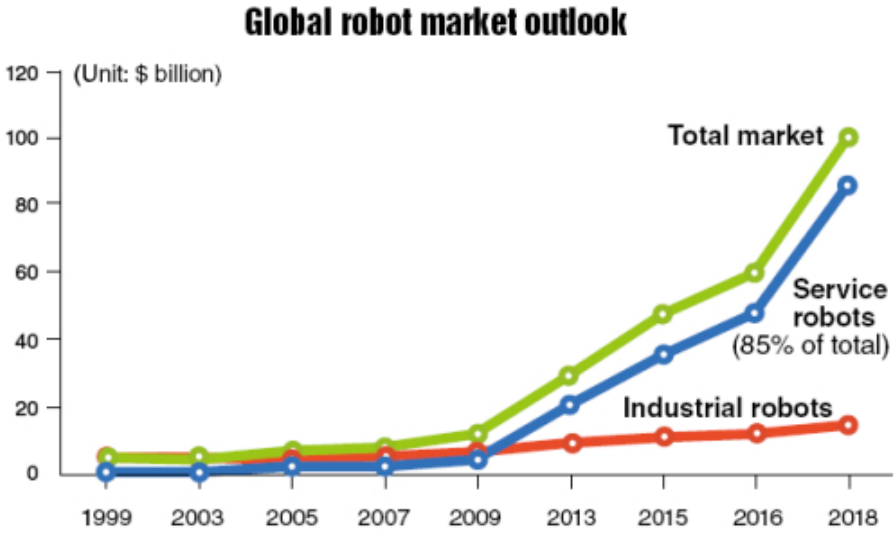

Figure 1.2: Global market outlook (KIRIA)

In order to fulfill this ever increasing global demand for service robots, not only the hardware, but the software/algorithmic half of these robots needs to be made more efficient. This work can be seen in a similar perspective, as it aims to implement a service robot algorithm to efficiently carry out a search and secure task that can be dull and dangerous for humans to perform. Hence, a solution is proposed for a similar problem in the chapters ahead.

1.4 Scope

The scope of this work is to implement a partitioning based strategy to partition a given environment into smaller regions that are easier to analyze and form a connectivity graph from these regions. After an analysis of the connectivity graph, the number of robots, their positions and paths are identified for an **Uncoordinated Decentralized Search** of the area. Each robot is assigned a specific role which can either be to stand and cover a certain location or to follow a certain path surveying different regions accordingly and are declared blocking robots or searching robots according to their defined roles. The results of the work are compared with other similar work done for validation purposes.

1.5 Thesis Outline

In this thesis, an algorithm for complete search-and-secure coverage of an environment is presented. A simulation is performed for the environment and the corresponding paths for mobile robots are accordingly identified.

In Chapter 1, the title of the research and the scope of work are described, along with a brief background of the research objectives and some important terminologies used and their definitions are listed.

In Chapter 2, the problem and corresponding objectives are stated. Afterwards, important fundamental concepts studied as part of the Literature Review and Research carried out for the study of various techniques and algorithms being used in this work are presented.

In Chapter 3, the Analytical Model and Research Methodology identified and implemented to accomplish this work are presented with brief results required to emphasize the outcome of each step.

In Chapter 4, case studies are presented as well as a comparison of results with a similar research project already carried out.

In Chapter 5, an implementation of the algorithm on an actual map of a specified location as well as its detailed results and discussions are presented.

In Chapter 6, the conclusion and different application areas of the research work are presented.

CHAPTER 2: OBJECTIVES AND LITERATURE REVIEW

2.1 Literature Review

The use of Unmanned Ground Vehicles (UGVs) for the "search and secure" problem is an application of great interest that allows searching an area and securing it using one or multiple autonomous robots based on the complexity of the area to be searched.

Search and secure involves two aspects; navigation and sensing. Developers have used different basic techniques to define navigation algorithms. These methods may use a variety of approaches, some of which may be same or similar. Yet, each method has distinct logical features, which differentiates it from other methods. The goal of this manuscript is to further the developments in robot navigation. Each navigation algorithm engenders different sensing and securing capabilities, hence the algorithms are treated as "search and secure" algorithms instead of just navigation. This section initially defines and describes the background for different search and secure algorithms as briefly mentioned in the previous chapter, which helps the reader identify why "partitioning strategies" are used in this study. Further, this chapter goes into the detail of partitioning strategies to bring the reader up to speed with the developments in the field up till now, so that he may do justice to the work presented in this manuscript.

2.2 Search and Secure Algorithms Overview

This section, as has already been described, takes a look at various navigation techniques employed for area searching and securing. All techniques have certain advantages and disadvantages. Some of them offer cost efficiency, while others focus on saving time. Some are more robust in terms of their tolerance to failure of one or more robots (or "agent"), while others are based on independence of each agent from the other in order to complete a task. Some strategies implement their navigation method based on global information of the area to be explored, while others use local decision making to autonomously make navigation decisions without knowledge of the global scenario. Yet, a few involve communication between different agents to tackle an environment, compared to independent partitioned tasks. The task accomplished in this section is to establish which of the navigation algorithms allow us to explore the requisite searching and sensing capabilities targeted in this thesis.

2.2.1 Cyclic Strategies

Cyclic strategies involve identifying the whole area to be explored and obstacles present in it as a set of vertices [3]. The path formed by joining of the vertices is termed as "Hamilton Path", and if a cyclic patrol is established, the cycle is known as "Hamilton Cycle" [4]. To compute the possibility of existence of Hamilton cycles within a certain map or "graph", extensive procedures have been established [5]. These help in determining a certain path for the robots to travel and, hence, are essential to the employment of this technique.

Once the Hamilton path or cycle is identified, a closed loop is established that covers all the vertices, and the robot team travels along that same path one-by-one. Each robot will "clear" all the possible hideouts of an intruder or locations of any static object by traversing through every vertex in the area. Yet, it is easily observable that an intelligent intruder may observe and identify the route taken by each robot and sneak in between two robots in the same cycle to remain unobserved at all times. Hence, this technique is only viable for static object searching, securing, and patrolling, but does not have the requisite properties to serve as a search, secure, and decontaminate technique for a dynamic intelligent intruder.

2.2.2 Markers-based strategies

As mentioned earlier, Markers-based strategies are multi-robot, decentralized, non-communicative, local search techniques, which employ deployment of tags to identify areas which have already been explored by one of the robots in the team. In this technique, there is no knowledge of the map prior to the start of exploration. The result is dynamic path identification by every robot based on tags or markers placed in its local region. Different studies have used this technique using different algorithms; one strategy [6] encompasses the algorithm: "A robot explores as long as there are open regions left. If all the regions are explored, then the robot picks the direction which was least recently explored." Another strategy is inspired by the exploration method used by ants [7], which also drop a marker fluid to define their path. The navigation probability of a robot in this strategy is highest in the direction most used by other robots.

As can be easily understood from the description, the target of this strategy is more towards defining patrolling routes and exploring an unknown area efficiently in terms of time. Though this technique may be used for searching and securing for static objects, its use for searching to decontaminate the area seem inadequate, as an intelligent dynamic intruder may easily observe the navigation path of the robots and find a pattern to identify unobserved

regions in the area at any given time. A dynamic intruder may use the pattern over time to keep hidden from the robots, thus making this strategy unfavorable for the study at hand, and rather more suited towards exploration and patrolling.

2.2.3 Communication-based decentralized strategies

The name of the technique makes it quite evident that there is at least some form of direct communication between robots at some point in time during their mission. As this technique is decentralized, it means that the communication is local, and not fed into a global monitoring and controlling system. The type and method of communication is solely the discretion of the specific algorithm developed for a team or a mission. Navigation strategies are based on the fact that the map is partially or completely unknown at the start of the mission, hence requiring use of local navigation algorithms.

There are two communication strategies used by Sgorbissa et al. [8]: goal sharing - which identifies to a robot what targets the other robots have to coordinate local movement; and state sharing - which helps robots without clearly established goals to communicate with nearby robots to get help with the identification and establishment of goals. Dollarhide et al. [9] have shown the viability of communication-based decentralized strategies for searching and securing for static objects, using different algorithms. Yet, this technique seems inadequate for searching to decontaminate, as ignorance of the map area and presence of loopholes in the dynamic search pattern may easily be exploited by an intelligent intruder.

2.2.4 Cooperation-Based Centralized Strategies

These techniques have a central monitoring and controlling unit, which preplans the navigation of all robots in the mission as well as makes requisite changes during the mission. Mapping of the environment is necessary for preplanning, and dynamic mapping is also featured in many algorithms and studies to allow for threat response [10] or enemy location based navigation [11].

This technique allows for dynamic changes in the search routes taken by any given robot based on malfunctioning of one or more robots, identification of a static or dynamic threat and the type of response initiated to deal with it, and emergence of environmental changes. Hence, this provides the capability to search and secure, as well as decontaminate a map. Yet, it is quite evident that the equipment is expensive, planning is quite difficult, and high intelligence in the algorithm is necessary to take care of the uncertainties of the dynamic situation.

2.2.5 Partitioning Strategies

Partitioning strategies are based on resolving the known area of a map into independent segments, which are then assigned to different robots for exploration and monitoring. This has been shown by Wurm et al. [12] to reduce the exploration time for unknown areas of the map significantly. Also for known areas of the map, searching and securing by this method is the most time efficient, which has been proven through simulations as well as real-time experiments. This is also a very good patrolling strategy as well, if the navigation algorithm is based on the probability calculations of least monitored area at any point in time [13]. This increases the probability of observation of each area and significantly reduces chances of external intrusion, as the route of each agent is dynamic and cannot be predicted by an intelligent intruder, unlike in cyclic patrolling techniques.

Contamination-based strategies are another set of algorithmic techniques, which are primarily based on partitioning. The basic strategy of decontamination is to assume that all edges and vertices in the map are contaminated and contamination may flow from a vertex to adjacent edges and vice versa. The objective is to observe and, hence, eliminate the possibility of or identify all contaminants present.

Once a vertex or edge is observed to be "clear", the algorithm has to make sure that there is no possibility of the cleared vertex or edge to be contaminated again. As any obstacle is a closed body, the edges form a cycle which may be traversed by an intruder or a contaminant without observation if a single agent attempts to observe it by itself. Hence, every obstacle serves as a cycle, which has to be broken by using multiple observers at the same time to clear it. To achieve this, a few agents are placed in the map area or "graph" as static "blockers", which break the cycles present in the graph. Other agents serve as "searchers" who move through the graph to decontaminate/clear the vertices and edges not observed by the static blockers, in a manner that the decontamination is permanent [14].

The blocker and searcher placement algorithm is based on furthering the partitioning strategy. A popular segmentation technique is triangulation, using Delaunay algorithm. Combining triangulated regions to form convex polygons, and then joining the centers of adjacent regions results in a diagram known as "Voronoi Diagram" [15]. This diagram is used to identify cycles present in the graph and locate suitable blockers positions and searcher paths. Optimal partitioning techniques are being developed [16], which are the focus of this study as well. The next section focuses on this very topic, and aims to identify the research that has already been done to develop a base for the work presented in this manuscript.

2.3 Developmental Stages of Partitioning-based Algorithms

Every search and secure operation involves determining how many robots will be required for it, what navigation routes will be implemented, how much time it will take, and what will the cost of the operation be. To make use of robots for such tasks feasible, it is necessary to keep the costs minimal, hence the use of the least number of robots for the least amount of time is the goal. Any "strategy" implemented to perform a "clearing" operation in an area or "graph" has an associated "strategy cost" [14].

2.3.1 Early Developments

Initial attempts at using partitioning algorithms for search and secure missions involved clearing contaminated edges in a graph [17]. These were improved upon in further researches [18]-[21], such that they led to the "graph-clear" problem. More complex methods evolved [22], allowing planar regions of simply connected environments to be cleared by a single mobile gap detector, when combined with static blockers as per requirement. The problem was shown to be NP-complete by Parsons [17], with strategies extracted from a minimum spanning tree constructed by the algorithm. Though the algorithm suggested good strategies for blocker placement use of moving blockers, it was proven to be strategically non-optimal and not as cost-effective as possible, hence leading to further research.

2.3.2 Objective-based Algorithm Development

Gerkey et al. [23] addressed the issue of algorithm development by revising sensory capabilities of robots and introducing new ones, while Ge and Fua [24] targeted the use of the least number of robots (say, one) and develop search algorithms that would sweep a given area completely, with least amount of repetition, and in the least amount of time. Isler et al. [25] have targeted minimizing the probable decontamination time of an area with an intelligent and fast intruder, using randomized navigation strategies for even a slow agent. This strategy employs the cost-efficiency principle by minimizing the number of agents used to one. Further, it involves the use of randomized movements, instead of pre-determined clearing movements, as might be the case with generic partitioning-based algorithms. The research showed that the unpredictability of the movement of the searching agent allows it to outsmart the faster moving intruder as it cannot use its intelligence effectively. Though time consumption was comparable to pre-determined navigation strategies, the reduction of preprocessing costs was deemed significant.

2.3.3 Search and Secure Using Static Robots

A research by Ganguli et al. [26] somewhat similar in spirit to what is considered in this manuscript is related to the "art gallery problem". This research presents its own method of partitioning a simple polygonal environment. The name given to the technique, "The vertex-induced tree", involves division of a polygonal environment into star-shaped subsets, the technique of which is described in detail by the researchers. The differences to the current research are communication (though limited) between different agents, agent-based decision making, low prior knowledge of the area they are to be deployed in, and static locations after deployment for observation of "an art gallery".

All the above researches give a clue regarding the variability among different techniques used for developing partition-based algorithms. It is evident that identification of the objective of the algorithm is of primary importance, as it ascertains the type of techniques which might be utilized. In lieu of this, a summary of the parent research for the current study is presented in the next section.

2.4 Search and Secure using Mobile Partitioning-based Robots

The search and secure algorithm developed for using mobile robots in the parent research by Katsilieris [27] targeted sweeping an area with rectangular obstacles. The objective of the research was not the sensing itself, but to develop a navigation algorithm, which could be coupled with independent sensory control, processing, and feedback algorithms. The output of the algorithm was purely navigation trajectories, not robot control based on environmental inputs. It was supposed that the map environment and obstacle location was known beforehand, which was an input parameter for the algorithm to produce navigation trajectories as outputs. All robots were to behave mutually independently, only performing based on their own pre-determined objectives. No inter-robot communication was involved in any way. This section describes the steps in the algorithm through which the area graph was converted to the navigation codes.

2.4.1 Segmentation and Merging

The segmentation technique employed in this study is Delaunay triangulation. Triangulation simply refers to joining sets of three vertices in a manner such that the whole region is divided into non-overlapping triangles. Generally, there would be multiple ways of dividing a map with obstacles into triangles. Hence, this process requires optimization, which is where the Delaunay algorithm comes in. It prevents the formation of triangles in which

very acute angles might be present, which hinders the next set of steps. Still, this methodology is open for further optimization, and specifics of the implementation of the Delaunay algorithm change with different studies.

After the formation of triangles within the graph, they are merged to form convex polygons of the largest possible areas. An optimal implementation of this step along with optimal triangulation would lead to the best segmentation. If the segmentation is optimal, lesser regions would be formed in a given graph than any other possible segmentation solution. This would lead to lesser points to cater for in the steps highlighted ahead. Hence, this is necessary for optimization of the algorithm, which has been targeted by the author of the current research.

2.4.2 Comparison of Voronoi and Delaunay Triangulation

When a triangulation segmentation technique is in consideration, it is always best to perform a Delaunay Triangulation first and then create the connectivity graph by joining the centroids of all adjacent triangles. This results in lowest errors, 10% less than those produced by Voronoi Diagram construction directly [28]. Hence, Delaunay Triangulation is preferred in the coming chapters for partitioning of an environment.

2.4.3 Creating Minimum Spanning Tree

The algorithm further takes the convex regions and joins the centroids of all adjoining regions, essentially creating a Voronoi Diagram. This diagram is analyzed for loops by the algorithm, which might be used by an intelligent intruder to evade a searching agent (a problem discussed earlier in cyclic strategies). They are broken by identifying "cover nodes" to form a "minimum spanning tree". Generally, each obstacle is a cycle, hence thumb's rule suggests that there would be as many cover nodes as the number of obstacles present in a graph.

2.4.4 Determining Robot Positions and Tree Components

Once the position of cover nodes are identified, robot positions are calculated by the algorithm catering for all cover nodes. This essentially reduces the number of cycle breakers from the number of cover nodes to the new number of blockers based on the calculations of this step. This results in the formation of a "reduced minimum spanning tree". The root, branches, and leaves of this tree are identified, using their standard definitions. This helps

determine the nodes of the reduced minimum spanning tree to be searched by the agents. The root is the starting point of all the searchers.

2.4.5 Identifying Robot Paths

Blocker paths originate from the root and follow the nodes on the minimum spanning tree to reach their destinations. The target is to spend the least amount of time in travelling, hence the least time consuming path should be used to take the blockers to their final positions. Yet, the research being summarized employed a non-optimal path formed through the minimum spanning tree nodes.

Searcher paths are determined based on the reduced minimum spanning tree. The number of searchers is determined by the number of branches in the tree. The waypoints determined by the spanning tree might have a collision with an obstacle, as they are based on joining centroids with straight lines. So, intermediate path points are established by the algorithm by taking midpoints of adjoining edges of the RMST.

The critical point to be noted is that there are multiple possible improvements that this algorithm may undergo and this research focuses on some of those possibilities, as described in the later chapters.

2.5 Problem Definition

The problem can be defined as: Given a boundary/area with obstacles (multi-dimensional polygons), search the area completely for any intruders such that the whole environment is surveyed and no intruders are able to sneak back into an area (without being detected) that has already been secured. Some important considerations in this regard are:

- The surface of the environment specified should be level/smooth and easily traversable for the robots.
- The robots are assumed to have 360 degree unlimited vision.
- The algorithm is not aimed at characterizing the control algorithms of the robots, rather it defines the paths/trajectories that they have to follow.
- The algorithm does not imply usage of image or video processing techniques to actually detect an intruder. It provides the robot paths and the corresponding video streams that are henceforth produced by the onboard cameras. These streams can be analyzed by an observer or by implementing a video processing algorithm.

2.6 Problem Solving Options

Two problem solving options are available for obtaining desired solutions. Developing code in C or MATLAB. MATLAB is chosen as the preferred option as it provides computational efficiency regarding matrices and a graphical interface to easily visualize the results in different ways.

2.7 Thesis Objectives

Following are the Objectives and Deliverable Milestones for this research:

- Analytical Analysis of an environment/map for abstraction in the form of a graph.
- Consolidation/Integration of this graph to get a higher abstraction level representation of the environment.
- Detection of loops/cycles in this representation.
- Identification of required number of robots/agents along with their positioning and trajectories for complete coverage of the environment.
- Case studies and comparison with results of existing targeted algorithms.
- Implementation of the algorithm on actual map of a specified location.

CHAPTER 3: ANALYTICAL MODEL AND RESEARCH METHODOLOGY

3.1 General

The Search and Secure algorithm presented in this chapter is based on division of the environment to be analyzed in convex regions. These convex regions are then abstracted in the form of a graph known as the regions graph with centroids of the regions forming the nodes/vertices of the graph. The cycles in this graph represent paths that begin and end at the same vertex, hence providing loops for the intruder to evade the searching robots / agents. These loops/cycles are broken using static blocker robots (referred to simply as blockers in the forthcoming text) so that the intruder cannot "sneak back" or escape the robots / agents pursuing or searching for the intruder. Once the blockers reach their identified positions to break the cycles, the searching robots (referred to simply as searchers in the forthcoming text) are deployed in the area to search for any intruders and secure it.

3.2 Analytical Model and Research Methodology

This section explains in detail the Research Methodology used and the Algorithm developed henceforth.

3.2.1 Algorithm Overview and Description

The flow diagram of the algorithm is depicted in Figure 3.1. Following are the basic sections in which the algorithm is divided and is discussed in detail in the later text.

- **Triangulation:** Divide the environment into triangles. After a basic triangulation, the triangles are analyzed according to the Empty Circle Property regarding **Delaunay Triangulation** and **Lawson Flip Algorithm** to finalize the triangulation results.
- **Merging to Regions:** The Delaunay Triangles are merged to form **convex regions**.
- **Graphical Abstraction:** The centroids of the merged regions are computed and an adjacency matrix is formed which enlists the neighbours of each region. The centroids are then connected according to the adjacency matrix to form a **graph**.
- **Cycle Detection:** The cycles in this graph are detected through a **Depth First Search** (DFS) of the complete graph. Then in order to get all **internal cycles**, an exclusive or (XOR) of the cycles obtained from the DFS is done.

- **Static Blocker Agents:** According to the cycles, **positions** of blocker agents or **blockers** are determined in order to break all cycles. Finding the cycles and blocker positions is an iterative process and blockers are placed intelligently until all cycles are broken.
- **Generate Reduced Minimum Spanning Tree:** Once the blocker positions are determined, the regions graph is converted into a Reduced Minimum Spanning Tree (**RMST**) by removing all the regions covered by blockers. The RMST has no cycles and forms the search tree which is to be searched and secured by searcher robots.
- **Search Tree:** The RMST is analyzed to identify the **branches** of the tree. The number of branches equals to the number of searcher robots required.
- **Blocker Paths:** Create the paths for blocker robots. These paths are created on the basis of **Dijkstra's Shortest Path Algorithm** so that the blockers reach their designated positions in minimal time.
- **Searcher Paths:** Create the paths for searcher robots. This also creates shortest paths for searchers keeping certain criteria in consideration.

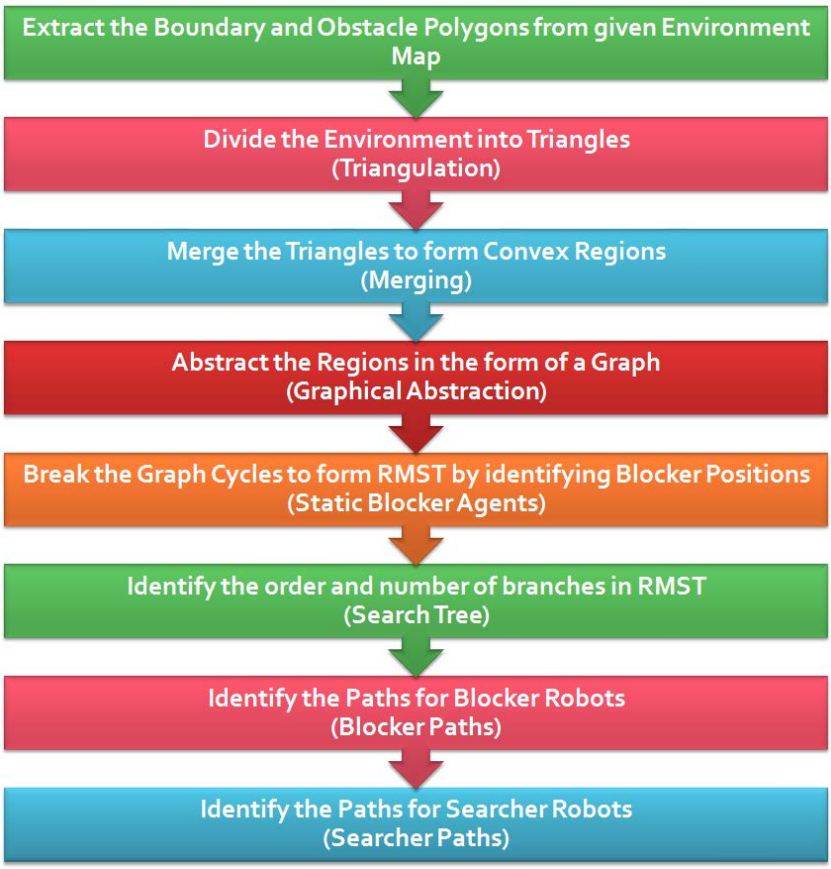

Figure 3.1: Flowchart of the algorithm

All these will be discussed in detail in the following sections.

3.2.2 Boundary and Obstacle Matrices

The first step is to generate boundary and obstacle matrices for further use. This involves the following matrices:

$$S\_obs_i = \{(x_j, y_j), i = 1\ to\ n\_obs\ , j = 1\ to\ n\_obs^i\}$$

$$S_{obstacle} = S\_obs_1 \cup \dots \cup S\_obs_i \cup \dots \cup S\_obs_{n\_obs}$$

$$S_{boundary} = \{(x_k, y_k), k = 1\ to\ n_{bound}\}$$

$$S_{all} = S_{boundary} \cup S_{obstacle}$$

And if an unpathable region also exists, then:

$$S\_unp_i = \{(x_m, y_m), i = 1\ to\ n\_unp\ , m = 1\ to\ n\_unp^i\}$$

$$S_{unpat\ hable} = S\_unp_1 \cup \dots \cup S\_unp_i \cup \dots \cup S\_unp_{n\_unp}$$

$$S_{all} = S_{boundary} \cup S_{obstacle} \cup S_{unpat\ hable}$$

where

n_obs: Total number of obstacles

n_obs^i: Total number of corners of obstacle *i*

n_unp: Total number of unpathable regions

n_unp^i: Total number of corners of unpathable *i*

n_{bound} : Total number of boundary corners

S_obs_i: Structure of corner points of obstacle number *i*

S_unp_i: Structure of corner points of unpathable number *i*

$S_{obstacle}$: Structure of all obstacles

$S_{unpat\ hable}$: Structure of all unpathables

$S_{boundary}$: Structure of all corner points of boundary

S_{all} : Structure of all corner points

x_j, y_j : x and y coordinates of j-th corner point of respective obstacle

x_k, y_k : x and y coordinates of k-th corner point of boundary

x_m, y_m : x and y coordinates of m-th corner point of respective unpathable

Every point is assigned a unique ID which is used for reference in the next steps. A vertices array is formed such that:

$$V_i^{all} = [(x_i, y_i), i = 1\ to\ n]$$

where

n: Total points

V_i^{all} : Vertices array of all points

i: ID of i-th point

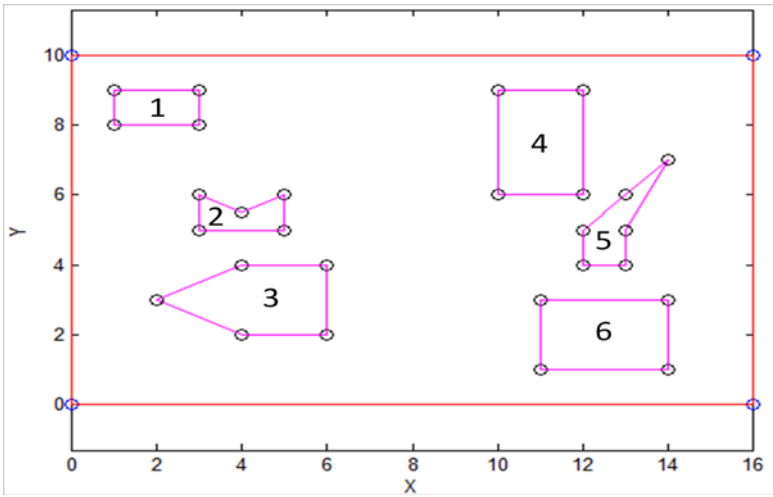

Figure 3.2: Example Case - Boundary and Obstacles

Figure 3.2 shows an example case of extracted Boundary (red) and Obstacles (magenta) numbered 1-6 from a given set of points.

3.2.3 Visibility Graph of Corners

After forming the aforementioned structures, a visibility graph is formed which basically checks the visibility of each corner with the other. To check the intersection mentioned in the algorithm below, all unpathable regions are assumed to be obstacles and are temporarily added to the $S_{obstacle}$ structure. The algorithm devised for that purpose is as follows:

1. $S^{i}_{visibility}$ = visibility_all_points()
2. j = i+1;
3. Loop until j <= n
4. line = {$[x_i\ x_j]$, $[y_i\ y_j]$};
5. INT = Check Intersection with all Obstacles, Unpathables and Boundary
6. if (INT == 1)
7. // i.e. the line passes through any obstacle, unpathable or boundary or // intersects multiple obstacles/unpathables or boundary
8. j is not visible to i;
9. break;
10. else
11. j is visible to i;
12. Add j to $S^{i}_{visibility}$ matrix;
13. Add i to $S^{j}_{visibility}$ matrix;

14. j = j+1;
15. end
16. end
17. return

Finally:

$$S_{visibility} = S^1_{visibility} \cup \ldots \cup S^i_{visibility} \cup \ldots \cup S^n_{visibility}$$

where

$S^i_{visibility}$: Visibility array of i-th corner

$S_{visibility}$: Visibility matrix of all corners

n: Total number of points

The $S_{visibility}$ matrix gives the visibility of all points w.r.t. each other and is then used in the next steps and the $S_{obstacle}$ structure is restored back by removing all unpathable regions from it.

3.2.4 Triangulation

The Triangulation function segregates the environment into triangles according to the **Delaunay Triangulation Algorithm**. The Delaunay algorithm creates non-overlapping triangle meshes for FEM (finite element method). The main feature of the Delaunay Triangulation Algorithm is that it maximizes the smallest angle in a triangle, thus minimizing the chances of creation of thin and long triangles. Firstly, an initial Non-Delaunay Triangulation is formed which is later converted into Delaunay Triangulation through the Lawson Flip Algorithm. The triangulation technique used follows these steps:

- Sorting the points in a lexicographical order.
- "Seeding" the triangulation with a triangle formed by the first three points of the lexicographical sort. Seeding means to start the triangulation, an initial triangle is formed. If the first three points selected for the seed triangle don't make a triangle (not visible to each other according to the $S_{visibility}$ matrix), then the next point is selected and checked until a seed triangle is formed. This becomes the current triangulation matrix.
- A convex hull is initiated from this seed triangle which forms the outer edge of the union of all triangles in the triangulation matrix.
- The convex hull is formed by using the **Graham Scan Algorithm**. The pseudo code for the algorithm used is given below.

- Adding points one by one to the current convex hull in lexicographical order according to the visibility matrix and checking the number of triangles thus formed. The triangles formed will depend on the visibility of the point being added w.r.t. the points in the current hull i.e. it is checked according to the matrix $S_{visibility}^{hull}$. and the triangles are thus formed according to the hull points visible to the point being added.
- Expanding the hull and triangulation matrix until all the corner points in the environment are added and all triangles are formed.

Following is the algorithmic representation of the initial triangulation:

1. Sort S_{all} in lexicographical order
2. $S_{triangle}^{m}$ = $\{V_l^{all}, V_p^{all}, V_n^{all}\}$;
3. Check visibility of $\{V_l^{all}, V_p^{all}, V_n^{all}\}$ with each other through $S_{visibility}$
4. $S_{triangle}$ = $S_{triangle}^{m}$;
5. S_{hull} = convex_hull($S_{triangle}$, pt_next);
6. Loop i < n
    a. $S_{remaining}$ = S_{all} - $S_{triangle}^{m}$;
    b. pt_next = $S_{remaining}^{i}$;
    c. Check visibility of pt_next w.r.t. hull and form $S_{visibility}^{hull}$
    d. S_{hull} = convex_hull(S_{hull} , pt_next);
    e. $S_{triangle}^{m}$ = $\{V_l^{all}, V_p^{all}, V_n^{all}\}$;
    f. $S_{triangle}$ = $S_{triangle}$ $\cup$ $S_{triangle}^{m}$;
    g. i = i+1;
7. end
8. return

where

V_l^{all}: First vertex in triangle

V_p^{all}: Second vertex in triangle

V_n^{all}: Third vertex in triangle

$S_{triangle}^{m}$: m-th triangle, where m = (1 to total_triangles)

$S_{triangle}$: Structure of all triangles formed

S_{hull}: The convex hull formed by all the triangles

$S_{remaining}$: Points that have not been included in the convex hull yet

In order to find the convex hull, the Graham Scan Algorithm [29] is used. All the points that form the hull are sorted in a lexicographical order and the first point is then selected (which is basically the lower-most point on the left). The angle of all other points from this point w.r.t. the positive X-axis is calculated (starting in CW direction) and the points are sorted on the basis of increasing angle. If two points are at the same angle, the point at the lesser distance is placed first. Then the points are sequentially added to make a hull, checking for other points lying to the LEFT of the current point being added. If there is any point to the left, it means it is to be added first to form a convex hull. The left or right (LorR) check is basic geometrical calculation and the hull is completed accordingly. If any points to be checked lie inside the hull, they are removed from the hull boundary array. The pseudo code for Convex Hull formation using the algorithm explained above is given below:

1. convex_hull(S_{hull} , pt_next)
2. V_1^{hull} = First point after sorting S_{hull} in lexicographical order
3. S_{hull_angle} = Calculate angle of all S_{hull} points w.r.t. V_1^{hull}
4. Sort S_{hull} w.r.t. angle calculated
5. if ($S_{hull\_angle}^{i}$ == $S_{hull\_angle}^{i+1}$)
    a. Sort i and i+1 points w.r.t. smaller distance
6. end
7. S_{hull_order} = V_1^{hull};
8. Loop i < total_hull_points
    a. P_0= V_i^{hull};
    b. P_1= V_{i+1}^{hull};
    c. P_2= pt_next;
    d. if(P_2 is present in $S_{visibility}^{i}$) // i.e. P_2 is visible to P_0
        i. LorR check P_2 point being added to S_{hull_order};
        ii. if (No other point to left of P_2)
            1. S_{hull_order} = S_{hull_order} ∪ $V_{pt\_next}^{all}$;
        iii. else
            1. Add the point P_1 on the left into the hull;
        iv. end
    e. else
        i. Add the point P_1 on the left into the hull;
    f. end

g. i = i + 1;

9. end
10. return

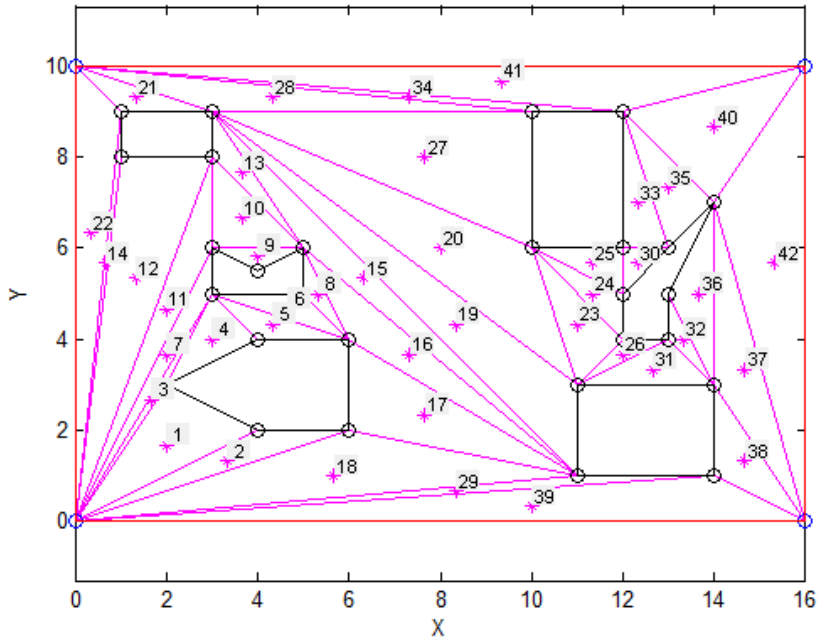

Figure 3.3: Example Case - Initial Non-Delaunay Triangulation

Figure 3.3 shows the results of basic Non-Delaunay Triangulation performed on the example case. A total of 42 triangles (magenta) are formed with their centroids marked as stars and numbered accordingly. This clearly shows that there are a large number of thin and long triangles formed this way.

After forming a basic triangulation, the triangulation is checked for the Empty Circle Property of Delaunay Triangulation.

Empty Circle Property: The triangulation of a finite set of points $S \subset R^2$ is a Delaunay Triangulation if the circumcircle of every triangle in the triangulation has no other point in it, i.e. it is empty [30].

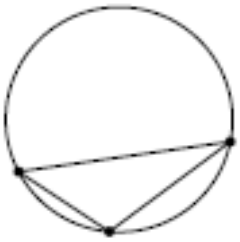

Figure 3.4: Circumcircle of a triangle

If any other point lies inside the circumcircle of a triangle, the triangulation is not Delaunay. So in order to make it a Delaunay Triangulation, the edges of the triangles need to be flipped. This is done according to the Lawson Flip Algorithm.

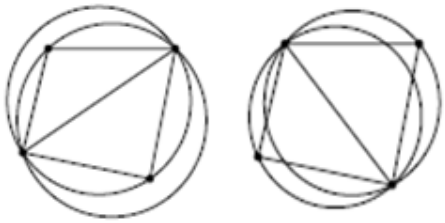

Figure 3.5: (a) Non-Delaunay Triangulation (b) Edge Flipped for Delaunay Triangulation

Lawson Flip Algorithm: The Lawson Flip Algorithm states that if there is a sub-triangulation of four points that is not Delaunay (Figure 3.5(a)), replace this sub-triangulation by the other triangulation of the four points [30].

So, Figure 3.5(b) shows a Delaunay Triangulation of the four points that formed a Non-Delaunay Triangulation in Figure 3.5(a).

In this way all triangles are checked for the Empty Circle Property and their Non-Delaunay Edges are flipped to form a Delaunay Triangulation.

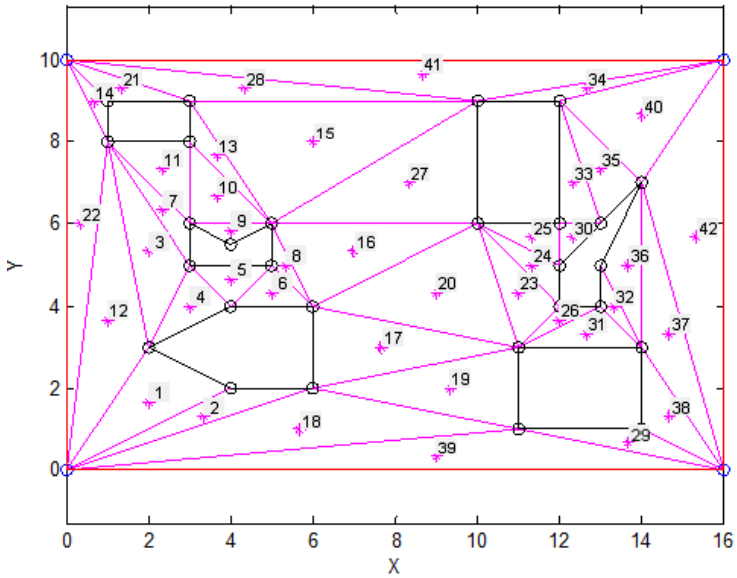

Figure 3.6: Example Case - Delaunay Triangulation

Figure 3.6 shows the results of example case after applying the Empty Circle Property and Lawson Flip Algorithm to the Non-Delaunay Triangulation. It can be seen that the thin and long triangles have been removed by flipping edges; for example the common edge between triangles 22 and 14 has been flipped. And by connecting the centroids of all these triangles, a **Triangles Connectivity Graph** is formed as shown in Figure 3.7. This is also the Voronoi Diagram as the Delaunay Triangulation and Voronoi Diagram have the property of duality. The triangles graph is modeled as:

$$S\_Graph_T = \{V_i^T, E_{ij}^T \ , i \ and \ j = 1 \dots total \ triangles, i \neq j\}$$

where

V_i^T: Centroid of i-th triangle

E_{ij}^{T}: Edge between V_i^T and V_j^T

The edges of this Triangle Graph also represent the adjacency matrix for each triangle.

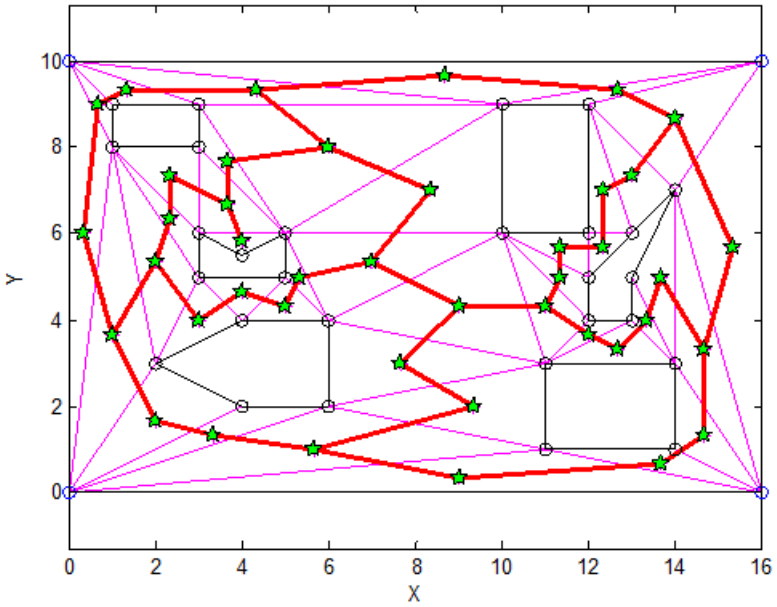

Figure 3.7: Example Case - Triangles Connectivity Graph (Voronoi Diagram)

3.2.5 Merging Triangles into Convex Regions

The triangles formed in the previous step are then merged together to form convex regions. Convex regions are important because any point within a convex region is visible by any other point inside the same region [27]. Consequently, a robot standing at any point in the convex region can secure it completely because any intruder cannot hide from it and can be detected clearly. The difference between a convex and non-convex region can be seen in [27].

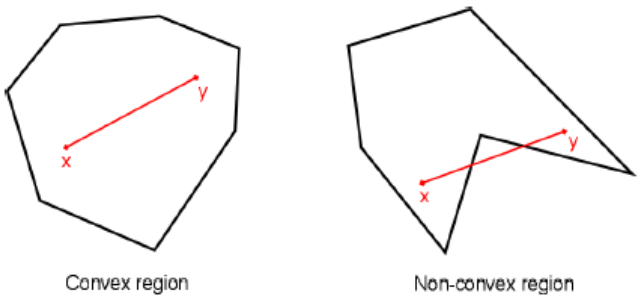

Figure 3.8: Convex and Non-convex Regions

In the parent study, convex regions are formed only by merging adjacent triangles once i.e. if a triangle is merged with another into a larger merged region, it is not further analyzed for merging with other triangles/regions. In the improved algorithm presented in this manuscript, in order to form **merged convex regions**, first of all adjacent triangles are

analyzed. If all their points are visible to each other, they are merged together to form a larger merged region that is convex. This larger region is then further analyzed with its neighbouring triangles and if any triangle can further be merged into this region, it is merged and the adjacency matrix is updated accordingly. This results in larger merged convex regions as compared to the parent study. All the triangles are checked and merged according to the criterion of forming convex regions.

The pseudo code for the algorithm implemented is as follows:

1. merge_to_regions()
2. S_{region} = $S_{triangle}$
3. while(i < total regions)
    a. j = i+1;
    b. while(j <= total regions adjacent to S_{region}^{i})
        i. S_{region}^{j} = j-th region adjacent to S_{region}^{i}
        ii. Check visibility of points in S_{region}^{i} and S_{region}^{j} from $S_{visibility}$ matrix
        iii. if (All points visible to each other)
            1. S_{region}^{i} = Merge S_{region}^{i} and S_{region}^{j}
            2. Delete S_{region}^{j}
        iv. else
            1. j = j+1;
        v. end
    c. end
4. end
5. return

where

S_{region} : Regions structure

$S_{triangle}$: Triangles structure

$S_{regi\,on}^{i}$: i-th region structure

S_{region}^{j} : j-th region structure

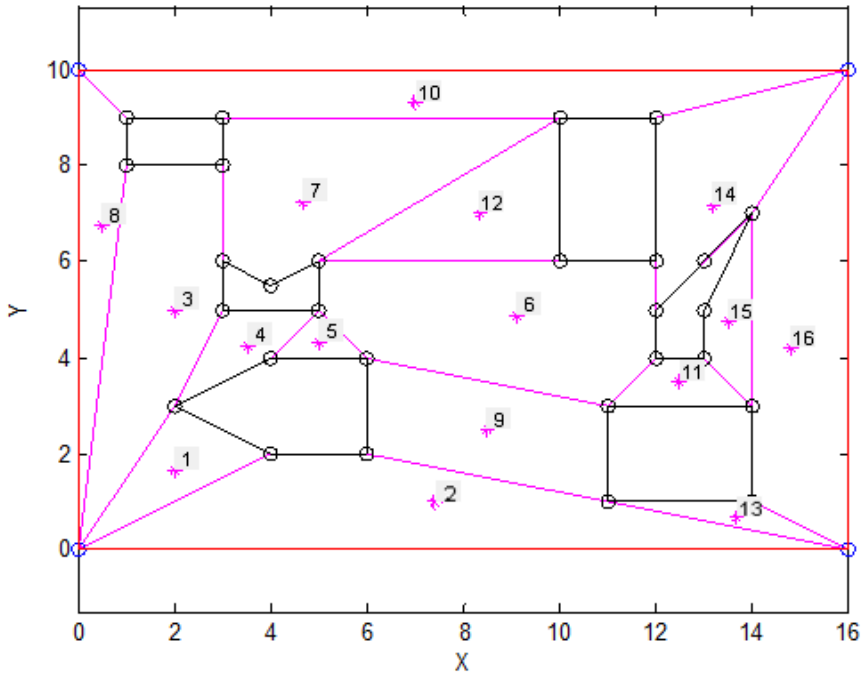

Figure 3.9: Example Case - Merged Regions

Figure 3.9 shows the results of merging the triangulation depicted in Figure 3.6. The 42 triangles have been merged into a total of **16** regions and as can be seen in the figure, all regions are convex.

3.2.6 Regions Connectivity Graph

A Regions Connectivity Graph or simply the **Regions Graph** is formed just as the Triangles Connectivity Graph was formed, since the Regions Graph and the Regions Connectivity Graph have the property of duality. It is formed by connecting the centroids of adjacent regions with each other. The centroids become the nodes/vertices and the lines connecting them are the edges of the graph. As the regions merging technique is improved (in previous step), it results in a better Regions Graph (Voronoi Diagram) in this step since the number of regions in the graph are reduced. The regions graph is modeled as:

$$S\_Graph_R = \{(V_i^R, E_{ij}^R)\ , i\ and\ j = 1 \ldots total\ regions, i \neq j\}$$

where

V_i^R: Centroid of i-th region

E_{ij}^R: Edge between V_i^R and V_j^R

The edges of this Regions Graph also represent the adjacency matrix for each region. The Regions Graph for the example case is shown in Figure 3.10. The stars (green) show the nodes and thick lines (red) show the edges between them. For the particular set of regions, this can also be called their Voronoi Diagram.

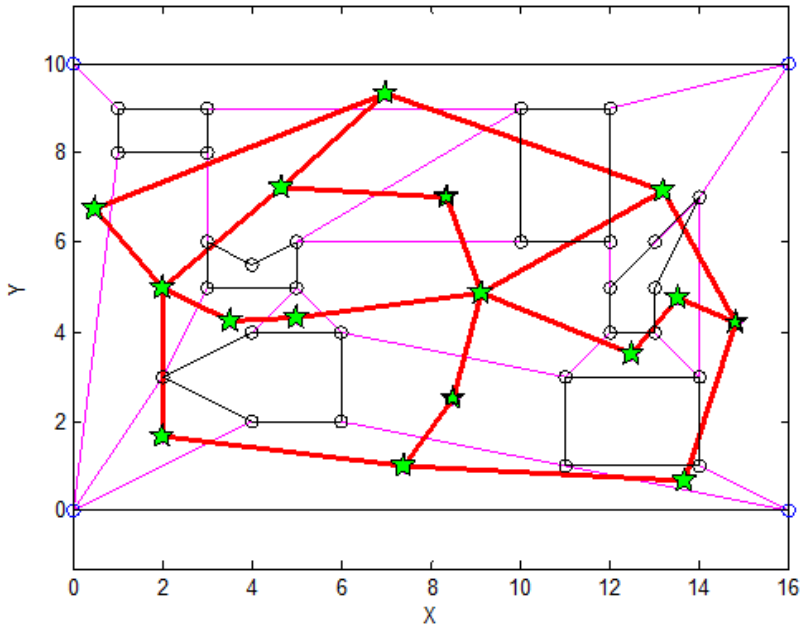

Figure 3.10: Example Case - Regions Graph

3.2.7 Cycle Detection

In this step, the regions graph is analyzed to detect loops/cycles. Cycles or loops in the graph result in paths that the intruder can use to avoid any searching agents. So, first of all the number of internal cycles present in the current graph are found, and then those cycles are broken at appropriate positions. By internal cycles we mean the cycles that are not subsets of each other and their combination results in the complete set of cycles present in the graph.

Various approaches can be used to find the number of cycles and their order in a graph. The **Kruskal's Algorithm** and **Prim's Algorithm** are the most used algorithms in detecting cycles and creating a Minimum Spanning Tree (MST). The Kruskal's Algorithm is a greedy algorithm based on selecting the globally optimal edge in a weighted edge graph. Similarly, the Prim's Algorithm selects the locally optimal edge. Both algorithms terminate when all nodes are visited once. The results of Kruskal's and Prim's Algorithms for an example case are as follows [31]:

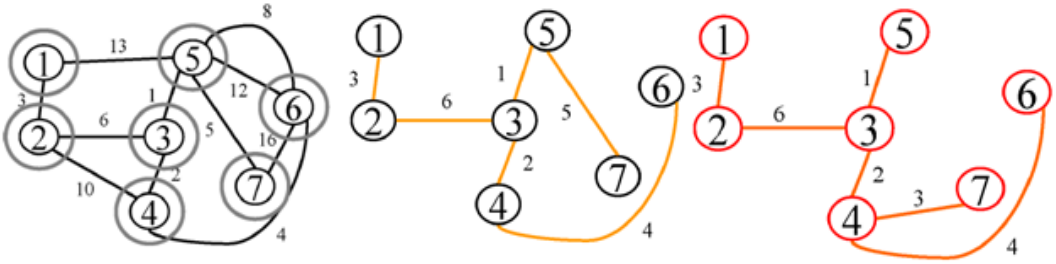

Figure 3.11: Results of Kruskal (yellow lines) and Prim's Algorithms (orange lines)

But instead of forming a MST as shown above, the algorithm in this manuscript simply detects the number of cycles and their order in this step and then creates a Reduced

Minimum Spanning Tree (RMST) in the next step by placing blocker robots at optimal locations. The number of cycles are calculated through a simple formula:

Number of Cycles in Graph = Number of Edges - Number of Nodes + 1

The cycles node order is found through the **DFS-XOR algorithm** explained below:

- Run a DFS on the whole graph. As the stack in the DFS fills up, exploration continues until all nodes are visited atleast once.
- Whenever a node is visited twice in the DFS, it shows that there exists a cycle; and all the nodes in between the node visited twice from the stack give us the cycle nodes order[Cycles Main Ref].
- When all cycles are found, these are not necessarily internal cycles; meaning there can exist a cycle in our cycles order array that might contain other smaller cycles inside it. So, all cycles are arranged in descending order w.r.t. cycle area.
- A XOR operation is performed between the edges of a cycle (parent cycle) and any other detected cycle (child cycle) if it lies inside the parent cycle. The XOR operation results in edges that are present in only one of these cycles, thus forming a new smaller cycle from the parent cycle. An example is presented below:

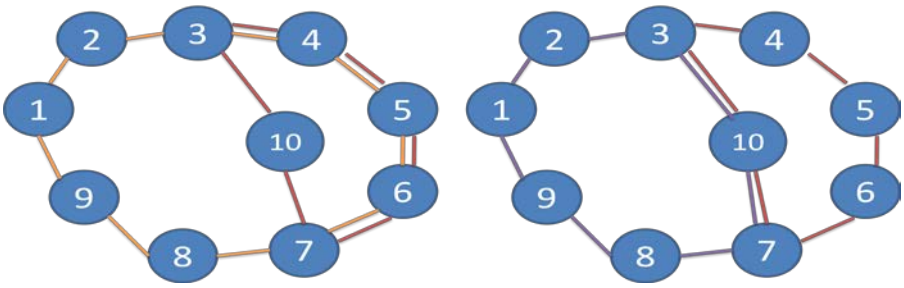

Figure 3.12: A XOR Example

Here the first graph shows a parent cycle (orange edges) and a child cycle (red edges). When their XOR is taken, the common edges between both cycles (3-4, 4-5, 5-6, 6-7) get eliminated from the resulting cycle and the uncommon edges form a new cycle (blue edges) shown in the second graph. This new cycle is saved in place of the parent cycle.

- This is done until all cycles are checked such that there is no other cycle within them, giving us the required internal cycles.

The psuedo code for the abovementioned technique is given as follows.

1. n = total_edges - total_nodes + 1;
2. i = 0;
3. dfs_stack = [];

4. while (i < n)
    a. Run DFS on whole graph and update dfs_stack
    b. if (last explored node is already in dfs_stack)
        i. Update $S^i_{cycles}.order$ from dfs_stack
        ii. Find area of cycle and Update $S^i_{cycles}.area$
        iii. Find centroid of cycle and Update $S^i_{cycles}.centroid$
    c. end
5. end
6. $S_{cycles} = S^1_{cycles} \cup ... \cup S^n_{cycles}$
7. $S_{area\_sorted\_cycles}$ = Sort S_{cycles} in descending order w.r.t. cycle area
8. i = 1;
9. while (i < n)
    a. updated = 0;
    b. for j = i + 1 : n
        i. if ($S^j_{area\_sorted\_cycles}.centroid$ lies inside $S^i_{area\_sorted\_cycles}$)
            1. Take XOR of edges of both cycles
            2. Update $S^i_{area\_sorted\_cycles}$ according to XOR result
            3. updated = 1;
            4. break;
        ii. end
    c. end
    d. if (updated == 0)
        i. i = i + 1;
    e. end
10. end
11. $S_{area\_sorted\_cycles} = S^1_{area\_sorted\_cycles} \cup ... \cup S^i_{area\_sorted\_cycles}$
12. $S_{cycles} = S_{area\_sorted\_cycles}$
13. return

where:

n: Total number of cycles in current graph

total_edges: Total number of edges in current graph

total_nodes: Total number of uncovered nodes in current graph

dfs_stack: Stack of explored nodes in DFS

S^{i}_{cycles} : Structure for i-th cycle

S_{cycles} : Structure of all cycles

$S_{area\ \_sorted\ \_cycles}$: Structure of all cycles sorted in descending order w.r.t. area

$S^{i}_{area\ \_sorted\ \_cycles}$: Structure for i-th area sorted cycle

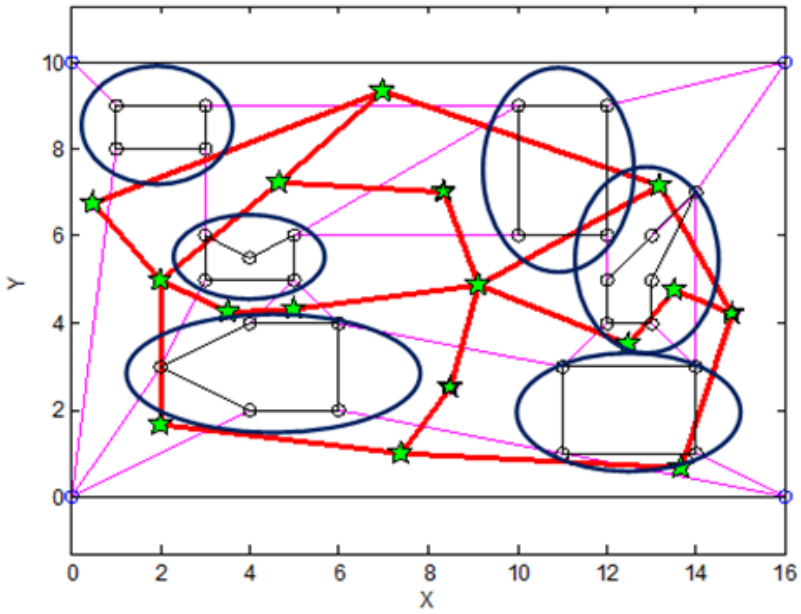

Figure 3.13: Example Case - Regions Graph (Cycles Depiction)

Figure 3.13 shows the cycles determined for the example case, depicting a cycle around each obstacle.

3.2.8 Static Blocker Positions

After finding the cycles in the graph and the regions/nodes that constitute them, positions for blocker robots or blockers are determined. In the parent study, the author first creates an MST by identifying cover nodes (nodes where the cycles are to be broken) and the blocker positions are determined only according to the set of cover nodes specified by the MST. This compromises the effectiveness of the blocker positions determined as they are limited only to the set of cover nodes determined (because multiple sets of cover nodes can exist for the same graph as shown by the Prim's and Kruskal's Algorithms briefly discussed in previous section).

In the algorithm discussed in this manuscript, an MST is not formed by determining cover nodes. Rather it is determined that which nodes effect the greatest number of cycles and then a vertex position for a blocker is finalized. The algorithm is explained below:

- From the structure S_{cycles} , the order of all cycles ($S^{i}_{cycles}.order$) is analyzed and an array which shows the number of times a region occurs in any cycle (region is a part of a cycle) is formed.
- The array formed in the last step is sorted according to descending order w.r.t. the number of cycles effected by a region.
- The corner vertices of regions effecting the maximum number of cycles are analyzed to prioritize the vertices according to the following criteria:
  - The maximum number of cycles effected by the vertex. If the number of cycles effected by any two or more vertices are equal, the maximum number of regions covered by a vertex. Furthermore, if the number of regions covered by any two or more vertices are also equal, they are prioritized according to the maximum area covered by a vertex (the total area of the regions visible from the vertex). Furthermore, if the area covered by two or more vertices is equal, they are prioritized according to the minimum distance from the root node.
- A blocker position is finalized for the vertex with the highest priority.

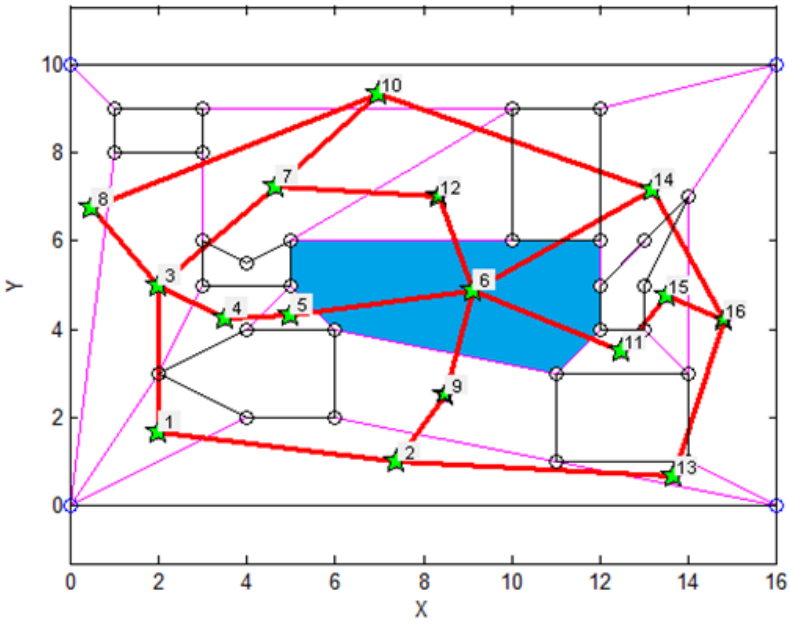

Figure 3.14: Example Case - Regions Graph and Regions Numbering

Figure 3.14 shows that Region 6 (highlighted in blue) effects the most number of cycles (five). Region 6 is formed by 8 vertices, and they all are analyzed for blocker position. After prioritizing the vertices w.r.t. number of cycles covered, we are left with only one vertex that effects 6 cycles. That vertex is finalized as a blocker position (shown below).

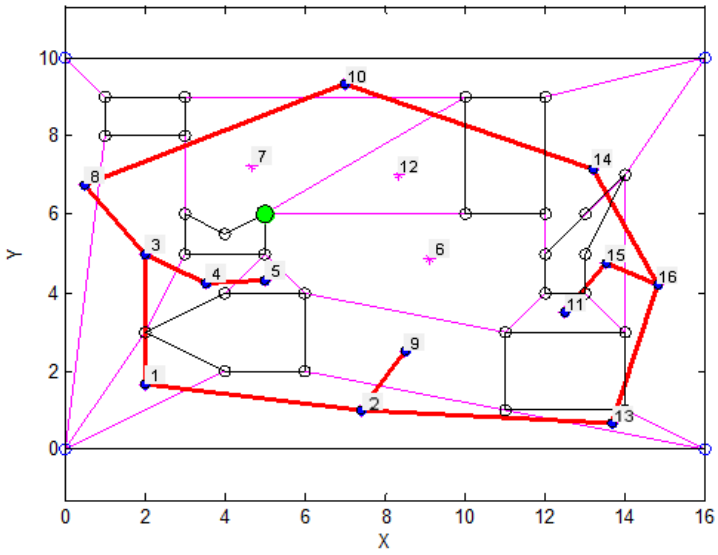

Figure 3.15: Example Case - Vertex effecting most cycles - Blocker Position (green)

All the regions covered by the blocker are removed from the graph and then it is iteratively re-analyzed for the number of cycles present. If the number of cycles present are more than zero, the cycle order and covered area are re-determined according to previous step and the structure S_{cycles} is updated. S_{cycles} is then again used in this step to determine further blocker positions until all cycles are broken. As in Figure 3.15, though all the inner cycles determined in the previous section are broken by one blocker, an outer cycle still needs to be broken (order: 1-2-13-16-14-10-8-3-1). After re-analyzing the graph, this cycle is broken and the final blocker positions are shown in the following figure.

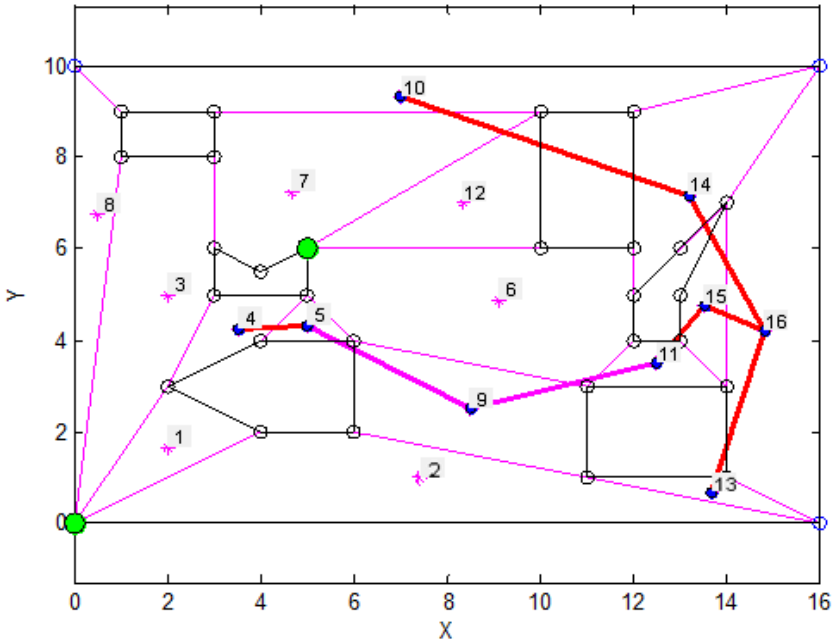

Figure 3.16: Example Case - Final Blocker Positions (green)

3.2.9 Reduced Minimum Spanning Tree

When Blocker Positions are finalized, all the regions covered by the blockers are removed from the Regions Graph. This results in a tree which is the Reduced Minimum Spanning Tree (RMST) as no cycles are present anymore. This can be represented as:

$$S_{rmst} = S_{region}$$

$$S_{rmst}\,(S_{rmst} == S_{reg\_blockers}) = []$$

In the parent study, the regions covered by blockers are removed from the Regions Graph unless they result in disconnected nodes. This results in some regions covered by blockers still being analyzed by searcher robots as they are a part of their branch. Consequently, the time to search and secure the complete environment increases unnecessarily. In the algorithm presented in this manuscript, all regions covered by blockers are removed from the Regions Graph regardless of disconnected nodes. The disconnected nodes are then checked for minimum distance to the RMST and connected to the tree node that is closest to them. As a result the searcher can move directly to the node needed to be searched rather than first searching a node in between that is already covered by blockers. RMST for the example case is shown in the following figure.

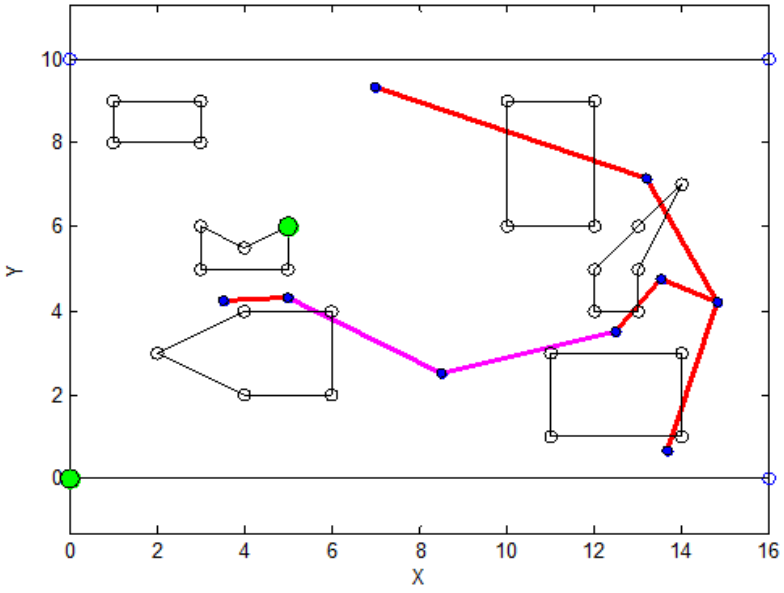

Figure 3.17: Example Case - RMST and Blocker Positions

The edges highlighted in magenta colour are the ones that have been added especially to connect the disconnected nodes directly to the RMST. In case of unpathable regions, **point regions** are added at the corners of unpathables in order to completely survey them. Point regions are regions that are points (area $\rightarrow 0$) and are used to update the RMST such that unpathable regions are surveyed completely.

3.2.10 RMST Root Optimization

In the parent study, an arbitrary root is selected, which can result in increased number of branches and greater number of searcher robots. In the algorithm presented in this manuscript, root selection is optimized keeping in consideration that:

- The root is not a leaf node, as it adds to the degree of all branches. Instead, the parent of this leaf can be used as the root node so that the branch length for all branches can be minimized. For example, in Figure 3.16, if node 13 is selected as root, it is preferable to select node 16 as root rather than node 13 and declare node 13 as a leaf node.
- The root is an end part of a branch, so that the branches can be minimized. This can be understood from Figure 3.16. If after the previous step node 16 is selected as root node, it results in the formation of two branches (16-15-11-9-5-4) and (16-13-14-10). Instead it is preferable to select node 5 or node 14 as root, resulting in a single branch only (5-4-9-11-15-16-13-14-10). This reduces number of branches and consequently the number of searchers required.

For the example, node 5 is finalized as root by the algorithm as in the following figure (cyan star).

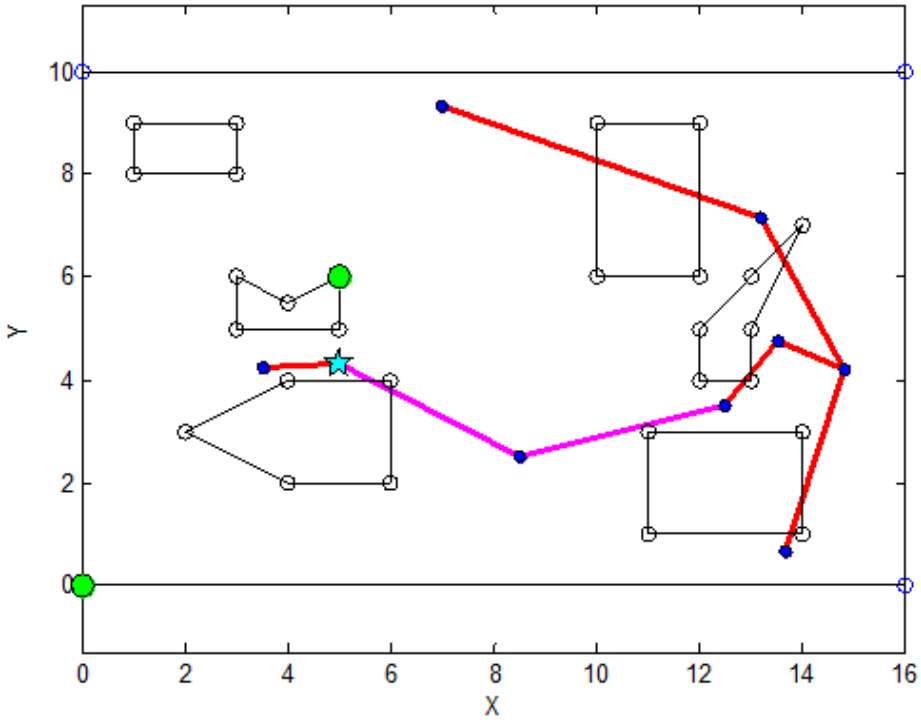

Figure 3.18: Example Case - Root Finalization (cyan star)

3.2.11 Number of Branches and Search Order

The RMST and root node are shown for the example case in Figure 3.18. In this section, the number of branches and their search order is determined.

- **Number of Branches:** A series of at least three connected points in a tree or graph is known as a branch. A branch starts from the root of a graph or tree and ends in a leaf node [27]. The termination leaf node of the branch should be such that all the children of its parent node are leaves, otherwise the branch will continue.
- **Leaf Nodes:** Leaf node of a tree is one that has only one neighbour (its parent node).

The number of searchers is equal to the number of branches in the RMST. In order to search a leaf node, no extra searchers are needed as the leaf node is searched by placing the searcher at a point such that it has vision of both the leaf node as well as its parent. The branches start from the root node and as there is no loop/cycle in the RMST, no intruder can contaminate an already cleared area without detection as the searchers move forward to clear the environment.

Furthermore, in order to accommodate the Unpathable regions, search points for each unpathable region are identified such that they get searched completely. These search points are declared point regions and are added to the search order. This is a major improvement compared to the parent study which doesn't cater for any unpathable regions.

In the example case, No. of branches = 01

After determining the number of branches, the search order S_{search_order} of each branch is determined according to the regions in the branch starting from the root node and ending at a leaf node. In the example case, the search order for the branch is:

Search Order: (5-4-9-11-15-16-13-14-10)

3.2.12 Blocker Paths

In this section, the Blocker Paths are finalized. Blocker paths are made according to the shortest path from root node to blocker positions and is based on the **Dijkstra's Shortest Path Algorithm**. The Dijkstra's Algorithm is a graph search algorithm that produces a shortest path tree from a root node to any destination node given a graph with a single root. Dijkstra's Algorithm itself only gives the length of the shortest path, but it is modified to return both the path length as well as the actual path coordinates. Hence, the paths from the root to blocker positions are determined. The pseudo code for the implemented algorithm is as follows.

1. dist = Array equal to the number of vertices (including root vertex)
2. Initialize dist to infinity (very large value)
3. dist(root) = 0;
4. previous_node = Array to save previous best node in path
5. for i = 1:n
    a. Q ← V (Q initialized to contain all vertices)
    b. while(1)
        i. index = Sort Q w.r.t. minimum distance in (dist)
        ii. u = Q(index(1)); //Minimum distance node
        iii. Remove u from Q
        iv. if (dist(u) == infinity)
            1. break;
        v. else if (u == $S_{blocker\_pos}$)
            1. $S^{i}_{blocker\_path}$ = Check and add nodes from previous_node to this array until last node added is the root node
            2. break;
        vi. end
        vii. break;
        viii. v = Check visibility of all adjacent nodes to u through $S_{visibility}\ (u)$
        ix. if (v is visible)
            1. d = Calculate distance from u to v
            2. if (dist(v) > dist(u) + d)
                a. dist(v) = dist(u) + d
            3. end
        x. end
    c. end
6. end
7. $S_{blocker\_path} = S^{1}_{blocker\_path} \cup \ldots \cup S^{n}_{blocker\_path}$
8. return

where

n: Number of blockers

$S^{i}_{blocker\_path}$: Path for i-th blocker

$S_{blocker\_path}$: Structure for all blocker paths

Blocker Paths for the example case are shown in the following figure. Red dot is the root node and the green dots show blocker positions. The green lines lead to the blocker positions from root node. The blockers move into position first, and then the searchers start searching the remaining regions.

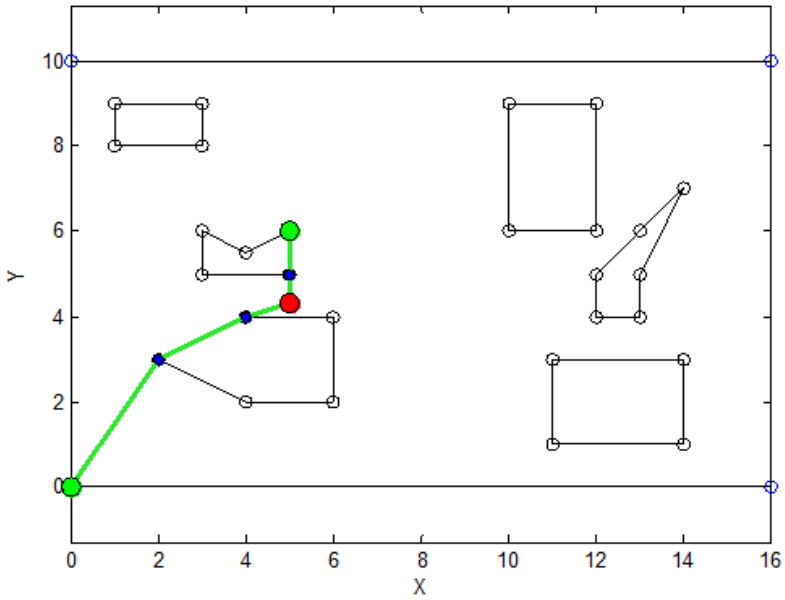

Figure 3.19: Example Case - Blocker Paths

3.2.13 Searcher Paths

Finally, searcher paths are calculated based on the RMST. Searcher paths are based on the shortest path algorithm, but the path is optimized for certain improvements.

- According to the $S_{searc\,h\_order}$ matrix, the searcher paths $S_{searc\,h\_pat\,h}$ are formed. Every searcher paths starts from the root node.
- As the next node from $S_{searc\,h\_order}$ is to be added to the $S_{searc\,h\_pat\,h}$ array, the next node is analyzed first. If it has any leaf node as its child, the next point in the path is determined such that it covers both the next node as well as its child leaf.
- For multiple leaves as children of the next node, the $S_{searc\,h\_pat\,h}$ is optimized such that maximum number of leaves are searched with minimum addition of search points to the path. All child leaves of the next node are analyzed before moving further.
- When the next node is analyzed completely, it is declared the current node and the node next to it is analyzed. This is done until the path termination leaf is reached. All leaves are analyzed such that the searcher does not have to enter completely into a leaf, rather it stays on the edge/corner of a leaf and searches it.
- This caters for unpathable regions as well as their points (point regions) have already been added to $S_{searc\,h\_order}$.

Following figure shows the searcher paths for the example case. As there was only one branch, so one searcher is required. The RMST with region numbers can be seen in Figure 3.16. The search order for the case is given below again for easier reference:

S_{search_order} : (5-4-9-11-15-16-13-14-10)

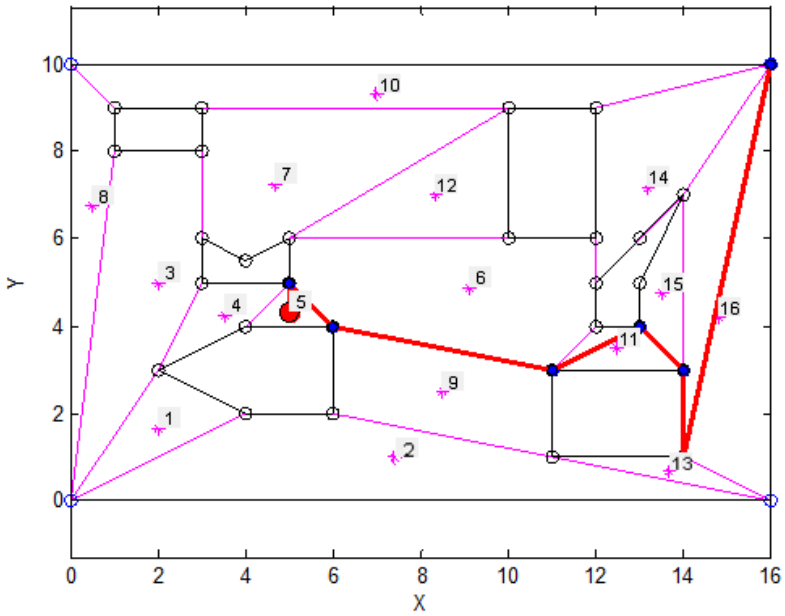

Figure 3.20: Example Case - Searcher Path (with region numbers)

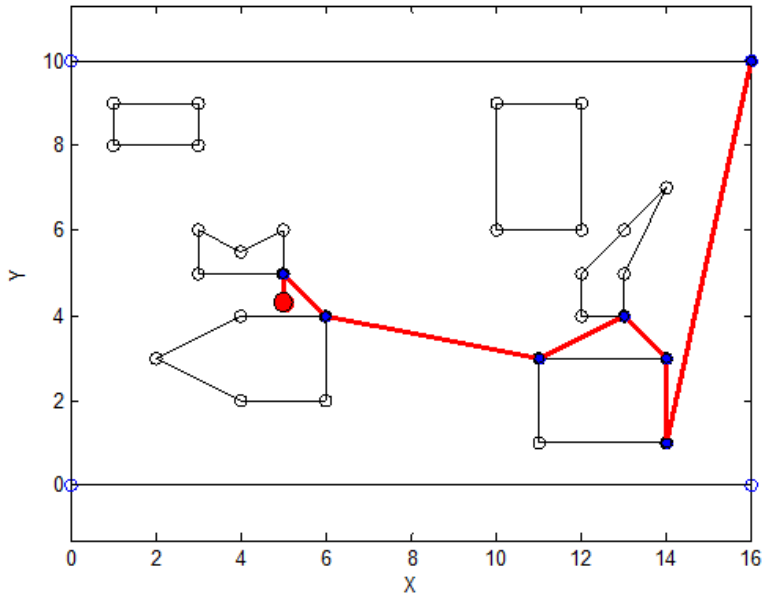

Figure 3.21: Example Case - Searcher Path

It can be seen in Figure 3.20 that in order to search node 14 after node 13, the algorithm analyzed the node such that instead of first going to the vertex of node 14 closest to node 13 and then onto node 10, it chose the top right corner point of the map which covers both node 14 as well as the branch termination leaf node 10. This significantly saves time and path cost in the whole search process.

CHAPTER 4: CASE STUDY FOR VALIDATION OF ALGORITHM

4.1 Case Study

In this chapter, a case study is presented to validate the results of the algorithm. It compares the results of the parent study with the algorithm discussed in this manuscript. The results are compared stepwise according to the steps discussed in detail in chapter 03.

4.1.1 Environment Map

The Environment Map used for the case study is shown below. It has a rectangular boundary with 9 obstacles.

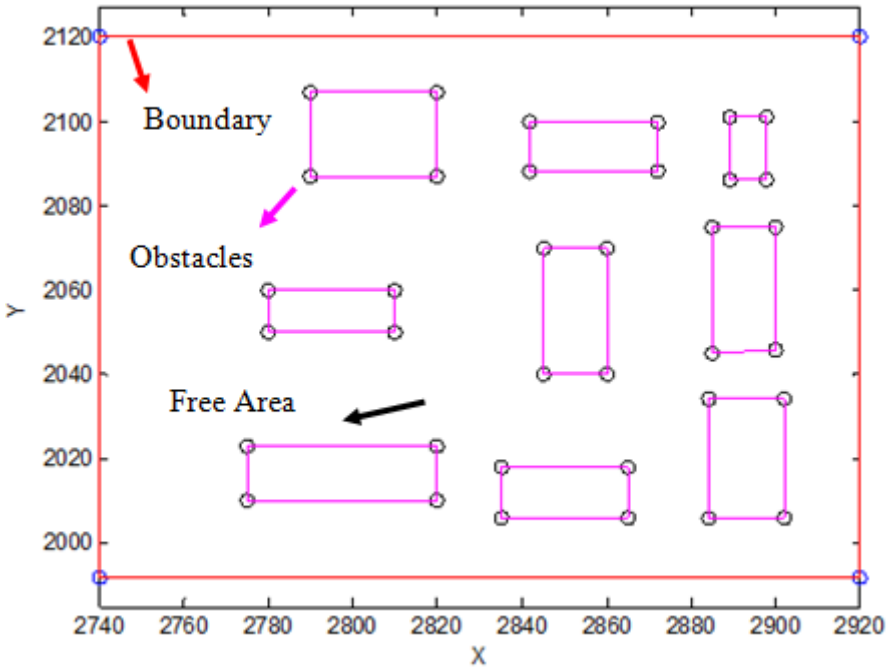

Figure 4.1: Case Study - Environment Map

4.1.2 Triangulation

The Triangulation results are presented in the following figures.

- Parent Study: The Delaunay Triangulation alongwith Triangles Graph in the parent study is shown in Figure 4.2 [27].

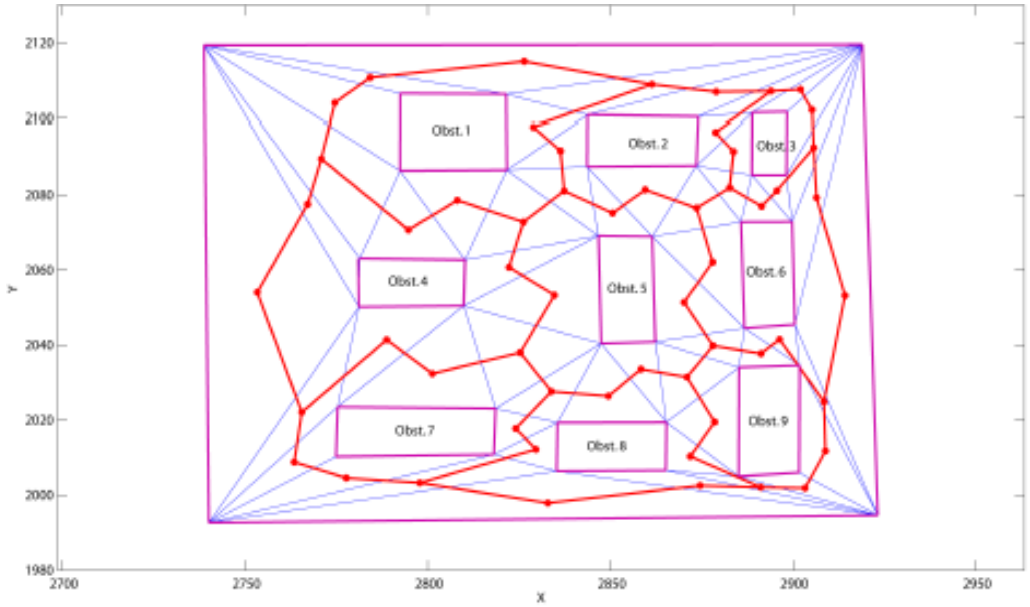

Figure 4.2: Case Study - Delaunay Triangulation and Triangles Graph (Parent Study)

- Presented Algorithm: The initial Non-Delaunay Triangulation results of the algorithm discussed in this manuscript are trivial and after application of the Lawson Flip Algorithm, the Delaunay Triangulation obtained is shown in the following figure. As it can be seen, a total of 56 triangles have been formed.

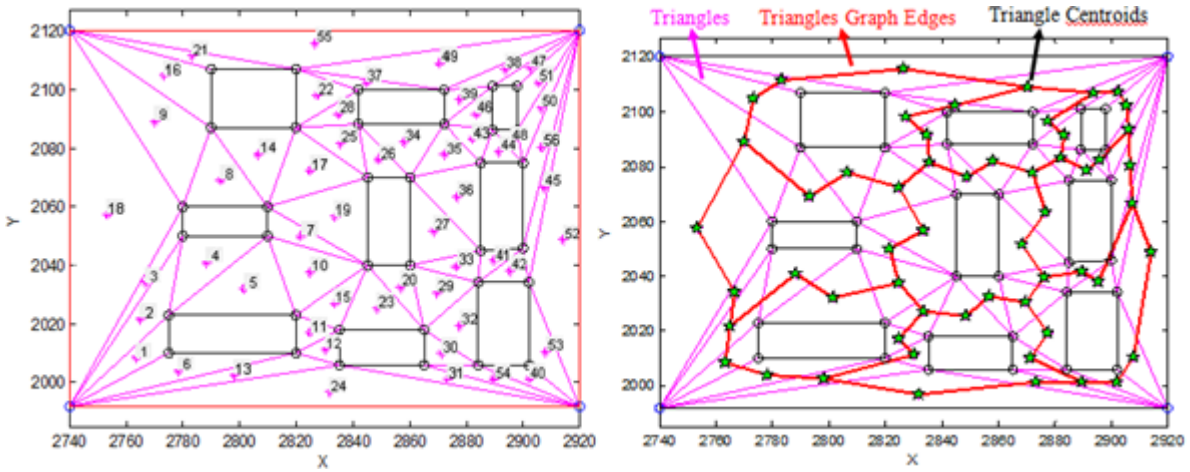

Figure 4.3: Case Study - Delaunay Triangulation and Triangles Graph

It can be analyzed from Figure 4.2 and Figure 4.3 that the Delaunay Triangulation results are very similar in the parent study and the presented algorithm.

4.1.3 Merging Triangles into Convex Regions

The results of Merging the Triangles into Convex Regions are shown below.

- Parent Study: The Convex Regions alongwith the Regions Graph formed in the parent study for this particular case are shown in the following figure [27].

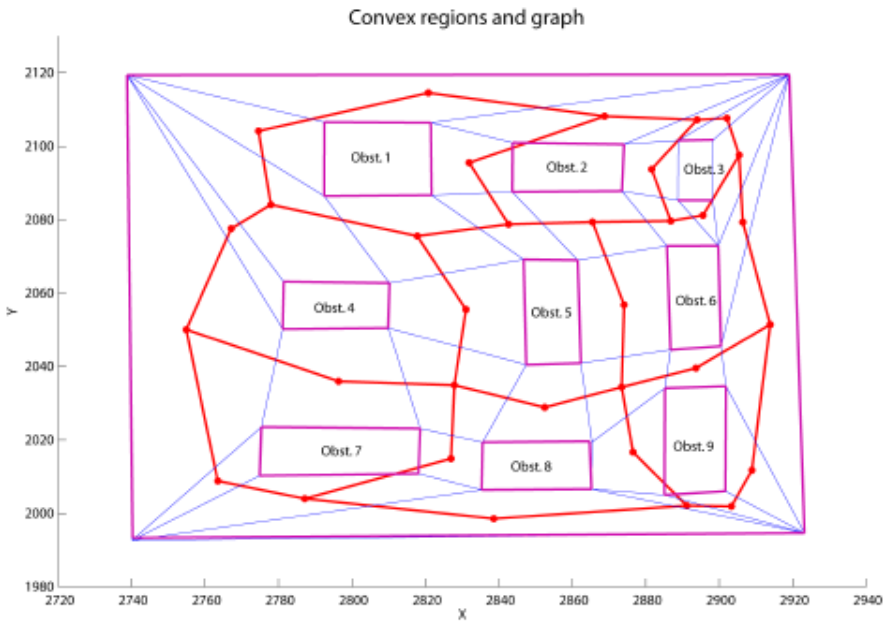

Figure 4.4: Case Study - Convex regions and Regions Graph (Parent Study)

A total of 33 regions are formed in the parent study as can be seen in the figure above.

- Presented Algorithm: The Merging results of the algorithm discussed in this manuscript are shown in Figure 4.5.

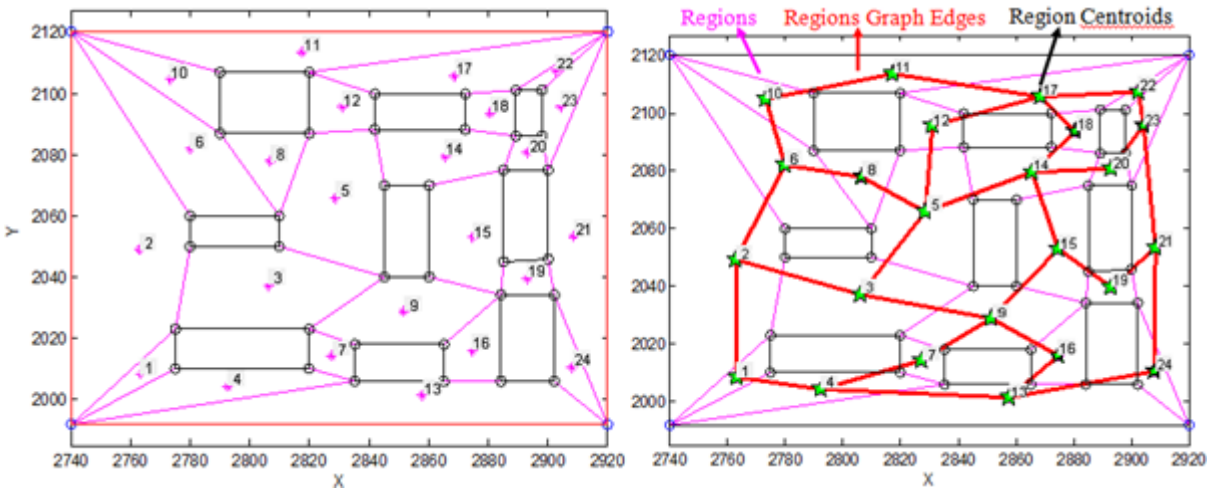

Figure 4.5: Case Study - Merging triangles into Convex Regions and Regions Graph

A significant improvement in the merging of triangles can clearly be seen by comparing Figure 4.4 and Figure 4.5. 52 Triangles have been merged into a total of **24 regions** as shown in Figure 4.5 compared to the 33 regions formed in the parent study. This is a significant improvement as it reduces the number of regions to be searched, in turn reducing the time as well as the number of robots required to search the complete environment.

4.1.4 Cycle Detection and Blocker Positions

The results of cycle detection, total blockers required and their positions are given in this section. There are a total of 9 cycles formed in this particular case study corresponding to the 9 obstacles in the environment (1 cycle around each obstacle). But the results differ in case of number of blockers and blocker positions which is depicted in the following figures.

- Parent Study: The blocker positions for this particular case study are shown in the following figure [27]. A total of 04 blockers are used to break the 9 cycles in the parent study, shown in green dots in Figure 4.6.

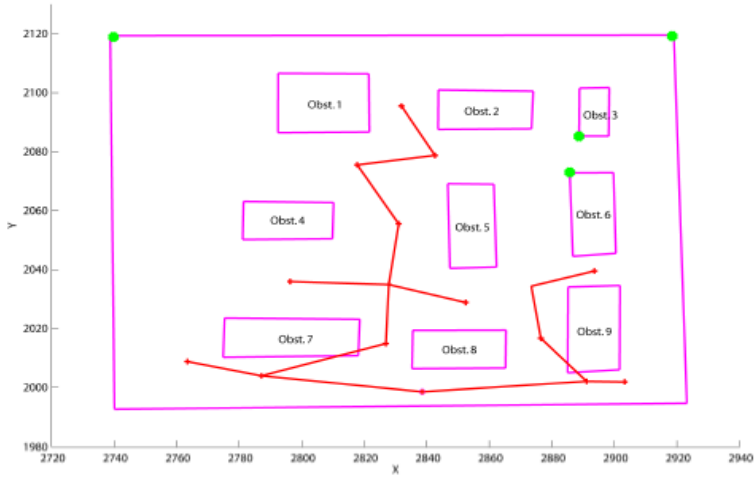

Figure 4.6: Case Study - Blocker Positions and RMST (Parent Study)

- Presented Algorithm: The blocker positions evaluated by the algorithm presented in this manuscript are shown in the following figure.

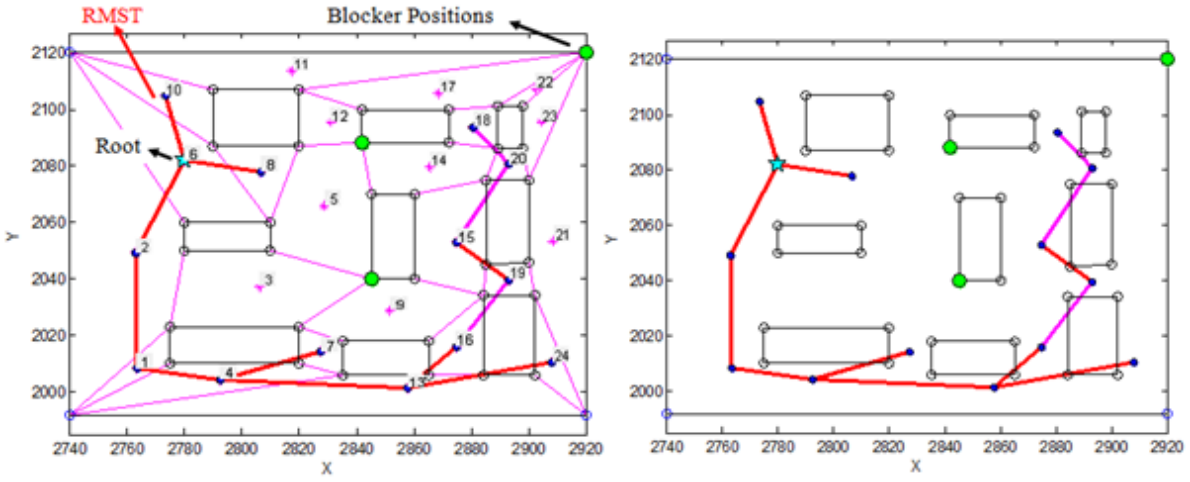

Figure 4.7: Case Study - Blocker Positions and RMST (with and without regions numbering)

In this case, a total of only **03 blockers** are used (shown as green dots) to break the 9 cycles present in the regions graph as compared to the 04 blockers identified by the parent study. Fewer number of robots required significantly improves performance and uses lesser resources. It reduces the time the blocking agents take to get to their positions as well.

4.1.5 Reduced Minimum Spanning Tree

The RMST computed for this case study are depicted in this section.

- Parent Study: The RMST formed for the case study after using 04 blockers is shown in Figure 4.6. A total of 16 regions (red dots) are to be analyzed in the RMST (red lines).
- Presented Algorithm: The RMST for the algorithm presented in this manuscript is shown in Figure 4.7.

A total of **14 regions** (blue dots and cyan star) are to be analyzed in the RMST (red and magenta lines) while using only 03 blockers; compared to the 16 regions to be analyzed according to the parent study after using greater number of blocker agents (04). Magenta lines in Figure 4.7 show the disconnected nodes that have been connected directly to the closest tree nodes (due to blocker coverage). Lesser number of regions to search reduces the number of searchers required as well as the total time and path cost.

4.1.6 Root Optimization

The analysis of root optimization for RMST is depicted in this section.

- Parent Study: As mentioned earlier, the parent study selects an arbitrary root node. In this particular case, the root node selected in the parent study is the node below Obstacle 08. It is depicted in the following figure (cyan star) [27].

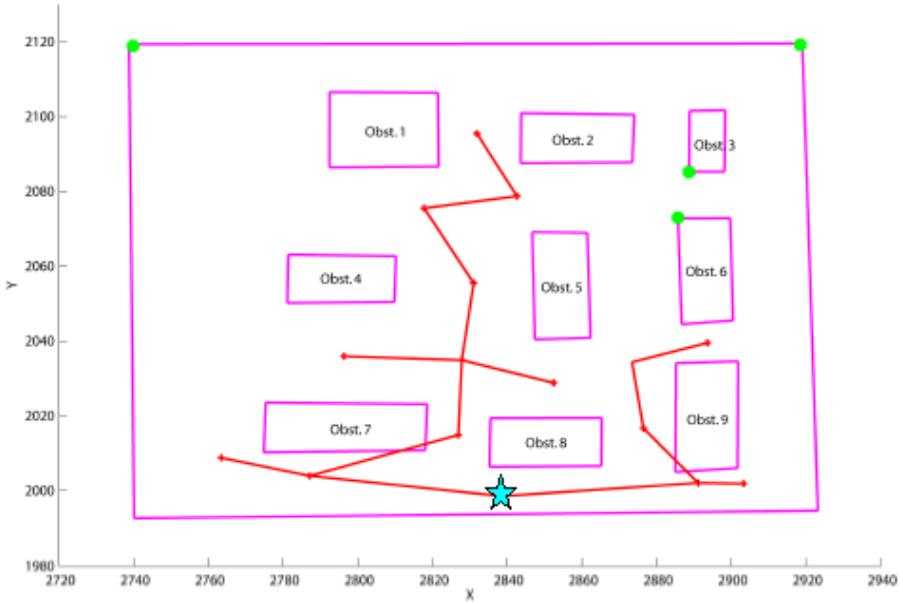

Figure 4.8: Case Study - Root Node Selection (Parent Study)

This is a suboptimal selection of the root node and can lead to an increase in the number of searchers required.

- Presented Algorithm: The algorithm presented in this manuscript performs root optimization before deducing number of branches and search order as discussed in chapter 03. The results of root optimization for this particular case are shown in the following figure with the root highlighted as a cyan star.

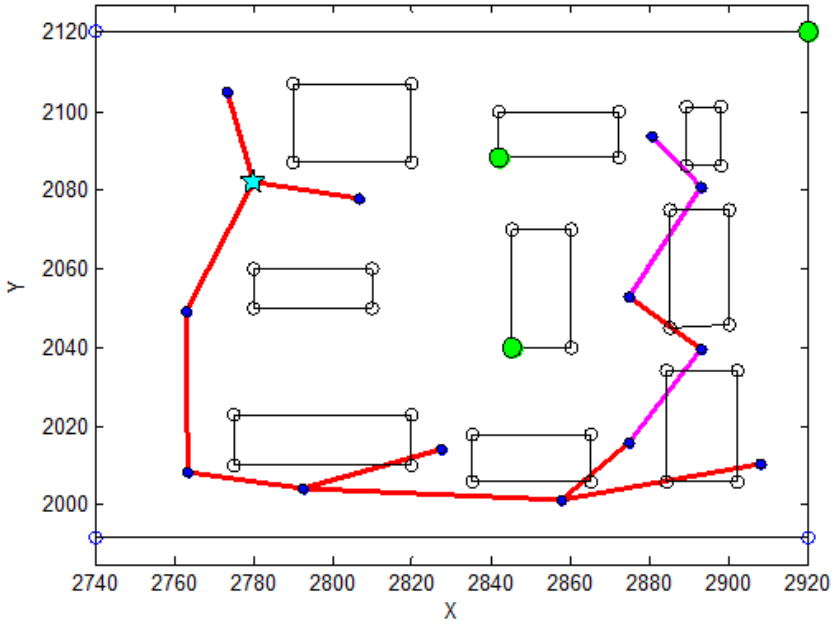

Figure 4.9: Case Study - Root Optimization and Selection

In this case, the root node selected is just to the left of Obstacle 01 and its region number is 06 (as can be seen in Figure 4.7).

4.1.7 Number of Branches and Search Order

This section compares the number of branches generated by both algorithms. The number of branches significantly depend on proper selection of the root node.

- Parent Study: The number of branches identified in the parent study are 02. This is clearly visible from Figure 4.8 according to the branch rules discussed in chapter 03. Therefore, this implies the requirement of 02 searchers.
- Presented Algorithm: The number of branches identified by the algorithm presented are only **01**. This can be seen from Figure 4.9 according to the branch rules discussed in chapter 03. Hence the presented algorithm requires only **01 searcher** compared to the 02 searchers required in the parent study.

4.1.8 Blocker Paths

This section shows the Blocker Paths used for this particular case study. Blocker Paths originate from the Root and end at Blocker Positions.

- Parent Study: The parent study uses suboptimal paths for blockers. They are not based on the shortest path algorithm and hence result in longer paths and increased time to reach the blocker positions. This increases the runtime of the algorithm significantly. The blocker paths for the case are shown in the following figure [27].

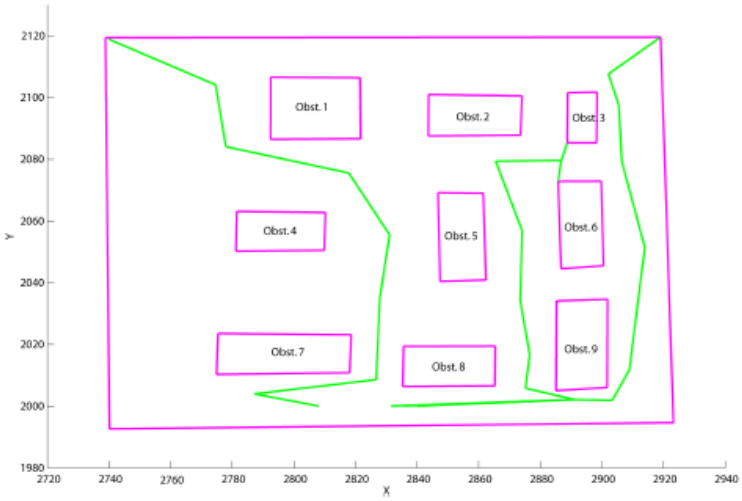

Figure 4.10: Case Study - Blocker Paths (Parent Study)

The paths for the 04 blockers according to the parent study are shown above. The maximum length for a blocker path in this case is approx. 235 meters and that is the critical path length as the searchers cannot start searching before all blockers reach their positions. The time taken by the robot (Groundbot by Rotundus) to reach its final position is 6 minutes and 8 seconds [27] at an average speed of approx 0.64 m/sec.

- Presented Algorithm: The algorithm presented uses the shortest path algorithm as discussed in chapter 03 to minimize the path length as well as time to reach the blocker positions. The resulting blocker paths are shown in the following figure.

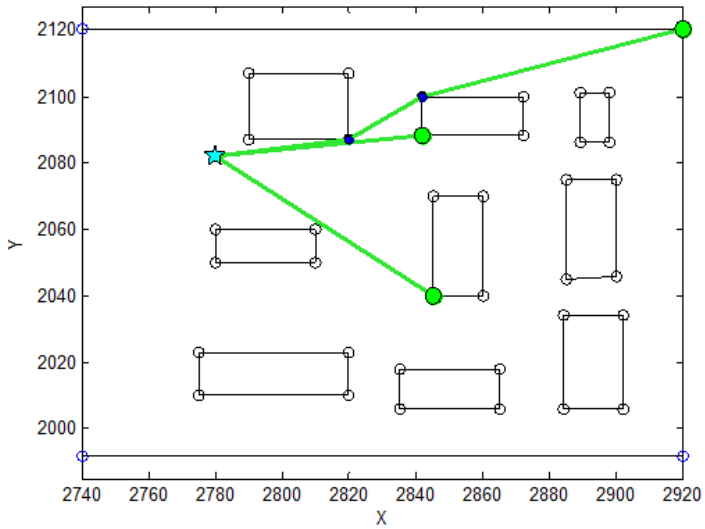

Figure 4.11: Case Study - Blocker Paths (Shortest Paths)

The blocker paths (green lines) originate from the root (cyan star) and end at the 03 blocker positions specified. As it can be seen from the figure, the paths are the shortest possible paths to reach the destination positions. The maximum path length for a blocker path in this case is approx **146 meters** compared to the 235 meter maximum path length generated by the parent study. If the same robot is used with the same average speed (0.64 m/sec) as reference, all blockers are in position by approx **3 minutes and 48 seconds**. This is a much improved performance compared to the parent study, reducing the critical path time by two and a half minutes.

4.1.9 Searcher Paths

Finally, a comparison of Searcher Paths for the case is presented in this section.

- Parent Study: The total number of searchers required in the parent study is 02 and their paths are shown in the following figure [27].

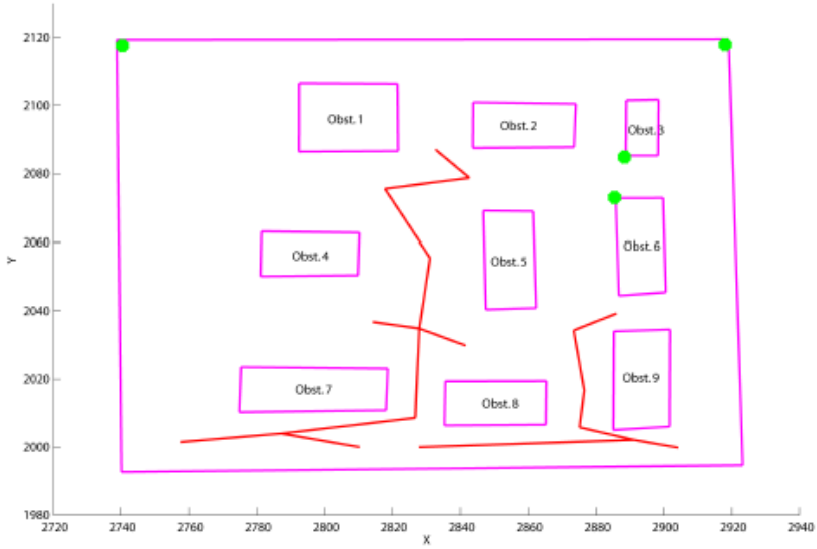

Figure 4.12: Case Study - Searcher Paths (Parent Study)

This figure shows the two searcher paths (red lines) alongwith the blocker positions (green dots). As is clear from the figure, the searcher paths are formed by using the midpoints of edges of adjacent regions to be explored. This results in a sub-optimal path that is not the shortest path. The longest searcher path is approximately 346 meters in this case taking approx 9 minutes to complete the search (average speed same as before). The total search time for the parent study comes out to be **15 minutes and 8 seconds**.

- Presented Algorithm: The total number of searchers required in the algorithm presented is **01** compared to the 02 searchers required in the parent study. The searcher path is shown in the following figures.

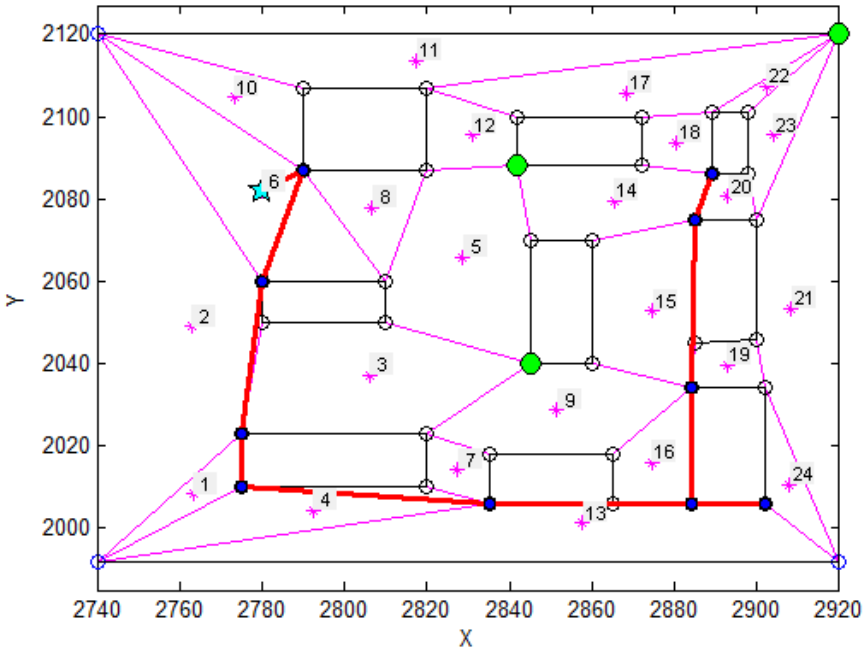

Figure 4.13: Case Study - Searcher Path (with regions numbering)

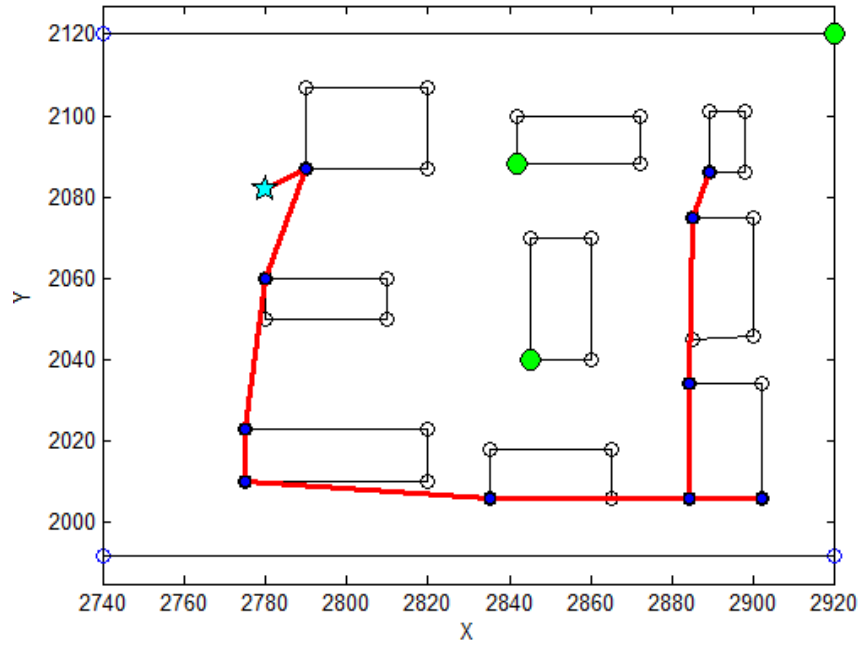

Figure 4.14: Case Study - Searcher Path

The figures above show the Searcher Path (red lines) alongwith the Blocker Positions (green dots). This path have been formed using the shortest path algorithm alongwith the important rules and considerations mentioned in chapter 03. This results in the shortest path for all possible Searchers. The Searcher path in this case is approximately **316 meters** taking approx **8 minutes and 13 seconds** to complete the search (average speed same as before) compared to the 9 minutes taken by the parent study. The total search time for the presented algorithm comes out to be **11 minutes and 51 seconds** as compared to the 15 minutes and 8 seconds for the parent study.

4.1.10 Conclusion of the Case Study

Table 4.1: Comparison of algorithms depicting time and cost efficiency

	Parent Study	**Presented Algorithm**
No. of Regions	33	24
No. of Blockers	04	03
No. of Searchers	02	01
Critical Path Length (CPL) Blockers	235 meters	146 meters
Time to cover CPL Blockers	06 min 08 sec	03 min 38 sec
Critical Path Length (CPL) Searchers	346 meters	316 meters
Time to cover CPL Searchers	09 min	08 min 13 sec
Total Time to Secure Environment	15 min 08 sec	11 min 51 sec
Total Robots required	06	04

The results of both algorithms can be summarized (Table 4.1) for this particular case study such that the algorithm presented in this manuscript is more time and cost efficient as it employs a better merging technique and uses the shortest path algorithm for blockers as well as searchers.

CHAPTER 5: RESULTS

In this chapter, the results of a targeted environment are presented. The results are compiled stepwise according to the steps discussed in detail in chapter 03.

5.1 Environment Map

The environment on which the algorithm was finally implemented is shown in the figure below. It is a scaled-down map of College of EME Block-01 to Block-07. (Units: ft/10)

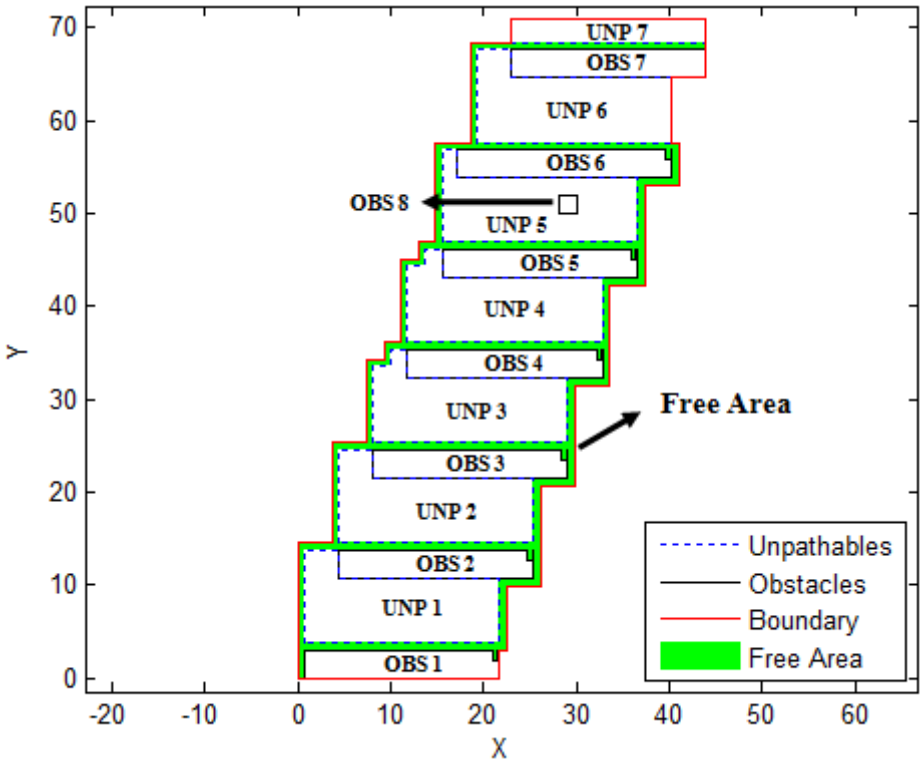

Figure 5.1: Results - Environment Map

As it can be seen from Figure 5.1, blocks 1 to 7 have been modeled as obstacles 1 to 7 and the lawns in front of the blocks are labeled as unpathable regions where the robots are not allowed to enter. Intruders can enter the lawns so they are to be surveyed by robots by staying at the edges/corners of the unpathable regions. Obstacle 8 is inside unpathable 5, so the cycle around that obstacle has to be catered for by the algorithm. The pathways are the only area where the robots can move (free area shown in green).

5.2 Triangulation and Triangles Graph

The triangulation that forms the Triangles Graph is presented in the following figure.

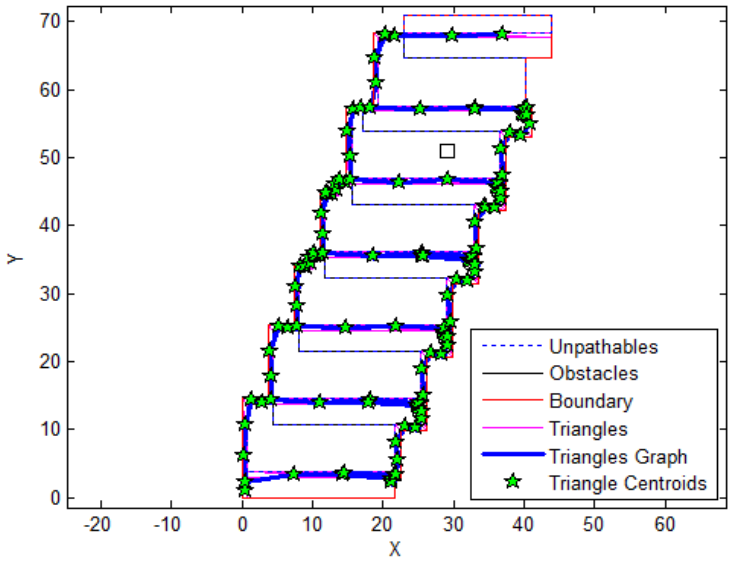

Figure 5.2: Results - Triangles Graph

A total of 131 triangles were formed in the free area of the environment. In the same way, all unpathable region were triangulated by treating each as a separate **sub-environment**.

5.3 Merging Triangles into Convex Regions and Regions Graph

The results of merging the triangles into convex regions are shown below.

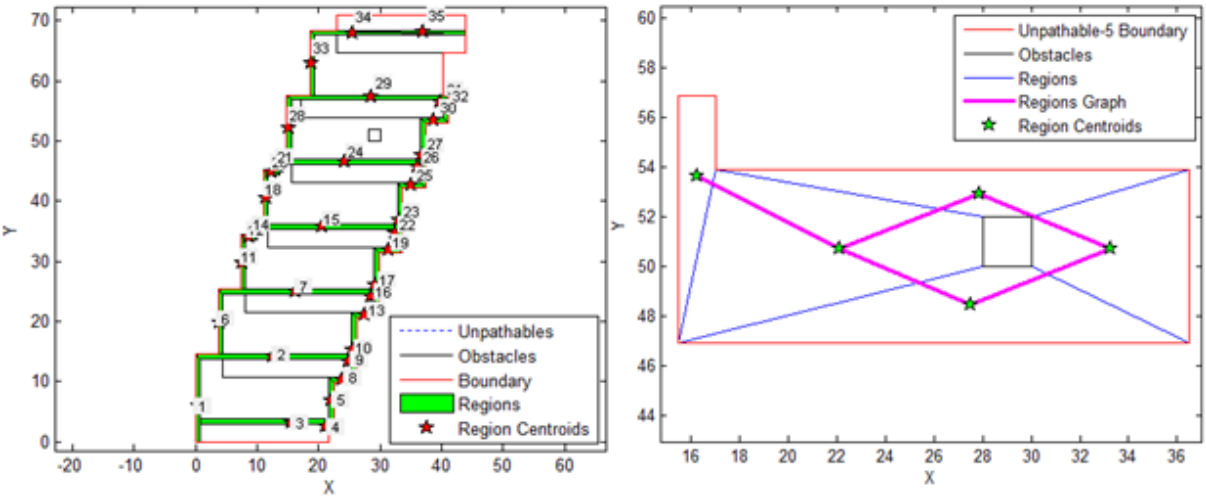

Figure 5.3: Results - (a) Merged Regions (b) Unpathable-05 Regions

As it can be seen in Figure 5.3 (a), 131 triangles in the free pathway area have been merged into a total of **35 regions**. Similarly, the triangles within all unpathable regions are also merged to form 18 more convex regions. Figure 5.3 (b) shows the 5 regions and regions graph formed within unpathable-05 that has obstacle-08 inside it. A cycle is formed around the obstacle within the unpathable. This cycle must be catered for (broken) in order to

completely survey this unpathable. Unpathable regions 1,2,3,4,6 and 7 have 2,2,3,3,2 and 1 regions respectively. Therefore, there are a total of **53 regions** in the complete environment. Figure 5.4 shows the corresponding Regions Graph.

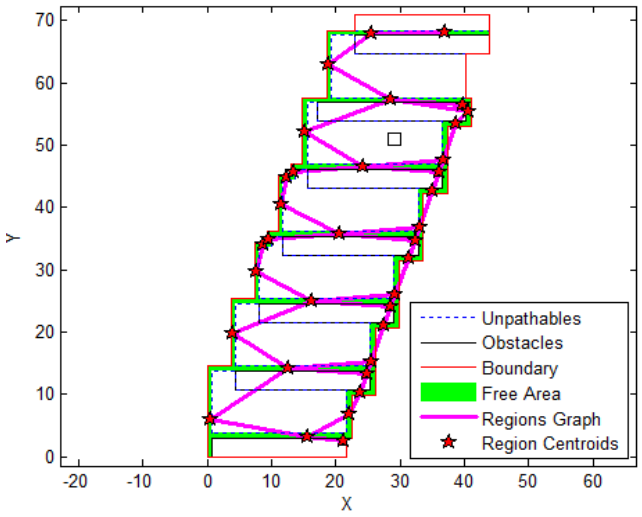

Figure 5.4: Results - Regions Graph (Voronoi Diagram)

5.4 Cycle Detection and Blocker Positions

The results of cycle detection, total blockers required and their positions are given in this section. A total of 6 cycles were formed (1 cycle each around unpathables 1 to 5 and one cycle around obstacle-08 within unpathable-05). The blocker positions evaluated by the algorithm presented in this manuscript are shown in the following figure.

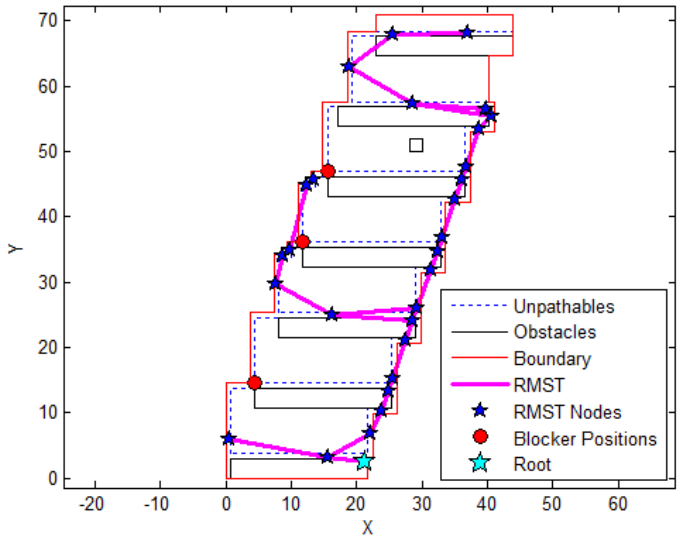

Figure 5.5: Results - Blocker Positions (and remaining Regions Graph)

A total of only **03 blockers** were required (shown as red dots) to break the 6 cycles present in the Regions Graph and are placed at corners of unpathable-02, 04 and 05 respectively.

5.5 Reduced Minimum Spanning Tree and Root Optimization

The RMST computed for the environment is depicted in the following figure. On comparison of Figure 5.5 and Figure 5.6, it can be seen that 6 region centroids have been added when finalizing the RMST. These are the **point regions** (numbered 54 to 59) added on the corners of unpathable regions required to completely survey the area within them. A point region is added at a corner of each unpathable except unpathable-02 that is completely covered by a static blocker. The root of the RMST is placed at an optimized location in order to minimize the number of branches in the tree.

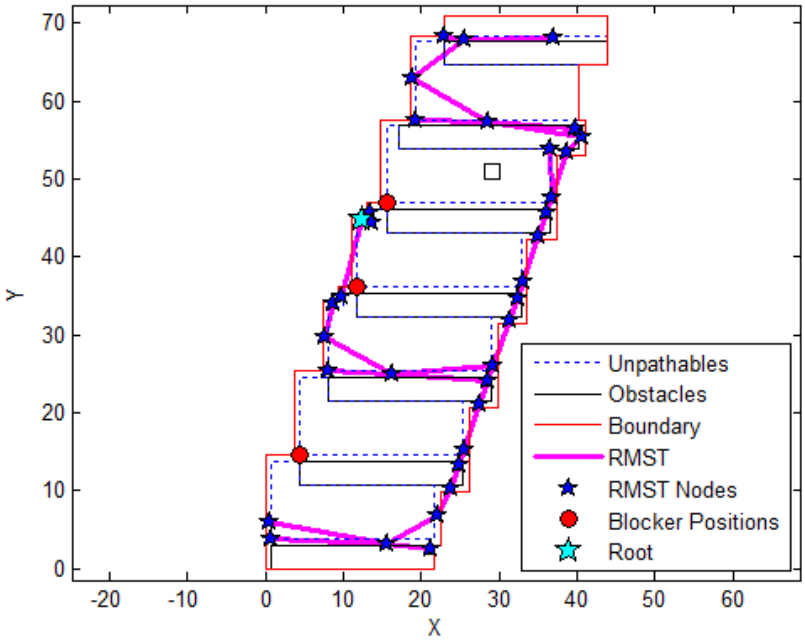

Figure 5.6: Results - RMST and Root Node Selection

5.6 Number of Branches and Search Order

As discussed earlier, the number of branches significantly depend on proper selection of the root node. The number of branches identified by the algorithm presented are **02**. This can be seen from Figure 5.6 according to the branch rules discussed in chapter 03. Hence it requires only **02 searchers** to survey the environment, and the search orders are as follows:

Searcher1: (20-56-21-14-12-11-7-55-16-17-13-10-9-8-5-3-54-4-1)

Searcher2: (20-14-12-11-7-17-19-23-22-25-27-57-26-30-32-29-58-31-33-34-59-35)

5.7 Blocker Paths

This section shows the blocker paths computed by the algorithm. Blocker paths for the 03 blockers originate from the root node and end at respective blocker positions as shown in the following figure. The maximum length for a blocker path in this case is approx. **105.5**

meters (CPL). Using the same average robot speed as earlier in chapter 04, all blockers reach their positions by **02 minutes and 44 seconds**.

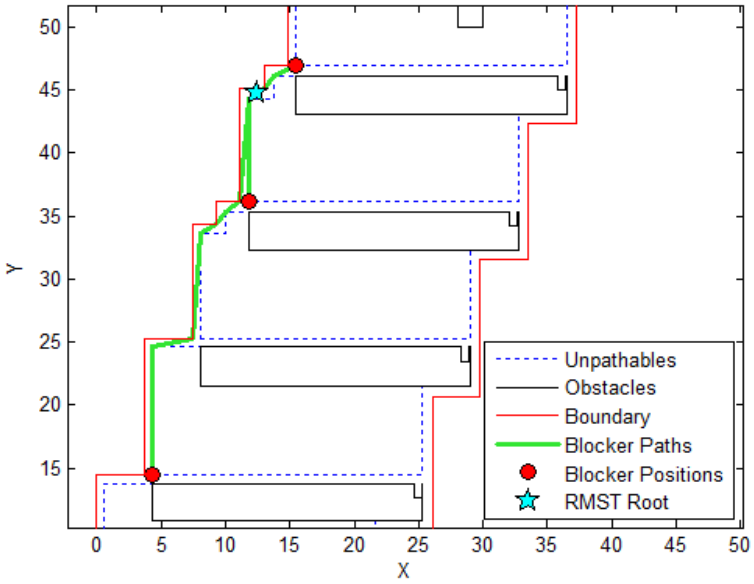

Figure 5.7: Results - Blocker Paths (Shortest Paths)

5.8 Searcher Paths

The total number of searchers required in the algorithm presented are **02** and their paths are shown in the following figures.

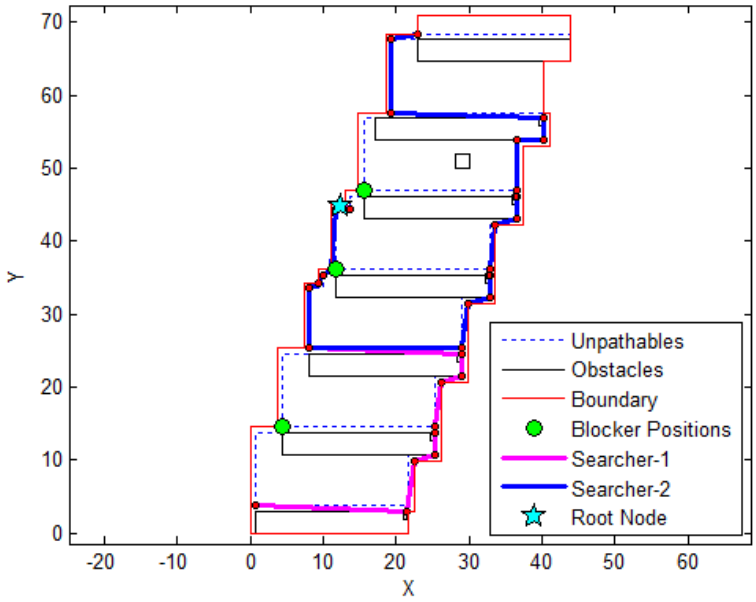

Figure 5.8: Results - Searcher Paths

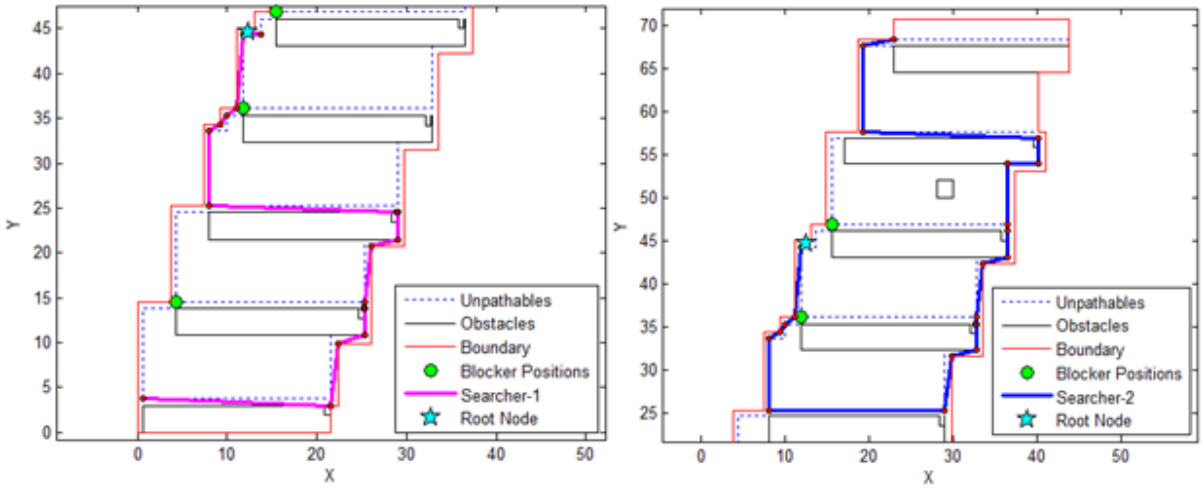

Figure 5.9: Results - (a) Seacher-1 Path (b) Searcher-2 Path

The figures above show searcher paths alongwith blocker positions. These path have been formed using the shortest path algorithm alongwith the important rules and considerations mentioned in chapter 03 according to the search orders identified. The searcher paths in this case are approximately **278 meters and 357.8 meters (CPL)** long respectively, and the time to cover the CPL is approx **9 minutes and 20 seconds**. The total search time for the environment comes out to be **12 minutes and 04 seconds**.

CHAPTER 6: CONCLUSION

The results of the Search and Secure algorithm presented in this manuscript can be summarized such that it is more effective and optimum because of the following:

- Better merging technique results in lesser number of regions in the regions graph.
- Improved blocker position evaluation results in lesser number of blockers required for breaking cycles.
- Reduced number of regions to be searched in the RMST.
- Root optimization for reducing the number of branches in the RMST and hence the number of searchers required.
- Shortest path algorithm for blockers resulting in minimum time.
- Shortest path algorithm with optimized leaf node rules for searchers.
- Handling of unpathable regions with internal obstacles.

All these points combine to form a more effective algorithm that significantly reduces the time and resource requirement for uncoordinated search of a particular environment. It can be improved even further by implementing a strategy to handle multiple obstacles within any unpathable region. Table 6.1 shows the summary of results of the environment analyzed in the previous chapter.

Table 6.1: Summary of results of the environment

No. of Blockers	03
No. of Searchers	02
Critical Path Length (CPL) Blockers	105.5 meters (Blocker-1)
Time to cover CPL Blockers	02 min 44 sec
Critical Path Length (CPL) Searchers	357.8 meters (Searcher-2)
Time to cover CPL Searchers	09 min 20 sec
Total Robots Required	05
Total Time to Secure Environment	12 min 04 sec

REFERENCES

[1] Trends in Robotics, International Federation for Robotics. Available: http://www.robotshop.com/blog/en/trends-in-robotics-9806

[2] The Korea Herald, Published: January 18, 2011. Available: http://www.koreaherald.com/view.php?ud=20110118000723

[3] Y. Elmaliach, N. Agmon, G. Kaminka, Multi-Robot Area Patrol under Frequency Constraints, *IEEE International Conference on Robotics and Automation* (ICRA 07), 385-390, Rome, Italy, April 10-14, 2007. ISBN 1-4244-0601-3.

[4] J. Bondy, U. Monty, *Graph Theory with Applications*, The Macmillan Press Ltd., 5th Edition, Elsevier Science Publishing Co. Inc, U.S.A., 1976. ISBN 0-444-19451-7.

[5] F. Rubin, A Search Procedure for Hamilton Paths and Circuits, *Journal of the Association for Computing Machinery*, 21(4), pp. 576-580, ACM, New York, U.S.A., October, 1974. ISSN 0004-5411.

[6] M. Batalin, G. Sukhatme, *Multi-Robot Dynamic Coverage of a Planar Bounded Environment*, Technical Report, Robotic Embedded Systems Laboratory, University of Southern California, 2002. Available: http://www.cens.ucla.edu/maxim/Publications/~papers/331.pdf

[7] M. Miazaki, *Sistema de controle multi-robo baseado em colonia de formigas artificiais*, Master Thesis, Instituto de Ciencias Matematicas e de Computacao - USP, Brazil, February, 2007.

[8] A. Sgorbissa, R. Arkin, Local Navigation Strategies for a Team of Robots, *Robotica*, 461-473, 21(5), Cambridge University Press, October, 2003. ISSN 0263-5747.

[9] R. Dollarhide, A. Agah, *Simulation and Control of Distributed Robot Search Teams*, Computers & Electrical Engineering, 625-642, 29(5), Elsevier Science Ltd., 1999.

[10] Y. Guo, L. Parker, R. Madhavan, *9 Collaborative Robots for Infrastructure Security Applications*, Studies in Computational Intelligence (SCI), Springer-Verlag Berlin Heidelberg, 185-200, Vol. 50, April 22, 2007. ISSN 1860-9503.

[11] M. Likhachev, A. Stentz, Information Value-Driven Approach to Path Clearance with Multiple Scout Robots, *IEEE International Conference on Robotics and Automation* (ICRA'08), 2651-2656, Pasadena, California, U.S.A., May 19-23, 2008. ISBN 978-1-4244-1646-2.

[12] K.Wurm, C. Stachniss, W. Burgard, Coordinated Multi-Robot Exploration using a Segmentation of the Environment, *IEEE/RSJ International Conference on Intelligent*

Robots and Systems (IROS 2008), 1160-1165, Nice, France, September 22-26, 2008. ISBN 978-1-4244-2057-5.

[13] N. Agmon, S. Kraus, G. Kaminka, Multi-Robot Perimeter Patrol in Adversarial Settings, *IEEE International Conference on Robotics and Automation* (ICRA 08), 2339-2345, Passadena, California, U.S.A., May 19-23, 2008. ISBN 978-1-4244-1646-2.

[14] A. Kolling, S. Carpin, The GRAPH-CLEAR Problem: Definition, Theoretical Properties and its Connections to Multirobot Aided Surveillance, *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS 2007), 1003-1008, San Diego, California, U.S.A., October 29 - November 2, 2007. ISBN 978-1-4244-0912-9.

[15] D. B. S. Portugal, *RoboCops: A study of coordination algorithms for autonomous mobile robots in patrolling missions*, M.S. Thesis, Dept. of Elec. and Comp. Engg., Fac. of Sci. and Tech., University of Coimbra, Coimbra, Portugal, 2009.

[16] A. Kolling, S. Carpin, Multi-robot Surveillance: an Improved Algorithm for the GRAPH-CLEAR Problem, *IEEE International Conference on Robotics and Automation* (ICRA 2008), 2360-2365, Pasadena, California, U.S.A., May 19-23, 2008. ISBN 978-1-4244-1646-2.

[17] T. Parsons, *Pursuit-evasion in a graph*, In Y. Alavi and D. R. Lick, editors, Theory and Application of Graphs, pp. 426–441, 1976.

[18] N. Megiddo, S. L. Hakimi, M. R. Garey, D. S. Johnson, and C. H. Papadimitriou, The complexity of searching a graph, *J. ACM*, 35–1, pp. 18–44, 1988.

[19] L. Barriere, P. Flocchini, P. Fraigniaud, and N. Santoro, *Capture of an intruder by mobile agents*, in SPAA ’02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures. New York, NY, USA: ACM Press, 2002, pp. 200–209.

[20] I. Suzuki and M. Yamashita, Searching for a mobile intruder in a polygonal region, *SIAM Journal on Computing*, 21(5), pp. 863–888, 1992.

[21] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani, A visibility-based pursuit-evasion problem, *International Journal of Computational Geometry and Application*s, 9(4/5), 1999.

[22] S. Sachs, S. Rajko, and S. M. LaValle, Visibility-based pursuit evasion in an unknown planar environment, *International Journal of Robotics Research*, 23(1), pp. 3–26, Jan. 2004.

[23] B. P. Gerkey, S. Thrun, and G. Gordon, Visibility-based pursuit-evasion with limited field of view, *International Journal of Robotics Research*, 2004, pp. 299–315.

[24] S. S. Ge and C. Fua, Complete multi-robot coverage of unknown environments with minimum repeated coverage, *Robotics and Automation (ICRA)*, April 2005, pp. 715-720.

[25] V. Isler, S. Kannan, and S. Khanna, Randomized pursuit-evasion in a polygonal environment, *Robotics*, 21(5), pp. 875–884, Oct. 2005.

[26] A. Ganguli, J. Cortes, and F. Bullo, Distributed deployment of asynchronous guards in art galleries, *American Control Conference*, June 2006.

[27] F. Katsilieris, *Search and secure using mobile robots*, M.S. Degree Project, Elec. Engg. KTH, Stockholm, Sweden, 2009.

[28] M. Barclay and A. Galton, Comparison of region approximation techniques based on Delaunay Triangulations and Voronoi Diagrams, *Journal of Computers, Environment and Urban Systems*, 32(4), pp. 261-267, July 2008.

[29] F. Preparata and M. Shamos, Computational Geometry: An Introduction, Chap. 3 "Convex Hulls: Basic Algorithms" (1985).

[30] B. Gartner, and M. Hoffmann, Delaunay Triangulations. In *Computational Geometry Lecture Notes HS 2013* (Ch.06).
Available: http://www.ti.inf.ethz.ch/ew/courses/CG13/lecture/Chapter%206.pdf

[31] A. Kamil, *Graph Algorithms*, CS61B, 14/04/03, UC Berkeley.