



*Social Graphs and Their Applications to Robotics*

ALWAFI, Fatma Ali Saad

Available from the Sheffield Hallam University Research Archive (SHURA) at:

<http://shura.shu.ac.uk/31906/>

## A Sheffield Hallam University thesis

This thesis is protected by copyright which belongs to the author.

The content must not be changed in any way or sold commercially in any format or medium without the formal permission of the author.

When referring to this work, full bibliographic details including the author, title, awarding institution and date of the thesis must be given.

Please visit <http://shura.shu.ac.uk/31906/> and <http://shura.shu.ac.uk/information.html> for further details about copyright and re-use permissions.

# **Social Graphs and Their Applications to Robotics**

**Fatma Ali Saad Alwafi**

A thesis submitted in partial fulfilment of the requirements of Sheffield Hallam University  
for the degree of Doctor of Philosophy

September 2022

## Declaration

I hereby declare that:

1. I have not been enrolled for another award of the University, or other academic or professional organisation, whilst undertaking my research degree.
2. None of the material contained in the thesis has been used in any other submission for an academic award.
3. I am aware of and understand the University's policy on plagiarism and certify that this thesis is my own work. The use of all published or other sources of material consulted have been properly and fully acknowledged.
4. The work undertaken towards the thesis has been conducted in accordance with the SHU Principles of Integrity in Research and the SHU Research Ethics Policy.
5. The word count of the thesis is 60,115.

Name	Fatma Ali Saad Alwafi
Date	September 2022
Award	PhD
Faculty	Industry and Innovation Research Institute/ Centre: MERI
Director(s) of Studies	Dr Lyuba Alboul

# **Social Graph and Their Applications to Robotics**

**Fatma Ali Saad Alwafi**

## **Abstract**

In this thesis, we propose a new method to design a roadmap-based path planning algorithm in a 2D static environment, which assumes a-priori knowledge of robots' positions, their goals' positions, and surrounding obstacles. The new algorithm, called Multi-Robot Path Planning Algorithm (MRPPA), combines Visibility graph VG method with the algebraic connectivity ( $\lambda_2$ ) of the graph Laplacian and Dijkstra's algorithm. The MRPPA implies sequential path planning for each robot based on the measured value of algebraic connectivity of the graph Laplacian, and the predefined weight functions to controlling the motion of robots while avoiding inter-robot collision, when planning the path of each robot, considers all the paths already planned for path correction and collisions' avoidance. The algorithm provides optimality of all planned paths because the paths depend on the order of planning, thus the choice of the right sequence for path planning of robots have significant impact on the performance of the team. VG has been selected because it produces solutions with optimal path lengths, i.e., short distances travelled from start positions to target, especially if combined with Dijkstra's algorithm. However, VG forces the robots to move near obstacles, and is computationally expensive, because it uses all vertices in the environment. Therefore, we have developed algorithms based on VG called the Central algorithm (CA) and its associate, the Optimisation Central algorithm (OCA). Contrary to VG, CA selects a relatively smaller number of vertices using the so-called Central Baseline (CB), in which the obstacles that intersect with the baseline only are considered, and it generates waypoints, travelling, through which the robots can avoid obstacles to reach their targets. Both algorithms employ a smaller number of obstacles, and this reduces the computational complexity of finding the optimal paths. Thus, it can create paths relatively fast and is convenient for path planning applications in obstacle-rich environments, whilst retaining the advantages of the VG. CA and OCA have made finding the shortest paths simpler because the process of path planning is equipped with pre-calculated step-by-step instructions. All these features make it more efficient than the VG. Simulation results show that the proposed algorithms can find safe and the shortest paths in 2D environments. Also, the results comparison with the VG confirmed that both the CA and OCA can find global optimal paths (short and safe) with computational efficiency.

## **Dedication**

I dedicate this work to God Almighty my creator, my strong pillar.

I also, wholeheartedly dedicated this study to spirit of my beloved parents, who have been our source of my inspiration and strength, who consistently provided me with moral, spiritual, emotional, and financial support ceaselessly so I reach the highest levels, may God have mercy on them.

To my husband who has encouraged me all the way and whose encouragement has made sure that I give it all it takes to finish that which I have started. To my children Sufian, Cerine, and Ghofran Alnmrush.

Thank you, my love for you all can never be quantified. God protect you all.

## **Acknowledgement**

Praise be to Allah who blessed us with science, faith, prayer, and peace upon our most honourable of the prophets and messengers.

Thank God Almighty who bestowed me strength and patience to overcome all difficulties that I faced to completion of this Thesis.

Firstly, I would like to express my deep sincere and gratitude to my research guide Dr Lyuba Alboul for continuous support, substantial efforts, immense knowledge, and her leading role in completing this thesis, her guidance helped me all the time to study. Also, deepest thanks for her patience to support me, despite the difficult circumstances that faced me during my studies, I cannot imagine a better supervisor.

Besides my supervisor, I would like to give special thanks to the late Prof. Jacques Penders for his insightful comments and encouragement to improve my thesis.

Also, I would like to express my special appreciation and thanks to Dr. Xu Xu for her brilliant comments and suggestions.

My sincere thanks also go to my dear husband Rasem Alnmrush who helped and encouraged me to continue my study, without his precious support it would not be possible to conduct this thesis.

Great thanks to my family: my brothers, sisters, and sisters-in-law for supporting throughout my study and my life in general.

To my university: Sirte University for providing a scholarship to pursue doctoral study, to Libyan cultural affairs in London who supported me during my study, and to Sheffield Hallam University.

Finally, thanks must be given to all the people who contributed by some way in the work described in this thesis.

## Contents

<b>1</b>	<b>Introduction.....</b>	<b>15</b>
1.1	Motivation.....	15
1.2	Brief Overview of Graph Theory.....	19
1.3	Graph Environment Models.....	21
1.4	Aim and Objectives.....	23
1.4.1	Aim.....	23
1.4.2	Objectives.....	23
1.5	Contributions of the Thesis to Existing Knowledge.....	24
1.6	Outline of the Thesis.....	25
<b>2</b>	<b>An Overview of Main Characteristics of Multi-Robot Systems.....</b>	<b>28</b>
2.1	Introduction to Multi-Robot Systems.....	28
2.2	Multi-robot systems versus single robot systems.....	28
2.3	Classification of Multi-Robot System.....	30
2.4	Homogeneous multi-robot systems versus heterogeneous multi-robot systems.....	33
2.5	Coordination control of MRSs.....	35
2.6	Formation Control.....	37
2.7	Centralised/Decentralised Approaches.....	39
2.8	Connectivity (Communication).....	44
2.9	Motion Planning Issues Overview.....	46
2.10	Multi-Robot Motion Planning.....	46
<b>3</b>	<b>Graph Theoretic approach to the study of multi-robot systems.....</b>	<b>49</b>
3.1	Introduction.....	49
3.2	Graphs approach for a multi-robot system.....	50
3.3	Importance of graphs for multi-robot systems.....	51
3.4	Path planning problem.....	54
3.4.1	Path planning classification.....	55
3.4.2	Path planning approaches.....	56
3.5	Configuration space (C-space).....	56
3.5.1	Advantages of C- Space.....	57
3.6	Classifications of robot path planning methods.....	58
3.7	Techniques of path planning methods.....	59
3.8	Roadmap.....	60

3.8.1	Voronoi diagram .....	61
3.8.2	Visibility graphs.....	61
3.9	Graph Search Algorithms.....	63
3.9.1	Dijkstra's algorithm.....	64
3.9.2	A* algorithm .....	66
3.10	Path planning comparative analysis: A* algorithm or Dijkstra's algorithm .....	68
3.11	Shortest path analysis .....	69
3.11.1	An example of finding shortest path:.....	70
4	<b>Multi-robot motion planning-Based on Roadmap methods.....</b>	<b>75</b>
4.1	Motion planning problem.....	75
4.2	Motion Planning Roadmap Method .....	76
4.2.1	Problem definition .....	76
4.2.2	Problem Statement Assumptions .....	76
4.2.3	Problem inputs and outputs.....	77
4.3	Roadmap Environment Model .....	77
4.4	Motion planning-based on the visibility graph method .....	78
4.4.1	Definition and designing a visibility-graph algorithm.....	78
4.4.2	Visibility-graph pseudo-algorithm.....	79
4.4.3	Visibility graph method for the problem of finding the shortest path .....	79
4.5	Connectivity .....	81
4.6	Measuring Connectivity with the algebraic connectivity.....	82
4.6.1	Algebraic connectivity and collision avoidance .....	83
4.6.2	Example of an undirected weighted graph: .....	84
4.7	Definition (Pseudo-code) Dijkstra's Algorithm.....	89
4.7.1	Pseudo-code of Dijkstra's algorithm .....	89
4.7.2	Flow Chart of Dijkstra's algorithm.....	90
4.7.3	Techniques of Dijkstra's algorithm for the motion planning problem.....	90
4.7.4	Application of Dijkstra's algorithm .....	91
4.8	Multi-Robot Path Planning Algorithm (MRPPA).....	94
4.8.1	MRPP Algorithm .....	97
5	<b>Implementation of the Multi-Robot Path Planning Algorithm.....</b>	<b>100</b>
5.1	Introduction .....	100
5.2	Path Planning Using MRPP Algorithm.....	101



5.3	Result discussion.....	111
5.4	Advantages and Disadvantages of the Visibility graph method .....	116
5.5	Central Algorithm (CA) .....	117
5.5.1	CA Algorithm .....	118
5.5.2	Path Planning Using CA .....	118
5.6	Performance Comparison.....	126
5.7	Optimisation Central algorithm (OCA).....	128
5.7.1	Safety Distance .....	129
5.7.2	Path Planning Using OCA .....	131
5.8	Conclusion.....	133
6	<b>Simulations and Experiments.....</b>	136
6.1	Introduction .....	136
6.2	Path Planning Software .....	136
6.3	Path Planning Software using MRPPA based on VG.....	137
6.3.1	Path planning for simple maps by MRPPA .....	148
6.4	Path planning Software by CA.....	152
6.5	Path planning Software by OCA.....	160
6.6	Conclusion.....	166
7	<b>Conclusion and Future work.....</b>	169
7.1	Conclusion.....	169
7.1.1	2D Path Planning Algorithm.....	169
7.1.2	2D Path Planning Algorithms Based on Visibility Graph Method.....	170
7.1.3	Software Package for 2D Path Planning environment.....	172
7.2	Future Work .....	172
8	<b>Appendix [A] Important Concepts from Graph Theory.....</b>	175
8.1	Graph Theory .....	175
8.2	Graph Rigidity.....	182
8.3	Algebraic Graph Theory.....	183
8.4	The importance of graphs in networks analysis .....	186
8.4.1	The connectivity of the network (graph) .....	186
8.5	Connectivity in multi-robot networks .....	187
8.6	Algebraic Connectivity of Graph .....	188
8.7	Dijkstra's Algorithm .....	189

8.7.1	Description of Dijkstra's algorithm .....	190
8.7.2	Runtime of Dijkstra's algorithm .....	191
8.8	The shortest path .....	194
9	<b>Appendix [B]</b> .....	198
9.1	Program 1:.....	198
10	<b>Appendix [C]</b> .....	206
10.1	Program 2:.....	206
11	<b>Appendix [D]</b> .....	212
11.1	Program 3 .....	212
12	<b>Bibliography</b> .....	221

## List of Figures

Figure 1. 1: Examples of social animals (2014) available online at <a href="http://www.peta.org/issues/animals-used-for-food/factory-farming">http://www.peta.org/issues/animals-used-for-food/factory-farming</a> .....	16
Figure 2.2: Centralised and decentralised control.....	44
Figure 3.1: Example of a multi robot system represented as a graph .....	50
Figure 3.2: Example of a workspace in a static environment .....	54
Figure 3.3: Example of a workspace in a dynamic environment.....	54
Figure 3.4: Examples of workspace and configuration space.....	58
Figure 3.5: Classification model of robot path planning method [118][121].....	59
Figure 3.6: Path planning techniques representation of path planning methods.....	60
Figure 3.7: An example of a Voronoi diagram .....	61
Figure 3.8: Example of Roadmap visibility graph for single robot .....	63
Figure 3.9: Google maps.....	66
Figure 3.10: A* Search algorithm [35][118] .....	68
Figure 3.11: Example of a public transport network in Zurich city [137]. .....	71
Figure 3.12: Example of public transport network in Zurich city [137]. .....	71
Figure 3.13: Example of a public transport network represented as a graph.....	72
Figure 3.14: Example of shortest path in graph from (BP) to (ST) .....	73
Figure 4.1: A sample visibility graph.....	79
Figure 4.2: Example of an undirected weighted graph .....	85
Figure 4.3: Flow chart of Dijkstra's algorithm.....	90
Figure 4.4: Example of a team of robots represented as vertices of graph .....	91
Figure 4.5: First step of Dijkstra's algorithm .....	92
Figure 4.6: Second step of Dijkstra's Algorithm .....	92
Figure 4.7: Third step of Dijkstra's Algorithm .....	93
Figure 4.8: Fourth step of Dijkstra's Algorithm .....	93
Figure 4.9: Fifth step of Dijkstra's Algorithm .....	94
Figure 4.10: Shortest paths by Dijkstra's Algorithm .....	94
4.11. The process of MRPP Algorithm.....	98
Figure 5.1: Scenario of workspace environment for path planning .....	101
Figure 5.2: Scenario of workspace environment represented as a graph. ....	102
Figure 5.3: Example of disconnected undirected weighted graph .....	103
Figure 5.4: The path planned for robot two by using MRPPA. ....	103
Figure 5.5: The path planned for robot three by using MRPPA .....	104
Figure 5.6: The path planned for robot one by using MRPPA .....	104
Figure 5.7: The paths planned used MRPPA.....	105

Figure 5.8: Scenario of workspace for path planning .....	105
Figure 5.9: Example of two disconnected components undirected weighted graph.....	106
Figure 5.10: The path planned for robot one and two using MRPPA.....	107
Figure 5.11: The shortest paths for three robots using Dijkstra's algorithm.....	107
Figure 5.12: Simple workspace environment .....	109
Figure 5.13: Two disconnected components undirected weighted graph .....	109
Figure 5.14: Paths planned for each robot by using MRPPA. ....	110
Figure 5.15: The shortest paths for three robots by using Dijkstra's algorithm.....	111
Figure 5.16: A scenario with two obstacles .....	113
Figure 5.17: Waypoints paths planned by MRPPA. ....	114
Figure 5.18: Waypoints paths planned by MRPPA. ....	115
Figure 5.19: Waypoints paths planned by MRPPA. ....	116
Figure 5.20: The process of Central algorithm (CA) .....	118
Figure 5.21: The steps of Central algorithm (CA) in workspace environment.....	119
Figure 5.22: Two possible paths for robots (1, 2, and 3) by Central algorithm .....	119
Figure 5.23: Computation of waypoint ( $u'(x', y')$ ) at an intersection point ( $u(x, y)$ ).....	120
Figure 5.24: Diagram illustrating $\tan \varphi = M$ .....	121
Figure 5.25: Visibility graph network created by Central algorithm. ....	121
Figure 5.26: Shortest paths for three robots calculated by using Dijkstra's algorithm.....	122
Figure 5.27: Waypoints paths planned by CA of workspace.....	123
Figure 5.28: Central algorithm in simple workspace environment.....	124
Figure 5.29: Visibility graph network created by CA.....	124
Figure 5.30: Shortest paths for three robots calculated by using Dijkstra's algorithm .....	125
Figure 5.31: Waypoints paths planned by CA .....	126
Figure 5.32: The paths planned by CA (red) and VG (orange) .....	127
Figure 5.33: The impact of placing obstacles on the generated paths.....	128
Figure 5.34: The algorithm reroutes the robots for find next shortest paths.....	128
Figure 5.35: The obstacles in workspace with safety distance. ....	129
Figure 5.36: The expanded obstacle with safety distance.....	130
Figure 5.37: A diagram illustrating the relationship between polar and Cartesian coordinates.....	130
Figure 5.38: The partial of visibility graph network by the OCA.....	131
Figure 5.39: paths planned and generated by the OC algorithm.....	131
Figure 5.40: Waypoints paths planned by OCA .....	132
Figure 6.1: Scenario of workspace environment for three robots with one goal .....	137
Figure 6.2: Divided workspace environment by MRPPA .....	138
Figure 6.3: Paths planned by MRPPA using MATLAB.....	139
Figure 6.4: Three shortest paths planned via Dijkstra's algorithm by MATLAB .....	140

Figure 6.5: Workspace details depicted in Figure 6.4.....	141
Figure 6.6: Robots reach the target point.....	141
Figure 6.7: Workspace details depicted in Figure 6.6.....	142
Figure 6.8: Scenario of workspace environment for three robots with three goals .....	143
Figure 6.9: Divided workspace environment using MATLAB .....	144
Figure 6.10: Stapes of Paths planned by MRPPA using MATLAB.....	145
Figure 6.11: Application of the Dijkstra Algorithm for paths planned by using MATLAB.....	146
Figure 6.12: Workspace details depicted in Figure 6.11.....	147
Figure 6.13: Robots reach the target points .....	147
Figure 6.14: Workspace details depicted in Figure 6.13.....	148
Figure 6.15: Simple workspace environment by use MATLAB .....	148
Figure 6.16: Workspace details depicted in Figure 6.15.....	149
Figure 6.17: Stapes of Paths planned by MRPPA using MATLAB.....	150
Figure 6.18: Paths planned by MRPPA using MATLAB.....	151
Figure 6.19: Workspace details depicted in Figure 6.18.....	151
Figure 6.20: Robots successfully reach the target point .....	152
Figure 6.21: Application of the CA using MATLAB.....	153
Figure 6.22: Workspace details depicted in Figure 6.21.....	154
Figure 6.23: Visibility graph by CA using MATLAB.....	155
Figure 6.24: Workspace details depicted in Figure 6.23.....	155
Figure 6.25: Paths planned by CA using MATLAB.....	156
Figure 6.26: Workspace details depicted in Figure 6.25.....	157
Figure 6.27: Robots reach the target points .....	157
Figure 6.28: Workspace details depicted in Figure 6.27.....	158
Figure 6.29: Waypoints generated by CA.....	159
Figure 6.30: Robots successfully reach the target points.....	159
Figure 6.31: Workspace details depicted in Figures 6.30 .....	160
Figure 6.32: Application of the OCA algorithm using MATLAB.....	161
Figure 6.33: Workspace details depicted in Figure 6.32.....	162
Figure 6.34: Visibility graph by OCA using MATLAB.....	162
Figure 6.35: Paths planned by OCA using MATLAB.....	163
Figure 6.36: Workspace details depicted in Figure 6.35.....	164
Figure 6.37: Robots reach the target points .....	165
Figure 6.38: Workspace details depicted in Figure 6.37.....	166
Figure 8.1: Example of a graph. The labelled circles represent the vertices $v_i$ ,.....	175
Figure 8.2: Example of the directed and undirected graph. ....	176
Figure 8.3: Example of degree of graph .....	177

Figure 8.4: Example of regular graph .....	177
Figure 8.5: Example of subgraph.....	177
Figure 8.6: Example of induced subgraph and independent set.....	178
Figure 8.7: Example of isomorphic subgraph.....	178
Figure 8.8: Example of a path and a cycle in a graph.....	179
Figure 8.9: Example of forest, tree and spanning tree graph .....	179
Figure 8.10: Example of simple, multiple, and loop graph and cycle.....	180
Figure 8.11: Example of a weighted graph.....	180
Figure 8.12: Example of connected and disconnected graphs .....	181
Figure 8.13: Example of Bipartite graphs.....	181
Figure 8.14: Example of planar and nonplanar graphs .....	182
Figure 8.15: Example of a rigid and a non-rigid graph.....	183
Figure 8.16: Example of adjacency matrix .....	185
Figure 8.17: Example of Beta index calculated for different graphs .....	187
Figure 8.18: Examples of connected graph.....	188
Figure 8.19: Example of steps of Dijkstra's Algorithm <a href="http://www.programminggeek.in/2013/08/java-implementation-of-dijkstra-shortest-path-algorithm-for-coursera-programming-assignment-5.html">http://www.programminggeek.in/2013/08/java-implementation-of-dijkstra-shortest-path-algorithm-for-coursera-programming-assignment-5.html</a> .....	194
Figure 8.20: Example of the shortest path in graph .....	196
Figure 8.21: Example of shortest path in graph.....	197

## **Chapter one**

# **1 Introduction**

## **1.1 Motivation**

Over the course of the 20th century, robotics and automation became a critical part of modern society; they have been used in many domains such as medical, manufacturing, logistics, aerospace, transportation (warehouses and trans-shipment in harbours), industrial (assembly lines) and agricultural [1]. Society therefore progressively looks to the use of robotics to accomplish many tasks that are complex and difficult to do. For instance, there are new application fields that relatively recently have appeared, such as underwater and space explorations, search and rescue, in particular, in hazardous environments, and service robotics. It is becoming increasingly common to use groups of robots to achieve these missions [2][3]. Typically, the required tasks involve several sub-tasks that can be performed in parallel with a small amount of coordination required between robots. Thus, it has become very important to find ways to describe individual behaviours of robots, which when deployed as a team, can their tasks as a group [2]. Although robots' use is widespread in some fields, most modern robots need well-organised environments to know the conditions that can be predicted to work whilst avoiding failure. Besides that, with technological advancements, robots have become less expensive and more capable, especially when being used to solve a wide range of important and complex issues [2][4]. The idea of creating sets of mobile robots that are cooperated to perform more complex and pre-defined task is everyday closer to become a reality. In addition, the design of a team of robots in executing cooperative tasks in an autonomous way has attracted the attention of many researchers in recent years [4][5]. The essential precept, beyond a new concept of coordinating mobile robots, was directly inspired through the surveillance of natural systems, where it is easy to see many forms in nature such as social animals that are amazing and beautiful examples of cooperative entities. These animals can arrange themselves to perform incredibly difficult and complex tasks. For example, this can be observed by swarms of birds in the air, schools of fishes in the sea or by colonies of the ants and herds of animals on the land (see Figure 1.1) [4]. Also, it can be readily observed that natural teams are execute team-level missions, and they are instinctively able to turn simple individual behaviours into impressive team-level feats [2]. Even though these animals have limited cognitive capabilities, they are act and collaborate to accomplish their missions in ingenious methods to access their common targets. Given these wonderful examples, the concept of simulating natural behaviours of animals and applying them to mobile robotics seems a very attractive idea [4]. For this reason, in the field of mobile robotics there is increased attention to the systems that



are consisted of many autonomous mobile robots called multi-robot systems (MRS), and the study of this system has grown significantly in size and importance recently [6]. Mobile robots are automatic machines that can move in the environment of workspace. In addition, an independent mobile robot is a physically autonomous system, equipped with various sensors and actuators, essential and sufficient to perform and achieve a certain task [1]. Multi-robot system approaches introduce several advantages over single-robot solutions. A multi-robot system works in a shared environment to accomplish some tasks that may be difficult to accomplish by a single robot [4].

Figure 1. 1: Examples of social animals (2014) available online at <http://www.peta.org/issues/animals-used-for-food/factory-farming>

Furthermore, a key ability of multi-robot autonomous systems is cooperative localisation in difficult unknown environments or complex partially known environments via sharing information to reach common targets. Also, the performance of individuals within the team can be remarkably improved through allowing them to perform complex tasks collaboratively in various areas in a more reliable manner. Cooperative work allows performing the missions in less time or with the least cost [3][5]. The main topic fields of multi-robot systems are communication, mapping, and exploration, architectures, task allocation, and control, localisation, object transport and manipulation, motion coordination, and reconfigurable robots [3]. The research works that have studied and developed multi-robot systems have made remarkable progress in recent decades. A team of scientists began investigating this trend of

the research in the late 1980s [3][6]. The research study began in the early 1990s, for example shown in [7], and it has received a rising interest since the mid-1990s [8]. Additionally, in the early to mid-1990s there was a boom of multiple robot systems inspired by the general intelligent behaviour of large biological insect communities, such as ants [9]. Besides that, multi-agent robot systems (MARS) control problems have obtained significant importance. Each MARS has a transport subsystem that contains many mobile robots. The controlling problem similar to that of a mobile robot team, can be divided into two key parts: (1) optimal global mission decomposition into sub-missions and distributing it optimally among separate robots in the team; (2) path planning, control, and movement correction for each mobile robot [10]. In computer science, the research for multi-agent systems usually uses software agents that have been widely studied in the 1980s and 1990s [11]. Also, multi-agent systems have replaced single agents as the computing paradigm in artificial intelligence [11][12]. In the robotic society, the agents in a multi-agent system can also be robots, thus multi-agent systems are referred to as multi-robot systems as well [11]. In [13] the authors have introduced a taxonomy that classifies multi-agent robotic systems based on their computational abilities and communication techniques that has led to the understanding of team behaviour, which emerges collaboratively, and have described theoretical issues that may be raised in the study of cooperation of multi-robot systems, illustrating the usefulness of the taxonomy in simplifying discourse about robot collective properties. Authors in [14][15] have proposed the classification of swarm, collective or robot collaboration research by defining a taxonomy or collection of axes [13]. This is not surprising or unexpected. The continued improvement of the technology and the infrastructure has enabled the deployment of multi-robot systems consisting of increasing numbers of robots. Also, the growing attention to these systems is expected to lead to significant progress in accomplishing complex tasks by them. What is more, a multi-robot system will be superior to a single robot by performing certain dangerous tasks successfully, such as environmental exploration, or demining tasks [8]. Within search applications, a multi-robot system offers many advantages over single robot's solutions, especially when used in a hazardous area or inaccessible area to humans, for example, disaster relief workers could use a swarm of robots to search and- rescue victims. Swarm robotics applied in search missions could offer many major advantages over traditional search techniques [16]. Multiple robots in swarm robotics collectively solve problems by forming benefits structures and behaviours like the ones observed in natural systems, such as swarm of fish, bees, or ants [17][18]. The team (group) behaviours appearing in the swarms emerge great robustness and flexibility [18]. However, the distributed nature of these systems makes the

design of the effective algorithms quite difficult, the total performance relies significantly on the issues arising from the complex interactions between the robots. The design of these algorithms has highly challenging requirements. One is that the robots must be highly mobile. Another one is that the robots must maintain a communication network across a large geographical domain. So, distributed algorithms must be robust to changing network topology. The third requirement is that the robots must estimate their physical composition. This means that the characteristics of the environment where robots operate must be evaluated, also the robots will need some geometric information about the locations of other robots, or they cannot coordinate their movements. The last one is that multi-robot distributed algorithms must be robust to population changes due to robots failures or the addition of new members. Fundamentally, in order for the algorithms to be able to achieve these requirements, they must operate at the intersection of physical mobility, communication networking, and distributed computation [19]. In addition to that, these algorithms need to overcome several challenges. One of these challenges is the development of distributed motion algorithms that guarantee connectivity of the overall network. The algorithm needs limited local information and communication among robots to determine the addition or deletion of network links through distributed consensus and market-based auctions [20]. The other one is how to design appropriate coordination strategies between the robots that enable them to perform operations efficiently in terms of time and working space [6]. Thus, all this requires a multi-robot system to have the ability to make decisions based on the situation of its surrounding environmental changes that may greatly enhance its autonomy [21]. The existing technologies can operate a system like a multi-robot system in a known and relatively regular environment. It also operates in a dynamic environment that has obstacles that may appear while performing the task; these technologies are not sufficient because robots do not have the ability to make their own decisions [21]. Therefore, particular attention has been presented to a multi-robot system developed to operate in dynamic environments, where uncertainty and unexpected changes can occur due to the presence of robots and other factors that are external to the MRS itself. Technological Improvements in both (hardware and associated software) are the main reasons behind the increasing interest in MR system, where the software techniques developed for robots, applications take advantage of the hardware improvements and introduce complicated and credible solutions for the basic missions, which robot must be capable to execute whilst working in real environments: path planning, object recognition, object transportation, localisation, and tracking, etc [22]. Besides, autonomous multi mobile robots are systems that operate in a partially unknown and unpredictable environment. This means that robots must

have the capability to move without disruption and have the ability to avoid any obstacle placed inside the confinement of movement. In addition, these systems have little or no human interference for their movement and are designed in a way that follows a predetermined path, whether in an external or internal environment [23]. Additionally, the autonomy of the system involves many domains such as communications, trajectory generation, and sensor fusion, cooperative tactics, task allocation, scheduling, and path planning. The path planning is an important domain in the control of mobile robots, and it is considered one of the main elements of autonomy and plays a key role in enhancing the autonomy level of a multi-robot system. Therefore, it is taken into consideration when designing multi-robot systems [21][24].

Even though the use of a multi-robot system has many important advantages that distinguish it from a single robot system, it has also some foundational problems, which require solutions, such as consensus protocol, flocking, formation control, task allocation, rendezvous, containment, centralised and decentralised control, mapping, exploration, optimisation and communications, and the motion planning problem [3][25]. These problems are still under study and analysis and researchers are still looking to find suitable solutions to address them in order to achieve a high quality of overall performance, although there are advanced studies on this subject and many articles address more than one of these problems [3][25]. For this reason, we investigate one of these problems - the problem of motion planning. The research investigates a multi-robot system in terms of its advantages, disadvantages, and problems. It focusses specifically on the problem of motion planning and how to solve it by using the concept of graphs. Graphs and algebraic graphs theory are efficient and powerful tools that are both used to solve a wide range of problems.

## **1.2 Brief Overview of Graph Theory**

Graph theory was originated in 1736 when Swiss mathematician Leonhard Euler solved the Königsberg bridge problem (a famous example is the 'Seven Bridges of Königsberg' problem): where the author in his published paper in 1736 discussed the possibility of crossing all the seven bridges of Königsberg just one time [2][26][27][28][29]. See Figure 1.2 for an illustration. The first paper of graph theory in history was written by Euler about the seven Bridges of Königsberg in 1736 [30]. Euler's main insight was that the islands and bridges could be modelled by a simple mathematical structure, the mathematical structure constructed for the problem is known as a graph model of the problem [27][28]. Since Euler's solution to this problem, graph theory has evolved to become one of the essential fields of applied mathematics [31]. Since then, graph theory has contributed to solving many mathematical issues [27].

Although graph theory is closely related to applied mathematics, it is a multi-disciplinary field among Mathematics, Operations Research, and Computer Science. Industrial and systems engineering uses graphs as well for optimisation [2][29][30]. Graphs are one of the important concepts that could help to solve problems in various fields. In mathematics and computer science, graph theory is the study of graphs, which are mathematical structures used to model relations between objects of a certain collection [2][29][30].

Figure 3.2: Königsberg bridges. Bóta, A. (2015) Methods for the description and analysis of processes in real-life networks, available online at

<https://www.google.co.uk/search?q=graph+theory+three&biw>

At present, graph theory is an effective domain in both theoretical and applied sciences. It is a rapidly developing area of research, and its various applications to networks, distributed computing, social networks, and web graphs partly explain the growing attention to it [2]. On the other hand, graphs are quite efficient tools for describing relationships between objects that are represented by nodes (vertices). In turn, relations among nodes are represented by connections [29]. In general, any mathematical object involving points and connections among them can be called a graph or a hypergraph. For example, physical networks, map colourings, databases, organic molecules, ecosystems can be modelled as graphs. All these examples require multi-graphs, like directed graphs or graphs which allow loops. Therefore, graphs can serve as mathematical models. Graph theory has been studied to solve many problems, such as traffic routing problems, payload transport, task assignment, air traffic control and many other applications, including robotics [2][29][30]. Motion planning is an eminently important topic for mobile robots since, by definition, a robot performs missions by moving in the real world [25]. The problem of motion planning is one of many difficult problems that can be solved by using concepts of graph theory. It is also considered a common problem in multi-robot systems and still requires more investigation [32]. Much research has been dedicated to this problem, because of its importance in different areas and not only in multi-robot systems. The main task

of motion planning is to produce a continuous movement or path that connects the start position to the target position without collisions. In robotics, the problem of motion planning includes producing a continuous robot motion from one configuration to another in a configuration space whilst avoiding collision with obstacles [21] [25]. Also, the problem at hand is to design a strategy that allows the robots to arrive their required locations through short collision-free paths [33].

### **1.3 Graph Environment Models**

There are several environment models that are widely known for purposes of path planning problems, such as vector (obstacles are represented by polygons), grid (occupancy cells), and graph. Each one of these environments has specific features and drawbacks. A graph is considered a suitable model for motion planning problems and path planning purposes, as a basis, the graph model contains only possible paths, i.e., information about obstacles is excluded when the graph is established [11]. Masehian, E., & Sedighizadeh, D, (2007) mentioned that more than 50 percent of all recent robot planning algorithms are dependent on classical methods. However, the implementation of the classical methods is in constant reduction if it is compared with heuristic approaches [33][34]. Therefore, many approximation algorithms have been proposed to address the motion planning problem [30]. In fact, there are three kinds of (classical methods) graph approaches that contribute to solving this problem cell decomposition, potential field, and roadmap [21][30][33]. Roadmap approaches are one of the main techniques that allow groups of robots to find the shortest path in the workspace to perform their tasks. Specifically, the roadmap algorithms for two-dimensional (2D) path planning that drive robots to move along the path designed in the configuration to reach required goals. This approach uses information from static obstacles, and it contains simple, collision-free path segments that are combined [32]. The most common roadmap methods are Visibility graphs (VG) and Voronoi Diagrams (VD) [21][33][34][35]. These methods graphically analyse the map to produce a connectivity graph or network. A connectivity graph is a set of feasible paths from the current robot location, through sets of consecutive vertices, to the goal location [33]. On the other hand, path planning utilising roadmap-based approaches represent the environment by establishing maps or graphs from sets of vertices and edges;( in each method, the vertices and edges are determined to build a road map in different manner [21][35]). Voronoi diagrams produce vertices (waypoints) that are equidistant to two or more objects (or VD of a set of geometric objects is a division of space into cells, each of which contain points closer to one particular object) [33][34]. Also, the roadmap contains paths or

Voronoi edges that are equidistant from all the points surrounding obstacles, the points where the paths meet are called nodes (i.e., VD determines vertices that are equidistant from all the points in obstacles region) [21][35][36]. Voronoi diagrams were first used by Canny, J [37]. Takahashi, O., & Schilling, R. J, have employed VD in many robots' path planning methods [38], whilst Lin, C.C., Chuang, W. J., & Liao, Y. D, have combined VD with heuristic methods [39]. Although VD generates long paths and are far from obstacles (this makes it relatively safe, due to increased distance among obstacles and the robot), however, the paths are not optimal (as it is long) and not efficient, thus this is the main disadvantage of this method [21][33][35].

Visibility graphs are considered to be one of the oldest roadmap methods that applied in a 2D environment, and they were used on robot Shakey [35]. VG considers obstacle vertices in the environment to be the vertices, through which the robots can arrive at their required locations (or it is the set of lines in the free space that links an advantage of object to another; these advantages are vertices of polygonal obstacles [11][33][34]). Visibility graph approaches proceed to link vertices that are visible to each other. These visible vertices have the property that a straight line connecting them does not intersect the interior of obstacles [33][35]. Asano, T., Asano, T., Guibas, L., Hershberger, J., & Imai, H, have firstly used VG in robot motion planning and, they ensured that the robot would find the shortest path to its target [40], Alexopoulos, C., & Griffin, P. M have introduced two algorithms for path search using a visibility graph established from a 'tessellation of contours and removal concave highs'[24] [41]; the first one was a V\*graph algorithm, which minimised the number of considered nodes, hence, minimising the computational complexity of the algorithm [42][33]. Also, the algorithm assumed that the obstacles were static. The second one was called E \*Graph and, it assumed the obstacles could move along linear paths at a constant speed [24][41]. In contrast to Voronoi paths, one advantage of VG is the ability of finding a path with the shortest distance if one exists [21]. In addition, there are several algorithms that are employed for solving optimisation problems on the graph such as Dijkstra's algorithm: A\*, and D\*algorithm, etc [11]. These algorithms are employed to obtain the optimal paths, or to find the shortest collision-free paths [21]. In addition, it is often possible to transform vector and grid environment models to the identical graph representation, and the possible paths can be represented by a visibility graph. Hence, the algorithms can be applied, for instance, on visibility graphs [11]. Furthermore, the visibility graphs methods can help the robots in the system move to the desired goal location while avoiding collisions [21]. The advantages of using a visibility graph for motion planning are that it's a well-understood and simple method that produces optimal paths in a two or three-

dimensional workspace [11]. Also, it is computationally effective, and guarantees to obtain an optimal path if there is one [21][35]. For this reason, this thesis focuses on the visibility graphs method in a 2D environment. This chapter does not discuss exact Visibility Graph (VG) method in detail. More information about as Exact VG method could be found in Chapter three.

## **1.4 Aim and Objectives**

### **1.4.1 Aim**

The overall aim of this project is to study and provide a solid mathematical background to the development of multi-robot systems based on graph theory and algebraic graph theory. In particular, this project aims to develop path planning algorithms for a multi-robot motion planning problem in a two-dimensional (2D) environment based on graph techniques and graph search algorithms. In addition, to demonstrate how path planning can be improved using graph theory to find a collision-free path through an environment with obstacles, from a specified starting position to the desired target destination with the achievement of certain optimisation criteria.

### **1.4.2 Objectives**

- Establishing a theoretical framework for a multi-robot system based on graph theory.
- Investigating the motion planning problem for a multi-robot system and discuss how to develop and find a solution to it based on graph techniques and graph search algorithms.
- Investigating the path-planning problem or collision avoidance between robots and address it through exploiting tools from the graph theory, such as the properties of weighted graphs, edge-weight functions, and the matrices associated with graphs and their eigenvalues, especially the Laplacian matrix and its second smallest eigenvalue and their important roles to determine the measure of robustness connectivity.
- Designing and execution of a roadmap-based multi-robot path planning algorithm in a 2D static environment, consisting of polygonal obstacles, which assumes a-priori knowledge of robots' positions, their goals' positions, and surrounding obstacles. The algorithm combines the visibility graph method with the algebraic connectivity (second smallest eigenvalue  $\lambda_2$ ) of graph Laplacian and the Dijkstra's algorithm for find the optimal path for robots' motion whilst avoiding a collision.
- Develop two path planning algorithms in two-dimensional (2D) workspace environments based on the visibility graph method, to improve its performance to generate paths safer and not too close to obstacles during movement to avoid collision.



## 1.5 Contributions of the Thesis to Existing Knowledge

In order to extend the area of knowledge and emphasise the important role of advanced graph algorithms, and to address the path planning problem for a multi-robot system, several solutions are developed which form the contributions of this thesis.

**The first contribution** is the development of an algorithm for multi-robot path planning in 2D space. The result of the proposed algorithm is an optimal, collision-free path. The proposed algorithm is based on the combination of the visibility graph method with the algebraic connectivity (second smallest eigenvalue  $\lambda_2$ ) and the Dijkstra's algorithm. The algorithm implies sequential path planning for each of robots (path by path) based on the measure value of algebraic connectivity of graph Laplacian, which controls the inter-robot's connectivity when it away from zero. In addition, predefined weight evaluation function (edge weights), when planning the path of each robot, considers all the paths already planned for path correction as well as avoiding collisions. The algorithm provides optimality of all planned paths because the paths depend on the order of planning, thus the choice of the right sequence for path planning of robots have significant impact on performance of the team. Additionally, these algorithms possess a standard of completeness and computational efficiency to path planning and are able to find optimal paths if the environment is known. It is also emphasised that the visibility graph method and Dijkstra's algorithm are chosen because they are guaranteed to produce an optimal path if one exists, where within the context of this thesis, an optimal path means the path that has the shortest distance from a start position to a goal position.

**The second contribution** of the thesis is the development of a set of path planning algorithms that are based on the visibility graph (VG) method. The algorithms are computationally efficient because the number of obstacles that are used for path calculation is relatively small. This means the algorithms find paths by reducing the number of obstacles (as well as edges) which lowers the computation time contrary to the visibility graph approach. On the other hand, the algorithms hold the completeness criterion as it will generate a path if one exists, hence they solve the problem of the conventional VG method, and they hold the completeness criterion. The outcome of the developed path planning algorithms is optimal (shortest) and collision-free (safe) paths that direct robots safely away from obstacles. In addition, they reduce the computational complexity of roadmap approaches and are also able to produce general solutions for different environments. The algorithms plan safe paths for the robots in the C-space; so that they can traverse through the vertices of obstacles in different scenarios of workspaces without the collisions, even with add new obstacles. It is also worth emphasising

that the algorithms possess the criteria of path planning and may be capable of finding a globally optimal path if the knowledge of the environment is fully and accurately known. Note, the optimal paths here mean the safe and shortest paths.

Additionally, the software packages and the simulations to realise the path planning algorithms have been developed. The proposed control algorithm strategies are designed to be user-friendly, equipped with step-by-step instructions using MATLAB in two 2D environments, where random or particular scenarios can be generated. Simulation results demonstrate the effectiveness of the proposed algorithm.

## 1.6 Outline of the Thesis

This thesis is structured as follows:

The thesis focuses on studying and providing a solid mathematical background to further develop a self-organising multi-robot system, initiated in the Centre for Automation and Robotics Research, establishing a theoretical framework for such a system based on graph theory approaches. Each chapter focuses on a specific topic and starts with an introduction section that includes detailed information about the robotics based on the literature review.

**Chapter 2** is a review of the relevant literature on multi-robot systems and main characteristics of these systems in terms of coordination and formation control, centralisation and decentralisation, communications, and the motion planning problem.

**Chapter 3** presents a comprehensive literature survey of the importance of graphs to study of a multi-robot system, and how problems of this system can be solved using graphs properties. Specifically, it provides a broad survey of literature about the motion planning problem and the importance of path planning and discusses how to solve this problem based on graph algorithms.

**Chapter 4** introduces the multi-robot motion planning and application details of the concepts of graph theory to this system. Particularly, the problem of motion planning is discussed and graph-theoretical techniques to solve the problem are presented. More specifically, it investigates: (1) the multi-robot motion planning based on roadmap methods in a 2D environment; (2) the key role of algebraic connectivity (second smallest eigenvalue  $\lambda_2$  of the graph Laplacian) to maintain connectivity and avoid collisions; (3) the multi-robot path planning proposed algorithm based on visibility graph methods in conjunction with the algebraic connectivity and the Dijkstra's algorithm. The chapter also presents the pseudocode

of the developed algorithm and provides details on how and why the algorithm works to find a collision-free path.

**Chapter 5** deals with direct applications of path planning based on the proposed algorithm to find the optimal path for a team of multiple robots and to drive them to reach their predetermined targets in a known workspace environment. In addition, it gives examples of randomly different scenarios of different workspace environments for groups of multi-robots and discusses the proposed path planning algorithms in 2D environments based on the VG method. The chapter also demonstrates the application of these algorithms to path planning. Furthermore, the improvement of the proposed path planning algorithms is highlighted. Additionally, it discusses and compares the results of the study conducted.

**Chapter 6** explains the simulations and experiments software package that has been developed for path planning to validate the effectiveness of the proposed control algorithms strategies using MATLAB and discussed the results that have been obtained.

**Chapter 7** concludes with comments on the work and provides conclusions based on the proposed work in this thesis. Lastly, the chapter combines final comments about the results obtained and the possible guidelines for future work.

**Appendix A:** summarises some of the main concepts and principles of graph theory, algebraic graph theory, and provides a detailed description of Dijkstra's algorithm.

**Appendix B:** presents a program for calculating the shortest paths using Dijkstra's algorithm for a team of multiple robots developed in MATLAB.

**Appendix C:** presents a program that implements the path following algorithm for Differential Drive robots to follow the desired path using a robot simulator developed in MATLAB.

## **Chapter two**

## **2 An Overview of Main Characteristics of Multi-Robot Systems**

### **2.1 Introduction to Multi-Robot Systems**

At the present time, the development of multi-robot systems (MRSs) is one of the most important research topics. It has raised the interests of many researchers and has seen tremendous progress of the related technology. An MRS can be described as a group of robots that are operating in the same working environment. Modern robotic systems might range from simple devices equipped with sensors, to ones that can obtain and process data, to complex mechanisms that are able to interact with the working environment using fairly sophisticated methods. It is also difficult to provide a definition of the level of autonomy needed for robots to be considered as ‘independent’ entities working in the environment, instead of simple machines that supply services to the operator [5][22][43]. This chapter focuses on the overview of the existing research on multi-robot systems. It starts with the description of main characteristics of these systems, their importance, and what potential problems may occur and how they can be solved.

### **2.2 Multi-robot systems versus single robot systems**

A single-robot system (SRS) consists of an individual robot only, which is capable to act in the environment and to interact with it. In this system, the robot is often designed to handle a mission on its own account. Whilst an SRS provides relatively robust performance, some missions might be inherently very intricate or even impossible for a single robot to perform, for instance, spatially separated missions [6][13]. According to Dudek, G., Jenkin, M.R., Milios, E. & Wilkes, D, the inherent constraint is the spatial limitation of a single-robot system (for more information, see [13]). An MRS consists of more than one individual robot. Most of researchers agree that a multi-robot system provides advantages over a single robot system for many reasons. An MRS focuses on the execution of tasks in a more efficient way by increasing the number of small and simple robots in simultaneous operations [6][43]. It brings flexibility and robustness to the system by taking advantage of natural parallelism inside the system. In addition, a multi-robot system has a desirable capability for accomplishing spatially distributed tasks, which cannot be achieved by a single robot; examples of these tasks are underwater discoveries, big area surveillance, and goal detection [22][43][44]. In addition, an MRS can solve several problems including establishing mobile communication networks, distributed sensing, and robotic search and rescue applications. The main reason for the growing interest in multi-robot systems, is their ability to withstand system failures by creating redundant processes and expanding responsibilities (if one of the robots fails, the others step in). Thus,

the system provides a decreased failure rate. For instance, multiple robots can localise themselves quite effectively, if they exchange information about their positions whenever they sense each other, this may lead to the increase of the overall system durability [43][45][46]. In the real world, an MRS is beneficial not only when they are performing different acts, but also when the robots have the same abilities. In addition, from an engineering viewpoint, an MRS can improve the effectiveness of the automated system, either in terms of performance in achieving appointed missions, or in the robustness and reliability of the system (this can be raised via modularisation). Furthermore, even when an individual robot can accomplish the given mission, the potential of deploying a group of robots can improve the performance of the whole system [22]. On the other hand, the efficient control strategies represent big challenges for the development of those systems, since the robotic agents mostly have limited sensing abilities, mobility, and communication. The robots might have no communication abilities or might be able to communicate with each other only at a specific distance or having communication links that might follow a certain random pattern. On top of that, the robots can just sense each other directly in their field-of-view [47]. Nevertheless, despite all these drawbacks, designs of MRSs proved to be very robust against the failure of robots or communications. Moreover, an MRS presents various attractive advantages in many ways, in particular, those that are linked to the ability of scaling up the systems. All these advantages have given researchers incentives to expand research activities in this direction recently [43][47][2][48][49][50]. Furthermore, there are other advantages that make MRS usage more widespread than SRS as follows [48]:

- A multi-robot system can accomplish complex tasks that are not achievable at all or would be too difficult for a single robot.
- An MRS has a better spatial distribution where the tasks are inherently distributed in time or space.
- Setting up many resource-bounded robots is easier than using a powerful and complex single robot and allows tasks executions with lower costs.
- Multiple robots can act together, where the cooperative action allows problems solving and performing tasks in shorter time due to parallelism.
- Different robots may provide complementary capabilities at the same time.
- Redundancy offered by the introduction of multiple robots can raise the overall robustness of the system [48].
- The possibility to work on repetitive tasks is another advantage of a multi-robot system.

All these advantages have contributed to the study of multi-robot systems in the last few decades [2][5][6][44][45][50][51][52][53][54]. Furthermore, there is increasing attention recently to the development of systems of multiple autonomous robots because they exhibit collective behaviour, with advantage over one single robot with multiple abilities which may waste resources. Whereas multiple different robots, each one has its own configuration, are more robust, flexible, and cost-effective. In addition, the tasks to accomplish may be too difficult for one individual robot, whilst they can be effectively done by multi-robots [55]. Moreover, multi-robot systems have been used in various real-life applications instead of single robots because of their efficiency and applicability. Also, the multi-robot system is defined as the team of robots organised in the shape of a multi-agent architecture so that they can act towards the same or different targets to execute a common mission [56][57]. In addition, over the last ten years, the multi-robot system has continued to attract attention due to expected special abilities such as robustness, communication, coordination, parallel operation, awareness, cooperative behaviour, and scalability. Thus, compared to a conventional SRS, MRSs can cover a wide range of fields during goal recognition, object transportation/relocation, or where a multi-robot solution is more efficient, more cost-effective, more reliable, and more robust than a single robot. All these characteristics are relevant to the efficiency of multi-robot systems [57][58][59].

Figure 2.1: Example of a Multi Robot System<sup>1</sup>

### 2.3 Classification of Multi-Robot System

In this part, we will describe many works related to MRS through collecting them according to their taxonomy. There are many kinds of MRSs, each capable of executing a wide range of missions. Since the vast variety of apparatus and configurations can be classified as multi -

---

<sup>1</sup> The images of robots used are images of robots e-pucks ( <https://e-puck.gctronic.com/>) that live in our robotics lab.

robots, it is essential to understand different factors that are important in MRSs, along with: how this system is classified, what its properties are, and also some shapes of classification to put these systems into perspective [56][60]. Many survey and research papers have been introduced that are related to MRS classification. In addition, several published research papers have organised and given a taxonomy of multi-robot [56]. Iocchi, L., Nardi, D., & Salerno, M., have presented a taxonomy for MRSs based on their cooperative abilities, where the classification allows for an accurate taxonomy of various typologies of multi-robot systems, with special attention to those design choices concerning cooperation inside the MRS [43]. Farinelli et al. in [22] also introduced a taxonomy to categorise approaches to coordination in MRS. The taxonomy introduced in [22] is characterized by two sets of dimensions: Coordination Dimensions and System Dimensions, as shown in Figure 2.2 [22].

Figure 2.2: Example of MRS taxonomy from [22]

Whilst Dudek et al. in [13] proposed a taxonomy that classify multi-agent systems based on communication, calculational capacity, and a little other parameter. They have proved that a cooperative effort may be more compelling when compared to a single entity of the collection, and they discussed the classification of multi-robot system focused on the calculation and communication aspects. Wang et al in [61] have categorised multi-robot coordination into four approaches, i.e., deliberative, hybrid, reactive, and behaviour-based. Cao et al in [62] have been suggested some dimensions for classifying the multi-robot system. Czarnecki, C. A. in [60] has presented a classification of multi-robot systems, the details about the proposed classification in [60] are shown in Figure 2.3.



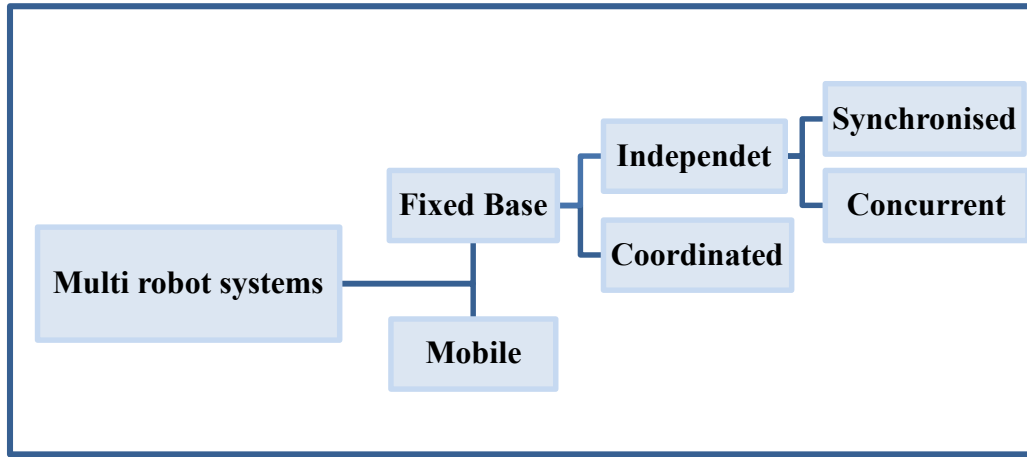


Figure 2.3: Multi-robot system classification from [60]

In contrast, Verma et al. in (2021) have focused on categorising the coordination approaches for multi-robot system. Figure 2.4 shows the general classification that is suggested in [56].

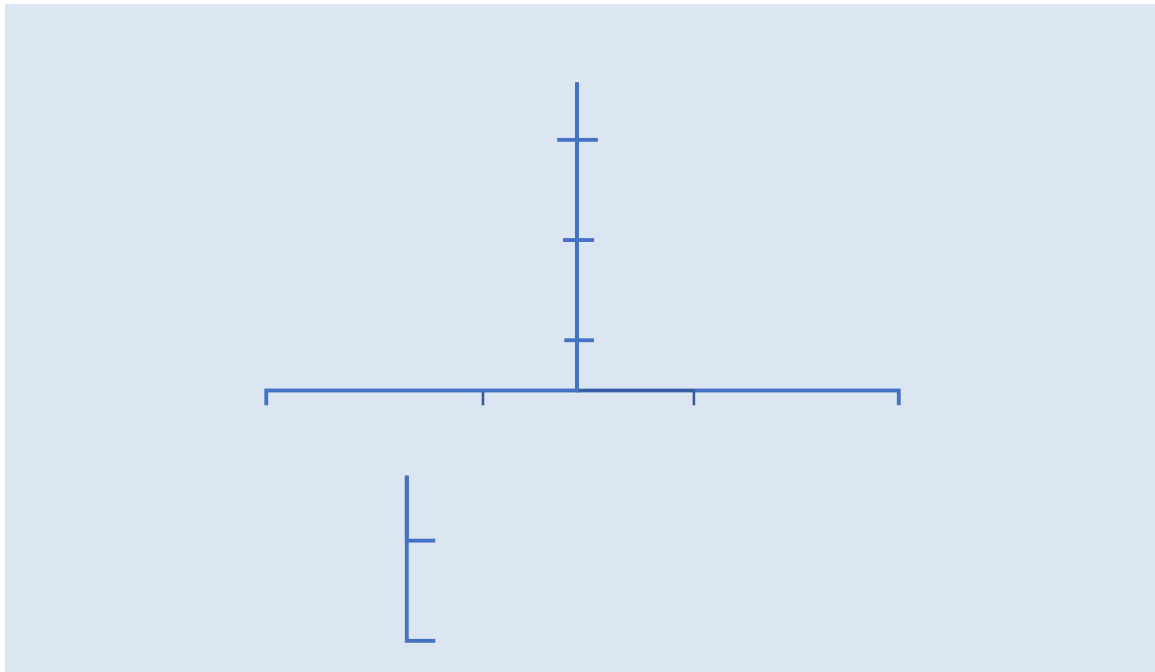


Figure 2.4: Classification of MRS [56]

The taxonomies that are proposed for classifying the MRSs have focused on the important aspects that influence the development of a MRS. The taxonomy in Figure 2.2, has considered the problem of coordination as a central problem in designing an effective MRS. In addition, a coordination considers as a cooperation where the actions executed by each robot consider the actions performed by the other robot. It is also defined the coordination as a set of rules that the robots must follow to interact with each other in the environment [22]. In Figure 2.3 the

taxonomy only given considered the independent fixed base multi-robot systems (IFMRS), see [60]. Whilst, in Figure 2.4 the taxonomy focusses on classification of the coordination approaches for MRS and considered the coordination an important and difficult element in designing effective MRSs, especially when dealing with difficult missions and large-scale systems. So, for MRSs to obtain widespread acceptance, the robots and their control systems must be adaptable and flexible in the environment of their working and to their missions it performs [56].

To provide a classification for MRSs, we need to know what different and important aspects that have effect on the development of this system. We have identified three dimensions related to MRS with some similarities to other previous taxonomies. The taxonomy we are proposed for classifying MRS is characterized by three dimensions: composition, coordination, and communication. The first aim at characterising includes the composition that influence team development, while the second one concerns the kind of coordination that is achieved in the MRS, the last one is communication type that affects the performance of MRS through exchange the information between the robots, see Figure 2.5 and Sections 2.3, 2.4, 2.6 and 2.7 for more details about the proposed classification.

Figure 2.5: Classification of MRS

## **2.4 Homogeneous multi-robot systems versus heterogeneous multi-robot systems**

Usually, the field of a multi-robot system includes mobile robots that can move and interact with each other to achieve a certain task and/or to reach a specific target with additional requirements to keep a particular formation [6]. It is often referred to as a team of robots; and

can either be arranged as heterogeneous or homogeneous, or by other mean, based on composition. MRSs can be categorised as heterogeneous and homogeneous robot systems [56][57]. Members in a heterogeneous multi-robot team/group differ in the hardware structure or in the control systems, or in both [43][56][63]. In addition, the abilities of the robots may vary, where the robots can be specialised for specific missions [6]. Moreover, in heterogeneous multi-robot teams, the specialisation is determined either via functional or structural differences, or both, between team members [56][63]. Also, each robot in a heterogeneous team performs a different function, thus it is important for the robots to coordinate their works to accomplish a collective mission [64]. A heterogeneous system may consist of diverse robots, for instance, different types of robots like aerial and land robots which can operate together to help coordinate the group and to determine probable goals to execute their tasks whilst maintaining a specific formation [6][22][43][63]. On the other hand, the team of homogeneous multi- robots contain the same configuration forms, capabilities, and properties [57]. A homogeneous team is often composed of small robotic units that share an identical control system and equivalent physical structures [63]. Every individual robot has its own controller that is an exact copy of those assigned to the other teammates. The specialisation in a homogeneous robot team emerges by a self-organising process or dynamic of mission /role allocation, where the team members autonomously allocate the roles between themselves [63]. Besides, in homogeneous multi-robot teams, the abilities of all robots are identical, because all the robots have the same hardware and software [6][43][63]. Moreover, if the robots of a deterministic homogeneous system are subject to the same inputs, they have the same behaviour through generating the same outputs. Moreover, they work differently just when they behave under different conditions in environment in which they operate, or if the system is non- deterministic [43]. Additionally, a homogeneous multi-robot team has many advantages such as that it can be easily amended; its composition is simple and less expensive, easy to process spreading, and easier to maintain the design process [6][22][43][63]. In general, the homogeneous or the heterogeneous of the members can influence the way in which robustness is achieved [43]. Heterogeneous multi-robot/team requires some shape of inter-robot communication, on the contrary, inter-robot communications for a homogeneous multi-robot team is not mandatory. But a main requirement for the homogeneous team with no inter-robot communication, to be strong (i.e., robust) is that the single robot of the team must be intelligent, i.e., be able to make choices and act accordingly. Thus, a strong interior control structure inside each robot in the team is required for a single robot to be intelligent [64]. Heterogeneous multi-robot systems are more complex than homogeneous multi-robot systems, therefore their

mission planning becomes very difficult. For example, the robustness of a heterogeneous robot group is fragile due to allocation, where if an individual in the group fails, it is difficult to replace its function and the whole group is likely to fail. Similarly, the group may fail, due to environmental changes and uncertainties. On the other hand, homogeneous robots' groups do not suffer from these restrictions because their members' ability to perform any role makes the group's performance less susceptible to individual robot failure and to changes in the operating conditions [6][22][43][63]. Figure 2.6 provides a classification of multi-robot system based on the homogeneous and heterogeneous robots [64]. As a result, our work takes into account only homogeneous robots due to their advantages over heterogeneous counterparts, and ease of handling.

Figure 2.6: Classification of MRS based on homogeneous and heterogeneous robots [64]

## 2.5 Coordination control of MRSs

Despite improvements of the modern technology used in multi-robot systems, there remain several challenges to overcome. One of the most interesting challenges is the control and coordination of a team of mobile robots. Coordination is known as collaboration in which the actions of the team are accomplished by all robots together such that, each robot takes into consideration actions of other robots in the team and does it in a coherent and high-performance manner [6][43]. In addition, coordination, and cooperation in multi robot are known as: “joint operation or work amid a team of robots”, or the coordination is the mechanism employ for cooperation [56]. There is also a set of predefined instructions that the robots must follow to interact with each other in the environment. Coordination is a core task for a multi-robot system, where the system performance may be directly impacted by the quality of coordination and control [6][43]. An essential control problem for distributing and coordinating a team of mobile

robots in an uncertain or complex environment is how to ensure and maintain the connectivity of mobile robots under constraints of communication to accomplish their tasks, without collisions with each other. Furthermore, coordination can be either dynamic or static, and centralised or decentralised. Dynamic coordination happens during the performance of the task, and it is often based on the synthesis and analysis of the information. Where information can be acquired via the means of communication, it is defined as reactive or online coordination [43][56][65][66][67]. This type of coordination has difficulty handling intricate tasks, and it is also divided into two classes: implicit coordination and explicit coordination. Implicit coordination is known as techniques that use the dynamics of the interaction between the environment and the robots to accomplish the required collective performance that usually manifests in the form of emergent behaviour [6][25]. It is often connected with implicit communication when the robot considers the behaviour and models of others, whereas explicit coordination is defined as techniques that use accredited collaboration ways and communication, such as those employed in multi-agent systems, and that are usually used to deal with relatively more sophisticated robots. It is also often connected with explicit communication that is produced through the active behaviour of robots. Integrating implicit and explicit information can improve the coordination performance for the entire robot system [25].

On the other hand, static coordination requires the adoption of an agreement before starting the task. For instance, traffic control that involves certain rules such as standing at the intersection and maintaining adequate distance between a robot and the corresponding robot and keeping it, for example, to the right. It is also defined as offline coordination or deliberative coordination. This type of coordination can deal with complex tasks, where the real-time control may be difficult to implement and substandard [65][67][68][69]. Additionally, control coordination in a multi-robot system has received tremendous attention lately; due to the many advantages which can be acquired when replacing a single robot system with a multi-robot system. In fact, many researchers have turned their attention towards nature to find ideas that may help to solve various coordination issues in a distributed manner. Challenging features remain concerning different aspects of coordinated control such as formation control, flocking, rendezvous, consensus, synchronisation, containment control, cooperative and simultaneous searching and reconnaissance, cooperative localisation, and mapping [5][45]. The challenges of a multi-robot system are to design suitable coordination strategies among robots that enable them to accomplish processes efficiently in terms of the working space and time [70]. Different control strategies have been proposed to accomplish the distribution of control coordination of a multi-

robot system; these strategies include many methods such as the graph-based method, behaviour-based, leader-follower, and virtual structure, etc. The graph-based method has become dominant over the other methods, since a multi-robot system can be modelled as a mathematical structure, known as a graph. Also, most aspects of coordination control can be studied via utilising the benefits of graph theory; the developments have lately been summarised for both graph theory and algebraic graph theory, both are playing core roles in order to provide a coherent profile of the distributed control coordination for a multi-robot system, especially in the formation control. For example, the Laplacian graph perspective that provides insight into different research issues of multi-robot systems, is of high importance [68][71][72][73] (see Chapter 3 and 4 for more detail on how graph theory concepts are employed in studies of multi-robot systems). Another concept of coordination control design that has the same importance, and is closely related to network connectivity, is formation rigidity. The rigid graph theory has played a key role in network localisation and analysis of the configuration performance [68].

Figure 2.7: Classification of Coordination in MRS [68]

## 2.6 Formation Control

Formation control of a multi-robot system is considered one of the core subjects in robotics, which has seen many research studies lately. It is also an important domain not just because of its theoretical importance, but also because it meets many practical requirements [48][74]. The concept of formation can be observed from natural life; it is not an invention or discovery of the human mind; it can be seen in flocks of migratory birds or flocks of animals or schools of fish. The idea behind formation control was directly inspired by observing these examples in nature, and then applying them to robots. Formations are designed to implement particular

tasks that may be hazardous to humans to perform [73][74][75]. Formation in a multi-robot system is defined as an overlapping physical structure, where the system is maintained in tightly pre-defined restrictions and provisions. In addition, it allows the capability to transfer big objects compared to a single robot and reduces the total effort and time needed to explore a large area and mapping [73][76]. On the other hand, formations of multiple robots are known as sets or teams of mobile robots, which have been structured and established to maintain some predefined geometric pattern, which is adapted to environmental constraints through controlling orientations and positions of every individual robot in the team, whilst allowing the team to move toward the specified target at the same time [43] [51][73].

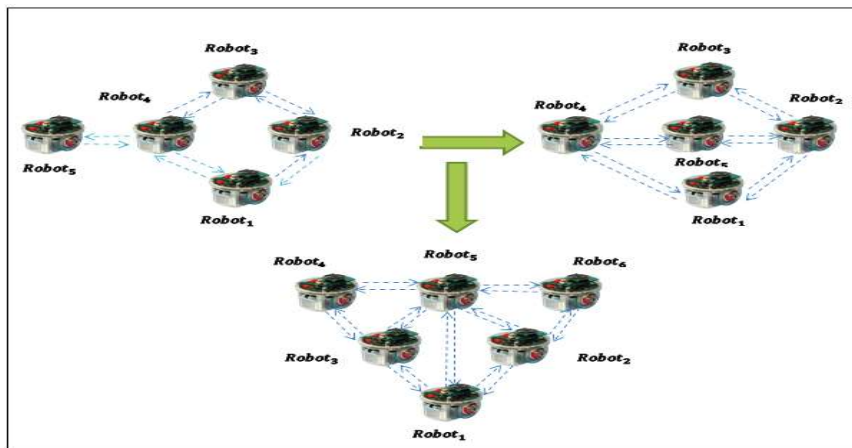


Figure 2.8: Example of formation control for multi robot system

Moreover, many formation control strategies have been developed to perform certain tasks via various ways [75]. One of the formation control strategies is known as scaling which allows the volume of the formation to increase or to contract. Another strategy controls the formation to move and rotate, where the formation is considered as a rotational constant [77]. There are other strategies which just require each robot to ‘sense’ particular aspects of other robots. In addition, control strategies that allow communication are more flexible but are also mostly more complex to implement [22][47][76]. Technological progress has also led to studies and research to improving these strategies for formation control to allow a team of robots to move in the working environment whilst keeping the formation and capability to change the formation scale if necessary. These strategies that allow the formation scale to alternate if needed, have a great practical importance when the previous formation may not suffice. For instance, if we have a team of autonomous robots moving inside a certain factory to perform a specific task, the team may require maintaining a particular configuration form, but it may also need to change the shaping scale due to the presence of obstacles [74][76]. Formation control

is a very well-studied problem, and several different approaches can be found in the literature [78]. In addition, the formation problem is defined as the coordination of a team of robots to get into and to keep a formation with a specific form. Application fields of formation control include security patrols, search and rescue operations, remote terrain and space exploration, landmine removal, area coverage and reconnaissance in military missions, control of arrays of satellites and UAVs [79]. One of the advantages of formation control is that a group of mobile robots can be used in order to accomplish and perform various tasks in a single time-span or at the same time, which cannot be achieved or are impossible to be completed by a single robot [47][73][75]. These capabilities require solving some important issues to successfully accomplish formation control for a team of mobile robots:

- How to change the pattern of formation automatically and adapt to an unknown environment.
- How to create and prepare transformation of formation during robots' transit.
- How to control and maintain formation pattern during movement.
- How to design any kind of formation shape and achieve it.
- How to change the motion plan to avoid obstacles.
- How to alter the form of the formation [51] [43][73].

All the above issues can be solved with the use of graphs which will be discussed in detail later in Chapter three.

## **2.7 Centralised/Decentralised Approaches**

In multiple robot system, the control of a multi robot team is a challenging issue. There are two strategies to this issue: Centralised and decentralised [80]. In addition, the main decision to be made when determining a team architecture of robots is whether the system is centralised or decentralised [6][61]. In centralised control strategy of multi robot system, the global information about the case of the entire system is maintained. The system collects information from all robots and saves track of their location in environment [80]. This strategy has an agent or leader that is in accountable of organising the act of the other robots; the leader is in charge for the decisional operation with the participation of the entire team, while other members work according to directions given by the leader [43][56][57]. All the decisions are made by the leader to define if a computation mission is to be executed by the leader or via a specific robot [57]. A leader has global information about the environment, and whole information about the robots. The leader can communicate with all the robots to share information [6]. In addition to that, this type of the control strategy can be applied to a fixed group, or it may be employed in



a hierarchical structure, where the robots can operate under the control of a leader, and they may act as leaders of subgroup of robots themselves in an MRS [43]. The leader then arranges a group of robots to reach a common target, plans missions for single group members and supervises the entire process. In a multi-robot system, it is possible to allow more than one member to acquire the role of a leader during the mission [80]. Although, the centralised strategy is relatively easy to design, it is not robust to communication failures and unexpected positions. The centralised strategy is appropriate when the number of robots is small, and the environment is known and constant. However, this strategy is ineffectual with large number of robots due to the high computation requirement of the leader, and the communication cost between the robots [6][43] [56] [80]. The centralised strategy has been applied in autonomous logistics, traffic control, and transportation systems in hospitals [56][80]. Moreover, this control strategy suffers from scalability problems and a drawback that it relies heavily on communication. Therefore, when a communication failure occurs, this may lead to the failure of the entire system [79][80]. On the other hand, a centralised strategy is an approach that depends on the leader which means that with the failure of the central unit, the system cannot accomplish their tasks [56][65][80]. A further classification can be introduced for centralised system based on the method that the leader role is played, which are strongly centralised or weakly centralised [43][56]. In strongly centralised systems, a constant leader is employed (which can be a robot or some remote server) which stays the same during the whole mission duration. It is possible that more than one member in the multi-robot system is allowed to obtain the role of leader, and they can plan the actions of other robots. However, in the case of strongly centralised systems, the role of a leader is assigned to a one robot at the beginning of the task. It remains the same until completion of the whole task [43][56]. Weakly centralised systems: in this approach more than single robot is permitted to become leader during the task. A leader is not selected in advance, but the role of the leader is dynamically assigned during the execution of the task based on some criteria, depending on the environment changes, communication, or forced by the failure of the current robot leader. There can be many policies to choose a leader such as some pre-set priorities, calculation force, etc. [43][56]. In addition, if there are multiple leaders and all of them are finally controlled by a single robot, these approaches are also classified as centralised. If these leaders work independently, and are not controlled by one single leader, then it is called hierarchical. Also, in several states, multi-robot system does not follow fully centralised or decentralised approaches [56]. Generally, both weak and strong, permitted for a simpler mission assignment among the team member because just one of them, the leader, is in accountable of it. Strongly centralised systems have the drawback

that they strongly depended on communication. Hence, when a communication failure takes place, it results in the failure of the whole process of the system. In addition, it can fail in accomplishing its task, if the leader is broken or when its leader goes out of order, and it has an advantage that a well-suited robotic agent can be realised to be the leader, for example, by having the appropriate computing capabilities to analyse the environmental data used to take decision. Weakly centralised systems attempt to recover from a leader failure by choosing a new leader. Therefore, weakly centralised systems are more robust than strongly centralised because it can choose a new leader in state of leader failure [43][56].

By contrast, decentralised control strategies consist of robotic agents that are completely independent in the process of adopting decisions, where each robot is an autonomous unit that operates according to its position in the team without a leader. Besides that, the decision-making is done with the participation of all members [56][80]. In this approach, the computation mission is executed by any single or multiple robots in the group. There are no certain leaders throughout the mission and hence it permits all other robots to proceed with their account even if one robot fails [57]. Decentralized systems can better respond to changing or unknown environments, and often have more superior robustness, flexibility, adaptability, and reliability. However, the solutions they reach are usually sub-optimal [6][43]. In addition, the decentralised strategy is extremely robust, and it can perform very well in complex and hazardous environments. This strategy is scalable; potentially many heterogeneous robots can collaborate to reach a common goal [56][79][80]. Besides, the decentralised approach is currently a dominant paradigm, and the behaviour of this approach is usually described employing such terms as “self-organisation” and “emergence” [61]. Decentralised multi-robot architecture is typical in swarm robotics. Often, robotic swarm contain several simple robots that have primitive behaviour [80].

Furthermore, this strategy is designed as in either a distributed way or a hierarchical way. In the first category there has no central control agent and treats all agents equally in terms of the communication and control: all the robots are equal with respect to each other and are completely autonomous in the decision-making process. Therefore, if one of the agents stops working, the other agents can still complete their task [6][43][56][61]. Although, the distributed architectures produce good robustness to the failure by allowing each robot to take decisions dependently, but many intricacies come to achieve the coordination among robots [56].

In the hierarchical design, robots are formed as small local groups and leaders are distributed to perform their tasks [56][74][81]. In addition, there exist one or more local central control agents which organise robots into clusters. The hierarchical architecture is a hybrid architecture,

intermediate between a centralised architecture and a distributed architecture [6]. When the operation of coordination is locally centralised, it is called hierarchical. Also, the MRS has local leaders, but they are not eventually controlled by single leader. Such approaches are generally used in MRS with multiple missions where a team of robots work on different mission, or a mission is divided within the robots' teams by negotiation, but, not by commands from the leader or the central system. This type of approaches is less robust than distributed, but it can be realised just with local communication or global communication with less complexity and cost [56]. Despite advances in the techniques for coordinating movements of multiple robots in the formation by the employment of decentralised control, decentralised approaches that are mostly studied are Leader Following, Behaviour-based, and Virtual Structure, where the approach of the Leader Following is the most studied in multi robot systems [73][82][83][84][85]. Tools from algebraic graph theory have been used for the study of the leader-follower system, where the followers are governed by the Laplacian based feedback law [49]. Figure 2.9 shows a leader-follower network with robot-followers indicated as

$V_f = \{R_2, R_3, R_4, R_5\}$  and robot-leaders as  $V_l = \{R_1, R_6\}$ . A good review of these approaches can be found in [49][75][85].

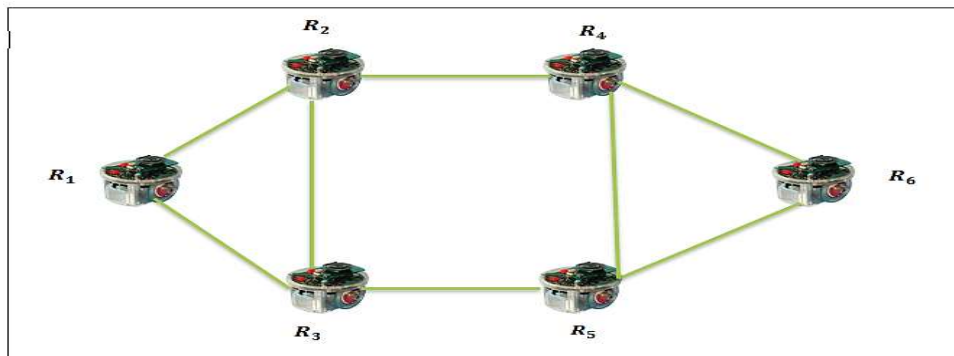


Figure 2.9: Example of a leader-follower network

Moreover, decentralized control strategies are the most popular approach for teams of MRS [44]. Recently, decentralised strategies have become more prevalent, where sensing hardware and communication have become easier to execute on groups of agents and less expensive [79]. In addition, these control strategies can accomplish a range of tasks including the location rendezvous [79]. What is more, when designing decentralised control strategies for multi-agent systems, it is important to have information about communication abilities of the agents and the sensing ranges. In addition, it is often very useful to model a multi robot system as a graph-

theoretic framework where each robot corresponds to a node, and each information link corresponds to an edge [79]. In addition, one important feature is that the latest current research addresses the combination of graph theory with decentralised controls [86]. This graph-theoretical approach is discussed in detail in Chapter 3. On the other hand, the decentralised system means that its every unit (robot) has the following attributes:

- Limited sensing/communication (information gathering);
- Limited computing power (information processing);
- Limited available memory (information storage).

These cases of work abilities, limited sensing, and communication that relates to the idea of decentralisation, distributed sensing (decentralised), and control, are also closely related to the concept of graphs. For example, the consensus protocol for a multi-robot system is one of the most important solutions for a team of a robot with the target to drive the entire system into a final common position. This problem has been resolved in a completely decentralised way. Good examples of decentralised formation control are shown in [6][49][59][67]. Next, in more detail, what the decentralised controller does exactly mean. For example, assumed that a team consists of five robots that represented as a graph for encoding the information flow among robots (communicated, sensed, and elaborated). The decentralisation is such that: on every edge, the volume of the information flow is constant (the number of robots). If we add the sixth robot, it does not increase the information, memory, or computing power that needed by robots (1, 2, 3, and 4). Thus, the amount of information grows linearly with the number of neighbours (see Figure 2.9) [6][67][86]. In general: decentralised approaches are more robust to robot failures, malfunctions, or communication failure [56]. The research communities have shown their increasing attention towards employing decentralised approaches for multi-robot coordination. But the cost of communication is a challenge faced by many researchers when using decentralised coordination by explicit communication. However, the utilise of implicit communication is also possible to achieve a decentralised approach. Utilising implicit communication is more scalable. Therefore, in practice, a combination of implicit and explicit communication may be more beneficial and efficient. When coordination is achieved by utilised both decentralised and centralised approaches, the coordination can be called hybrid coordination [56]. In addition, it is exceedingly claimed in previous studies that the decentralised strategy has many inherent features over the centralised strategy, involving scalability, fault tolerance, reliability, and natural exploitation of parallelism. But we are not

aware of any published theory or empirical comparison that directly supports these claims [61]. For this reason, our work in this thesis focuses on centralised control strategies, as we deal with a small number of robots [56].

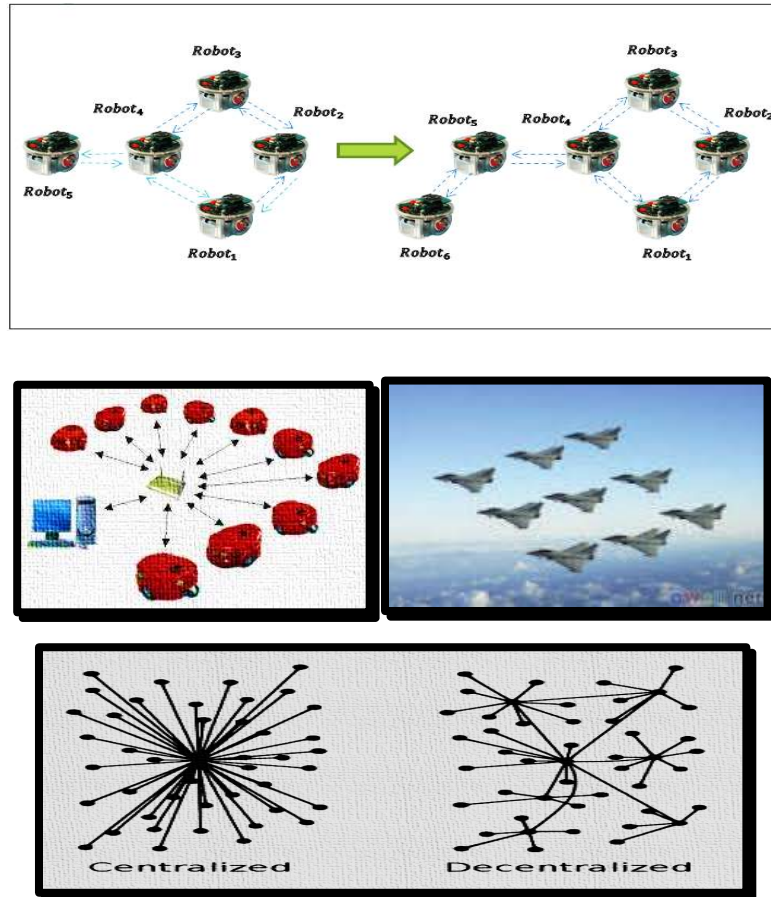


Figure 2.2: Centralised and decentralised control

<http://www.https://www.softwareadvice.com/resources/it-org-structure-centralize-vs-decentralize/>

## 2.8 Connectivity (Communication)

Communication in a multi-robot system is considered an essential and important issue, and it has received major attention from researchers recently. The concept of communication for a multi-robot system has several meanings, where this concept is defined as the process of transferring information from one robot to another robot, based on the way robots' sense or share information directly or indirectly. Robots can cooperate by communication mechanisms that enable them to share information among themselves. Communication also means any way that the robots can exchange or sense some information about each other [56]. In addition, there are several different methods to establish communication links between robots. The importance

of communication lies in the knowledge of two main parts. The first one is: how robots can communicate with each other, and the other one is: how robots can exchange information with each other to participate in performing various operations [13][87][88]. Communication can be explicit, for example, via Wi-Fi, or implicit, for instance by sensing each other by sensors, or via the environment, where the environment itself is a communication medium [6][56][89]. Explicit communication utilises additional communication hardware - a device intended for signals that other team members can understand. The robots interchange information directly utilising unicast and broadcasting intentional messages in explicit communication. Whilst, in implicit communication, robots gain information about other member robots via the environment. This communication uses staggered interaction between team members and can be gained by using specific sensors in the robot [56]. In addition, the communication structure of a team defines the possible modes of inter-agent interaction. There are three characteristic key types of interactions that can be supported: interaction via Sensing, interaction via Communications, and interaction via Environment [6][61]. The simplest, most limited sort of interaction happens when the environment itself is the communication medium, and there is no explicit communication or interaction among agents. Some researchers have called this modality “cooperation without communication” [61]. Dudek et al. in [14] have suggested a more detailed taxonomy of communication structures. On the other hand, dynamic networks have lately appeared as an effective method of modelling different shapes of interaction within a group of mobile agents, like communication and sensing. Communication is a flexible and powerful way for exchange of the information among the members of the team to perform networks’ operations [89]. These networks consist of several mobile robots that are determined as continuous dynamical systems of continuous space with communication abilities [61][88][90]. These robotic networks are assigned to perform a variety of tasks, such as search and rescue missions. In addition, information exchange in these systems can be of two types: either the effective communication or passive sensing between agents. Communication among agents may be stronger and more flexible than passive sensing. Also, it requires robots to follow a common protocol to handle communications between agents, which might be a difficult requirement because it needs communication hardware that often is wireless [61][88][91]. Moreover, communication can help robots to collaborate through learning information that is observed or inferred from others to configure the system [78][83][92]. The design of control composition algorithms is a hard task for multi-robot systems, even when dealing with a single network. Wireless communication plays a vital role in distributing these algorithms.

## **2.9 Motion Planning Issues Overview**

Motion planning is a common problem in the development of autonomous robotics, which has been addressed in many types of research. It also has importance and great value not just in robotics but also in various fields such as maintenance planning, computer-aided design, and virtual environments. Motion planning comprises sensor-based planning, configuration spaces, decomposition and sampling methods, matrices computation and their properties, and advanced planning algorithms. The main function of motion planning is to produce a continuous path that links a start location (S) to a goal location (G) where the robot movement is free from collisions with surrounding obstacles [21][93]. In addition, it is an ability to build collision-free paths that connect robots to their target destinations. Moreover, motion planning is one of the important tasks in the intelligent control of autonomous mobile robots [94][95]. With a technical standpoint, the problem of motion planning is to properly define a path for a robot in a specific environment from a start position to a goal position while avoiding a collision. Or, in other words, the motion planning problem in its simplest form is how the robot moves from an initial configuration to the other configuration until it reaches a target configuration in the optimal path without collision. To describe the problem of motion planning in a correct and simple manner, this requires knowing and identifying two commands. The first one is what information is possessed by the robot to do its mission in a workspace environment, and the second is what capacities possessed by the robot to move inside the environment of the workspace without any collision with known or unknown obstacles to reach the goal [21][93][96][97][98]. Often the motion planning of the robot is separated into two categories: one category is trajectory planning which is aimed to schedule the motion of the robot along the planned track in a workspace, and the other category is the path planning that guides the robots to find the optimal path among two points in a workspace [99]. Furthermore, the motion planning is related to several expressions or terms such as Path planning, Trajectory planning, Navigation, Global path planning, and Local navigation [21].

## **2.10 Multi-Robot Motion Planning**

In a multi-robot system, to find paths (shortest) for a team of robots that are operating together in a shared workspace that allows robots to move from their initial position to the goal positions whilst avoiding collision with obstacles and with each other, it must be taken into consideration that MRS forms itself a graph. This graph will be dynamically changing whilst robots progress to their goals, where the robots and obstacles are geometric entities, and the robots operate in the configuration space. Furthermore, each robot has its own start and target positions, and it

computes its own movement based on the information at its disposal to move along its path whilst avoiding mutual collisions [87][100]. The motion planning problem of a multi-robot system can also be classified into two categories: centralised and decentralised, where in the first category a multi-robot system is considered as a single robot system instead of several autonomous entities. Within this category, a single robot (a leader or master) or a central based station has all information of the whole system and control all robots, as mentioned above in section (2.7). Some algorithms based on randomised sampling have used this type of approach to improve system performance. The centralised motion planning often addresses coordination via setting the robot speeds along their respective paths, where the path of each robot is calculated independently, and it uses the sampling algorithms to coordinate previously built paths [6][87][100]. The introduction of algorithms based on sampling techniques such as the roadmap method had a major effect on the domain of motion planning because of their simplicity, efficiency, and applicability to a wide area of issues, such as the multi-robot systems case [95], which we will use in this thesis. Whereas a multi-robot system in the second category operates in a distributed and independent manner and more rapidly than in the first category [6][88][96]. In addition, the decentralised category is considered more common than the centralised one [6][87][100].

In fact, in order for a multi-robot system to collaboratively achieve a certain mission in a common workspace, one of the key missions for each robot is to reach its single target without colliding with a static obstacle or another robot. In addition, in an environment of a multi-robot, path-planning or collision avoidance is an important problem. The main important problem that faces robots when they move in the environment is that they must consider the presence of obstacles, and any dynamic objects such as a moving robot [100]. The collision avoidance problem has been widely investigated for a multi-robot system in the literature and many different approaches have been used, with typical solutions relying on the possibility of avoiding collisions with obstacles and between robots [78]. The developed algorithms for collision avoidance can help avoid the collision from any static obstacles and inter-robot collision avoidance. Besides, path planning and collision avoidance with polygonal obstacles in most of the known methods involve the use of graph search algorithms to find the shortest path from the possible paths [101]. Moreover, the collision avoidance among robots is gained through exploiting tools from the graph theory such as the properties of weighted graphs, edge-weight functions, the Laplacian-based algorithm used for graph connectivity to represent the link between the robots [78]



## **Chapter Three**

### **3 Graph Theoretic approach to the study of multi-robot systems**

#### **3.1 Introduction**

Graphs and algebraic graph theory are fundamental and strong tools to study and analyse stability of the formation control involving all above-mentioned aspects in Chapter two. Graphs are used to control a group of mobile robots in keeping a desired formation and altering formations and providing information exchange when needed while navigating in an environment with obstacles. They can help accomplish transformations between different formation patterns [55][71][75][102][103]. The use of graph theory has been highly advantageous to determine relations among the individual robots, because of its algorithms and definitions that have been developed to find minimal constraints and rigid structures [76]. Besides that, directed graphs have been used to describe the topologies and configuration patterns, where the description of graph topology has been used for stability analysis of the controllability of robot formations and for selection of suitable controllers for certain formation patterns [74][104]. The problem of formation control is handled by exploiting strategies that are based on graph theory. For example, delays in the communication channels may make the system unstable, for this reason, the formation control strategy is described on the basis of weighted-edge graphs, where the weighted edge has been used for both of formation control and collision avoidance [102]. The problem regarding formation modelling has been studied at the beginning of this century [74]. Moreover, some aspects of the formation control of a team of mobile robots can be improved by taking advantage of algebraic graph theory. For instance, weighted graphs properties can be used to obtain a formation shape and avoid collision among robots moving in the environment [22][105]. According to Desai et, al. in [103], directed acyclic graphs have been adopted to represent the control graph between mobile robots, and for the design of the control strategy. In addition, it can employ graph algorithms by adding a specific geometric pattern, to maintain formation generation and control the consensus problem. Furthermore, the tools of algebraic graph theory have been used successfully for formation control, which aims to drive the whole system (consisting of  $N$  robots) to a common final state, where locally distributed and scalable formation controllers can be designed using properties of the Laplacian matrix [22][74][105]. Moreover, graph theory in both classifications, geometric and topological, has a fundamental role in the design of control algorithms [89]. Additionally, the communication architecture among robots often relies on the approach of modelling systems as graphs, where every robot is represented as a node (vertex) of a graph and the communication links between robots are represented edges of the graph, which are

defined according to the predetermined communication model. Graphs have been used broadly to study sensors and robots' networks. They have also been utilised as models of wireless communication [84][90][106]. For instance, the connectivity of the communication graph is a property that enables coordination between robots; communication has an essential role in keeping formation or coordination among robots. That means, to perform a task that requires keeping formation successfully, the robots should ensure that the communication graph is connected, for the exchange of information between the robotic systems [55][74][103][106]. Also, there is a complex interplay among communication and mobility because the communication network is occasionally subject to change, each time a robot moves. Besides, robots need to coordinate their movement, and determine the right place to move to perform their tasks correctly which ensures information exchange. All these can be achieved via communication. Where the communication lets the robots share information to find which robots are nearby, and to assess their ability to transmit information to other robots [51][106]. Normally, understanding complex systems such as multi-robot systems require a bottom-up analysis, i.e., how these systems are connected, and their interactions are best characterised as networks whilst maintaining the desired formation and changing formations during navigating an environment with obstacles.

### 3.2 Graphs approach for a multi-robot system

Multi-robot system can be represented as a graph, where each robot can be considered as a vertex of the graph, and the communication structure between robots can be described as an edge of the graph [61][102], see figure 3.1. A graph  $(G)$  represents a linkage between a set of vertices, where vertices relate to each other, and these vertices can be connected if there are paths (edges) between them.

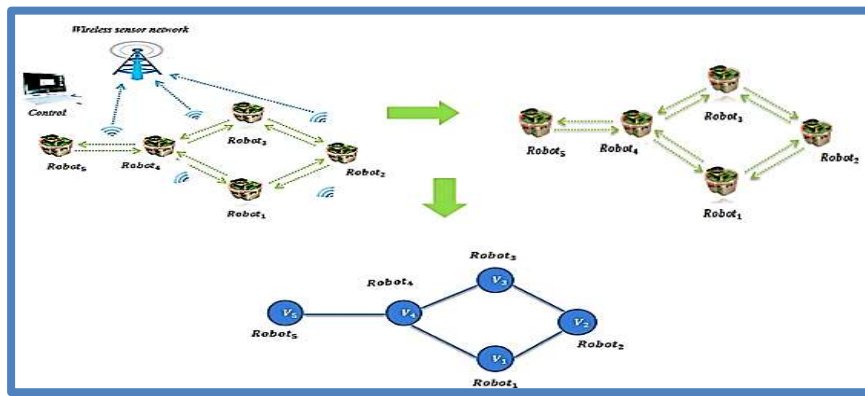


Figure 3.1: Example of a multi robot system represented as a graph

All vertices that are near a vertex are defined as a set of its neighbours which relate to others by edges. Therefore, the link between any two robots is represented as an edge of a graph. In addition to that, this system can be adapted to various sorts of graphs for communication architecture such as directed graphs and undirected graphs. Many types of research are using undirected graphs in robotics systems because it is easy to deal with information exchange between the robot and their neighbours. There are two models of undirected graphs, weighted graphs that consist of weighted edges and unweighted graphs do not have weighted edges [5][102]. According to Coogan, S, many different graphs can be associated with multiple robots (agents)[47]. Example are sensing graphs where each sensor encodes to what the robots (agents) can measure, or sense, i.e., relative speeds and positions of other robots inside its sensing graph without active communication, as well as detection of obstacles and boundaries [47]. Also, the communication topology between robots can be modelled as a graph, commonly referred to as the communication graph, and communication links can be established as unidirectional (moving or operating in a single direction) or bidirectional (operating in two directions), (see Appendix 8 for more information) [102]. Hence, the problem of ensuring the exchange of information in mathematical terms means that the communication graph should be connected. Graph theory is used to encode the information flow among robots, also it is an important tool in the analysis of the stability and control of robot formations [47][102]. There is an example of the graph that is a random graph in which the communication links are modelled through random operations [47]. In addition, weighted-edges graphs are exploited to lead a team of mobile robots to establish the required shape while collisions avoiding [102].

### **3.3 Importance of graphs for multi-robot systems**

Graphs are strong tools for increasing the power of the communication links for information flow between robots for a workflow to perform their tasks. In addition, the matrices that are related to graphs have important and useful properties for development of multi-robot systems such as adjacency matrix, diagonal matrix, incidence matrix, and the last but not the least, the Laplacian matrix<sup>2</sup> which has an important role to solve problems of networked systems such as consensus protocol, flocking, formation control, and the rendezvous problem, in addition to the motion planning problem that is considered in this thesis(see Chapter four for more details). These problems have been studied widely in the field of decentralised control and attracted considerable attention in many types of research. The rendezvous issue has been discussed in

---

<sup>2</sup> Graph properties are described in detail on Appendix 8.

[84][107][108][109], where they have been explained that the problem of formation control may be expressed as consensus issues, where the formation error is determined for every robot and the aim is to stabilise the formation (i.e., control the formation error for each robot to each zero) [107][108][109]. According to Rahmani, A., Jia, M., Mesbahi, M., & Egerstedt, M, the control method for multi-agent rendezvous is Laplacian-based (see the definition of the Laplacian matrix in Appendix 8). Their study was aimed to identify effects of graph-theoretic concepts on system theoretic properties of the robotics system. They have shown how the symmetry structure of the network directly relates to the controllability of the corresponding robotics system, for more information see example in [49]. On the other hand, Fax, A. and Murray, M mentioned that they had applied the tools of graph theory to associate the communication network topology with formation stability [59]. They proved a Nyquist standard by using the eigenvalues of the Laplacian matrix of the graph to define the impact of the graph on formation stability; they also proved a separation principle which stated that if the information flow was stable for the given graph, then the stability of formation had been achieved, and the information flow become rendered highly strong among robots. A good example of a consensus-based formation control strategy is given in [59] confirming that even in the presence of delays in the communication, the agreement is reached. Moreover, a multi-agent system shows a stable behaviour, even in the presence of a varying communication topology [59]. The Adjacency and Laplacian matrices have been widely used in multi-robot system graphs (see [59][107][110][111]). Fax, J. A., & Murray, R. M, also pointed out that the eigenvalues of the Laplacian matrix could display the system's stability with laws governing the closest neighbours (see [59][110]). Jadbabaie, A., Lin, J., & Morse, A. S, have exploited the properties of the Laplacian matrix to demonstrate the convergence of boids (bird-oid objects) speed [107][111][112]. Examples of consensus protocol problems are given in [102][103]. In addition, Falconi, R., Sabattini., Secchi, C., Fantuzzi, C., & Melchiorri, C, stated that the problem of the consensus protocol for multi-agents (robots) could be solved in a totally decentralised method with the Laplacian-based feedback way. They described a consensus-based algorithm for the formation control of teams of robots through the definition of suitable edge weight functions, where they proved and achieved formation control and collision avoidance between robots by exploiting tools of graph theory, see examples in [78][113]. Zavlanos, M. M., & Pappas, G. J, pointed out that the controlling network problem of robots (agents) can be solved using an algebraic graph theory, where the condition of the connectivity has translated through the dynamics of a Laplacian matrix and its spectral characteristics [114][115]. In addition, in distributed networks, the effectiveness of collaboration like networks

and multi-agent systems is based on vertices' capability to exchange information. The availability of different communication protocols with diverse technical characteristics opens the possibility to guarantee connectivity during a system's operation in any condition [116]. Communication can be represented by a graph, in which connectivity can be expressed by a well-known algebraic connectivity value or Fiedler value, called the second smallest eigenvalue of the Laplacian matrix, which is indicated by  $\lambda_2$ , and is a measure of connectivity of the communication graph [115][116]. It is one of the most substantial tools employed in several applications that require maintaining connectivity. This value is determined by the graph topology and the parameters of a graph, for instance, the number of vertices  $n$ , number of edges  $m$ , minimal degree  $d_{min}$ , etc [116]. Also, for many common graph topologies, such as cycle or cube graphs, and complete graphs, the algebraic connectivity is known and can be defined by the number of vertices  $n$  in the graph, such as, the algebraic connectivity for a cycle graph:  $\lambda_2 = 2(1 - \cos \frac{2\pi}{n})$ , whilst for a complete graph  $\lambda_2 = n$  [116]. However, the graph topology can be changed by allowing processes on the graph, which may involve the addition or removal of vertices and edges. Thus, in an incomplete graph, the maximal value of the algebraic connectivity is upper bounded by the graph parameters (i.e., the upper-bound of for  $\lambda_2$  an incomplete graph with  $n$  vertices is defined by  $\lambda_2 \leq n - 2$ , whereas the bound related to the minimal degree is indicated by the following inequality:  $2d_{min} - n + 2 \leq \lambda_2 \leq \frac{n}{n-1}d_{min}$ ) [116]. Furthermore, second smallest eigenvalue is usually used to catch connectivity of dynamic networks. For a weighted graph  $G = (V, E, W)$  the entries of the Laplacian matrix (L) are often related to the weights in  $W$  so that the  $i, j$  entry of L is indicated by  $[L]_{ij} = \sum_{j=1}^n w_{ij}$  if  $i = j$ , and  $[L]_{ij} = -w_{ij}$  if  $i \neq j$  [90][115]. Fiedler, in 1973, has recognised the importance of  $\lambda_2$  in connectivity properties of the graph and called it algebraic connectivity because of its connection to the vertex and edge connectivity: correspondingly the number of vertices or edges that require to be removed to disconnect a graph. Also, the properties of the graph Laplacian for connectivity and partitioning of graphs are important, specifically on the number of connected components, and measures of how easy it is to partition a graph into two disconnected subgraphs [115] [116]. Some of the key concepts in the algebraic graph theory which are used in this thesis are summarised in Chapter 4 and Appendix A. All these studies have demonstrated the importance of graph theory for the development of robotics systems.

### 3.4 Path planning problem

The path planning problem is a case of motion planning problems, which is still an open problem to be studied widely. It is an important problem that covers a wide area of robotics research. Path planning plays a main role in enhancing robotic navigation systems in both dynamic and static environments. The static environment consists of only static obstacles in the domain of the workspace, whereas the dynamic environment contains both dynamic and static obstacles in the workspace domain [21][35][117][118].

Figure 3.2: Example of a workspace in a static environment

Figure 3.3: Example of a workspace in a dynamic environment

In addition, path planning is still one of the challenging problems in the field of robotic applications [35][118]. Furthermore, path planning is essential for robots in a multi-robot system to find a safe route to be traversed from the starting point to the goal point, which does not result in collision with any obstacles based on the amount of the information available about the environment that may be partially or completely known or unknown, the approaches to path planning vary considerably [21] [35][117][118]. Path planning in multi-robot systems has been often tackled as an optimisation problem focused on finding the shortest collision-free path [119]. Additionally, the design of path planning is a key topic that has gained extensive attention in the field of mobile robotics. The main feature of a multi-robot system is the capability to plan its own motion to perform specific tasks. For instance, in the path planning

problem, a collision-free path is calculated for a robot to move from a start location to a goal location between static obstacles; here the task of robots is to avoid a collision and to reach the target destination. In addition, mobile robots can navigate by themselves with a perfect path planning system without human intervention to access their goal destination. The performance of path planning can be described via several algorithm properties such as optimality, speed, and completeness [117][118]. Moreover, path planning indicates to the computation of the path that an object must follow in navigation from a starting point to a given destination/configuration. This is considered a principal problem in robotics, since the moving object may be a robot itself, a part of it (e.g., its robotic arms) or an object being carried by one or more robots [120]. The key stages in path planning are choosing a suitable map representation for the application and decreasing the robot to a point-mass that allows planning in the configuration space.

### **3.4.1 Path planning classification**

The problem of the path planning can be classified into two classes, depending on the range of knowledge of robots for all information around the surrounding environment. Class one is local path planning, and the second class is global path planning [117][118][121]. A local path planning requires the robot to move in both static and dynamic environments [122]. This path planning method is employed for solving the problem of robot path planning to avoid obstacles in a real-time environment via sensors measurements to obtain information for the location of the robot in the workspace, and the form and size of the obstacles in a partially known environment [123]. After that, this information is used to proceed with local path planning. The essential point here is that the first step is to sense obstacles in the environment and then define a collision-free path [124]. In the local path planning problem, algorithms used for the path planning work in response to the change of environment whilst a robot is moving with the objective of finding an optimal path. In contrast, a global path planning needs total knowledge of the environment and, so a collision-free optimal path is created within the environment to move robots from initial location to goal location before the robots start moving [35][118][121][125]. This means, all robots have prior information about their work environment such as location of the obstacles and targets. Many researchers have processed the problem of global path planning and have found some suitable solutions such as Rapidly expanding Random Tree (RRT) methods, and probabilistic roadmap (PRM) [117][118][121]. An early model of the global path planning is Piano's Mover problem which is studied widely [121]. Even though details of path planning algorithms are totally different, most of them



follow a common framework for finding of the shortest path between the start and the goal points [35][99][118]. Furthermore, an important consideration for algorithms of the path planning is completeness. The first stage determines a map for robots, which may have a complex geometric form, in the configuration space [35][99][117][118][126].

### **3.4.2 Path planning approaches**

In practice, different approaches have been introduced to execute path planning for robots. Such approaches rely on the environment, type of sensor and robot abilities. In addition, they gradually contributed for improve performance in terms of distance, cost, time, and complexity [121]. In general, the problem of path planning is to find a path joining some points for collision avoidance with obstacles in a workspace. The map of the workspace can be transformed into different types of search spaces, to reduce the search space size. A search space is established by roadmap methods, cell decomposition methods, or artificial potential field methods. The roadmap method transforms the workspace into a set of vertices and paths that enable a graph search [118]. Besides, most methods of path planning consist of two-stage operation to establish collision-free paths. Stage one is called the pre-treatment stage. In this stage the environment is represented that includes objects/obstacles. Then it defines free spaces which introduces workspace ( $W$ ) in dimensional space with a graph that we call the environmental graph, where the vertices and edges are established in  $W$  with considering the configuration of robots ( $R$ ) and obstacles ( $O$ ). In addition, the concept of configuration space ( $C$ -space) is applied to represent  $O$  and  $R$  in  $W$  [21][35][117][118]. Typically, in the  $C$ -space, the size of robot is reduced to a point, while the size of obstacles is enlarged according to the robots 'size. These techniques are used to build a so-called roadmap from the environmental graph (see subsection 3.8 for more detail on a roadmap), and each technique uses a different way to determine vertices and edges. The second stage is the query stage. At this stage, the first position ( $S$ ) and the goal position ( $G$ ) are integrated into the roadmap if it is present, or into the environmental graph. The path is then calculated in the represented environment by using one of the graph search algorithms [21]. Moreover, the workspace  $W$  is defined as the world that consists of obstacles that occupy some space in the environment and of a free space, where a robot works in, or all points that a robot can reach (i.e., the space where robot can move in, which are in Euclidean space 2D or 3D) [21][72].

### **3.5 Configuration space (C-space)**

To plan a robot's motions when there are several degrees of freedom, a construction called a configuration space is used. A configuration space ( $C$ -space) represents each possible

configuration as a single point and contains all the possible robot's configurations (or a C-space is the space containing all possible configurations of a robot and obstacles region in  $W$ ). It is used to ensure that a robot does not collide with obstacles  $O$  in  $W$  [21][72][118]. Lozano-Perez [127] proposed that instead of dealing with a complex geometrical representation of a robot in the Euclidean representation of the workspace, the robot can be considered as a point in its configuration space [118]. Path planning led to the development concept of the C-space through polygonal obstacles. The concept of C- space is a key formalism for robot motion planning that allows determination of positions of robots and the obstacles. A robot configuration  $C$  is a specification of the physical state (the positions of all robot points) relative to a fixed coordinate system (fixed environment frame) [21][72]. Generally, the common concept behind most of the path planning methods to represent  $W$  is C-space [21]. In C-space the configuration of a robot is a set of parameters that especially determine the position of each point in the robot. The position of a robot is represented as a configuration illustrate the form of the robot, and the motion is described as a path in configuration space [21], whereas the robots and its environment (e.g., the geometry of the obstacles), are represented in a (2D or 3D) dimensional workspace. This means that the configuration space is the area of all possible specification of obstacles and robots in  $W$  [21][128].

### 3.5.1 Advantages of C- Space

The advantage of using the C-space in path planning is to ensure that robots do not intersect with obstacles in  $W$ . It supplies a uniform framework for path planning, and it represents an essential formalism and substructure that allows evaluation and comparison of diverse algorithms [21][129]. In addition, one way the configuration of a single robot may be described is by a moving point which means the robot has zero size. Besides, if a robot moves in a two-dimensional space, the configuration can be expressed using two parameters, or in other words, its position can be represented as two coordinates  $(x, y)$ , which means that the configuration has two parameters. If the robot moves in a three-dimensional workspace the exact position can be represented as a configuration that has three parameters  $(x, y, \text{ and } z)$  [48]. The planning problem increases exponentially with the number of dimensions in the configuration space. For this reason, the possible configurations are often reduced to simply the  $(x, y)$  coordinate space for global path planning. See Figure 3.4 for an example of workspace and configuration space [130].

Figure 3.4: Examples of workspace and configuration space

<http://ais.informatik.uni-freiburg.de/teaching/ss11/robotics/slides/18-robot-motion-planning.pdf>

### 3.6 Classifications of robot path planning methods

The mission of the mobile robot path planning is to find a free-collision path by an environment with obstacles, from a given start position to a desirable target destination whilst satisfying certain optimisation criteria. Path planning method may be divided into various types depended on different situations and based on the environment where the robot is in [121][130]. One is an online type: path planning in a dynamic environment that has both dynamic and static obstacles in the map. Second is an offline type: path planning in static environment that just consists of the static obstacles in the map [118]. In addition, each type could be further classified into two sub-sets based on how much information the robot has about the surrounding environment, as follows: (1) path planning in a partially known or uncertain environment as the robot explores the environment using sensors to obtain the local information of the position, form, and size of obstacles, and then use the information to proceed local path planning; (2) path planning in a clearly known environment as the robot already knows the position of the obstacles prior it starts to move. The path for the robot may be the global optimised result because the whole environment is known [118]. In fact, this thesis focusses only on the offline kind, which means static and known environment. Because in the static and known environment, the robots know the whole information of the surrounding environment before they start travelling to execute their tasks. Consequently, the optimal paths can be calculated offline before the robots moves. In addition, the techniques of path planning in this environment are relatively mature such as roadmap method, which is used for static environment. Figure 3.5 illustrates the hierarchy of a classification model of Robot path planning [118] [121]

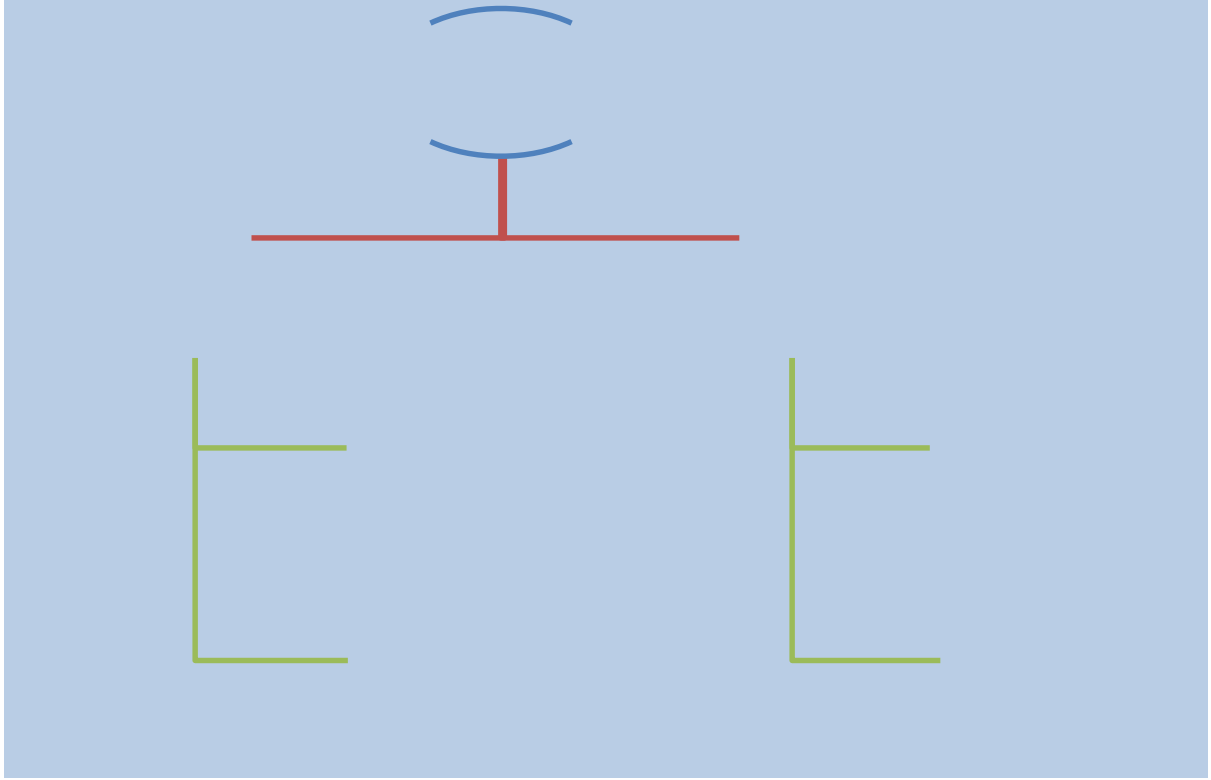


Figure 3.5: Classification model of robot path planning method [118][121]

### 3.7 Techniques of path planning methods

Although techniques of path planning for a mobile robot system have become more efficient, the sampling-based motion planning ways approaches become more successful because they depend on the idea of either resolution or probabilistic complete-ness [128][130]. Path planning methods for a clearly known static environment includes three classical techniques that can solve the problems of robot path planning in the C-space [118]. Most planning methods employ one of these three techniques to represent the configuration space (C-space), which are roadmaps, cell decomposition, and potential fields. Figure 3.6 shows the techniques representations of Path Planning Methods [21][118][128][130]. From these three methods, roadmap is one of popular approach to find a path for a robot, and it is widely used for path planning problems. In our work, we will consider roadmap method for solving the motion planning problem.

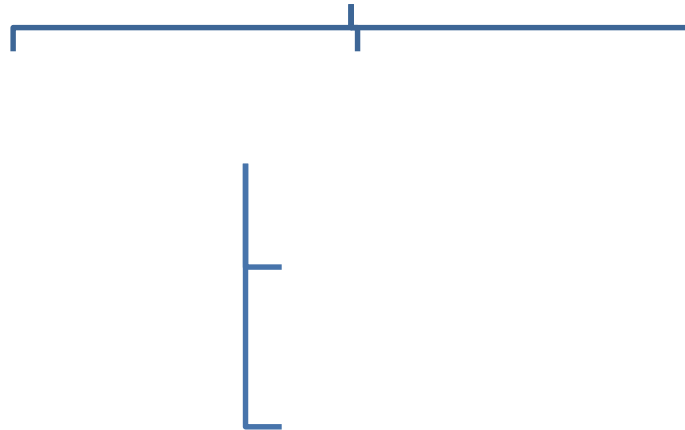


Figure 3.6: Path planning techniques representation of path planning methods

### 3.8 Roadmap

Roadmaps are graphs that represent how to move from one place to another [118]. This approach consists of constructing a graph whose vertices are collision-free configurations. In addition, the roadmap is an extension of the skeleton approach, and it is a collection of paths that allow for effective navigation in C-space [131]. The idea behind the roadmap approach is to find an optimal path in D-dimensional configuration space whilst maintaining the connectivity of robots in the free space. Based on the roadmap, a collision-free path from start to the target configurations is determined by graph search algorithms. In addition, the roadmap approach provides solutions to a multitude of scientific problems like the motion planning problem. Also, roadmap methods are fast and most of them are easy to execute and have useful and valuable features [21][118]. Besides, it is the best approach as the convergence speed is known, and it does not have any problematic heuristics [21][132]. This approach is probabilistically complete which means that if there is solution it will be found with high probability in bounded time. Furthermore, the algorithms that utilise random sampling techniques are notably used to construct a roadmap for mobile robots. Due to the fact that these algorithms work well in practice, most of the studies in this area addressed the problem of motion planning for a single robot between static obstacles [129][132][133][134]. There are different types of roadmaps that are suggested to achieve this task, such as visibility graph, Voronoi diagram or retraction approach, and silhouettes [128]. Though these methods are

different, they have one thing in common: they try to convert the free space from the workspace to a graph representation (e.g., a roadmap) [21].

### 3.8.1 Voronoi diagram

A Voronoi diagram (VD) is one of the most important and popular methods for generating a roadmap from a C-space. This diagram is defined as the collection of points equidistant from two or more objects, see Figure 3.7 [135]. It can be built as the robot enters a new environment. It creates roadmaps that joint the start and target configurations by shaping paths containing line segments and parabolic arcs (for polygonal obstacles) that maximise the clearance among the obstacles and the robot. In addition, this diagram is a set of locations in the plane that is a collection of areas that divide up the plane, each area corresponds to one of the locations and all the points in one area are closer to the area representing the area than to any other location [35][118]. The Voronoi diagram partitions the space into regions, where each region contains one object. For each point in a region, the object within the region is the closest to that point than any other object. Also, VD paths are far away as possible from the obstacles, hence, there is no required growing obstacle boundaries. Therefore, if a robot follows the edge of the VD, it will not collide with obstacles. This feature makes VD safe, however the paths generated are inefficient and not optimal in terms of path length. This is why VD is not considered in our study [21][33][35][118].

Figure 3.7: An example of a Voronoi diagram

### 3.8.2 Visibility graphs

Visibility graph is one of the earliest roadmap methods, which applied to 2D spaces. It is utilised in robot motion planning when the geometry of environment is known [118]. In addition, visibility graph is a main concept in calculational geometry, for a given group of geometrical objects (e.g., segments, polygons, points, rectangles) they encode which objects are visible to each other [136]. In contrast to the Voronoi diagram, the visibility graph (VG) uses the nodes of the obstacles including the starting and goal points in the C-space as the

vertices [21]. Using this approach, Lozano-Perez and Wesley proposed standard complexity  $O(n^3)$  to find a path in a C-space with N vertices [137], which was further reduced by Lee to  $O(n^2 \log n)$  and by Guibas and Hershberger to  $O(n^2)$  time in two dimensions, where n is the number of edges [138]. This technique is often used for motion planning in 2-dimensional configuration spaces in practice [21][121][138]. In computational geometry, a visibility graph is a basic concept that is used in various sorts of issues, algorithms, and structures [138]. The key concept of the visibility graph method is that if there is a free-collision path among two points, then there is a polygonal path that bends just at the obstacle's nodes. Free-collision path (in curves) can be transformed into line segments (straight line) [130]. In addition, visibility is defined as the capacity to draw a straight line among two vertices without crossing any other edge in the input of graph G, where two visible vertices are said to be unobstructed by any obstacle, and a line is drawn between them in the output of graph called visibility graph (VG) [21][137]. It is one of the techniques for path planning representing a C-space of an environment with polyhedral obstacles. Moreover, a visibility graph is a very important structure which is applied mainly to 2D configuration spaces with polygonal obstacles [137][138]. There are  $O(n^2)$  edges in the visibility graph, and it can be established in  $O(n^2)$  time and space in 2D, where n is the number of vertices [135]. The configuration space (2-dimensional) of the VG is known as a network that is established from sets of vertices (V) and edges (E) and it includes a set of polygonal obstacles (O) [129][131]. This network or VG is an undirected graph where the edge is a linear segment that connects a pair of mutually visible vertices. Also, the edges of the VG network are edges of the obstacles. In addition, the VG has many applications to solve problems of multi-robot systems, such as the problems of robotic motion planning and finding the shortest path to move robots inside an environment while avoiding obstacles [127][131]. The problem of the shortest path for mobile robots can be formulated as to how to find an optimal continuous path through the environment with obstacles without collision or intersection of their interior, and the 'task' of visibility graph is not just to find the path but to find the minimum distance free path from an initial point S to the endpoint (target) E when the visibility graph is constructed [21][127][129]. There are many examples of using the method of visibility graphs in the problem of robotic motion planning. The famous example is given by Nilsson for the Shakey Robot project; Nilsson introduced the visibility line (VL), where the graph is established depending on a planar map called the grid model [139]. This method evolved via studies of the problem of the path planning for ground robots or UAVs. Also, Thompson used the VL to establish a roadmap and applied search

algorithm to point robot to find the optimal path [138]. Lozano-Perez and Wesley suggested a VL-based algorithm to solve the problem of finding the least path (minimum path) to a polyhedral object moving from a start position to a target position between known polygonal obstacles considering the dimension of the object [127]. Moreover, Tokuta used the VL method as well, he introduced the VGRAPH method in a two-dimensional workspace. This method included a start position and a goal position for a robot in the roadmap and used a search algorithm to define the containment of the point, which included visibility of the point from the node. Where a point is joined by defining its visibility from all obstacle nodes [140]. Rosli bin Omar worked on the development of path planning for unmanned aerial vehicles (UAVs) using a visibility line-based method, and he applied this method on a single robot (a single UAV) [21][134]. See Figure 3.8 for an example of a visibility graph. In our work, we selected the visibility graph (VG) approach to the representation of the C-space of a multi-robot system as it is particularly beneficial for polygonal environments, and it is reducing the problem of motion planning to a graph search. In addition, it is a well-understood and simple method that produces optimal paths in a two-dimensional workspace [11][21][99][137]. Visibility graph method considers obstacle vertices, in the map, to be the vertices through which the robot can reach its desired position. These visible vertices have the property that a straight line connecting them does not intersect the interior of obstacles [33][35]. Additionally, this method guarantees that the robot will find the shortest path to its goal [40]. Visibility graph have been used to reduce the number of considered vertices, thus reducing the computational complexity of the algorithm [33][41]. Also, it is computationally effective, and guarantees to obtain an optimal path if there is one, optimal path means; the safe and shortest path [21][35].

Figure 3.8: Example of Roadmap visibility graph for single robot

### 3.9 Graph Search Algorithms

Once a method of representing the environment has been created, the second step is to search for the best path for this representation using graph search algorithms [118]. These algorithms



come from a wide range of applications, and they have also received wide attention, due to their importance in path planning [21][118][120].

In robot planning positions, the cost function (usually depended on cost considerations and time) that is typically used is the length of a path. Thus, algorithms for extracting this shortest path are desired to allow efficient robot navigation. Several graph search algorithms need to investigate each vertex in the graph to define the optimum path [118]. In addition, they search paths in a systematic way to ensure that the solution exists and uses the lowest number of iterations when calculating the perfect possible path and it considers each vertex (V) in the graph (G) [21][118][132]. This may occupy much time to establish a path, especially for a large environment that consists of many vertices. Therefore, there are several search algorithms to address this problem, for example, Breadth First Search (BFS), Depth First Search (DFS), Dijkstra's algorithm and A\* algorithm [21][35][118]. These algorithms plan the entire path before starting physical movement in any place [141][142]. BFS expands the vertices based on a heuristic approximation of the cost to the target [141][142], and it is a restriction of generic search in that it explores all neighbours of a selected node prior it goes deeper in the graph. It utilises a queue as its data structure to gain the restriction. But it does not define in which order to push select the neighbours of a chosen node [21][35][118]. DFS is exploring the graph differently than BFS. In DFS the deepest vertex is expanded first [21][118]. It moves forward by the graph, backtracking just when requisite whilst BFS explores close nodes prior going deeper to the far more nodes [118]. It utilises stack as its data structure to gain this restriction [35][118]. DFS progresses towards the target as fast as possible, looking for a route until it found a dead end [21]. Among the popular known and most used algorithms for path planning problem are Dijkstra's and A\* algorithms, and many variants and expansions have been proposed for these algorithms (e.g., D\*, or the jump point search) [119]. These algorithms depend on the environment where the map represents a graph and then finds a path in that graph and these algorithms just consider a subset of vertices. During a series of technological development, research-based planning has benefited significantly from these algorithms to create an advanced path, where Dijkstra's algorithm was the first algorithm that uses the best approach to find optimal shortest path [21][121].

### **3.9.1 Dijkstra's algorithm**

Dijkstra's algorithm is considered a simple and excellent one for solving path planning problems. In 1959, the Dutch computer scientist Edsger Dijkstra introduced Dijkstra's Algorithm, which he used to find the shortest path based on the cost (weight) assigned to

vertices to move from the first point to all points in the graph. All costs at the vertices are positive and are gradually calculated during the implementation of the algorithm. Also, this algorithm is considered complete, i.e., if there is a solution, it will find it [21][121][143]. Dijkstra's algorithm works by visiting vertices in the graph. It begins from the first point (node) where the path must start, while all direct neighbours of this node are marked to calculate distances to reach from the start point to the neighbouring vertices. After that, it moves from the first node to all its neighbouring vertices to check the costs (distances) and mark them to choose the least cost. Once all neighbours of the node have been checked once, the algorithm proceeds to the next node with the least cost, where it is repeatedly testing the closest vertex that is not tested and chooses one with the least cost until it reaches the target. Furthermore, there exist a variety of applications and special states where the Dijkstra's algorithm can be employed. As well as the computation of distances and direction, it also can be used for other concepts. For example, in the cases where the Euclidian distance is not the required cost, but, for instance, time is the cost [121] [143]. A commonly utilised algorithm for finding the shortest path is based on Dijkstra's algorithm and employs a full visibility graph [103]. In addition, the important condition when obstacles are static is that the path is established by a sequence of line segments linking a subset of points of edges visible, among them obstacles. Many algorithms depending on this condition have been developed; the best known is named Algorithm V Graph using a visibility graph built from the image taken on the environment, and the shortest path is defined by the method of Dijkstra's algorithm [24].

There are several different application areas for Dijkstra's algorithm; the following examples explain some applications of this algorithm according to [127].

- Routing Systems: In graphs, the Dijkstra's Algorithm is used to find the least path (shortest distance) from one vertex to another vertex. According to this result, the shortest path algorithm is used extensively in network routing protocols [144][145][146].
- Dijkstra's algorithm is applied to find directions automatically between actual sites like driving directions on websites and mapping as MapQuest or Google Maps. Often, the algorithm is used to create all the roads by calculating the distance and collecting data to get the least or shortest distance. For instance, if cities are represented as vertices in the graph and the driving distances among pairs of cities connected by direct roads that are represented as edge path costs, thus Dijkstra's algorithm can be used for measuring the shortest road between one city and all other cities [145], see Figure 3.9.

Figure 3.9: Google maps

<https://cse.buffalo.edu/faculty/miller/Courses/CSE633/Muthuraman-Spring-2014-CSE633.pdf.pdf>

- In applications of communication networks, Dijkstra's algorithm is used to solve the minimum delay path problem, which is the problem of the shortest path. For instance, in data network directing the target is to find the path of data packets that pass through a switching network with a least delay [146].
- Traffic information systems: to calculate the most efficient road, the systems use the graph theory and Dijkstra's Algorithm.
- Flight Schedule: A travel agent needs software to prepare the flight schedule for customers. Also, the agent has the right to access the database of all airports and flights such as the flight number, source (origin) airport, and destination. In addition, the flights have leaving and arrival time. Therefore, the agent needs to determine the earliest arrival time for the destination given a source airport and start time exactly [144][145][146].
- Determination of a file server: To determine a file server in a local area network, the time transferring files from one computer to another computer is considered. Therefore, Dijkstra's algorithm is used to reduce the number of hops from the file server to each other computer on this network [146].
- Robot Path Planning: Dijkstra's algorithm is used to solve different problems related to the determination of shortest paths e.g., in factories, transport, and facilities layouts [99][146].

### 3.9.2 A\* algorithm

The A\* algorithm is one of a family of graph search algorithms which is a variation of Dijkstra's algorithm in which the value of the heuristic,  $h(n)$ , is zero. This algorithm is the most popular choice for finding a path, due to the fact that it can be used in a wide range of contexts, and it

is totally flexible. The algorithm uses the actual distance from the start point as well as the estimated distance to the target point; also, it could be used to find the shortest path like Dijkstra's algorithm [21][99][118]. As a result, the total cost function at the current node  $n$  can be expressed as follows:

$$f(n) = h(n) + g(n)$$

where  $f(n)$  is the goodness of the node,  $h(n)$  is the heuristic value of the node (nearness to the goal and it is computed using information available in the roadmap), whilst  $g(n)$  is the cost from the start position to the node. The algorithm will evaluate the node in the graph for which the resultant  $f(n)$  is the best. The heuristic value is a calculation of what the straight-line distance from the current vertex to the target will be if there are no obstacles in between them [21][35][118]. However, there is no manner to estimate the true heuristic value in advance. In addition, the heuristic function minimises the total number of states/vertices required to be explored by A\*. As both forward and backward costs are employed, it thus combines the Best-first search and Dijkstra's algorithm. If the backward ( $g(n)$ ) cost is dominant, then A\* is equivalent to be Dijkstra's algorithm and the result is the shortest path from the source vertex to the target, but the search process occupies longer. This situation is called acceptable heuristic which means the estimated distance  $h(n)$  among vertex  $n$  and the source vertex does not underestimate the true distance from the vertex to the target. In the extreme case, A\* becomes Dijkstra's algorithm if the heuristic value  $h(n)$  is zero. Moreover, if the forward cost or heuristic weighting is dominant, A\* is similar to the Best-first search, and it is not guaranteed to produce a shortest path, even though the path is produced faster. A\* becomes Best-first search if the backward cost is zero. Like Dijkstra's algorithm, A\* has a priority queue that stores the list of nodes, it is complete if a solution exists, provided that the time and memory are unlimited. It will find the target point if it can be possibly found in the map or graph. In an optimal sense, A\* is guaranteed to produce a path with the least cost from a starting point to a target point if the heuristic value is admissible, which means smaller than the actual value [21][35].

Figure 3.10: A\* Search algorithm [35][118]

A\* some properties, e.g., it will always find a solution, given one does exist, and it will be complete if  $h(n)$  is not less than the estimate of how close the vertex is to the target. In addition, it is optimal in that it will provide a quicker search of any other shortest path algorithm which utilises the same heuristic if a closed set is not utilised. Therefore, the algorithm is commonly used in mobile robotics [35][118] [122].

### 3.10 Path planning comparative analysis: A\* algorithm or Dijkstra's algorithm

Typically, when we traverse from one situation to another, we look for the shortest path between two situations to reach the goal task in a timely manner. In fact, when we need to find the shortest path among two vertices, we represent it in a graph that models something like time or distance between situations, and then we introduce path-finding algorithms, namely, Dijkstra's algorithm and A\*Search. These algorithms are efficient; their applications are widespread, and they are universally applicable [118]. Hence, this makes them of considerable importance. Dijkstra's algorithm is especially the same as A\* algorithm, except there is no heuristic ( $h(n) = 0$  always), thus, it searches by extending out equally in every trend, whilst A\* scans the region just in the trend of destination. Dijkstra's is simple compared to A\*, and it is efficient enough to use for relatively considerable problems because it has an order of  $n^2$ , where  $n$  is the number of vertexes. The key drawback of the algorithm is the fact that it consumes much time, (i.e., ends up exploring a much larger region prior the goal is found), so, this makes it slower than A\*. A\* uses Best First Search, whilst Dijkstra's uses Greedy Best First Search, hence this makes A\* algorithm faster if compared with Dijkstra's algorithm [148]. It can be observed that the difference between Dijkstra's algorithm and A\* algorithm is that we add the heuristic to the cost that is used to order vertices in the priority queue. A heuristic is an intuitive concept to problem solving. It is an instinct about what makes sense mathematically and can be applied when traditional methods to solve the problem fail, it may solve a problem faster. A heuristic may be some key information about how close a vertex is to our desirable destination. Indeed, the shortest distance among any two points is a straight line. The straight-line distance from a vertex to the goal gives us a good estimation about how much farther that we may have to cross on a graph, which encodes distances as edge weights. Adding our estimate for the distance that we have left to the distance that we crossed from the source gives us a better approximation for how costly a shortest path going through that vertex may be. Thus, if we need to ensure that the path is the shortest, our estimated cost must underestimate the true cost. In addition, if a heuristic always underestimates the true cost, it is

called an admissible heuristic. Although admissible heuristics ensures that A\* algorithm will provide a shortest path (or by other means, leaving this algorithm unmodified, will always provide the shortest path), however, it expands far too many vertices. Besides, increasing the distance multiplier can dramatically minimise vertex extension, but finds expensive paths, whilst decreasing it to zero will run Dijkstra's algorithm, for more information see [149].

In general, both algorithms have the importance of their own, for example, Dijkstra's is often utilised when we do not know where our goal destination is, whereas A\* is utilised when we know the source and destination. We will assume that we have a resource-gathering unit that requires to go to obtain some resources of some type. It might know where many resource areas are, but it desires to go to the nearest one. In this case, Dijkstra's algorithm is better than A\* because we do not know which one is the nearest [148].

### **3.11 Shortest path analysis**

Basically, the shortest path problem in the graph theory is known as the problem of finding a path among two vertices in a graph G. This means that the total of the weights of its constituent edges is reduced. To understand the shortest path problem, we give practical examples of driving directions: if someone wants to go to a specific place in the city, they will determine the best route between two junctions on the route map, streets junctions, and highways. Here, the geographically defined aim is to obtain the fastest route to reach the target destination. This is similar to the problem of finding the shortest path between two vertices when referring to the graph: where the route network can be considered as a graph with positive weights with different terminologies like the best versus shortest, the weights in the graph may represent distance, estimated time, or some other cost [21][141]. Also, the vertices may represent route intersections and each edge of the graph is associated with a part of the route among two intersections. Additionally, this approach is applied in networking systems that using routing protocols: The Internet can be represented as a graph where vertices are represented as computers or network vertices, and edges represented as a direct connection (link) [141][142]. A particular path between two vertices of computer networks might be unavailable because one computer in the path might be overloaded with excessive movement or might be paused. For this reason, we consider the shortest path as the first choice, and not as the only choice. In addition, these graphs are particular in the sense that some edges are more significant than others, for example, for a long-distance travel such as motorway [21][144][145]. The difficulty of the shortest path is to find a road with the lowest travel cost from one destination to many destinations through the network. Furthermore, the analysis of the shortest path is important

due to its wide applications in the field of transport. According to Alija, A. S [141], the shortest path helps computing an optimal path, and optimal directing is the process of determining the best path to move from one place to another. This path may be faster or shorter based on how it is defined. Due to the nature of applications, there is a requirement for techniques to determine the shortest path, whether from a point of view of processing time or in terms of the needs of the memory. For example, if someone is new and does not know the place, they may face many obstacles and perhaps waste much more time in determining the endpoint. For this reason, it is very important to detect the route to the end destination in a road network so as not to face the complexity of calculating the shortest paths, or difficulty of finding an endpoint in the real route network. There are some solutions that have been established to overcome these obstacles, such as to provide a map for the area and then, after entering the starting point and the final location, it is probable to obtain the shortest path [141][142]. Besides, to solve the problem of the shortest path, there are many algorithms that can be used, which range from simple to complex such as Dijkstra's algorithm, A\* algorithm etc. [21][141][142]. The simple approach is to walk towards the target until encountering any type of obstructions, the direction then will be changed [141][142].

In fact, in problems of motion planning, the edge weight may be a distance between locations, a cost connected with the travel, or a time needed to travel. While in electrical networks, the edge weight is likely to be impedance or resistance of the edge, and the path weight is the total of the weights of each edge in the path. On the other hand, in a weighted graph the lowest weight path between two vertices is called the shortest path if this path has a minimum weight among two vertices. Additionally, the graph is weighted if a non-negative (positive) number, called the weight, is linked to each edge [72][132], by considering the problem of how to calculate the shortest paths from one vertex to all other vertices in the weighted graph.

### **3.11.1 An example of finding shortest path:**

According to Flitter, H. & Grossmann, T [142], there is someone who wants to travel from Bucheggplatz to Stauffacher in Zurich city via the tram in a short time. They applied the Dijkstra's algorithm to find the shortest distance road from Bucheggplatz to Stauffacher, see Figure 3.11.

Figure 3.11: Example of a public transport network in Zurich city [142].

There are many possible paths inside the network from Bucheggplatz to Stauffacher as shown in Figure 3.12, but whichever is the shortest path that person must start from the source (Bucheggplatz) to reach the target (Stauffacher). Dijkstra's algorithm defines that the shortest path among two areas on a given network which is the public transport network of a city (From Bucheggplatz over Helvetiaplatz to Stauffacher [BP→HE→ST] (see Fig 3.12) [142].

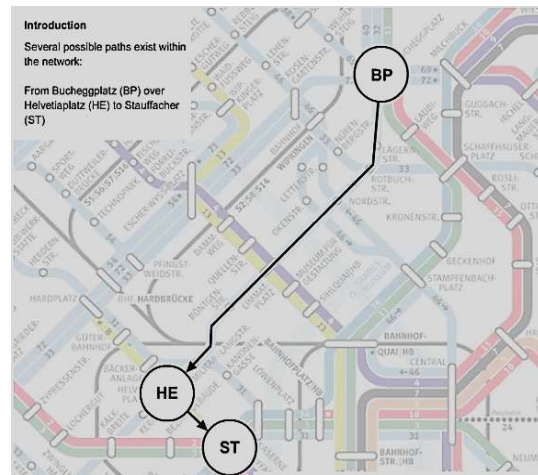


Figure 3.12: Example of public transport network in Zurich city [142].

To show how this algorithm works step by step, we illustrated in Figure 3.13 how a path can be found from the source Bucheggplatz (BP) to the target Stauffacher (ST). We represent the transport network as a directed graph, where each link (edge) refers to a direct connection between two places, and each link carries a weight that it refers to the travel time in minutes, as shown in Figure 3.13.



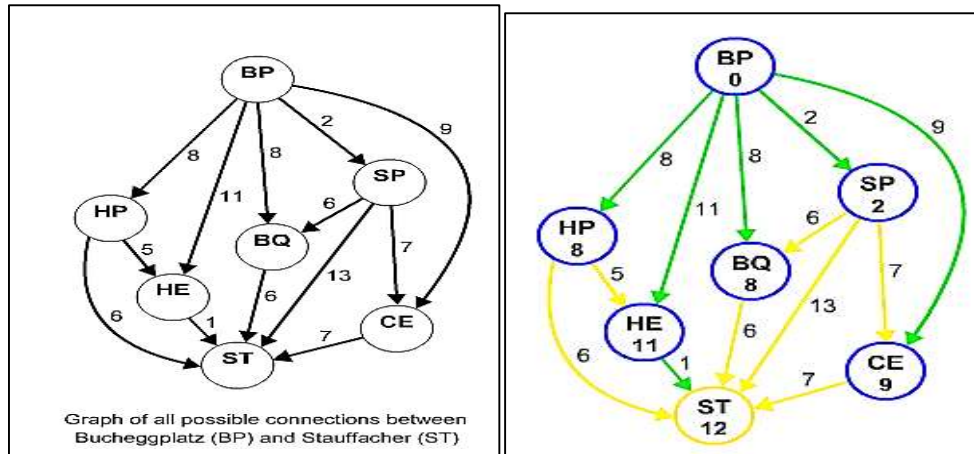


Figure 3.13: Example of a public transport network represented as a graph

Here with ● Vertex visited (tested) is marked, with ● Vertex in progress, → (yellow arrow) indicates Edge in progress, and with → (green arrow) the shortest path is indicated.

Dijkstra's algorithm starts from the source vertex (BP); hence the vertex is placed in the priority queue. The adjacent vertices to source vertex (BP) are vertices BQ, SP, HP, HE and CE which are shown in the graph above. It can be seen that vertex (SP) contains the lowest cost which is (2) as shown in Fig 3.13. Then vertex (SP) is kept in a Priority queue. The neighbours of vertex (SP) are vertices (BQ), (CE) and (ST) (see Fig 3.13). From these three vertices, vertex (BQ) has the lowest cost from vertex (BP) which is labelled as (8), so (BQ) is stored in a priority queue. After that, vertex (BQ) is extended to its adjacent vertices, which in this case only namely, vertex (ST). But in this stage, there are three vertices that are unvisited yet, which are (HP), (HE) and (ST) [141]. Between the three remaining vertices, vertex (HP) has the lowest cost which is (8) from the source vertex (BP). Hence vertex (HP) is put in a priority queue. The neighbours of vertex (HP) are vertices (HE) and (ST), where vertex (HE) has the lowest cost from vertex (BP) which is (11). Thus, vertex (HE) is stored in a priority queue and then vertex (HE) is extended to its adjacent vertex which is vertex (ST). Vertex (ST) is the last one that is examined and put in a priority queue with the vertices (HE) and (BP). As (ST) is the target vertex, then vertices (HE) and (BP) are backtracked [127]. The shortest path found with the lowest cost by vertex (HE) which is (12), (see Figure 3.13). Note that this path can be called 'the path of waypoints' [25]. The shortest path in this graph is  $[BP+HE+ST] = [11+1] = 12$  (see Fig 3.13).

Figure 3.14: Example of shortest path in graph from (BP) to (ST)

## **Chapter Four**

## 4 Multi-robot motion planning-Based on Roadmap methods

### 4.1 Motion planning problem

In this chapter, we start our investigation into the problems of motion planning of a multi-robot system with a specific focus on how utilisation of graph theory and algebraic graph theory concepts can contribute to their solutions. As was mentioned in Chapter 3, this problem is considered one of the multi-robot problems which graph theoretic approaches have played a significant role in developing and solving [95][96]. Multi-robot motion planning is the problem of computing feasible paths for a team of robots between a start location and a target location in an environment. It is a major technique that enables robots to move in an environment [131] [150]; it also includes determining what motions are appropriate for the robot so that it reaches a target state without collisions into obstacles [151]. Therefore, the motivation behind our interest to study the motion planning problem is to design mobile robots that can implement specific and detailed instructions. One of those instructions is to move a robot from one point (v) to another point (u) inside the workspace environment (W), or in other words, determine the robot path [150]. The motion planning problem for a multi-robot system is the path planning problem that can be formulated as the problem of finding a collision-free path for the robot movements from a start position to a goal position in a given environment according to some criteria (e.g., distance and time), in such a way that the straight-line path between two consecutive positions do not intersect with the interior of the obstacles [95][96]. Path planning problem is one of the critical problems in robotics, which has been under extensive study until nowadays. To achieve that, two skills are needed: the first one being the ability to know how to apply the properties of graph algorithms, linear algebra, and geometry to multi-robot systems; the second one being the ability to formulate and solve the motion planning problems in a multi-robot system. This chapter presents the motion planning algorithm that is referred to as a roadmap method. The roadmap is built by a set of paths each of which consists of collision-free area connections. This method demonstrates the ability of robots to move between obstacles in their configuration spaces. In order to achieve this, it is assumed that each robot is able to gain all the necessary information around it (locations of neighbouring robots and obstacles) through its knowledge of free workspace [129][147]. We consider the fundamental task of moving robots from start point S to endpoint E in an environment with static obstacles. In this task, the failure or success of robots' motion depends on their knowledge ability of the workspace [152].

## 4.2 Motion Planning Roadmap Method

A roadmap as we discussed in Chapter 3 is a set of locations in the configuration space along with the paths that connect them [146]. It is a general approach to solving the problem of motion planning. Roadmaps are representing the connectivity of free space for robots which consists of a workspace where robots will be working along with the start and goal configurations for each robot. It determines paths that robots should follow to reach their goals whilst avoiding collisions with obstacles as well as with other robots. In addition, a roadmap can be reached for each point  $s_{start}$  in free space, if a path from  $s_{start}$  to some other vertices in a roadmap with access to the  $g_{goal}$  exists. Furthermore, a roadmap is connected if any two locations are connected via a path [146][153][154][155]. Besides, the introduction of algorithms based on sampling techniques such as the roadmap method had a major effect on the domain of motion planning because of their simplicity, efficiency, and applicability to a wide area of issues, such as the MRS case [95].

### 4.2.1 Problem definition

Multi-robot motion planning is an issue that concerns with finding paths for multiple robots from their given starting positions to their goal positions without colliding with obstacles or each other in the environment [156]. Our aim is to plan the movement of robots from the first position to the target position in a workspace environment through a sequence of steps that will be implemented as follows in section 4.2.

### 4.2.2 Problem Statement Assumptions

In the development of the approach presented in this thesis the following assumptions are made.

- The workspace  $W = 2D \text{ or } R^2$  is a bounded polygon, mostly just a rectangle.
- The motion of the multi-robot begins at the initial position  $s_{start}$  and continues until the final positions  $g_{goal}$ .
- Within the workspace  $W$ , there are a limited number of obstacles
- The beginning position and goal position exist within the workspace  $W$  and outside all obstacles.
- All obstacles in the environment are considered static, and they can be represented by a set:  $O_i = O_1, O_2, \dots, O_n$ .
- The obstacle  $O_i \subset W = R^2$  is geometrically represented as a convex polygon.
- The team of robots consists of a set  $R_i$  of mobile robots.

- There are  $N$  robots denoted as:  $(R_1, R_2, R_3, \dots, R_N)$ , described by a moving point (that is a robot has zero size) at  $(x, y)$ .
- The robot knows the start site and the target site.
- The robots here do not have sensors, but they have total knowledge of the free space.
- Every robot has exact knowledge of the workspace environment (the location and the geometric representation of the obstacles).
- The robot movement is omnidirectional which means that the robot can move in all possible directions.
- Each robot can access information about any robot in the team by connectivity.

#### 4.2.3 Problem inputs and outputs

**Inputs** into multi-robot pathfinding problem are:

A graph  $G(V, E)$  where  $|V| = N$ . The vertices  $V$  of the graph are all the possible positions for robots, and the edges  $E$  are all possible paths between the positions  $R$  robots, each labelled as  $R_1, R_2, \dots, R_n$ . Each of these robots has a start position  $s_i \in V$ , and a goal position  $g_i \in V$ .

**The output** of this problem is a plan, that specifies position of every robot, where at the beginning all robots are at their initial positions and at the end all robots are located in their target positions [156].

#### 4.3 Roadmap Environment Model

Graph models are more appropriate for path and motion planning problems because as a rule, graph model just contains possible paths, since the information about obstacles is excluded during the graph establishing [10]. Points (places) in the environment and permissible paths between them are represented by the graph, vertices of which represent certain places in the environment and edges represent permissible possible paths. The environment model can be represented by a visibility graph method [10][150]. A visibility graph is beneficial in many applications because of its simplicity, visualisation, and completeness. In addition, the advantages of using this method for motion-planning are the following: it is a very simple method, and a well-understood method that produces optimal paths in a configuration space [10]. In addition, by using this approach, we can efficiently compute roadmaps in environments with polygonal obstacles and find an optimal path from the start positions to the goal positions [72][93][131]. A visibility graph is a graph, of which vertices represent vertices of polygonal obstacles, and its edges represent possible straight paths (i.e., visibility lines) connecting the obstacle vertices [157]. Once the graph is created, the starting and goal points are added and

the visible edges, connecting them with other graph vertices, are computed [10]. In fact, a visibility graph formation has two stages: The first stage is the construction of the graph itself (the configuration space that consists of the set of obstacles, where all visibility lines represent a C-space according to forms of obstacles). At the second stage, the start and goal configurations are assumed to be present as vertices to choose the optimal path in the configuration space defined by polygonal obstacles [21][158].

#### 4.4 Motion planning-based on the visibility graph method

The visibility graph method used for multi-robot path motion planning is described as an undirected weighted graph  $(G)$ , where  $(V)$  is the set of vertices representing robot configurations,  $(E)$  is the set of edges representing paths between vertices, and  $w$  is a function that assigns the weight (length path) to each edge in  $E$  [10][128][151][157].

The environmental movement of the robot is viewed as a graph  $(G) = (V; E; w_E)$ , where

- $V$  is the set of vertices (with which we will typically associate a location (physical position in  $2D = R^2$ ), or, in other words, the set of vertices corresponding to the position of the obstacles the robot must avoid, the starting points from which the robot should move, and the end points of which the robot should move towards [128][157].
- $E \subset V \times V$  is a set of edges in the graph, which is the route surrounded by obstacles.
- $w_E: E \rightarrow R^+$  is the weighted cost associated with each edge [128][157].

##### 4.4.1 Definition and designing a visibility-graph algorithm

A visibility graph is defined as a graph whose nodes include the start location, the goal location, and the nodes of polygonal obstacles. Its edges are the edges joining all pairs of nodes that is mutually visible and the edges of the obstacles [157].

A Visibility-graph algorithm is designed by increasing a given undirected weighted graph  $G$  with an additional set of edges ( $E_{vis}$ ), where:  $G_{visibility-gr} = (V, E, w_E)$ ,

$$E_{vis} = \{(v_i, v_j) \in V \times V \mid \text{visible}(v_i, v_j, E_{obs})\}$$

Function  $\text{visible}(v_i, v_j, E_{obs})$  returns true if the edges in  $E_p$  do not cut (does not intersect the presumptive) edge  $(v_i, v_j)$ , or in other means  $(v_i, v_j)$  form an edge if it can go from  $v_i$  to  $v_j$  without being hindered by any obstacle whatever (for every pair of vertices in  $V$ , an edge is added to  $E_{vis}$  that can see each other).  $E$ , the set of edges in graph  $G$  that is divided into  $E_v$ , the edges added via the visibility graph algorithm, and  $E_{obs}$ , the edges that shape the boundaries of the polygonal obstacles:  $E = \{E_{vis} \cup E_{obs}\}$ ;  $w_E: E \rightarrow R^+$  [10][128][151][157].

Figure 4.1: A sample visibility graph

Considering a set of polygonal obstacles  $(O_1, O_2, \dots, O_n)$ , the visibility graph  $G = (V, E, w)$ , with the set of vertices  $V$  that are all “visible” convex vertices from the corners of polygonal obstacles, and the start point and the end point, which is formed by connecting vertices via edges (E) [72][95][158][159][160]. Note, that a vertex of a polygon is a convex vertex if the interior of angle is exactly smaller than  $\pi$  radians while if the vertex has an inner angle larger than  $(\pi)$  then it is not a convex vertex. In addition, in a visibility graph, non-convex vertices of polygonal obstacles are not considered [95][158][159].

#### 4.4.2 Visibility-graph pseudo-algorithm

**Input:**  $s$  start,  $g$  goal, polygonal obstacles.

**Output:** Visibility Graph  $VG$ .

```
1:   for every pair of vertices,  $v_i, v_j$  where  $i, j$ 
2:       for every obstacle  $O$ 
3:           if segment  $(v_i, v_j)$  intersect  $O$ 
4:               go to (1)
5:           end if
6:       end for
7:           Insert edge( $v_i, v_j$ ) into  $VG$ 
8:   end for
```

#### 4.4.3 Visibility graph method for the problem of finding the shortest path

The shortest path in the graph  $G$  is the shortest distance between the source (start position) and the end point (target position) i.e., a path of minimal length, or in other words, the set of waypoints that connect the start position to goal position in a 2D workspace environment. The



Visibility-graph method is considered as one of the roadmaps towards the approach based on combinatorial optimisation, best suited to the problem of finding the shortest path in terms of distance. With this method, a minimum cost path will be found according to certain criteria among the start vertex (start position) and the end vertex (target position) of the robots. Hence, to find the shortest path in workspace, many obstacles require two key steps when solve the problem of finding the shortest path in the form of the visibility graph as above in section 4.1.1):

**First:** Create visibility-graph in workspace  $W = 2D$ .

**Second:** to find the shortest path, an algorithm such as Dijkstra's algorithm based on cost corresponding to each edge (distance between vertices) can be applied [157][160].

The visibility graph method is considered one of the path planning concepts that producing the shortest path if it is combined with Dijkstra's algorithm. In addition, when it is complete, it means that a path will be found if it is available [21][33]. This is the reason that the visibility graph method and Dijkstra's algorithm are chosen for path planning.

Note that: the commonly employed heuristic method in robot motion planning is the A\* algorithm, where the heuristic methods use particular assumptions to minimise the complexity of an issue. These methods have a disadvantage in that they employ multiple variables and coefficients, which must be selected by the algorithm designer. In addition, there is no literature that determines a certain manner for variable selection and hence the consequences are not regular for different scenarios. As a result, heuristic methods do not make general solutions. For different cases, the variables of a heuristic algorithm might need modification [33]. In addition, as we mentioned in chapter 3, the A\* algorithm searches a graph efficiently based on the chosen heuristic (with respect to a selected heuristic). If the heuristic is "good," then the search is effective; if the heuristic is "un good," even though the path will be found, the search will take longer time than probably required and may return a suboptimal path. This means "If the heuristic is optimistic, then A\* will result in an optimal path, where an optimistic, or admissible, heuristic always returns a value less than or equal to the cost of the shortest path from the present vertex to the target vertex inside the graph" [161][162]. This may seem useful, but in some cases, it is impossible or difficult to find a heuristic that is efficient to evaluate and produces good search guidance. Thus, when  $h(n)$  becomes closer to  $g(n)$ , and  $h(n) = 0$ , then A\* degenerates to Dijkstra's algorithm [151]. For this reason, A\* search algorithm is not considered in our work, because if it is combined with the visibility graph method, the resultant path might not be optimal, as it is difficult to compute the heuristic of A\*, where the heuristic value is normally a computation of what the straight-line distance to the target would be if there

are no obstacles. Therefore, there is no method to measure the cost of straight lines that connect vertices to the goal point (g) in an environment where the lines pass through obstacles. Also, if the heuristic cost is not admissible, i.e., higher than the real cost, the produced path may not be optimal in terms of the path length [21][35][118]. Accordingly, a multi-robot motion planning problem becomes the problem of finding the optimal (safe and shortest) paths on workspace for each robot which avoid collision with obstacles and with each other. Hence, based on these requirements of the problem it is important that the new created graph called a visibility graph algorithm covers well connectivity to avoid collision and calculated the optimal path as required [163].

#### **4.5 Connectivity**

The connectivity is critical for a team of multi robots, so, robustness connectivity must be established to distribute information effectively among a team of robots [163]. Maintaining connectivity, i.e., the possibility for robots to exchange information with each other is often an essential requirement, and inter-robot communication enables multi-robot systems to coordinate and execute complex missions efficiently. Besides, the quality and robustness of the connectivity have a high impact on the team's capability to complete their tasks. Whenever increase the number of robots, the communication links increase, thus the number of possible configurations increases dramatically. Furthermore, connectivity maintenance is an important part to consider whilst controlling a multi-robot system; a multi-robot system must be connected to get to a certain common target. Here the tasks can represent objectives such as tracking a target, collisions avoidance and reaching a desired position [163][164]. Maintaining connectivity during the operation of an MRS can be executed utilising two approaches: (1) maintaining the local connectivity approach, and (2) maintaining the global connectivity approach [116], the last one is considered in this thesis. The global connectivity approach depended on mathematical formalism; in this, the communication network of an MRS is represented by utilising graph theory [116]. Any multi-robot formation can be represented as a communication graph where each robot in the formation is identical to a vertex, the capability to communicate among pairs of robots is identical to edges [163]. In addition, if an MRS is represented by an undirected weighted graph, then the algebraic connectivity of the graph Laplacian represents the measure of communication network connectivity, where maximising the algebraic connectivity in an undirected weighted graph gives rise to information flooding in the information exchange process [116].

Let us consider a multi-robot system, which has a limited communication range model as undirected weighted graph  $G = (V, E, w)$  where  $V = \{1, \dots, n\}$  is the set of vertices representing the number of robots,  $E \subseteq \{V \times V\}$  is the set of edges representing paths between vertices, where  $e_{ij}, i \neq j$ . An edge exists between vertices if robot  $i$  interacts with robot  $j$ , this means two robots can communicate if and only if they are within the communication distance, also the presence of the edge  $e_{ij}$  implies to the presence of the edge  $e_{ji}$  because we use an undirected graph.  $w$  is a function that assigns the weight (length path) to each edge in  $E$ .  $w = \{w_{ij} \mid (i, j) \in V \times V\}$  is a set of weights so that  $w_{ij} = 0$  if  $(i, j) \notin E$  and  $w_{ij} > 0$  otherwise. If we consider a team of  $N$  robots, we define the set of neighbours of the  $i$ -th robot as  $N_i = \{j \in V, j \neq i \mid e_{ij} \in E\}$ , which represent all the robots that can communicate with it. Hence, each robot is assumed to be able to interchange data with their neighbours, i.e., with all the robots that are in its neighbourhood [115][163][164]. One method to represent such an undirected weighted graph is by using the graph Laplacian  $L$ , and to use its algebraic connectivity as an indicator of the system connectivity. The algebraic connectivity is defined as the second smallest eigenvalue  $\lambda_2(L)$  of the graph Laplacian. Graph Laplacian  $L \in R^{n \times n}$  be the principal matrix, which combines the adjacency and the degree matrices:  $L = D - A$ . The Adjacency matrix  $A \in R^{n \times n}$  of the weighted graph  $G$  is defined by:

$$A_{ij} = w_{ij} = \begin{cases} a_{ij} > 0 & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases} \quad (4.1)$$

Each element  $a_{ij}$  is defined as the edge weight of  $e_{ij}$ , where  $a_{ij}$  is a positive number,  $a_{ij} = a_{ji}$ ,  $a_{ii} = 0$  for all  $i$ . Here  $A_{ij} = w_{ij} \in R^{n \times n}$  is the weight function and can be seen as function of the distance between robots  $i$  and  $j$ . The degree matrix  $D \in R^{n \times n}$  is a Diagonal matrix showing the number of edges connected to each vertex  $v_i$ . and each element of the diagonal is equal to the  $(i, i)$  entry in  $D = \text{diag}(d_{ii})$ , given by [115][163][164]:

$$d_{ii} = \sum_{j=1}^n a_{ij} \quad (4.2)$$

#### 4.6 Measuring Connectivity with the algebraic connectivity

The second smallest eigenvalue is called the algebraic connectivity value of the system. A significant property of  $\lambda_2$  is that it does not only provides a measure (indicator) of whether the graph is connected, but it also provides a measure of how well-connected the graph is. The value of the algebraic connectivity equals to zero ( $\lambda_2 = 0$ ) if the graph has disconnected components, i.e., there is no paths among vertices, or the graph has two disconnected components. If  $\lambda_2$  is very small, this refers to the graph being nearly disconnected, that it has

two components that are not very connected to each other. Non-zero connectivity refers to a path that exists among every pair of vertices (robots in the system) in the graph. Higher connectivity refers to a more robust graph as a larger number of edges, i.e.,  $0 < \lambda_2 < N$ . In addition, connectivity refers to the number of vertices in the graph if the graph is completely connected, thus the maximum value of  $\lambda_2 = N$ , and it is obtained when the entries  $(i, j)$  of the adjacency matrix are all equal to 1, which means all the possible edges are present in it [115][163][164].

#### 4.6.1 Algebraic connectivity and collision avoidance

The relation between the connectivity of graph and  $\lambda_2$  can be used to control and maintain connectivity. In general,  $\lambda_2$  is a function of the state of the whole system, and it is an important parameter that affects the performance and robustness properties of dynamical systems working over an information network [115]. In addition, the algebraic connectivity maintains connectivity and determines its robustness between robots, which enables them to execute tasks whilst maintaining connectivity inside the system. In our approach, we considered a multi-robot system as an undirected weighted graph. To control connectivity of the system, the connectivity modifies through choosing the best edges to add by measuring the second smallest eigenvalue of the Laplacian matrix to communicate the system robustness [115]. Hence, this enables the robots to obtain whole information about the workspace environment to achieve collision avoidance and to find the best safe paths, using the edges' weights to control the motion time of robots, where edges' weights are the functions of the inter-robot distances. In other words, algebraic connectivity introduces a type of continuous measure of how well a team of robots is connected while maintaining connectivity within the system. Indeed, the weight functions can be determined to make the inter-robot distances converge to a desirable value. In addition, the weight functions can be selected since collisions between the robots are always avoided. More specifically, the edge-weight functions are designed with the purpose of controlling the motion of robots while avoiding inter-robot collision [78].

We consider a multi-robot system consisting of  $N$  mobile robots moving in a 2D space, and define the state vector of the multi-robot system  $p_i = [p_1^T, p_2^T, \dots, p_N^T]^T \in R^n$ ,  $p_i \in R^n$  to represent the position of robot  $i$ . The team of robots can be described by single integrator models as:  $\dot{p}_i = u_i$ , where  $u_i \in U \in R^n$  refers to the control input of the system:

$$\dot{p}_i = u_i = \sum_{j \in N_i} w_{ij}(p_i - p_j) \quad (4.3)$$

We define the edge weights of the graph  $G$  as:  $a_{ij} = \begin{cases} e^{\|p_i - p_j\|} & \text{if } d_{ij} \leq R \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$

where  $R > 0$  is the communication range, and  $d_{ij} = \|p_i - p_j\|$  is the Euclidean distance between position of robot  $R_i$  and robot  $R_j$  [163][164]. In this system, the robots should avoid collisions with each other to stay safe during movement when performing their tasks. In the environment of the workspace, we assume that the obstacles are convex and static, and the distance among two obstacles is greater than the size of a robot. We will consider two sorts of collisions: one is a collision between an obstacle and a robot. The second is inter-robot collisions (i.e., collision among two robots). Each robot can determine the presence of an obstacle and measure its relative location and the distance from its boundary, within the communication range  $R > 0$  [10]. Therefore, to solve the problem of a team of multi-robot, our aim is to maintain a certain value of the algebraic connectivity of the working of the MRS, of which an initial configuration where the team is connected ( $\lambda_2 > 0$ ), keeps connected, whilst being controlled to achieve the desirable objective of collision avoidance until reaching the target configuration. Collision avoidance mechanism is executed, that prevents robots from collision between each other with communication defined based on the edges' weights (which defines the quality of the communication links between robots), and when  $\lambda_2$  is away from zero, whilst every robot tracks their paths to reach the goal location [115][116][163][164].

Note that, in a connected component of an undirected weighted graph, with graph Laplacian (Laplacian matrix)  $L$ , if all edges weights are positive, the second smallest eigenvalue,  $\lambda_2 > 0$ .

#### 4.6.2 Example of an undirected weighted graph:

This example will illustrate how to find the Laplacian matrix and the eigenvalues of the Laplacian matrix, especially the second smallest eigenvalue ( $\lambda_2$ ). Figure 4.2 shows the example of an undirected weighted graph:  $G = (V, E)$  where  $i = 1, \dots, 16$ ,  $j = 1, \dots, 49$ .

Figure 4.2: Example of an undirected weighted graph

The vertices set of G is:  $(V) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}\}$ .

The edge's weights  $(E) = \{e_1, e_2, \dots, e_{49}\}$ , the number on the edges represents the weights of the edges (the distance between every two vertices).

**Table 4.1:** Showing the edges set E of weighted graph G above is:

$e_1 = (v_1, v_2)$	$e_2 = (v_1, v_3)$	$e_3 = (v_1, v_4)$	$e_4 = (v_1, v_5)$	$e_5 = (v_1, v_6)$
$e_6 = (v_1, v_7)$	$e_7 = (v_2, v_3)$	$e_8 = (v_2, v_4)$	$e_9 = (v_2, v_5)$	$e_{10} = (v_2, v_7)$
$e_{11} = (v_3, v_4)$	$e_{10} = (v_3, v_5)$	$e_{13} = (v_3, v_6)$	$e_{14} = (v_4, v_5)$	$e_{15} = (v_4, v_6)$
$e_{16} = (v_4, v_7)$	$e_{17} = (v_4, v_8)$	$e_{18} = (v_4, v_{11})$	$e_{19} = (v_5, v_7)$	$e_{20} = (v_5, v_9)$
$e_{21} = (v_6, v_8)$	$e_{22} = (v_6, v_{10})$	$e_{23} = (v_6, v_{12})$	$e_{24} = (v_7, v_8)$	$e_{25} = (v_7, v_9)$
$e_{26} = (v_7, v_{11})$	$e_{27} = (v_7, v_{13})$	$e_{28} = (v_8, v_9)$	$e_{29} = (v_8, v_{10})$	$e_{30} = (v_8, v_{11})$
$e_{31} = (v_8, v_{12})$	$e_{32} = (v_8, v_{15})$	$e_{33} = (v_9, v_{11})$	$e_{34} = (v_9, v_{13})$	$e_{35} = (v_{10}, v_{11})$
$e_{36} = (v_{10}, v_{12})$	$e_{37} = (v_{10}, v_{14})$	$e_{38} = (v_{11}, v_{12})$	$e_{39} = (v_{11}, v_{13})$	$e_{40} = (v_{11}, v_{15})$
$e_{41} = (v_{11}, v_{16})$	$e_{42} = (v_{12}, v_{14})$	$e_{43} = (v_{12}, v_{15})$	$e_{44} = (v_{12}, v_{16})$	$e_{45} = (v_{13}, v_{15})$
$e_{46} = (v_{13}, v_{16})$	$e_{47} = (v_{14}, v_{15})$	$e_{48} = (v_{14}, v_{16})$	$e_{49} = (v_{15}, v_{16})$	—

The communication graph can be described by means of the adjacency matrix  $A \in \mathbb{R}^{n \times n}$  of the weighted graph  $G$  defined by:  $[A(t)]_{ij} = w(t)_{ij}$ , as we mentioned in section 4.5 above. It is square and symmetric i.e.,  $A = A^T$  (only for undirected graphs). Communications among robots  $i$  and  $j$  are possible if the  $(i, j)$  entry in  $A(a_{ij})$  has a value of one (1), however if the  $(i, j)$  entry in  $A(a_{ij})$  has a value of zero (0) then there are no connections between robot  $i$  and  $j$ . Here a graph is symmetric that means all connections exists in both directions and thus each robot  $i$  that can receive data from another robot  $j$ , can transmit the data back to that robot  $j$ . The Adjacency matrix corresponding to the undirected weighted graph in Figure 4.2 is given by:

$$[A(t)]_{ij} = \begin{bmatrix} 0 & 8 & 7 & 3 & 9 & 6 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 7 & 9 & 7 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 7 & 0 & 8 & 9 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 9 & 8 & 0 & 5 & 8 & 6 & 7 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 9 & 7 & 9 & 5 & 0 & 0 & 6 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 1 & 8 & 0 & 0 & 0 & 8 & 0 & 9 & 0 & 6 & 0 & 0 & 0 \\ 8 & 2 & 0 & 6 & 6 & 0 & 0 & 4 & 8 & 5 & 8 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 8 & 4 & 0 & 9 & 4 & 8 & 7 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 7 & 0 & 8 & 9 & 0 & 0 & 3 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 5 & 4 & 0 & 0 & 3 & 5 & 0 & 7 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 8 & 8 & 3 & 3 & 0 & 4 & 8 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 7 & 0 & 5 & 4 & 0 & 0 & 5 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 8 & 0 & 8 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 & 5 & 0 & 0 & 9 & 7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 6 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 3 & 5 & 7 & 6 & 0 \end{bmatrix} \quad (4.5)$$

The elements of the adjacency matrix represent a weight of the link between robots in network. As we can see, the columns and rows in above indicate the weight (distance) between two vertices, where if there is a path between two vertices it has a value  $D$  and if there is no path, it has a zero value. For example, in the first column and the first row there is a zero value ( $a_{11} = 0$ ), while  $a_{12} = 8$ .

Now,  $D(t) = \text{diag}(d_{ii}) = \sum_{j=1}^n a_{ij} = w_{ij}(t)$ , where  $D$  is the Diagonal matrix of vertex degrees, the value of  $D$  is calculated based on how many edges are linked to that vertex. The Diagonal matrix corresponding to the undirected weighted graph in Figure 4.2, is given by:

$$[D(t)]_{ij} = \begin{bmatrix} 41 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 33 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 43 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 38 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 50 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 53 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 35 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 33 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 37 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 48 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 42 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 \end{bmatrix} \quad (4.6)$$

Note, for vertex 1, there are six edges connected from vertices 2, 3, 4,5,6 and 7. Thus, the degree of vertex 1 is:  $d_{ii} = \sum_{i=1}^n a_{ij} = w_{ij}(t)$

$$d_{11} = \sum_{i=1}^n a_{12} + a_{13} + a_{14} + a_{15} + a_{16} + a_{17} = 8+7+3+9+6+8 = 41$$

A matrix that plays a central role in many graph-theoretic treatments of MRSs is the Laplacian matrix, which defined by:  $L(t) = D(t) - A(t)$ , where  $D(t)$  is the Diagonal matrix and  $A(t)$  is the Adjacency matrix. The essential role of the Laplacian matrix is to measure the connectivity in the team of multiple robots. The biggest role is played via  $\lambda_2$  of the Laplacian matrix, if  $\lambda_2 > 0$  then a graph is connected. It also means that robots have a stronger relationship to receive and share information with their neighbours. The Laplacian matrix corresponding to the undirected weighted graph in Figure 4.2 is given by:  $[L(t)]_{ij} =$

$$\begin{bmatrix} 41 & -8 & -7 & -3 & -9 & -6 & -8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -8 & 33 & -7 & -9 & -7 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -7 & -7 & 32 & -8 & -9 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -9 & -8 & 50 & -5 & -8 & -6 & -7 & 0 & 0 & -4 & 0 & 0 & 0 & 0 & 0 \\ -9 & -7 & -9 & -5 & 43 & 0 & -6 & 0 & -7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -6 & 0 & -1 & -8 & 0 & 38 & 0 & -8 & 0 & -9 & 0 & -6 & 0 & 0 & 0 & 0 \\ -8 & -2 & 0 & -6 & -6 & 0 & 50 & -4 & 8 & 5 & 8 & 0 & -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -7 & 0 & 8 & -4 & 53 & -9 & -4 & -8 & -7 & 0 & 0 & -6 & 0 \\ 0 & 0 & 0 & 0 & -7 & 0 & -8 & -9 & 35 & 0 & -3 & 0 & -8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -9 & -5 & -4 & 0 & 33 & -3 & -5 & 0 & -7 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 0 & -8 & -8 & -3 & -3 & 47 & -4 & -8 & 0 & -6 & -3 \\ 0 & 0 & 0 & 0 & 0 & -6 & 0 & -7 & 0 & -5 & -4 & 37 & 0 & 5 & -7 & -3 \\ 0 & 0 & 0 & 0 & 0 & 0 & -3 & 0 & -8 & 0 & -8 & 0 & 32 & 0 & 8 & -5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 & 5 & 0 & 48 & -9 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -6 & 0 & 0 & -6 & -7 & 8 & 9 & 42 & -6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & -3 & -5 & -7 & -6 & 24 \end{bmatrix} \quad (4.7)$$

The eigenvectors of the Laplacian matrix are  $\{\vec{v}_i, \text{ where } i = 1, \dots, 16\}$ , and eigenvalues are  $(\lambda_i)$ , where  $\{\lambda_i, i = 2, \dots, 16\}$ . The multiplication of eigenvalues in the formula of



$[L(t)]_{ij}$  in (4.7) starts with  $\lambda_2$ . The smallest eigenvalue  $\lambda_1$  is always zero (0) because of the construction of the Laplacian matrix we have zero row sums  $\sum_j L_{ij} = 0$ , thus,  $L(t)1 = 0$ , which means that there is at least one eigenvalue  $\lambda_1 = 0$ . In addition, if  $\lambda_1$  is a simple eigenvalue (i.e.  $0 = \lambda_1 < \lambda_2$ ), then the graph is connected. Moreover, in this case,  $null(L) = span\{1\}$ , where  $\vec{1} = [1 \dots 1]^T$  and  $\vec{0} = [0 \dots 0]^T$  are vectors of  $N$  elements all equal to 1 and 0 respectively. This implies  $L1 = 0$  [49][50][113]. Note that the  $\lambda_2$  of the Laplacian matrix given above has a value greater than zero. The advantage of this value is to measure connectivity of the associated graph, where in case of undirected weight graph  $\lambda_2$  is larger in case of a highly connected graph. In addition, the connectivity between robots is measured by  $\lambda_2$ , where the robots have perfect connectivity if  $\lambda_2$  has a great value. So, this enables the robots to obtain all the information that they need to perform their tasks to the fullest. Table 4.2 represents the eigenvalues of the Laplacian matrix.

**Table 4.2:** the eigenvalues of the Laplacian matrix

$\lambda_2$	7.6143
$\lambda_3$	17.9264
$\lambda_4$	21.3275
$\lambda_5$	32.4816
$\lambda_6$	35.0564
$\lambda_7$	36.3646
$\lambda_8$	38.8115
$\lambda_9$	41.8012
$\lambda_{10}$	43.7309
$\lambda_{11}$	44.2198
$\lambda_{12}$	50.8069
$\lambda_{13}$	53.2039
$\lambda_{14}$	58.0177
$\lambda_{15}$	61.1491
$\lambda_{16}$	65.4882

**Note:**  $\lambda_2 = 7.6143$ , which means that the graph in Figure 4.2 is more connected than in other situations. If we delete the edges  $\{e_{41}, e_{44}, e_{46}, e_{49}\}$  from the graph in Figure 4.2 then  $\lambda_2 =$

4.5747 the graph is connected because  $\lambda_2 > 0$ , but if we delete also the edges  $\{e_{41}, e_{44}, e_{46}, e_{48}, e_{49}\}$ , then the graph will be not connected, due to  $\lambda_2 = 0$ . The graph is connected if and only if  $\lambda_2 > 0$ .

#### 4.7 Definition (Pseudo-code) Dijkstra's Algorithm

Let us begin with some notes before explaining the details of the pseudo-code of Dijkstra's algorithm, as it is important to understand how the algorithm works. As we mentioned previously, the algorithm is working via solving a sub-problem that calculates the shortest path from the source vertex to the nearest vertices. Besides that, for the algorithm to work, the graph must be a weighted graph, and all edges must have positive weights because the algorithm solves only the problems with positive weights or nonnegative costs. This means that:

$$c_{ij} \leq 0 \text{ for all } (i, j) \in E \quad (4.8)$$

The following pseudo-code gives a summarised description work of Dijkstra's algorithm [70][154].

##### 4.7.1 Pseudo-code of Dijkstra's algorithm

**Input:** Dijkstra (weighted graph  $G$ , Source ( $S_i = R_1, R_2, R_3$ ) Robot position, Target)

**Output:** the set of visited vertices and distance values for each vertex in the graph  $G$

// Initialisation of value of shortest path, not the path itself:

**for each vertex  $v$  in  $G$  :**

2:  $dist[s] = 0$  (distance to source vertex is zero)

3: **for all  $v \in V - \{s\}$**

4:  $do dist(v) = \infty$  (Set all other distances to infinity)

5:  $S = \emptyset$  ( $S$ , the set of visited vertices is initially empty)

6:  $Q = v$  set of all vertices in  $G$

7: **while  $Q \neq \emptyset$  is not empty :**

8:  $\_$  find vertices  $u$  in  $Q$  with smallest  $dist(v)$

9:  $S \leftarrow S \cup \{u\}$  (add  $u$  to list of visited vertices)

10: **for all  $v \in neighbors[u]$**

11: **for each vertex  $u$  connecte to  $v$  by an edge :**

12: **if  $dist(v) > dist(u) + w(u; v)$  :** (if new shortest path found)

13:  $dist(v) = dist(v) + w(u; v)$  (Set new value of shortest path)

14:  $(Q) = v$

15: **return** *dist values for all vertices  $v$*

#### 4.7.2 Flow Chart of Dijkstra's algorithm

In this work, MATLAB is being used to implement the Dijkstra's algorithm. Figure 4.3 represents the flowchart that shows stages involved in the algorithm.

Figure 4.3: Flow chart of Dijkstra's algorithm

#### 4.7.3 Techniques of Dijkstra's algorithm for the motion planning problem

In the application field of motion planning or network analysis, a common problem is the computation of shortest paths in the workspace for a team of robots to perform their tasks to the fullest. One of the standards of path planning is to provide an optimal path which connects the first position  $s_{start}$  and the goal position  $g_{goal}$ . A multi-robot system deployed in the work environment to accomplish robots' tasks whilst following their respective optimal paths will obtain two properties: (1) minimum transit duration to reach the target; (2) the avoidance of collision. The visibility graph with Dijkstra's algorithm is available to achieve this [21][165]. The algorithm is implemented to calculate the shortest distance between two places (vertices) in a network (graph) and takes into consideration the weights of edges which are positive values (cost or distance between two places or vertices). The importance of Dijkstra's algorithm is measured by its ability to solve the shortest path problem, which makes this algorithm a powerful and general tool [21][165].

#### 4.7.4 Application of Dijkstra's algorithm

The visibility graph has a purpose to determine the set of obstacles that will be utilised to calculate the path from the starting point to the target point based on Dijkstra's algorithm. This path must be secure and satisfies three criteria such as path optimality, completeness, and computational efficiency. So, before designing path planning algorithms these criteria must be considered. To control a team of multiple robots in an environment of workspace, we first define the position for each robot and obstacle, the moving path for each robot, and target position. We assume that there are  $N$  robots  $R_i$ ,  $i = 1, \dots, N$ , each situated in a 2D space. We associate an undirected weighted graph in Figure 4.2 above,  $G = (V, E)$ , with the robots, where  $V$  represents the vertices: sets of robots and obstacles,  $E$  is the edge set that defines the communication (paths) between the robots (see Figure 4.4) below. An undirected graph means that if  $E \subseteq V \times V$ ,  $(v_i, v_j) \in E \Leftrightarrow (v_j, v_i) \in E$ . A link  $(v_i, v_j)$  between robots  $R_i$  and  $R_j$  implies that robots can measure their relative positions, speed, and exchange or share information. In addition, we assume that the inter-robot distances are specified as a set of  $D$ , where

$$D \{d_{ij} \in R \mid d_{ij} > 0, i, j = 1, \dots, N, i \neq j\}, d_{ij} = d_{ji} \quad (4.9)$$

Figure 4.4: Example of a team of robots represented as vertices of graph

The graph  $G = (V, E)$  consists of vertices  $V = \{v_1, \dots, v_{16}\}$  starting from  $(S_i = Robot_1, Robot_2, Robot_3)$ , where  $Robot_1, Robot_2, Robot_3$  are indicated as  $(v_1, v_2, v_3)$  respectively, the goal is indicated as  $(g = v_{16})$ ,  $E = \{e_1, \dots, e_{49}\}$ , and edge cost are the terms used for the distances between two vertices, such as the edge cost between  $R_1 = v_1$  and  $v_4$  of the weighted graph in Figure 4.2 is equal to 3. The graph has four polygonal (triangular)

obstacles ( $O = O_1, O_2, O_3, O_4$ ), and three robots ( $R_1, R_2, R_3$ ) represented as vertices. Each robot has an initial position ( $S_i$ ) and the goal position ( $g_i$ ); in this graph the robots have the same target which is ( $v_{16}$ ). We aim at finding collision-free paths for robots to move from the starting point to reach the goal point. Dijkstra's algorithm is applied to determine the optimal paths to move between vertices. The aim is to find the shortest paths from the original point (start point  $S_i$ ) for each robot to all other vertices. The algorithm assigns zero to first vertex and vertices  $S_i=R_1, R_2, R_3=0$ , whilst assigns infinity to all other vertices which are not visited. Then, it will assign gradually a value to obtain the smallest value up to the target or (it will choose smallest value up to the target destination), which is the vertex  $g = v_{16}$ . The algorithm processes each step taken and measures the distance from  $S_i$  to  $g$ . To explain how this algorithm works, consider a scenario of an undirected weighted graph in Figure 4.2.

**First step:** vertex ( $S_i; R_1 = v_1, R_2 = v_2, R_3 = v_3$ ) is set to become the current vertex and put in the priority queue. Zero is assigned to vertex  $S_i$  and infinity to all other vertices (See Figure 4.3).

Figure 4.5: First step of Dijkstra's algorithm

**Second step:** The temporary distance is calculated by considering all unvisited adjacent vertices to the vertices ( $R_1 = v_1, R_2 = v_2, R_3 = v_3$ ), and the least distance is chosen. Then the previously registered value is replaced with a new value less than infinity.

Figure 4.6: Second step of Dijkstra's Algorithm

**Third step:** All neighbours of vertex  $S_i = (Robot_1, Robot_2, Robot_3)$  have been taken into consideration, they are denoted as visited and will not be examined again. The neighbours to vertices  $S_i$  are vertices  $(v_4, v_5, v_6, v_7)$ . It is found that  $(v_4, v_6, v_7)$  have the least distance, i.e. (3, 1, 2). The vertices  $(v_4, v_6, v_7)$  now are marked as current vertices for each robot and are stored in the priority queue.

Figure 4.7: Third step of Dijkstra's Algorithm

**Fourth step:** The neighbouring vertices of vertices  $(v_4, v_6, v_7)$  are vertices  $(v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13})$ . From these vertices,  $(v_{11}, v_{12}, v_{13})$  have the least distance, i.e. (7, 7, 7), thus they will be marked as current vertices and kept in the priority queue.

Figure 4.8: Fourth step of Dijkstra's Algorithm

**Fifth step:** Now vertices  $(v_{11}, v_{12}, v_{13})$  will be 'expanded' to their neighbours, i.e. vertices  $(v_{14}, v_{15}, v_{16})$ , vertex  $(v_{16})$  has the least distance from these vertices  $(v_{11}, v_{12}, v_{13})$  and will place in the priority queue. Recall, that  $g = (v_{16})$  is the goal vertex. Since all the vertices have been visited, the shortest paths from vertices  $(S_i = R_1, R_2, R_3)$  to the goal vertex  $g = v_{16}$  is found with the lowest distance, i.e., 10.

Figure 4.9: Fifth step of Dijkstra's Algorithm

Figure 4.10: Shortest paths by Dijkstra's Algorithm

#### **4.8 Multi-Robot Path Planning Algorithm (MRPPA)**

To address the motion planning problem for a multi robot team, and to find collision-free optimal path the MRPPA based on visibility graph method has been proposed in this thesis.

Our proposed roadmap algorithm includes the following main steps:

- Establish a free space map.
- The algorithm defines the start  $s$  and goal  $g$  positions for each robot, and obstacles vertices numbers and locations.
- All obstacles in the map are modelled as polygons. The algorithm analyses the position of each obstacle's vertices. The starts and goals positions of the robots are known relative to the obstacles in the surrounding environment. Each robot is considered a dynamic obstacle.
- Use the constructed free space and visibility graph algorithm, through which the robots can navigate without colliding with obstacles.
- The workspace environment is divided into two disconnected components of the undirected weighted graph. Then choosing the best edges to add between these two components of the graph to find paths for each robot, based on the measure value

of algebraic connectivity of graph Laplacian, which controls the inter-robot's connectivity when it is greater than zero.

- When planning a path for any robot, vertex weights are changed just as in the single-robot path planning algorithm. The vertices' weights of the graph are initialised with a maximum possible value  $\infty$ , whilst the start vertex is initialised by the start time value  $s_i = w_0 = t_0$ . According to known edge weights, Dijkstra's algorithm will be applied to find the shortest path based on cost corresponding to each edge (distance between vertices), where the shortest path is a path of minimal length, so it is required to find a vertex sequence (series waypoints), which denotes the shortest path from the starting point to the goal point.

Note that, if Dijkstra's algorithm is used as a basis to find the shortest path, the path for robots can be changed based on distance, which corresponds to environment model correction. In addition, during path planning, vertices' weights  $w_i$  change and equate to the moments of time, at which the robot  $R_i$  passes through these vertices [10].

$$w_j = \begin{cases} w_i + w_{ij}, & \text{if } (w_i + w_{ij}) < w_j \\ w_j, & \text{if } (w_i + w_{ij}) \geq w_j \end{cases} \quad (4.10)$$

where  $w_i$  is vertex weight, and  $w_{ij}$  is the edge weight of the graph that corresponds to motion time from vertex  $v_i$  and vertex  $v_j$ . This value is variable and can be changed during planning the path. In addition, it can have two different weights  $w_{ij}$  and  $w_{ji}$ , which based on direction of motion among vertex  $v_i$  and vertex  $v_j$ . Beside  $w_{ij}$ , each edge of graph is characterized by distance (  $d_{ij} = e_{ij}$  ) between vertex  $v_i$  and vertex  $v_j$ .

- To provide collision avoidance, the edges' weights can be modified during path planning, either by path correction, where a robot is not allowed to move on the edge that occupied by another robot or through control robot's motion time on some edges by controlling the distances between vertices to allow it to free up the way for others, the paths of which are planned earlier [10]. This means the increased time of the robot moves on the graph edge from vertex  $v_i$  to  $v_j$ . So, we have two principal conditions that must be considered for path correction, and the robot motion time needs be controlled to avoid collision. This is done through the following steps:

**First:** it is not possible for two robots to pass crossroads simultaneously on the same vertex of a graph, thus if this happens, to avoid collisions, let  $T_{R_n}$  be the arrival time: (i.e., a time when robot  $R_n$  passes through the vertex  $v_i$ ), and  $R_n$  be robots, ( $n = 1, 2, \dots, m$  represent the number of robots), if  $T_{R_n} = w_i + w_{ij}$ , we will assume that  $\epsilon > 0$  is a minimum value, which



determines the time interval among robots passing the same crossroads. Then  $w_{ij} = w_{ij} + \epsilon$ , it must provide a safe passage (safe distance) for robots when crossing the crossroads through increased weight edge (distance) on the graph from vertex  $v_i$  to vertex  $v_j$  to increase motion time of the robot on a graph edge by  $\epsilon$  time units that correspond to its motion time change. Therefore  $\epsilon$  is the safety value from which robots  $R_n$  and  $R_m$  will never collide, the weight  $w_j$  of vertex  $v_j$  is calculated according to formulae (4.10).

**Second:** Two robots are not allowed to move together on the same edge in opposite directions (two different directions). Therefore, if two robots are moving in opposite directions on a graph edge (straight roads) at the same time. If  $(w_i > T_{R_{ni}}) \wedge [(w_i + w_{ij}) > T_{R_{nij}}], (n = 1, 2, \dots, m)$ , then  $T_{R_m} > T_{R_n}$  and no collision happens. Due to the robot  $R_n$  will pass through the edge before the robot  $R_m$ , whose path is being planned, and drives onto the edge. Hence in this case, the edge weight does not require changing. Then the vertex weight  $w_j$  is calculated as (4.10).

Note that, the time ( $T_{R_n}$ ) depended on the distance ( $d_{ij} = w_{ij}$ ) between the edges in the graph. If  $T_{R_m} > T_{R_n}$ , this means the distance travelled by the robot  $R_n$  is less than the distance travelled by the robot  $R_m$ , hence, the arrival time of robot  $R_n$  is shorter than the arrival time of the robot  $R_m$ .

On the other hand, if  $T_{R_m} < T_{R_n} \wedge \left[ T_{R_m} \leq \frac{w_i(T_{R_n} - T_{R_m}) - T_{R_m} \cdot w_{ij}}{T_{R_n} - T_{R_m} - w_{ij}} \leq T_{R_n} \right]$ , then collision occurs because the robot  $R_n$ , whose path is being planned will follow robot  $R_m$  on the edge and collide with it, due to the distance travelled by it is short, so its arrival time is short. To avoid collision, it is important to modify the edge weight of the current robot (reduce the movement of this robot, which is being calculated). That means to increase its arrival time by increasing distance in this edge as

$$w_{ij} = \frac{(w_i - T_{R_n} - \epsilon)(T_{R_n} - T_{R_m})}{T_{R_n} - T_{R_m} - \epsilon} \quad (4.11)$$

then the vertex weight  $w_j$  is defined as in (4.10).

In addition, if  $(w_i < T_{R_{ni}}) \wedge [(w_i + w_{ij}) < T_{R_{nij}}], (n = 1, 2, \dots, m)$ , then  $T_{R_n} > T_{R_m}$ . This state occurs when two robots move in opposite directions and the robot  $R_m$  whose path is being planned will cross through the edge earlier than robot  $R_n$ . There is no collision that occurs; thus, the edge weight does not need to change. The weight of the next vertex  $w_j$  is calculated as (4.10).

In contrast, if  $T_{R_n} < T_{R_m} \wedge \left[ T_{R_n} \leq \frac{w_i(T_{R_m} - T_{R_n}) - T_{R_n} \cdot w_{ij}}{T_{R_m} - T_{R_n} - w_{ij}} \leq T_{R_m} \right]$ , then the collision is possible: robot  $R_n$  will follow the robot  $R_m$ , of which its path is being planned, and collide it on the edge. To avoid collision, it is important to modify the edge weight of the current robot (i.e., changing the arrival time through increasing distance) according to (4.11), and then the vertex weight  $w_j$  is defined as (4.10). In addition, if  $(w_i < T_{R_{n_i}}) \wedge [(w_i + w_{ij}) > T_{R_{n_i}}]$ , ( $i = 1, 2, \dots, n$ ), then  $T_{R_n} < T_{R_m}$ , then the collision is possible: robot  $R_n$  will follow and crash into the robot  $R_m$ , of which its path is being planned, before the crossroads. To avoid collisions, the arrival time of the current robot must be increased. So, the edge weight must be changed based on (4.11), and then the vertex weight  $w_j$  is calculated as (4.10) [10].

#### 4.8.1 MRPP Algorithm

**Inputs:** Start positions( $S$ ), goal positions ( $g$ ), polygonal obstacles( $O$ ).

**Outputs:** Visibility graph ( $VG$ ), Optimal paths from  $s$  start to  $g$  goal

Our algorithm mainly includes the following steps:

1. **Establish** a free space map.
2. **Determine** the start  $s$  and goal  $g$  positions for each robot, and obstacles' vertices numbers and locations.
3. **Divide** the workspace environment into two disconnected components of undirected weighted graphs  $\{G_1, G_2\}$ .
4. **Select** the best edges ( $w_{ij}$ ) to add between these two components of the graph  $\{G_1, G_2\}$  based on the measure value of algebraic connectivity of graph Laplacian ( $\lambda_2$ ).
5. **Create** the Visibility graph ( $VG$ ).
6. **Find** a vertex sequence (series waypoints) from start( $s$ ) to goal ( $g$ ) by using Dijkstra's algorithm, which denotes the shortest paths.
7. **End:** paths calculated  $\widehat{W} = \{w_0, \dots, w_n\}$ , where  $w_0$  = start point and  $w_n$  = goal point.

#### 4.11. The process of MRPP Algorithm

## **Chapter Five**

## 5 Implementation of the Multi-Robot Path Planning Algorithm

### 5.1 Introduction

The previous chapter has explained the proposed algorithm (MRPPA) in this thesis, which combines the advantages of the roadmap visibility graph approach with algebraic connectivity (second smallest eigenvalue) of the graph Laplacian, and Dijkstra's algorithm for planning short and safe paths for a team of a multi-robot system in a two-dimensional workspace.

The proposed paths planned contain two main components: a global planner and path optimisation. The global planner gathers information about the surrounding environment such as the information about the robot's positions and targets as well as all information about the obstacles. Depending on the analysis of the Roadmap to find the path with minimum cost. Where it finds the optimal path with a prior knowledge of the environment and static obstacles, so a collision-free optimal path is created before the robots start moving. All robots have prior information about their work environment such as location of the obstacles and targets. After that, it proceeds to generate sets of possible ways (visible) in which the robots can reach their targets by using a visibility graph. The MRPP Algorithm analyses all possible ways and chooses the most suitable path, based on the measure of algebraic connectivity, and the predefined weight evaluation function. It implies sequential path planning for each robot one by one (i.e., path by path), considering all already planned paths (i.e., when planning the path for the next robot it considers all the paths already planned to exclude collisions). Consequently, the first path in the sequence is planned for the first robot; the path of the second robot is planned with a concern of the first robot's path. While, when planning the path for the third robot, MRPPA will take into account the paths of the first and second robot. The algorithm provides the optimal paths for each robot. It means that the currently planned path is the optimal of all possible at this phase. The paths here, their lengths, and motion times are based on the order of planning. The choice of the right sequence for path planning of robots has a significant impact on the performance of the robot team.

This chapter aims to produce optimal paths that connect the starting nodes to the target node for a team of multi-robots with collision avoidance. The robots that follow paths in the workspace environment to accomplish their missions will obtain the following benefits:

- Perform their tasks in a short time and reach their targets. This is because VG can calculate path with shortest length (the paths have the least distances because they contain a

set of vertices of obstacles). It is also complete which means it guarantees that the robot will find the shortest path to its target.

- Maintain their energy and reduce their consumption. Due to that the VG has the ability of finding a path with the shortest distance if one exists. Thus, this makes robots take a short time to move and perform their tasks, and this will reduce their energy use.
- Low exposure to a collision, because the visibility graphs methods can help the robots in the system move to the desired goal location while avoiding collisions. VG considers obstacle vertices in the environment to be the vertices, through which the robots can arrive at their required locations. These visible vertices have the property that a straight line connecting them does not intersect the interior of obstacles.

To achieve all these advantages, different scenarios of workspace environments for a team of a multi-robot system are given.

## 5.2 Path Planning Using MRPP Algorithm

To illustrate how the algorithm works, a scenario consisting of six obstacles where their sizes and positions are generated together with s start and g goal is considered as per depicted in Figure. 5.1. To control a team of multiple robots, we assume that there are  $N$  robots  $R_i$ ,  $i = 1, \dots, N$ , each situated in a 2D space. Where the position for each robot, obstacles, the moving path for each robot, and target position are predefined. So if we consider the scenario in Figure 5.1 the workspace consist of three robots ( $S_i = Robot_1, Robot_2, Robot_3$ ) each one has starting positions indicated as red points, the green point denotes the goal position g, and six Obstacles ( $O_i = O_1, O_2, O_3, O_4, O_5, O_6$ ) indicated as blue shapes.

Figure 5.1: Scenario of workspace environment for path planning

If we created visibility graph according to the definition and algorithm as mentioned in the previous chapter, the environmental movement of the robot is viewed as an undirected weighted graph  $(G) = (V; E; w_E)$ , where  $V$  is a set of vertices corresponding to the position of the obstacles, and starting and the end points of which the robot should move from and to,  $E$  a set of edges in the graph, which is the route surrounded by obstacles, and  $w_E$  is the weighted cost, value, or number associated with each edge. Where edge  $e_i = (v_i, v_j) \in E$ , and the weight  $w(e_i)$  is the distance from vertex  $v_i$  to vertex  $v_j$ , and all weights are nonnegative. Thus, the scenario of Figure 5.1 can be represented as a graph in Figure 5.2.

Figure 5.2: Scenario of workspace environment represented as a graph.

The graph  $(G) = (V; E; w_E)$  in Figure 5.2 has ten components that are not very connected to each other. The vertices  $V = \{v_1, \dots, v_{28}\}$ , where  $v_1, v_2, v_3$ , and  $v_{28}$  are isolated vertices, and the edges set  $E = \{e_1, \dots, e_{24}\}$ , the edges of obstacles. We will divide the workspace environment into two disconnected components of the undirected weighted graph by using visibility graph to add visible edges between vertices. This is because, if the graph has two disconnected components,  $\lambda_2 = 0$ . And if  $\lambda_2$  is small, this suggests the graph is nearly disconnected, that it has two components that are not very connected to each other, such as in Figure 5.3. Now, will use algebraic connectivity to choose the best edges to add to find paths for each robot one by one.

Figure 5.3: Example of disconnected undirected weighted graph

The graph  $G = (V, E)$  in Figure 5.3 contains vertices  $V = \{v_1, \dots, v_{28}\}$  marked from  $S_i = (v_1 = R_1, v_2 = R_2, v_3 = R_3)$  to  $(v_{28} = g)$ ,  $E = \{e_1, \dots, e_{67}\}$ , there are six (6) polygonal obstacles  $(O_i = O_1, O_2, O_3, O_4, O_5, O_6)$ . Each robot has initial position  $(S_i)$  and the goal position  $g$ , here we have just one goal for three robots. The second smallest eigenvalue of the graph in Figure 5.3 has zero value ( $\lambda_2 = 0$ ) because the graph has two disconnected components (the graph not connected). Note that, the  $(Robot_1 = v_1)$  can find way to reach the target because it is exists in the component that consists of vertices  $\{v_1, v_7, v_9, v_{11}, v_{13}, v_{15}, v_{17}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{28}\}$ , where  $(v_{28} = g = \text{Goal})$ . Whilst  $(Robot_2 = v_2, Robot_3 = v_3)$  exist in the component that consists of vertices  $\{v_2, v_3, v_4, v_5, v_6, v_8, v_{10}, v_{12}, v_{14}, v_{16}, v_{18}, v_{24}, v_{25}, v_{26}, v_{27}\}$ , which do not have paths to reach the target  $g$ . Therefore, if we add a bridge (edge) between vertices  $v_{12}$  and  $v_{22}$  to form a weak link between the two graphs,  $\lambda_2$  increases to 0.3128. But if we add a path between vertices  $v_8$  and  $v_{21}$  instead of  $v_{12}$  and  $v_{22}$ ,  $\lambda_2$  increases to 0.3652, thus we create a good link between the components because vertices  $(v_8, v_{21})$  have more connections in their respective components which enable robot two  $(Robot_2 = v_2)$  to find a path to reach the target  $g$  by using Dijkstra's algorithm and it is shown in Figure 5.4.

Figure 5.4: The path planned for robot two by using MRPPA.



In addition, if we add a bridge between vertices  $v_8$  and  $v_{21}$  and vertices  $v_{12}$  and  $v_{22}$  together (i.e., two paths),  $\lambda_2$  jumps to 0.8835. But if we exchange the vertices  $v_{22}$  and  $v_{24}$  instead of the vertices  $v_{12}$  and  $v_{22}$ ,  $\lambda_2$  jumps to 0.8987 because the vertices  $v_{22}$  and  $v_{24}$  have more connections in their respective components. Thus, enables robot three ( $Robot_3$ ) find a path to reach the target  $g$  by using Dijkstra's algorithm and it is shown in Fig. 5.5.

Figure 5.5: The path planned for robot three by using MRPPA.

Moreover, if we add a bridge between vertices  $v_1$  and  $v_6$  and vertices  $v_6$  and  $v_{11}$  together (i.e. two paths),  $\lambda_2$  jumps to 1.7695, and this enables robot one ( $Robot_1 = v_1$ ) to find a path to reach the target  $g$  by using Dijkstra's algorithm and it is shown in Fig. 5.6.

Figure 5.6: The path planned for robot one by using MRPPA.

Although each robot gets a path to reach the target  $\lambda_2$  is still small which refers to two components not being very connected to each other. Thus, the communication is not enough to exchange and share all information between the robots to avoid collision. Therefore, the more paths we add between the vertices of the graph, the more the second smallest eigenvalue increases, and results in a more strongly connected graph, such as the graph shown in Figure

5.6, where  $\lambda_2 = 3.1262$ . By using Dijkstra's algorithm then, shortest paths are found, and it is shown in Fig. 5.7.

Figure 5.7: The paths planned used MRPPA.

Note that, the aim of adding paths between vertices is to illustrate the ability and impact of the algebraic connectivity on determining the best edges to add between vertices to find a path for each robot respectively, how controlled connectivity within the system to guidance each robot to avoid collision, and increase strength communication, which enables robots to obtain the whole knowledge of the environment to perform their tasks. In the scenario of the workspace in Figure 5.2, the algorithm has found a path for each robot one by one, it planned the first path for *Robot<sub>2</sub>*, second path for *Robot<sub>3</sub>*, and third (last) path for *Robot<sub>1</sub>*. Each robot goes through a different path and in different directions that do not intersect with each other. Thus, there is no collision occurring because robots do not move together on the same edge in opposite directions, and do not pass crossroads simultaneously on the same graph vertex, and algebraic connectivity is away from zero ( $\lambda_2 > 0$ ). Let us consider the scenario of workspace environment in Figure 5.1 above with add three goals instead of one goal, to explain how the algorithm will work to avoid collision between robots, as represented in Figure 5.8.

Figure 5.8: Scenario of workspace for path planning

In this scenario we have three robots and three goals; we will first divide the workspace into two disconnected components of undirected weighted graph by using visibility graph such as Figure 5.9.

Figure 5.9: Example of two disconnected components undirected weighted graph

The graph  $G = (V, E)$  in Figure 5.8 consist of vertices  $V = \{v_1, \dots, v_{30}\}$  marked from  $S_i = (v_1 = R_1, v_{28} = R_2, v_3 = R_3)$  to  $(g_1 = v_{30} = g_2 = v_2 = g_3, v_{29})$ ,  $E = \{e_1, \dots, e_{69}\}$ , and there are six (6) polygonal obstacles ( $O_i = O_1, O_2, O_3, O_4, O_5, O_6$ ). Each robot has initial position ( $S_i$ ) and the goal position ( $g_i$ ), here we have three goals for three robots. The second smallest eigenvalue of the graph in Figure 5.9 has zero value ( $\lambda_2 = 0$ ), which means the graph is disconnected and have two connected components. The robots ( $R_1 = v_1, R_2 = v_{28}, R_3 = v_3$ ) exist in the first component that contains vertices:

$\{v_1, v_3, v_7, v_9, v_{11}, v_{13}, v_{15}, v_{17}, v_{19}, v_{28}, v_{29}\}$ , where vertex ( $v_{29} = g_3$ ) is a goal for robot three. Subsequently, robot three ( $R_3$ ) can find a way to reach its target, but robot one ( $R_1$ ) and robot two ( $R_2$ ) do not have paths to reach their targets. Whilst the second component contains vertices:

$\{v_2, v_4, v_5, v_6, v_8, v_{10}, v_{12}, v_{14}, v_{16}, v_{18}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}, v_{25}, v_{26}, v_{27}, v_{30}\}$ ,

vertices ( $v_2 = g_2, v_{30} = g_1$ ) are goals for robot one and robot two. When adding an edge between vertex  $v_6$  and vertex  $v_{14}$ ,  $\lambda_2$  increases to 0.0867, and this enables robot one ( $R_1 = v_1$ ) to find path to reach its target ( $g_1 = v_{30}$ ). Whereas if add two edges ( $v_8, v_{10}$ ) and ( $v_8, v_{17}$ ),  $\lambda_2$  increases to 0.1808, this allows robot three  $R_3 = v_3$  to find a path to reach its target ( $v_{29} = g_3$ ). Furthermore, when adding three edges together  $\{(v_2, v_{10}), (v_8, v_{20}), (v_{20}, v_{28})\}$ ,  $\lambda_2$  increases to 0.3472 and robot two ( $R_2 = v_{28}$ ) will find a path to reach its goal ( $g_2 = v_2$ ), see Figure 5.10.

Figure 5.10: The path planned for robot one and two using MRPPA.

If add all possible paths between the vertices of the graph, the second smallest eigenvalue increases, and this will create a strong connectivity in the graph, where  $\lambda_2 = 6.3802$ . By using Dijkstra's algorithm then, the safe shortest paths are found, and it is shown in Fig. 5.11.

Figure 5.11: The shortest paths for three robots using Dijkstra's algorithm.

The MRPPA has planned a path for each robot, first path planned for  $R_1$ :  $\{R_1 = v_6 \rightarrow v_{14} \rightarrow v_{25} \rightarrow v_{18} \rightarrow v_{24} \rightarrow v_{30}\}$ , second path for  $R_3$ :  $\{R_3 = v_3 \rightarrow v_{10} \rightarrow v_8 \rightarrow v_{17} \rightarrow v_{13} \rightarrow v_{29}\}$ , and third (last) path for  $R_2$ :  $\{R_2 = v_{28} \rightarrow v_{21} \rightarrow v_8 \rightarrow v_{10} \rightarrow v_2\}$ . Although there is intersection (crossroad) between the path planned of robot one ( $R_1$ ) and the path planned of robot three ( $R_3$ ), and also opposite directions on the graph edges (straight roads) between the

path planned of robot three ( $R_3$ ) and the path planned of robot two ( $R_2$ ), but no collision happens, because the algorithm has planned a path for each robot sequentially (one by one). Hence, when planning the next path, it considers all the paths that have already been planned to prevent collisions and keep  $\lambda_2 > 0$ . There is a crossroad when robot one ( $R_1$ ) will pass the edge ( $v_6, v_{14}$ ), and robot three ( $R_3$ ) will pass the edge ( $v_{10}, v_8$ ), but no collision occurs because robot one will pass before robot three. The arrival time ( $T_{R_n} = w_i + w_{ij}$ ) of robot one when passed the vertex ( $v_6$ ) is:  $T_{R_1} = w_1 + w_{(1,6)} = 2$ , and when passed the vertex ( $v_{14}$ ):  $T_{R_1} = w_6 + w_{(6,14)} = 4$ . Whereas the arrival time of robot three when passed the vertex ( $v_{10}$ ) is:  $T_{R_3} = w_3 + w_{(3,10)} = 4$ , and when passed the vertex ( $v_8$ ):  $T_{R_3} = w_{10} + w_{(10,8)} = 7$ . Consequently,  $T_{R_1} < T_{R_3}$  (this means that the arrival time of robot one ( $R_1$ ) to the vertex ( $v_{14}$ ) is shorter than the arrival time of robot three ( $R_3$ ) to the vertex ( $v_8$ ), because the distance (edge weight) that robot one ( $R_1$ ) has passed the vertex ( $v_6$ ) = 2 is less than the distance (edge weight) that robot three ( $R_3$ ) has passed the vertex ( $v_{10}$ ) = 4, thus when robot one arrived at the vertex ( $v_{14}$ ) = 4, robot three will arrive at the vertex ( $v_{10}$ ), for this reason no collision occurs and change of the edge weight is not necessary. If  $T_{R_1} > T_{R_3}$ , then the collision is possible (i.e., if arrival time of robot one on the vertex ( $v_6$ ) = 4), then the change of the edge weight is necessary to avoid collision. Also, there are opposite directions (straight roads) on the edge ( $v_8, v_{10}$ ) between robot three ( $R_3$ ) and robot two ( $R_2$ ).  $R_3$  will pass the edge earlier than the  $R_2$ , where  $T_{R_3} = \{(w_{10} + w_{10,8}) = (4 + 3) = 7\}$ , and  $T_{R_2} = \{(w_8 + w_{8,10}) = (9 + 3) = 12\}$ . Thus, the arrival time of robot three when passed the edge ( $v_8, v_{10}$ ) shorter than the arrival time of robot two, due to the distance that robot three has passed to arrive at the vertex ( $v_8 = 7$ ) is less than the distance that robot two ( $R_2$ ) has passed the vertex ( $v_8$ ) = 9. Thus,  $(w_8 < T_{R_2}) \wedge (w_{10} + w_{(10,8)} < T_{R_2})$ , then  $T_{R_2} > T_{R_3}$ .

In addition, there is a crossroad on the vertex ( $v_8$ ), where  $T_{R_3} = w_{10} + w_{10,8} = 7$ , and  $T_{R_2} = w_{21} + w_{21,8} = 9$ , hence  $T_{R_3} < T_{R_2}$  the arrival time of robot three to the vertex ( $v_8$ ) before the robot two. Accordingly, there is no need to change the edge weight since no collision occurs. If  $T_{R_3} > T_{R_2}$ , then the collision is happening, so the change of the edge weight is important to collision avoidance.

Let us now change the workspace environment in Figure 5.1 to the simple workspace environment to see how the MRPPA will work to find the shortest safe paths for a team of

multi robots. Assume we have three robots and three goals in a simple workspace environment such as Figure 5. 12.

Figure 5.12: Simple workspace environment<sup>3</sup>

To apply the MRPPA, we will first represent the workspace as an undirected weighted graph, and then divide it into two disconnected components of undirected weighted graph by using a visibility graph such as Figure 5.13.

.

Figure 5.13: Two disconnected components undirected weighted graph

The graph  $G = (V, E_j)$  in Figure 5.12 consists of vertices  $V = \{v_1, \dots, v_{32}\}$ , where  $S_i = (R_1 = v_{24}, R_2 = v_{19}, R_3 = v_{31})$  are robot initial positions,  $(g_1 = v_{11} = g_2 = v_1 = g_3, v_{20})$  are goals positions,  $E = \{e_1, \dots, e_{70}\}$ , and there are five polygonal obstacles  $(O_i = O_1, O_2, O_3, O_4, O_5)$ . The second smallest eigenvalue of the graph in Figure 5.12 has zero value ( $\lambda_2 = 0$ ), because the graph has two disconnected components. The robots  $(R_1 = v_{24}, R_2 = v_{19}, R_3 = v_{31})$  exist in the component that contains on the vertices  $\{v_2, v_4, v_6, v_8, v_{10}, v_{12}, v_{14}, v_{16}, v_{18}, v_{19}, v_{20}, v_{22}, v_{24}, v_{26}, v_{27}, v_{28}, v_{29}, v_{30}, v_{31}, v_{32}\}$ , vertex  $(v_{20} = g_3)$  is a goal for robot three. Consequently, robot three ( $R_3$ ) has a path to reach

<sup>3</sup> The environments presented in this figure are part of MATLAB 'exampleMaps', <https://www.mathworks.com>

its target, whilst robot one ( $R_1$ ) and robot two ( $R_2$ ) have their targets in the second component, which consist of the vertices  $\{v_1, v_3, v_5, v_7, v_9, v_{11}, v_{13}, v_{15}, v_{17}, v_{19}, v_{21}, v_{25}\}$ , vertices ( $v_{11} = g_1, v_1 = g_2$ ) are goals for robot one and robot two. If add an edge between vertex  $v_{10}$  and vertex  $v_{15}$ , then robot one ( $R_1 = v_{24}$ ) can find a path to reach its target ( $g_1 = v_{11}$ ), and from this  $\lambda_2$  increases to 0.5212. Also, if add the edges  $(v_1, v_8)$ ,  $(v_8, v_{17})$ , and  $(v_{17}, v_{19})$  this enables robot three ( $R_3 = v_{19}$ ) to find a path to reach its target ( $v_1 = g_2$ ), and  $\lambda_2$  increases to 1.2747, see Figure 5.14.

Figure 5.14: Paths planned for each robot by using MRPPA.

Additionally, when adding all possible edges between the vertices of the graph, this will create a strong connectivity in the graph, and  $\lambda_2$  jumps to 2.5907. By using Dijkstra's algorithm then, shortest paths are found, and it is shown in Fig. 5.15.

Figure 5.15: The shortest paths for three robots by using Dijkstra's algorithm.

The MRPPA planned the first path for robot three ( $R_3$ ):  $\{R_3 = v_{31} \rightarrow v_{28} \rightarrow v_{10} \rightarrow v_{16} \rightarrow v_{20}\}$ , second path for robot one ( $R_1$ ):  $\{R_1 = v_{24} \rightarrow v_{10} \rightarrow v_{15} \rightarrow v_{11}\}$ , and third path for robot two ( $R_2$ ):  $\{R_2 = v_{19} \rightarrow v_{17} \rightarrow v_8 \rightarrow v_1\}$ . There is obviously a crossroad between the robot one ( $R_1$ ) and the robot three ( $R_3$ ) when they passed the vertex ( $v_{10}$ ), but no collision happen since the robot one will pass earlier than the robot three, the arrival time of the robot one when passed the vertex ( $v_{10}$ ):  $T_{R_1} = w_{24} + w_{(24,10)} = 5$ . While the arrival time of the robot three when passed the vertex ( $v_{10}$ ):  $T_{R_3} = w_{28} + w_{(28,10)} = 8$ . Accordingly,  $T_{R_3} > T_{R_1}$  (this means that the arrival time of robot one ( $R_1$ ) to the vertex ( $v_{10}$ ) is shorter than the arrival time of the robot three ( $R_3$ ) to the vertex ( $v_{10}$ ), because the distance (edge weight) that robot one has passed to the vertex ( $v_{10}$ ) = 5 less than the distance (edge weight) that robot three has passed to the vertex ( $v_{10}$ ) = 8, so robot one will leaved the vertex ( $v_{10}$ ) before robot three arrive to the vertex ( $v_{10}$ ), therefore no collision occurs and change of the edge weight is not necessary. Besides, there is intersection between a path planned of robot one ( $R_1$ ) when passed the edge ( $v_{10}, v_{15}$ ) and a path planned of robot two ( $R_2$ ) when passed the edge ( $v_{17}, v_8$ ), in this robot two will arrived at the vertex ( $v_8$ ) before robot one passed the edge ( $v_{10}, v_{15}$ ), and  $T_{R_2} = w_{17} + w_{(17,8)} = 5$  and  $T_{R_1} = w_{10} + w_{(10,15)} = 10$ , ( $w_{10} > T_{R_2}$ )  $\wedge$  ( $w_{10} + w_{(10,15)} > T_{R_2}$ ), then  $T_{R_1} > T_{R_2}$ . Hence, no collision occurs, because the algorithm has controlled the arrival time of each robot by controlling the edges weight (distance), and keeping  $\lambda_2 > 0$ .

### 5.3 Result discussion

In the scenario of a workspace environment in the first scenario, the robot one ( $Robot_1$ ) has path to reach the goal ( $R_1 \rightarrow V_{15} \rightarrow V_{13} \rightarrow V_{17} \rightarrow V_{28}$ ), and the total distance is (24). However, this path is not short and optimal, thus when we chose two edges between the components of graph the algorithm found the shortest and optimal path for robot one ( $Robot_1$ :  $R_1 \rightarrow V_6 \rightarrow$



$V_{11} \rightarrow V_{17} \rightarrow V_{28}$ ), and the total distance is (20). This happen in the second scenario of work space, where robot three has way to reach its target: ( $R_3 \rightarrow V_4 \rightarrow V_7 \rightarrow V_{29}$ ), and total distance is (20) but instead this path the algorithm find shortest and optimal path for *Robot*<sub>3</sub>:  $R_3 \rightarrow V_3 \rightarrow V_{10} \rightarrow V_8 \rightarrow V_{17} \rightarrow v_{13} \rightarrow v_{29}$ , and total distance is (15). Accordingly, the MRPPA chooses path for each robot respectively to collisions avoidance.

In addition, the visibility graph method considers obstacles vertices in the map to be the vertices through which the robots can reach their required positions. It precedes to link the vertices that are visible with each other, where the visible vertices are vertices with the property that a straight line (edge, bridge, path, etc.) connecting them does not intersect with any obstacles. Therefore, the calculated paths contain a set of waypoints ( $\widehat{W}$ ), which also has the shortest length. These waypoints ( $\widehat{W}$ ) are determined like a series of consecutive points which begin from the lowest number of the first point to the goal number, the waypoints are given by  $\widehat{W} = \{w_0, \dots, w_n\}$ , where  $w_0$  is the start point and  $w_n$  is the goal point. Hence, waypoints are a set of vertices of obstacles. For this reason, the paths (path1, path2, path3) have the least distances because they contain a set of waypoints, which are a set of vertices of obstacles that are found by using a visibility graph with combination of Dijkstra's algorithm in a C-space (2D environment), for more information see the tables 5.1, 5.3, and 5.5 below. Also, these waypoints do not include the start points ( $R_1, R_2, R_3$ ) and the goal point (g), so they are always at specific vertices of obstacles, thus they are able to produce the shortest paths in terms of the Euclidean distance, the important condition for a path to have a lower Euclidean distance from starting point to goal point in C-space, where each waypoint is a vertex of an obstacle (O) [21]. For more understanding we will give an example to explain it.

**Example:** Suppose that a set of waypoints ( $\widehat{W}$ ) that contains on a sequence of point, which are not vertices of the obstacles in configuration space. Let  $V_2$  be the first such point in the sequence.  $V_1$  and  $V_3$  are the points directly before and after  $V_2$ , respectively. the vertex  $V_2$  will not be on the straight line  $V_1V_3$ , because otherwise  $V_2$  should not be a waypoint. Without losing generality, consider a path  $V_1V_2 + V_2V_3$  created by vertices ( $V_1, V_2, V_3$ ) as illustrated in Figure 5. 16, if there is no obstacle among  $V_1$  and  $V_3$ , then  $V_3$  must be the next waypoint after  $V_1$ . In addition, let we consider the vertices  $V_6$  and  $V_7$  in the sequence. It can be observed the following arguments from Figure 5.16.

$$\begin{aligned}
V_1V_2 + V_2V_3 &= V_1V_2 + V_2V_6 + V_6V_7 + V_7V_3 > (V_1V_4 + V_4V_6) + V_6V_7 + V_7V_3 \\
&= V_1V_4 + (V_4V_6 + V_6V_7) + V_7V_3 > V_1V_4 + (V_4V_5 + V_5V_7) + V_7V_3 \\
&= V_1V_4 + V_4V_5 + (V_5V_7 + V_7V_3) > V_1V_4 + V_4V_5 + V_5V_3
\end{aligned}$$

Figure 5.16: A scenario with two obstacles

The above arguments illustrate that the path  $V_1V_4 + V_4V_5 + V_5V_3$  created by vertices  $(V_1, V_4, V_5, V_3)$ , since  $V_1, V_4$  and  $V_5$  are the vertices of the obstacles, is less than the paths that consist of vertices (points), which are in the sequence [21].

The tables below will summarise the details of the waypoints and the paths planned of each robot in different scenarios of the workspace environment in Figures 5.6, 5.11, and 5.15 respectively.

**Table 5.1:** The waypoints of the scenario of Figure 5.6 generated by MRPPA.

<i>waypoints (path1)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path2)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path3)</i>	<i>x</i>	<i>y</i>
$w_0$	8.7	0.3	$w_0$	11	1	$w_0$	2	1
$w_1$	7.8	4	$w_1$	10	5	$w_1$	4.5	4
$w_2$	5	6	$w_2$	9	7.5	$w_2$	3.3	7
$w_3$	4.5	12.6	$w_3$	10	10	$w_3$	3.2	9.5
$w_4$	5	16	$w_4$	10.8	12.9	$w_4$	5	16
-	-	-	$w_5$	5	16	—	-	-

Figure 5.17 shown waypoints and paths planned of Figure 5.6.

Figure 5.17: Waypoints paths planned by MRPPA.

**Table 5.2:** The calculated paths planned (path1, path2, path3) for each robot in Figure 5.6

Initial and endpoint	Shortest Path	Total distance
$Robot_1$ to Goal	$R_1 \rightarrow V_6 \rightarrow V_{11} \rightarrow V_{17} \rightarrow v_{28}$	$P_1=5+3+6+6=20$
$Robot_2$ to Goal	$R_2 \rightarrow V_{10} \rightarrow V_8 \rightarrow V_{21} \rightarrow v_{28}$	$P_2=4+5+3+8=20$
$Robot_3$ to Goal	$R_3 \rightarrow v_{25} \rightarrow V_{18} \rightarrow V_{24} \rightarrow V_{22} \rightarrow v_{28}$	$P_3=5+3+3=5+4=20$

Note that, the distance from  $R_1 \rightarrow V_6 = 5$ ,  $V_6 \rightarrow V_{11} = 3$ ,  $V_{11} \rightarrow V_{17} = 6$ , and  $V_{17} \rightarrow v_{28} = 6$ . Thus, the total distance is  $5+3+6+6=20$  which is the shortest distance that found by Dijkstra's algorithm.

**Table 5.3:** The waypoints generated by MRPPA in Figure 5.11

waypoints (path1)	x	y	waypoints (path2)	x	y	waypoints (path3)	x	y
$w_0$	3	1	$w_0$	3.4	16	$w_0$	8.7	0.3
$w_1$	4.5	4	$w_1$	4.5	12.6	$w_1$	7.8	4
$w_2$	7.5	6	$w_2$	5	6	$w_2$	5	6

$w_3$	10	5	$w_3$	7.8	4	$w_3$	3.2	9.5
$w_4$	9	7.5	$w_4$	11	1	$w_4$	2	8.5
$w_5$	10	10	—	-	-	$w_5$	0.5	9
$w_6$	12	12.7	—	-	-	—	-	-

Figure 5.18 shown waypoints and paths planned in Figure 5.11.

Figure 5.18: Waypoints paths planned by MRPPA.

**Table 5.4:** The calculated paths planned (path1, path2, path3) for each robot in Figure 5.11

<b>Initial and endpoint</b>	<b>Shortest Path</b>	<b>Total distance</b>
$Robot_1$ to Goal	$R_1 \rightarrow V_6 \rightarrow V_{14} \rightarrow V_{25} \rightarrow v_{18} \rightarrow v_{24} \rightarrow v_{30}$	$P_1=2+2+3+2+4+5=18$
$Robot_2$ to Goal	$R_2 \rightarrow V_{21} \rightarrow V_8 \rightarrow V_{10} \rightarrow v_2$	$P_2=3+6+3+8=20$
$Robot_3$ to Goal	$R_3 \rightarrow v_{10} \rightarrow V_8 \rightarrow V_{17} \rightarrow V_{13} \rightarrow v_{29}$	$P_3=4+3+5=2+1=15$

**Table 5.5:** The waypoints generated by MRPPA of simple workspace in Figure 5.15

<b>waypoints (path1)</b>	<b>x</b>	<b>y</b>	<b>waypoints (path2)</b>	<b>x</b>	<b>y</b>	<b>waypoints (path3)</b>	<b>x</b>	<b>y</b>
------------------------------	----------	----------	------------------------------	----------	----------	------------------------------	----------	----------

$w_0$	6.20	1.38	$w_0$	10.6	4	$w_0$	9.50	11.44
$w_1$	10	3.5	$w_1$	7	6.5	$w_1$	6.80	9.500
$w_2$	7	6.5	$w_2$	4.5	9.5	$w_2$	5	6.5
$w_3$	10	6.5	$w_3$	1.50	10.5	$w_3$	1	3.51
$w_4$	11.65	11.29	$w_4$	-	-	-	-	-

Figure 5.19 shown waypoints and paths planned in Figure 5.15.

Figure 5.19: Waypoints paths planned by MRPPA.

**Table 5.6:** The calculated paths planned (path1, path2, path3) for each robot in Figure 5.15

Initial and endpoint	Shortest Path	Total distance
$Robot_1$ to Goal	$R_1 \rightarrow V_{10} \rightarrow V_{15} \rightarrow V_{11}$	$P_1=5+5+5=15$
$Robot_2$ to Goal	$R_2 \rightarrow V_{17} \rightarrow V_8 \rightarrow v_1$	$P_2=3+2+9=14$
$Robot_3$ to Goal	$R_3 \rightarrow v_{28} \rightarrow V_{10} \rightarrow V_{16} \rightarrow v_{20}$	$P_3=3+5+4+3=15$

#### 5.4 Advantages and Disadvantages of the Visibility graph method

The advantage of the visibility graph method is that the calculated path has the shortest length (lowers distance) if it is coupled with Dijkstra's algorithm. It is also complete which means it always produces a path if one exists. This feature is essential because it will guarantee that the robot will accomplish a task in a scenario where path creation is feasible (i.e., it guarantees that the robot will find the shortest path to its target). On the other hand, the Visibility Graph

produces paths that contain waypoints that pass during obstacles' vertices. Thus, the drawback of this method is that it plans a path that forces the robots to cross and to move as near the obstacles. This is not safe and might lead to collisions among robots and obstacles. Besides, its calculation time increases dramatically with the growth of obstacles. To address these problems, an algorithm called the Central Algorithm has been designed. This algorithm can create paths relatively fast and is convenient for path planning applications in obstacle-rich environments because it is considered a narrow area towards the goals positions, for this reason, it uses a small set of obstacles and vertices when the paths are computed.

### **5.5 Central Algorithm (CA)**

To use the Central algorithm, the robot's initial and goals positions, and the number of obstacles vertices and position are given. The algorithm generates a straight line connecting the start position and target position that is called Central Baseline path (CB), which is a simple and short path between the two these positions. The Central Baseline is not collision-free, it pass-through the obstacles, hence, the obstacles are defined based on the CB paths, and the intersection points among the obstacles and the Central Baseline paths [21][33]. The CA generates a set of waypoints in free C- space around obstacles, these waypoints generated from each vertex of each obstacle that intersect with the central baseline path (CB). Waypoints are used to establish a partial visibility graph network from a specific area of the configuration space. The waypoints compute so that the line connecting the waypoint and its corresponding intersection point (vertex of obstacle) is orthogonal to the original central baseline path. Each Central baseline path has two intersection points, and each point has two waypoints; hence the total number of computed waypoints is four. The part of the Visibility graph is established based on the waypoints that are defined by the central baseline paths, which joins the initial and goals positions. CB allows using less set of obstacles during the path computation rather than the entire obstacles (O) as used by VG. Sequential waypoints are linked together to create multiple possible collision-free paths for the robots around the obstacles. It then finds shortest paths by using Dijkstra's algorithm.

### 5.5.1 CA Algorithm

**Inputs:** Start ( $S$ ), goal ( $g$ ), polygonal obstacles, Central Baseline  $CB$

**Outputs:** Waypoint  $\hat{W}$ , Visibility graph,  $VG$

Our algorithm mainly includes the following steps:

1. **Create** a Central Baseline ( $CB$ ) from starting point  $s$  to the goal point  $g$ .
2. **Determine** the intersection points between the central baseline ( $CB$ ) and obstacles.
3. **Construct** a set of waypoints, from each vertex of each obstacle that intersect with the  $CB$
4. **Calculate** the waypoints for those intersection points from each obstacle that lies on the  $CB$ , and their extensions including  $s$  start and  $g$  goal.
5. **Establish** the partial of the Visibility graph  $VG$  based on the waypoints ( $\hat{W}$ ) that are defined by the Central Baseline ( $CB$ ) and obstacles ( $O$ ).
6. **Find** paths from start( $s$ ) to goal ( $g$ ) by using Dijkstra's algorithm.
7. **End:** paths calculated  $\hat{W} = \{w_0, \dots, w_n\}$ , where  $w_0$  = start point and  $w_n$  = goal point.

Figure 5.20: The process of Central algorithm (CA)

### 5.5.2 Path Planning Using CA

To illustrate how the algorithm works, consider the previous scenario of Figure 5.11, that consists of three robots each one has start positions  $S_i = (R_1, R_2, R_3)$ , goals positions  $g_i = (g_1, g_2, g_3)$ , and six (6) polygonal obstacles ( $O_i = O_1, O_2, O_3, O_4, O_5, O_6$ ). First, we create the central baseline ( $CB$ ) that are joining the start positions to the goal positions, which are

highlighted in red colour. Then the obstacles that overlap with CB are determined. As a result, the intersection points between the obstacles and the central baseline paths are known, which are also highlighted in grey colour, hence it is required to calculate the waypoints for those intersection points. The waypoints generated from each vertex of each obstacle intersect with the central baseline path (CB), calculated for each half of the map, and the lines linking waypoints and their intersection points (vertex of obstacle) are orthogonal to the original Central Baselines paths, as shown in Figure 5. 21. Since there are two intersection points lying along each CB path, each point having two waypoints in each half of the map, so the total of four waypoints are computed, which are marked in blue colour. Successive waypoints are connected with each other to create multiple possible collisions-free paths for robots around obstacles, see Figure 5.21.

Figure 5.21: The steps of Central algorithm (CA) in workspace environment

**Note that**, for each robot there are two possible collision free paths. Each path passes through two waypoints, as shown in Figure 5.21.

Figure 5.22: Two possible paths for robots (1, 2, and 3) by Central algorithm



In addition, the CB those linking the start ( $s_i$ ) and goals ( $g_i$ ) positions intersect with some of the obstacles, which allows using less set of obstacles when the path computation rather than all obstacles ( $O_i$ ), and also that results in an even least number of vertices. Besides, the feature of this method is that the distance between the obstacles and the generated path can be controlled. It also allows robots to approach obstacles within an appointed and acceptable distance, whilst minimising the travelled paths based on the complexity of obstacles, and as well as the information about the robots' positions. Figure 5.23 illustrates waypoint calculation, where the Central Baseline joining the start point  $S$  and goal point  $g$  intersect an obstacle, and its edges are shown as dotted lines, at point  $u(x, y)$ . It is required to calculate a point  $u'(x', y')$  that creates an orthogonal line to the Central Baseline at point  $u(x, y)$ .

Figure 5.23: Computation of waypoint ( $u'(x', y')$ ) at an intersection point ( $u(x, y)$ )

The normal distance between the point  $u'(x', y')$  and the straight line must exceed the maximum distance  $m$  between any node in that object and the Central Baseline (CB) by a safe distance( $\delta$ ). The normal distance  $m$  is computed as:

$$m = \frac{|(g-s) \times (s-v)|}{|g-s|}, \text{ Or}$$

$$m = \left| \frac{ax + by + c}{\sqrt{a^2 + b^2}} \right|$$

The waypoint  $u'$  is computed on either side of the intersection point by given:

$$u' = u \pm (m + \delta) \begin{bmatrix} -\cos(90 - \varphi) \\ \sin(90 - \varphi) \end{bmatrix}$$

$$\text{Or } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \pm (m + \delta) \begin{bmatrix} -\cos(90 - \varphi) \\ \sin(90 - \varphi) \end{bmatrix}$$

In the given equations above  $u, u', v, s,$  and  $g$  are position vectors for the intersection point, waypoint, vertex, start, and goal locations. Whereas  $a, b,$  and  $c$  are constants and  $x, y$  are variables. Also, the angle  $\varphi$  between (CB) and the positive  $x - axis$  is acute, so  $\sin \varphi$  is positive, where  $\sin \varphi = \frac{opposite}{hypotenuse}$ , and  $\cos \varphi = \frac{adjacent}{hypotenuse}$

Angle  $\varphi$  is the inclination of the line with slope, and since the slope defines as:  $M = opposite/adjacent$ , so  $\tan \varphi = \frac{opposite}{adjacent} = M$

Figure 5.24: Diagram illustrating  $\tan \varphi = M$

Now after waypoints calculation, the part of the visibility graph is established based on the obstacles that are defined by CB. In the Figure 5.21 scenario the CB allows using three of obstacles during the path computation rather than six obstacles (O). If we represent the scenario of Figure 5.21 as undirected weighted graph, the set of obstacles that lie along CB with vertices  $\{4, 5, 6, 10\}$ ,  $\{7, 9, 11, 13\}$ , and  $\{8, 12, 14, 16, 18\}$ , which will construct a visibility graph network around them. The resultant VG using CA is illustrated in Figure 5.25. There are several paths generated by the path planning part of the algorithm, which will evaluate based on the weight function.

Figure 5.25: Visibility graph network created by Central algorithm.

Finally, the Dijkstra algorithm will be employed to find the safe and shortest paths that is referred to as the optimal paths, see Figure 5.25

Figure 5.26: Shortest paths for three robots calculated by using Dijkstra's algorithm

The algorithm has find paths for each robot, where path of robot one ( $Robot_1$ ):  $\{R_1 = v_1 \rightarrow v_{40} \rightarrow v_{37} \rightarrow v_{33} \rightarrow v_{30}\}$ , path for robot two ( $Robot_2$ ):  $\{R_2 = v_{28} \rightarrow v_{34} \rightarrow v_{38} \rightarrow v_{37} \rightarrow v_2\}$ , and third (last) path for robot three ( $Robot_3$ ):  $\{R_3 = v_3 \rightarrow v_{37} \rightarrow v_{38} \rightarrow v_{36} \rightarrow v_{29}\}$ . It clear that there is intersection (crossroad) between the path planned of robot one ( $R_1$ ) and the path planned of robot three ( $R_3$ ), in the vertex ( $v_{37}$ ) but no collision occurs because the robot three ( $R_3$ ) will passes the vertex ( $v_{37}$ ) before robot one ( $R_1$ ). The arrival time of the robot three when passed the vertex ( $v_{37}$ ):  $T_{R_3} = w_3 + w_{(3,37)} = 2$ , whereas, the arrival time of the robot one when passed the vertex ( $v_{37}$ ):  $T_{R_1} = w_{40} + w_{(40,37)} = 5$ . Consequently,  $T_{R_3} > T_{R_1}$  (this means that the arrival time of robot three ( $R_3$ ) to the vertex ( $v_{37}$ ) is greater than the arrival time of the robot one ( $R_1$ ), so the distance (edge weight) that robot three ( $R_3$ ) has passed to the vertex ( $v_{37}$ ) = 2 less than the distance (edge weight) that robot one ( $R_1$ ) has passed to the vertex ( $v_{37}$ ) = 5, thus when robot one arrived to the vertex ( $v_{37}$ ) = 4, the robot three already left the vertex ( $v_{37}$ ), for this reason no collision occurs and change of the edge weight is not necessary. Also, there is opposite directions (straight roads) on the edge ( $v_{37}, v_{38}$ ) between robot three ( $R_3$ ) and robot two ( $R_2$ ), the  $R_3$  will pass the edge earlier, than the  $R_2$ , where  $T_{R_3} = \{(w_{37} + w_{37,38}) = (2 + 2) = 4\}$ , and  $T_{R_2} = \{(w_{38} + w_{38,37}) = (6 + 2) = 8\}$ , the arrival time of robot three when passed the edge faster than the arrival time of robot two, due to the distance that robot three has passed to arrival the vertex ( $v_{38} = 4$ ) less than the distance that robot one ( $R_2$ ) has passed to the vertex ( $v_{38}$ ) = 6, thus  $(w_{38} > T_{R_3}) \wedge (w_{37} + w_{(37,38)} > T_{R_2})$ , hence  $T_{R_2} > T_{R_3}$  the arrival time of robot three to the vertex ( $v_{38}$ )

before the robot two. Therefore, it is not essential to change the edge weight since no collision happens.

**Table 5.7:** The waypoints by the CA of workspace in Figure 5.26

<i>waypoints (path1)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path2)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path3)</i>	<i>x</i>	<i>y</i>
$w_0$	3.5	1	$w_0$	8	0.8	$w_0$	3.4	13
$w_1$	3.9	5.1	$w_1$	7.3	4.7	$w_1$	4.8	9
$w_2$	7.3	4.7	$w_2$	4.90	5.90	$w_2$	4.90	5.90
$w_3$	8.3	6.8	$w_3$	3.6	7.4	$w_3$	7.3	4.7
$w_4$	10	11.5	$w_4$	1.9	9.7	$w_4$	8.6	2.5

Figure 5.27: Waypoints paths planned by CA of workspace

**Table 5.8:** The calculated paths planned (path1, path2, path3) for each robot in Figure 5.22

<b>Initial and endpoint</b>	<b>Shortest Path</b>	<b>Total distance</b>
<i>Robot<sub>1</sub> to Goal</i>	$R_1 \rightarrow v_{40} \rightarrow v_{37} \rightarrow v_{33} \rightarrow v_{30}$	$P_1=3+2+1+4=10$
<i>Robot<sub>2</sub> to Goal</i>	$R_2 \rightarrow v_{34} \rightarrow v_{38} \rightarrow v_{37} \rightarrow v_2$	$P_2=4+2+2+1= 9$
<i>Robot<sub>3</sub> to Goal</i>	$R_3 \rightarrow v_{37} \rightarrow v_{38} \rightarrow v_{36} \rightarrow v_{29}$	$P_3=2+2+1+3= 8$

Furthermore, we also applied the Central algorithm on the scenario of a simple workspace environment in Figure 5.12, to illustrate how the algorithm is worked and its impact on the different environments. We first created the Central Baseline for each robot from start points to goals points that were highlighted in red colour. Then we define the obstacles that overlap with CB, then the intersection points between the obstacles and the Central Baseline are determined, which are also highlighted in blue colour, hence it is required to calculate the waypoints for those intersection points, see Figure 5.28.

Figure 5.28: Central algorithm in simple workspace environment

Waypoints have been calculated and marked by small blue points; thus, the part of the visibility graph is established based on the obstacles that are defined by CB. If we represent the scenario of Figure 5.28 as an undirected weighted graph, the set of obstacles that lie along CB with vertices  $\{8, 10\}$ ,  $\{15, 17\}$ , and  $\{16, 22\}$ . The resultant VG using CA is illustrated in Figure 5.29. There are several paths generated by the path planning part of the algorithm, which will evaluate based on the weight function.

Figure 5.29: Visibility graph network created by CA

Finally, Dijkstra algorithm is employed to find the safe and shortest paths that referred to as the optimal paths, see Figure 5.30

Figure 5.30: Shortest paths for three robots calculated by using Dijkstra's algorithm

Dijkstra's algorithm has found paths for each robot, where path of robot one ( $Robot_1$ ):  $\{R_1 = v_{24} \rightarrow v_{35} \rightarrow v_{36} \rightarrow v_{11}\}$ , path for robot two ( $Robot_2$ ):  $\{R_2 = v_{19} \rightarrow v_{40} \rightarrow v_{38} \rightarrow v_{32} \rightarrow v_1\}$ , and third (last) path for robot three ( $Robot_3$ ):  $\{R_3 = v_{31} \rightarrow v_{33} \rightarrow v_{37} \rightarrow v_{44} \rightarrow v_{20}\}$ . It clear that there are intersections (crossroads) between the path planned of robot one ( $R_1$ ) and the path planned of robot three ( $R_3$ ), and also between the path planned of robot one ( $R_1$ ) and the path planned of robot two ( $R_2$ ) but no collision occurs; because the algorithm controlled the arrival time of each robot by controlling the edges weight (distances) to avoid collisions between robots.

**Table 5.9:** The waypoints by the Central algorithm of simple workspace in Figure 5.30

<i>waypoints (path1)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path2)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path3)</i>	<i>x</i>	<i>y</i>
$w_0$	9.4	3.8	$w_0$	8.5	11.44	$w_0$	7.30	3
$w_1$	9.5	6.8	$w_1$	7.9	9.5	$w_1$	7.4	5
$w_2$	6.9	7.3	$w_2$	6.3	9	$w_2$	9.5	6.8
$w_3$	4.6	8.5	$w_3$	4	6	$w_3$	10.2	7.5
$w_4$	3.5	10.5	$w_4$	3	3.51	$w_4$	11.65	10.29

Figure 5.31: Waypoints paths planned by CA

**Table 5.10:** The calculated paths planned (path1, path2, path3) for each robot in Figure 5.25

Initial and endpoint	Shortest Path	Total distance
$Robot_1$ to Goal	$R_1 \rightarrow v_{37} \rightarrow v_{35} \rightarrow v_{36} \rightarrow v_{11}$	$P_1=3+1+2+3=9$
$Robot_2$ to Goal	$R_2 \rightarrow v_{40} \rightarrow v_{38} \rightarrow v_{32} \rightarrow v_1$	$P_2=1+1+2+3=7$
$Robot_3$ to Goal	$R_3 \rightarrow v_{33} \rightarrow v_{37} \rightarrow v_{44} \rightarrow v_{20}$	$P_3=4+3+1+2=10$

## 5.6 Performance Comparison

Comparison between the performance of VG and CA in terms of the resulting paths planned and computation time. As formerly mentioned, one drawback of the visibility graph method is that the calculation time is related to the number of obstacles. The greater number of obstacles in the C-space causes a longer time to find paths (i.e., the computation time increases dramatically with the growth of obstacles). For example, the scenario of workspace in Figures 5.11, and 5.15 consists of (6 obstacles, and 99 edges), and (5 obstacles, and 85 edges) respectively. Whilst the scenario of workspace in Figures 5.26, and 5.30 considered just three obstacles and they contain 70, and 67 edges respectively. Accordingly, the calculation time to find the paths are less in figure 5.26 and 5.30 because it uses just three obstacles for computed paths. Whereas the calculation time to find paths in Figures 5.11, and 5.15 is great, due to the increased number of obstacles. Besides, CA in both scenarios of Figures 5.26 and 5.30 provides the shortest collision free paths while maintaining a safe distance from obstacles, whereas the paths planned by VG method in Figures 5.11, and 5.15, are close to the obstacles, see Figure

5.32, where the paths planned by CA highlighted as red while the paths planned by VG marked as orange colour.

Figure 5.32: The paths planned by CA (red) and VG (orange)

**As a result**, the main reason why the Central algorithm is more efficient than the conventional visibility graph method, resulting in general solutions for various scenarios and generates visibility lines (edges) for calculation of the paths because it uses a smaller number of obstacles during the calculation of the paths. It also provides the shortest collision-free paths whilst maintaining a safe distance from the obstacles. The shorter the paths, the safer the robots are from collisions. On the other hand, the algorithm has a drawback, that is: it considers just the obstacles that intersect with the CB but ignores other obstacles in the workspace. In addition, the algorithm generates a collision free path based on the complexity of the obstacles and the confirmation of information about the obstacles and robots positions, hence may not generate free-collisions paths in other situations of different maps. For example, the algorithm has found the shortest safe paths around the obstacles as shown in Figures 5.26, and 5.30 whilst maintaining a safe distance away from the obstacles. If we add other obstacles to the scenario of workspace in Figure 5.26, the paths of robots will not change unless the added obstacle interferes with those paths, as shown in Figure 5.33 below.



Figure 5.33: The impact of placing obstacles on the generated paths

The algorithm redirects the robots, when obstacles intersect the shortest collision free paths, to find the next shortest paths, see Figure 5.33. Both Figures 5.33 and 5.34 clarify the impact of adding other obstacles.

Figure 5.34: The algorithm reroutes the robots for find next shortest paths.

Consequently, the Central algorithm can be improved to address this problem.

### **5.7 Optimisation Central algorithm (OCA)**

As has been demonstrated in Section 5.4, the Central algorithm is relatively faster in planning paths in comparison with the visibility graph method. This is since the CA uses a smaller

number of obstacles during calculation of the paths. To optimise the CA to generate planning paths which are safer and not close to the obstacles, an improvement to CA is proposed in this section. It has been formerly emphasised that the Central algorithm employs CB that joins the start locations to g goals locations to determine a set of obstacles for paths calculation in an environment with many obstacles. This makes CA insensitive to the number of obstacles in the workspace environment. Hence it may not generate collision-free paths in other situations or in different scenarios. To address this, we need to expand the size of obstacles in the workspace environment by a certain distance before the paths are planned, to provide a safe path and avoid collision near the obstacles. This distance is called the Safety Distance that could be used to remove the drawback of the Central algorithm.

### 5.7.1 Safety Distance

Safety distance ( $D_s$ ) is important for the safety of the robots to be capable of safely passing the planned paths without colliding with any obstacles in the C-space and to ensure these paths are safe for the robots so that they can traverse through the vertices of obstacles in different scenarios of workspaces, even with added new obstacles. A safe distance produces general solutions and acceptable results for different maps because it generates waypoints around obstacles in the C-space. To demonstrate the implementation of the safety distance in a path planning process, consider a scenario of 5.11 that consists of six obstacles as shown in Figure 5.35.

Figure 5.35: The obstacles in workspace with safety distance.

Next, Figure 5.36 shows the expansion of the obstacle by using the safety distance ( $D_s$ ).

Figure 5.36: The expanded obstacle with safety distance

The safety distance  $D_s$  is calculated as follow:

$$D_s = PQ = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Alternatively, the Cartesian coordinates  $(x_1 - x_0)$  and  $(y_1 - y_0)$  can be converted to the polar coordinates  $r$  and  $\theta$  using the trigonometric functions sine and cosine as Figure 5.36.

Figure 5.37: A diagram illustrating the relationship between polar and Cartesian coordinates.

where  $r = PQ = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$ , so  $(x_1 - x_0) = r \cos \theta$ , and  $(y_1 - y_0) = r \sin \theta$ . The safety distance is calculated as follow:

$$r = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

$$r = \sqrt{(r \cos \theta)^2 + (r \sin \theta)^2}$$

### 5.7.2 Path Planning Using OCA

The partial visibility graph using the Optimisation Central algorithm is depicted in Figure 5.38. The path planning starts with calculating initial paths by using OCA as shown in red in Figure 5.38, where the red point represents the starting positions whilst the green points denote the goals positions.

Figure 5.38: The partial of visibility graph network by the OCA

As demonstrated, the safety distance produces many safe paths for each robot. It is changing the paths from semi collision-free to fully collision-free. In addition, even if we add other obstacle, the paths of robots will not change, because with the Safety distance the placing of obstacles will not affect the generated paths. Finally, there are several paths generated by the path planning part of the algorithm, which will evaluate based on the weight function. Then, Dijkstra's algorithm is used to find paths from the constructed network as shown in Figure 5.39.

Figure 5.39: Paths planned and generated by the OC algorithm

Dijkstra's algorithm has find paths for each robot, where path of robot one ( $Robot_1$ ):  $\{R_1 = v_6 \rightarrow v_{12} \rightarrow v_{30}\}$ , path for robot two ( $Robot_2$ ):  $\{R_2 = v_{16} \rightarrow v_{18} \rightarrow v_2\}$ , and third (last) path

for robot three ( $Robot_3$ ) :  $\{ R_3 = v_{10} \rightarrow v_8 \rightarrow v_{11} \rightarrow v_{29} \}$ . It clear that there are intersections (crossroads) between the path planned of robot one ( $R_1$ ) and the path planned of robot three ( $R_3$ ), and also between the path planned of robot one ( $R_1$ ) and the path planned of robot two ( $R_2$ ) but no collision occurs; because the algorithm controlled the arrival time of each robot by controlling the edges weight (distances) to avoid collisions between robots.

**Table 5.11:** The waypoints by the Optimisation Central algorithm

<i>waypoints (path1)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path2)</i>	<i>x</i>	<i>y</i>	<i>waypoints (path3)</i>	<i>x</i>	<i>y</i>
$w_0$	3.5	1	$w_0$	3.66	12.85	$w_0$	8.0	0.8
$w_1$	3.9	4.7	$w_1$	6.8	9.5	$w_1$	7.3	4.7
$w_2$	5.4	9.4	$w_2$	8.3	7.5	$w_2$	4.85	5.98
$w_3$	9.6	11.2	$w_3$	8.6	2.5	$w_3$	3.6	7.4
$w_4$	-	-	$w_4$	-	-	$w_4$	2.3	9.87

Figure 5.40: Waypoints paths planned by OCA

**Table 5.12:** The calculated paths planned (path1, path2, path3) for each robot by OCA

Initial and endpoint	Shortest Path	Total distance
$Robot_1$ to Goal	$R_1 \rightarrow v_6 \rightarrow v_{12} \rightarrow v_{30}$	$P_1=3+2+4= 9$
$Robot_2$ to Goal	$R_2 \rightarrow v_{16} \rightarrow v_{18} \rightarrow v_2$	$P_2=3+1+3= 7$
$Robot_3$ to Goal	$R_3 \rightarrow v_{10} \rightarrow v_8 \rightarrow v_{11} \rightarrow v_{29}$	$P_3=2+1+1+4= 8$

**Note that:** the paths planned generated by OCA are the shortest paths (i.e., have the least distances) in comparison with paths planned by visibility graph and are safer as well.

In summary, the Optimisation Central algorithm provides the shortest collision-free paths while maintaining a safe distance from obstacles, which makes it safer. Also, it employs a smaller number of obstacles, and this reduces the computational complexity of roadmap approaches, which means the calculation time decreases when calculating paths. All these features make it more efficient than the visibility graph method.

## 5.8 Conclusion

We have proposed an MPP Algorithm for 2D path planning based on the advantages of one of the most commonly used roadmap algorithms, the visibility graph method that combines with algebraic connectivity of the graph Laplacian (second smallest eigenvalue) as well as Dijkstra's algorithm to calculate (finding) the shortest safe paths for robots from the start positions to the goals positions consequently (one by one) to avoid collisions with each other. VG is a path planning method that can produce optimal paths if they exist, especially if it is combined with Dijkstra's algorithm. However, the paths planned by the visibility graph method may not be collision-free, because it is planning paths that pass through the vertices of obstacles, and it is also computationally expensive when the environments are obstacles-rich. So, this chapter has proposed an algorithm called the Central algorithm that has been designed for 2D path planning based on the visibility graph method. This algorithm is used to find new paths that will be re-planned to collisions avoidance with obstacles. The algorithm can address the disadvantage of the visibility graph as it produces general solutions for different scenarios and plans collision-free paths in a computationally tractable way. The proposed algorithm, CA, is employed to find paths from a set of obstacles that are defined by the so-called central baseline (CB). CB is a simple line that joins the starting points and the goal points. All obstacles that interfere with

CB are the set of obstacles that will be used for paths planning. As the set consists of a relatively small number of obstacles, the paths are fast computed. The idea behind this approach is to produce optimal shortest paths that direct robots safely away from obstacles. Also, it reduces the computational complexity of roadmap approaches, where a less set of obstacles is considered for path calculation. This makes CA insensitive to the number of obstacles in the environment, which is its main drawback because it just considers the obstacles that intersect with CB, whilst it neglects other obstacles that may cause a collision due to, they may be near to paths planned. In addition, the algorithm generates collision-free paths based on the complexity of the obstacles and the confirmation of information about the obstacles and robots' positions, hence may not generate free-collisions paths in other situations of different scenarios or maps. Therefore, to ensure the completeness of the algorithm, an improvement to CA has been proposed to improve its performance to generate planning paths safer and not close to the obstacles. Another algorithm called Optimisation Central algorithm (OCA) has been proposed to address this problem, the size of obstacles has been expanded by a certain distance before the paths are planned. This distance is called the Safety Distance that could be used to optimize the drawback of the Central algorithm. Safety distance can produce general solutions and acceptable results for different maps, and it is planning safe paths for the robots in the C-space, so that they can traverse through the vertices of obstacles in different scenarios of workspaces without collisions, even with adding new obstacles. In addition, it uses a smaller number of obstacles, and this reduces the computational complexity when calculating paths in the environment workspace. All these advantages make the OCA more effective than the visibility graph method to reduce optimal and complete paths.

## **Chapter six**



## **6 Simulations and Experiments**

### **6.1 Introduction**

This chapter describes a software package that has been developed in this project for robot path planning, which serves two purposes: first, it validates the effectiveness of the proposed control algorithms; second, it executes and displays the algorithms in an intuitive way using MATLAB. This software package contains two strategies; each one has its own objective. The first strategy is used to implement the Multi-Robot Path Planning Proposed Algorithm (MRPPA), while the second executes the CA and OCA algorithms for finding safe shortest paths for robots within the environment. We implement and provide simulations and experiments using the Visibility Graph method in conjunction with Dijkstra's algorithm respectively. The chapter is arranged as follows. The software packages and their functionalities are introduced first in section 6.2 to illustrate the method in which the path planning process in the 2D space is carried out based on the proposed algorithms to find paths in different random scenarios. Then, the 2D path planning process is demonstrated to find 2D paths through the application of MRPPA, CA, and OCA based on the VG method followed by Dijkstra's algorithm in sections 6.3, 6.4, and 6.5, respectively. Additionally, to explain that the designed software package has been developed to implement the process of path planning systematically and in an easy-to-use manner.

### **6.2 Path Planning Software**

The proposed control algorithm has been validated for path planning by means of extensive MATLAB/Simulink simulations. In general, to find the shortest collision-free path in a workspace environment using path planning proposed algorithms requires many inputs such as starting points, number of obstacles, positions and dimensions, goal location, and speed of robots. The obstacles are defined as geometric shapes such as triangles, rectangles, squares, or zigzag lines, etc. All required inputs must be supplied in order to perform and complete the process of path planning. In addition, all these inputs must be in accordance with a specific logical order, for instance, it is important to determine all obstacles data in terms of their dimensions and spatial locations before setting the starting points and target points so that these points do not fall in the obstacles area, causing it not to complete the path planning process. Consequently, if any of the required inputs is not supplied, the path planning process cannot be implemented.

### 6.3 Path Planning Software using MRPPA based on VG

It is important that before entering the process of explaining the simulation in detail, the main concepts that have been used at each stage should be presented. At the first stage, we defined a workspace for three robots to move with obstacles that have been placed on their paths based on the proposed 2D path planning algorithms. The robots move in an environment with six obstacles that are placed in the middle of the workspace in locations known and marked in blue with different labels. The position of each robot is marked by a red point and the target position is denoted by a green point. The environment is visualised using MATLAB, as shown in Figure 6.1.

Figure 6.1: Scenario of workspace environment for three robots with one goal

**Table 6.1.** describe details of workspace information depicted in Figure 6.1

```
>> Simulation workspace environment
```

```
figure1 = Figure (1) with properties:
```

```
Number: 1
```

```
Name: "
```

```
Colour: [1 1 1]
```

```
Position: [403 246 560 420]
```

```
Units: 'pixels'
```

```
Show all properties
```

```
G = graph with properties:
```

```
Edges: [24×1 table]
```

```
Nodes: [28×0 table]
```

At the second stage, the workspace environment is divided into two disconnected components by adding visible edges between vertices of obstacles such as in Figure 6.2. Then, the algorithm will choose the best edges sequentially to connected components of the workspace by measuring algebraic connectivity to create paths for each robot, see Figure 6.3.

Figure 6.2: Divided workspace environment by MRPPA

**Table 6.2.** describe details of workspace depicted in Figure 6.2

>> path planned by MRPPA

G = [graph](#) with properties:

Edges: [68×2 table]

Nodes: [28×0 table]

The MRPPA will planned paths (path 1, path 2, and path 3) respectively for each robot (one by one), as shown in Figure 6.3.

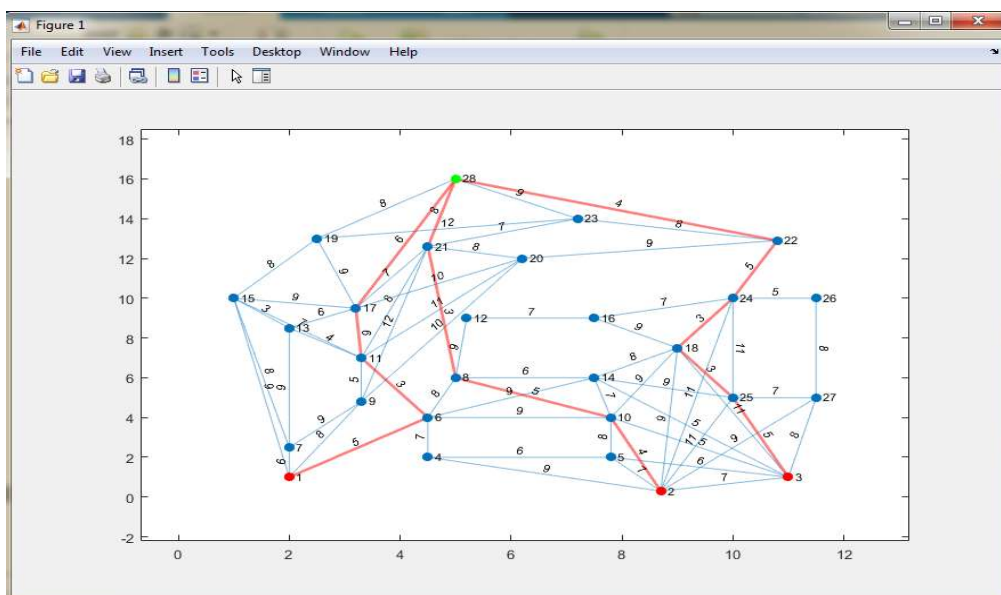


Figure 6.3: Paths planned by MRPPA using MATLAB

**Table 6.3.** describe details of workspace information depicted in Figure 6.3

```
>> Paths planned by MRPPA
```

G = [graph](#) with properties:

Edges: [71×2 table]

Nodes: [28×0 table]

After that, the algorithm determined and connects all possible visibility lines (visibility graph) from the start point to the target point. Finally, Dijkstra's Algorithm is applied to test and process each step taken and measured distance from start points to goal points to find the shortest path by using MATLAB (see the detailed explanation of the algorithmic procedure in Chapter 2). It marks every path that has the shortest distance for robot one ( $R_1$ ), robot two ( $R_2$ ), and robot three ( $R_3$ ) are marked with red. The algorithm determines the shortest path for each robot to reach its goal, as shown in the scenario Figure 6.4.

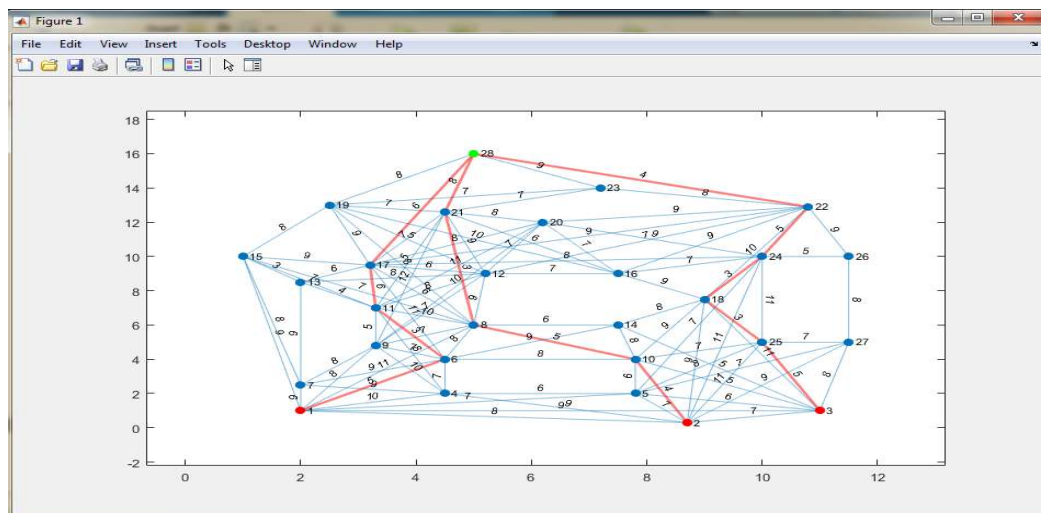


Figure 6.4: Three shortest paths planned via Dijkstra's algorithm by MATLAB

**Table 6.4.** describe details of workspace information depicted in Figure 6.4

```
>> Paths planned by MRPPA
```

G = [graph](#) with properties:

Edges: [108×2 table]

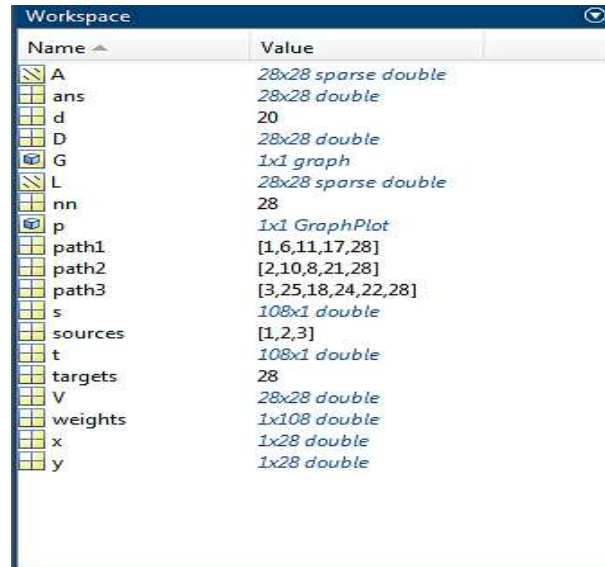
Nodes: [28×0 table]

```
path1 = 1 6 11 17 28      d = 20
```

```
path2 = 2 10 8 21 28      d = 20
```

```
path3 = 3 25 18 24 22 28  d = 20
```

**Note:** each calculated path is a series of consecutive waypoints starting from the first point to the target point. For example, the path1 for robot one from start point to the goal point is  $R_1 \rightarrow v_6 \rightarrow v_{11} \rightarrow v_{17} \rightarrow v_{28}$ , and the distance from start point to the goal is  $D = 20$



Name	Value
A	28x28 sparse double
ans	28x28 double
d	20
D	28x28 double
G	1x1 graph
L	28x28 sparse double
nn	28
p	1x1 GraphPlot
path1	[1,6,11,17,28]
path2	[2,10,8,21,28]
path3	[3,25,18,24,22,28]
s	108x1 double
sources	[1,2,3]
t	108x1 double
targets	28
V	28x28 double
weights	1x108 double
x	1x28 double
y	1x28 double

Figure 6.5: Workspace details depicted in Figure 6.4

The result of the simulation has shown that each robot has performed its tasks and reached its target required without collision with obstacles or with other robots. Although each robot has taken a different path and with different distances from other robots, they succeeded in reaching their goal, as shown in Figure 6.6.

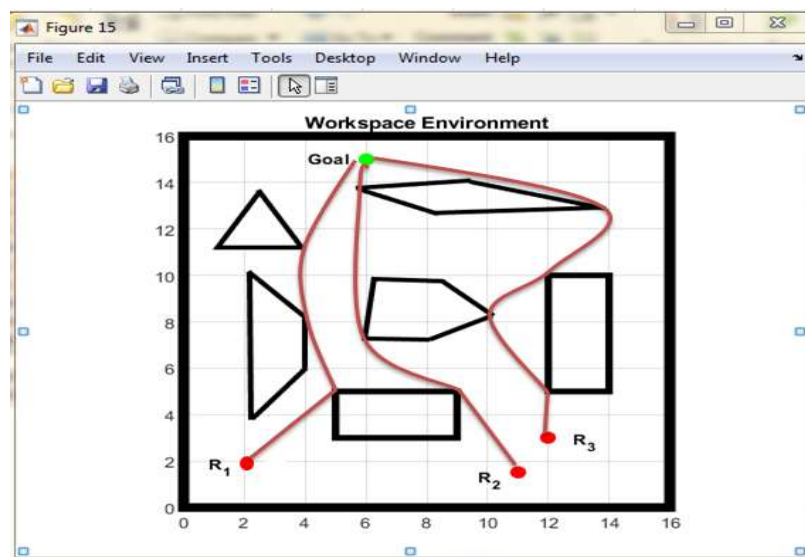


Figure 6.6: Robots reach the target point

**Table 6.5.** describe the details of workspace information depicted in Figure 6.6.

>> Simulation Paths planned by MRPPA

Robot1 = 3.00000 2.0000

Robot2= 11.000 0 1.5000

Robot3 = 12.2000 3.0000

figure2 = [Figure](#) (2) with properties:

Number: 2

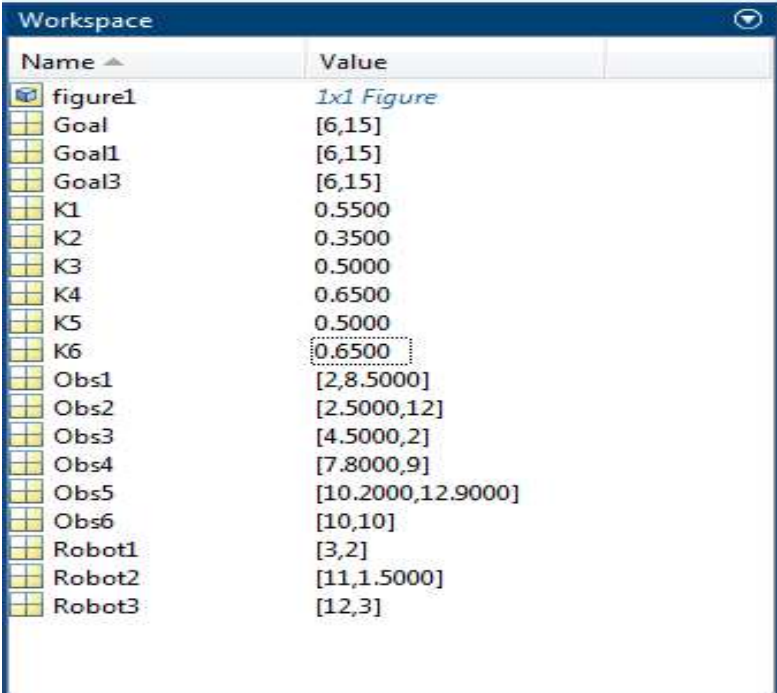
Name: "

Colour: [1 1 1]

Position: [403 246 560 420]

Units: 'pixels'

Show [all properties](#)



Name	Value
figure1	1x1 Figure
Goal	[6,15]
Goal1	[6,15]
Goal3	[6,15]
K1	0.5500
K2	0.3500
K3	0.5000
K4	0.6500
K5	0.5000
K6	0.6500
Obs1	[2,8.5000]
Obs2	[2.5000,12]
Obs3	[4.5000,2]
Obs4	[7.8000,9]
Obs5	[10.2000,12.9000]
Obs6	[10,10]
Robot1	[3,2]
Robot2	[11,1.5000]
Robot3	[12,3]

Figure 6.7: Workspace details depicted in Figure 6.6

We re-simulated the scenario of workspace environment in Figure 6.1 but with change in the position of robots ( $R_1$ ,  $R_2$ ,  $R_3$ ) and we add three targets to test the effectiveness of algorithms. In this scenario, we first provide a simulation environment for a team of three robots that appear as three red points, moving in the environment between six obstacles highlighted as blue, and three goals represented with green points, as depicted in Figure 6.8.

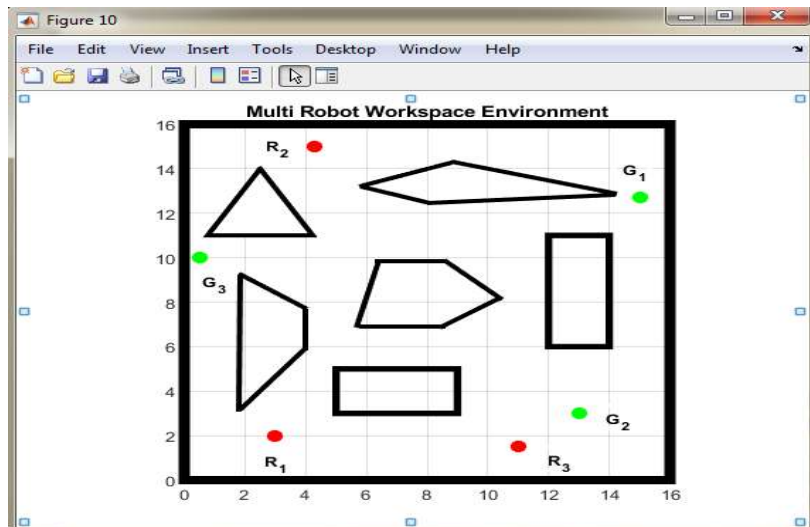


Figure 6.8: Scenario of workspace environment for three robots with three goals

**Table 6.6.** describe details of workspace information depicted in Figure 6.8.

```
>> Simulation workspace environment

figure1 = Figure (1) with properties:
    Number: 1
    Name: ''
    Colour: [1 1 1]
    Position: [403 246 560 420]
    Units: 'pixels'
    Show all properties

G = graph with properties:
    Edges: [24×1 table]
    Nodes: [30×0 table]
```



Secondly, the algorithm divided workspace environment into two disconnected components by adding visible edges between vertices of obstacles such as in Figure 6.8. Then, it is choosing the best edges to join the components of the workspace respectively via measuring algebraic connectivity to create path for each robot, see Figure 6.9.

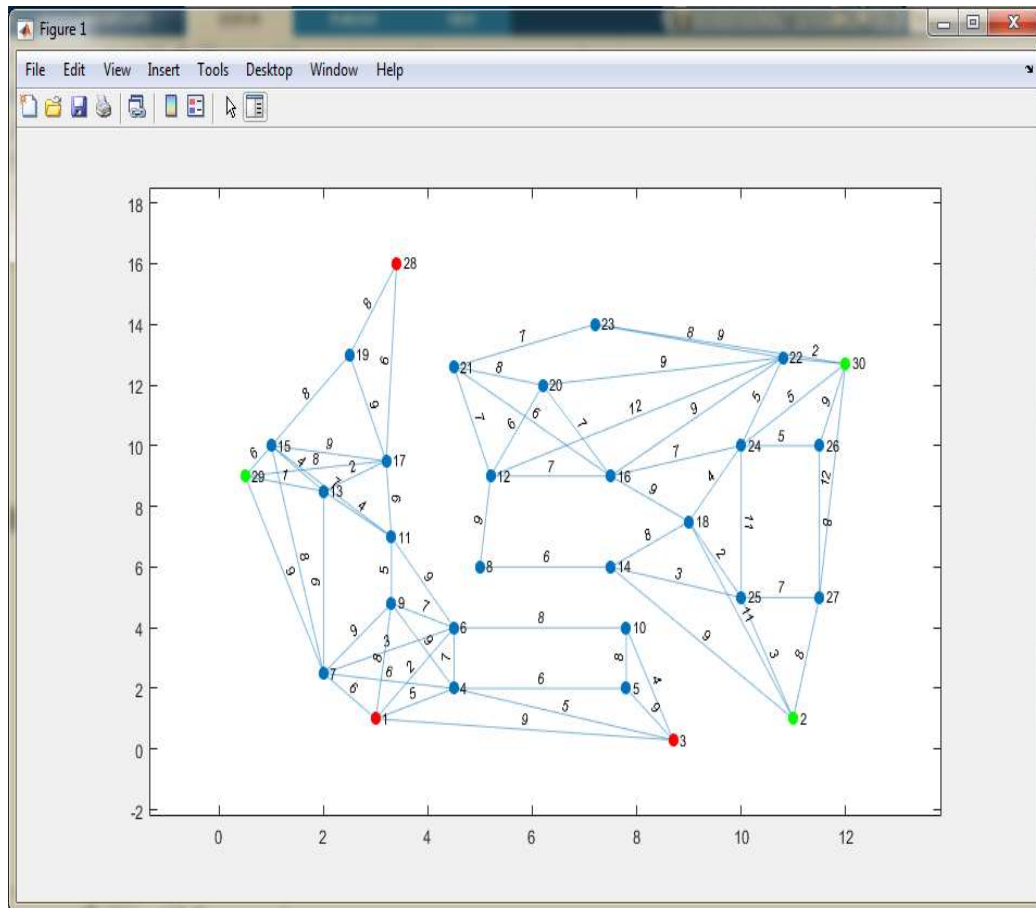


Figure 6.9: Divided workspace environment using MATLAB

**Table 6.7.** describes details of workspace information depicted in Figure 6.9.

```
>> Path planned MRPPA
```

G = [graph](#) with properties:

Edges: [69×2 table]

Nodes: [30×0 table]

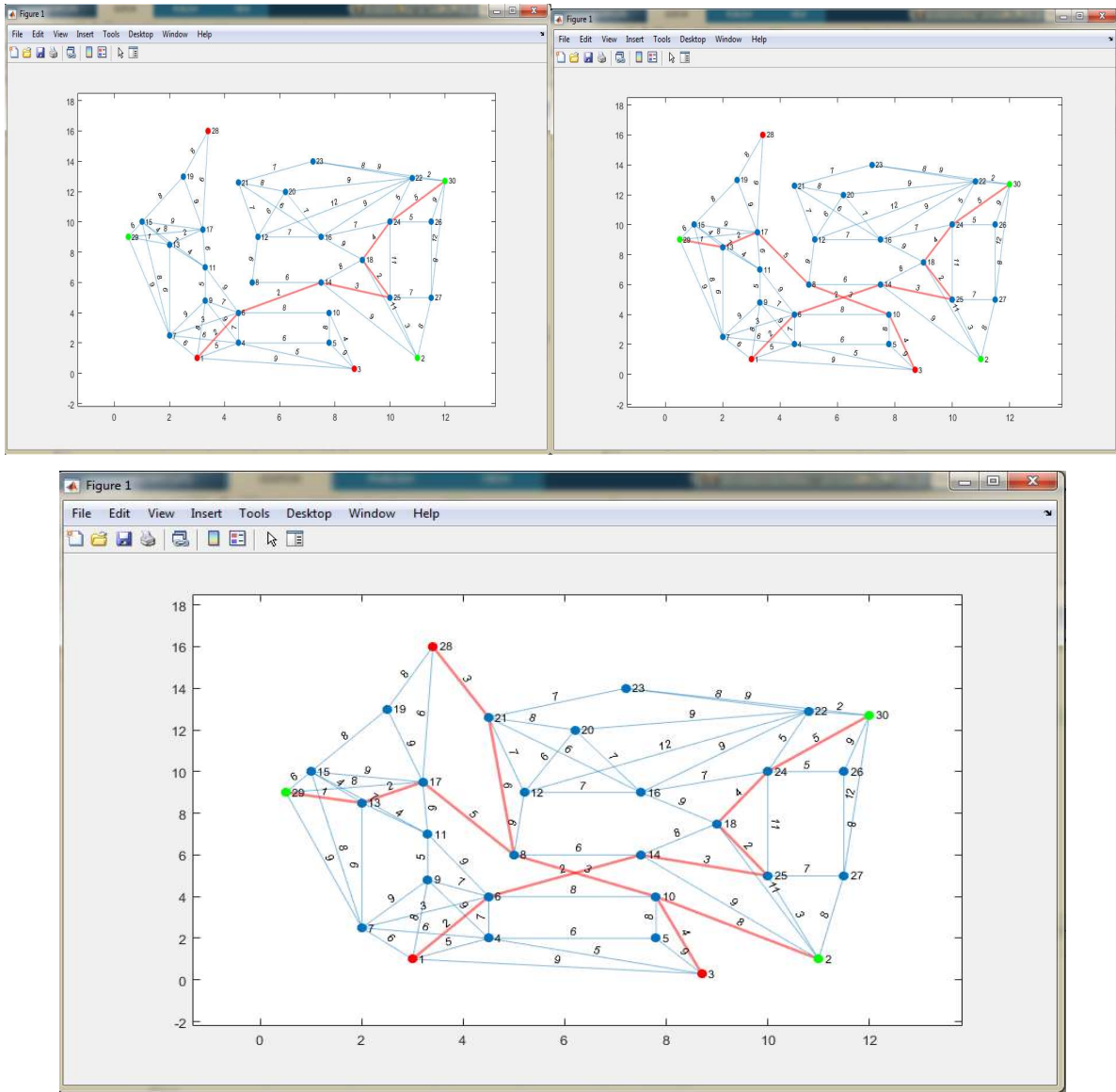


Figure 6.10: Stapes of Paths planned by MRPPA using MATLAB

**Table 6.8.** describes details of workspace information depicted in Figure 6.10.

```
>> Paths planned by MRPPA
```

G = [graph](#) with properties:

Edges: [74×2 table]

Nodes: [30×0 table]

Next, the algorithm connects all possible visibility lines (visibility graph) from the start point to the target point. Lastly, Dijkstra's Algorithm begins processing each step that is taken and measures the distances from start to the goal to find the shortest safe path. The algorithm has marked each shortest path with a red colour, such as the shortest path of robot one ( $R_1$ ) is

marked with a red colour, the shortest path of robot two ( $R_2$ ) with red, and the shortest path of robot three ( $R_3$ ) is marked with red. The algorithm has determined the shortest path for each robot to reach its goals that are also coded with a green colour. Figure 6.11 shows the application of the Dijkstra's algorithm for finding shortest paths for three robots ( $R_1$ ,  $R_2$ ,  $R_3$ ) that reach their targets without collision.

Figure 6.11: Application of the Dijkstra Algorithm for paths planned by using MATLAB

**Table 6.9.** describes details of workspace information depicted in Figure 6.11. For example, the path3 for robot two from start point to the goal point is  $R_2 = v_{28} \rightarrow v_{21} \rightarrow v_8 \rightarrow v_{10} \rightarrow v_2$ , and the distance from start point to the goal is  $d = 20$

```
>> Paths planned by MRPPA
G = graph with properties:
    Edges: [99x2 table]
    Nodes: [30x0 table]
path1 = 1    6   14   25   18   24   30        d = 18
path2 = 3    10    8   17   13   29        d = 15
path3 = 28   21    8   10    2        d = 20
```

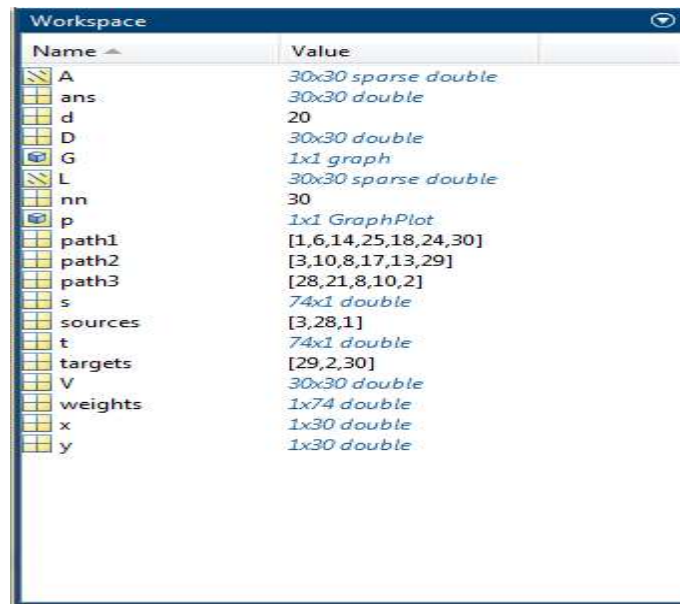


Figure 6.12: Workspace details depicted in Figure 6.11

The result of the simulation shows that the robots succeeded to reach their targets even as their locations and goals changed, as shown in Figure 6.13.

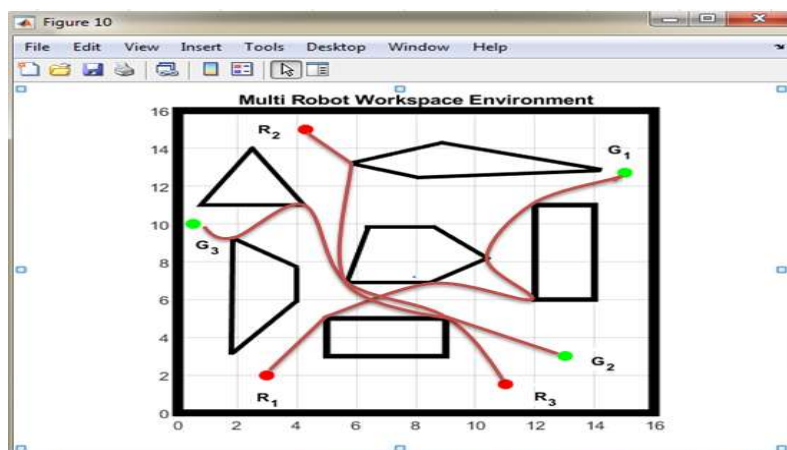


Figure 6.13: Robots reach the target points

**Table 6.10.** describes details of workspace information depicted in Figure 6.13, for example, Robot1 has the coordinate (3.000, 2.000).

```
>> Simulation Paths planned by MRPPA
```

```
Robot1 = 3.000 2.0000
```

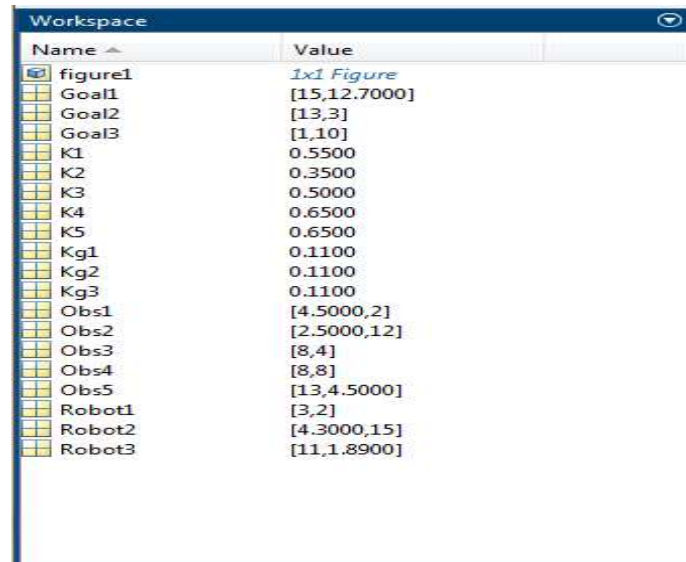
```
Robot2 = 4.3000 15.0000
```

```
Robot3 = 11.0000 1.8900
```

```
figure2 = Figure (2) with properties:
```

```
Number: 2
```

Name: "
Colour: [1 1 1]
Position: [403 246 560 420]
Units: 'pixels'
Show <a href="#">all properties</a>



Name	Value
figure1	1x1 Figure
Goal1	[15,12.7000]
Goal2	[13,3]
Goal3	[1,10]
K1	0.5500
K2	0.3500
K3	0.5000
K4	0.6500
K5	0.6500
Kg1	0.1100
Kg2	0.1100
Kg3	0.1100
Obs1	[4.5000,2]
Obs2	[2.5000,12]
Obs3	[8,4]
Obs4	[8,8]
Obs5	[13,4.5000]
Robot1	[3,2]
Robot2	[4.3000,15]
Robot3	[11,1.8900]

Figure 6.14: Workspace details depicted in Figure 6.13

### 6.3.1 Path planning for simple maps by MRPPA

To test the effectiveness of the proposed algorithm, we give a simulation of the simple workspace environment that has three robots highlighted in blue and three goals highlighted in green such as Figure 6. 15.

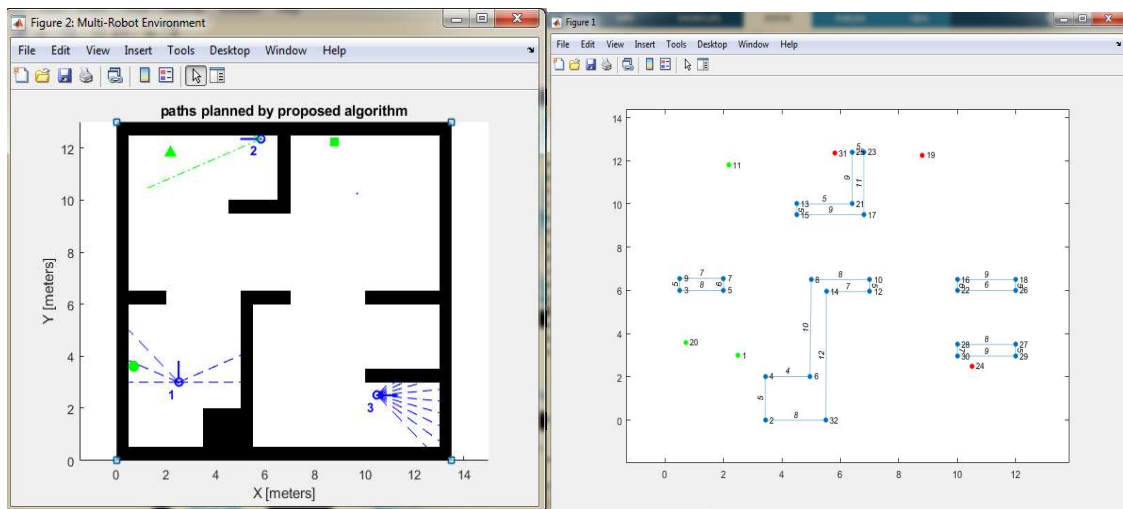


Figure 6.15: Simple workspace environment with the use of MATLAB

**Table 6.10.** describes details of workspace information depicted in Figure 6.15.

>> Path planned MRPPA	
G = <a href="#">graph</a> with properties:	
Edges:	[26×2 table]
Nodes:	[32×0 table]

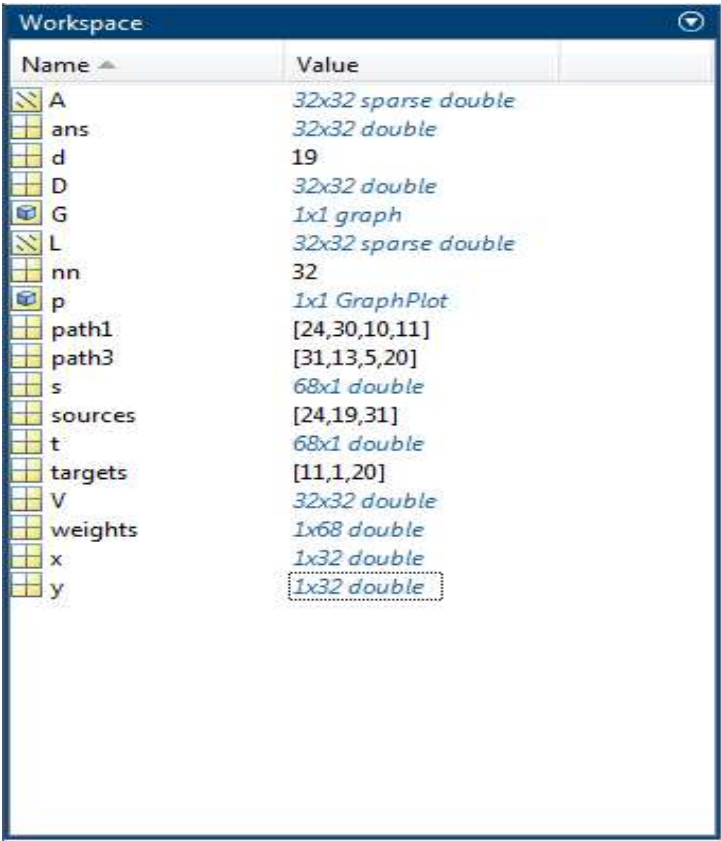


Figure 6.16: Workspace details depicted in Figure 6.15

Here, the algorithm will divide workspace environment into two disconnected components by adding visible edges between vertices of obstacles such as in Figure 6.17. In addition, it is choosing the best edges to connect the components of the workspace separately via measuring the second smallest eigenvalue to generate path for each robot.

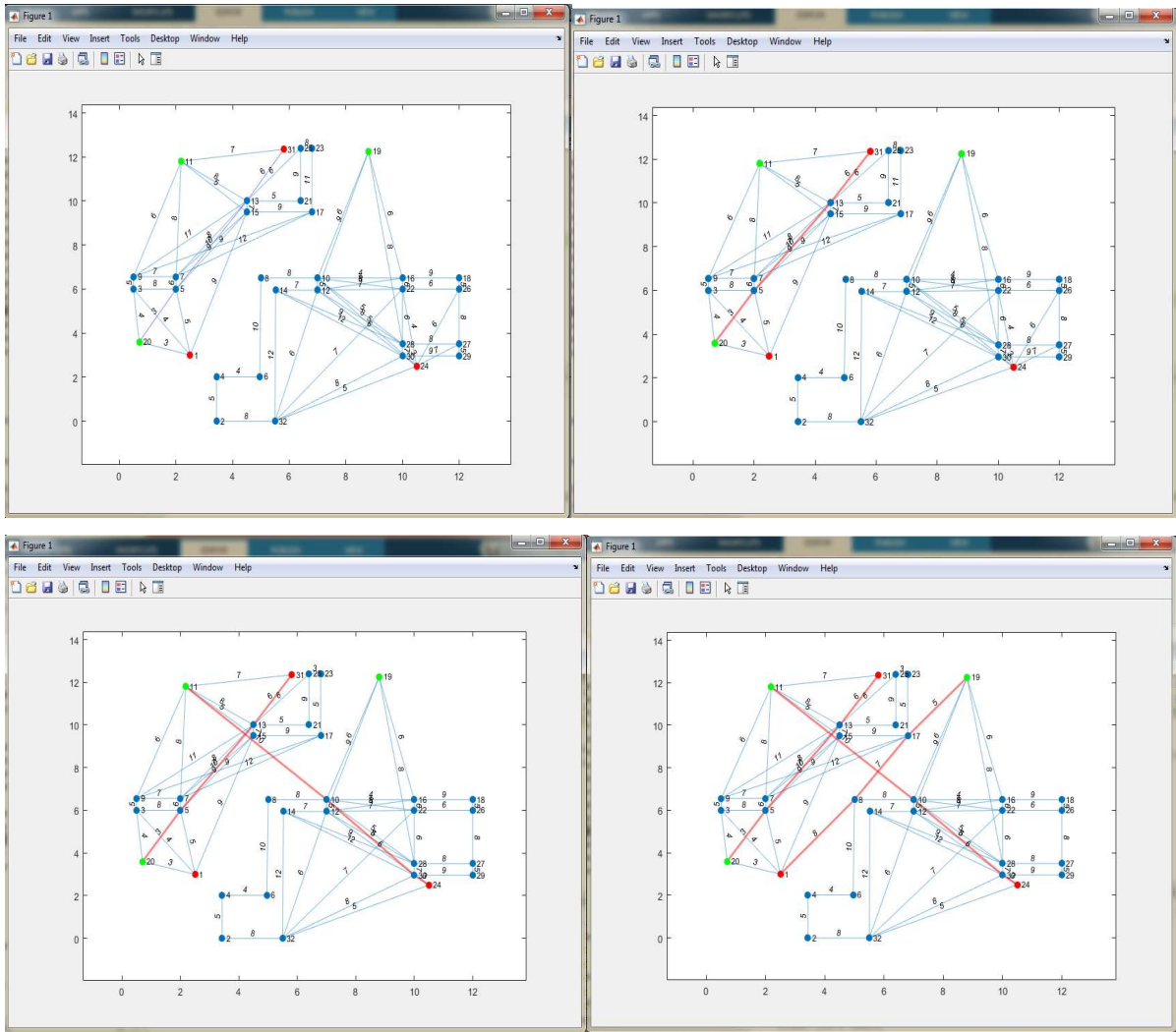


Figure 6.17: Paths planned by MRPPA using MATLAB

**Table 6.11.** describes details of workspace information depicted in Figure 6.17.

```
>> Path planned MRPPA
```

G = [graph](#) with properties:

Edges: [74×2 table]

Nodes: [32×0 table]

The algorithm connects all possible visibility lines (visibility graph) from the start point to the target point. Then, Dijkstra's Algorithm is applied to test and process each step taken and measured distance from start to goal to find the shortest path by using MATLAB, as Figure 6.18.



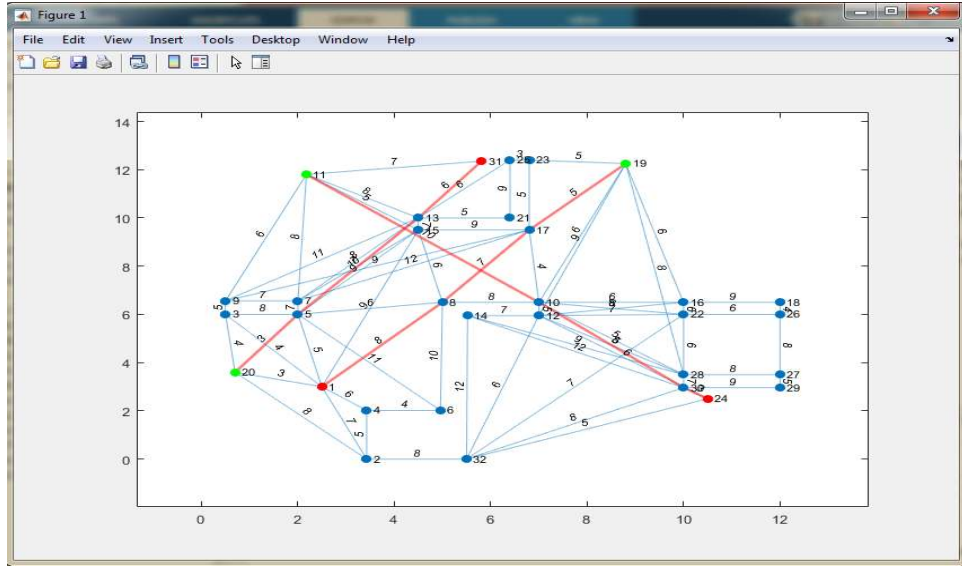


Figure 6.18: Paths planned by MRPPA using MATLAB

**Table 6.12.** describes details of workspace information depicted in Figure 6.18. For example, the path3 for robot three from start point to the goal point is  $R_3 = v_{31} \rightarrow v_{13} \rightarrow v_5 \rightarrow v_{20}$ , and the distance from start point to the goal is  $d = 19$

$G = \text{graph}$  with properties:

Edges: [85×2 table]

Nodes: [32×0 table]

path1 = 24   30   10   11                       $d = 18$

path2 = 1   8   17   19                       $d = 20$

path3 = 31   13   5   20                       $d = 19$

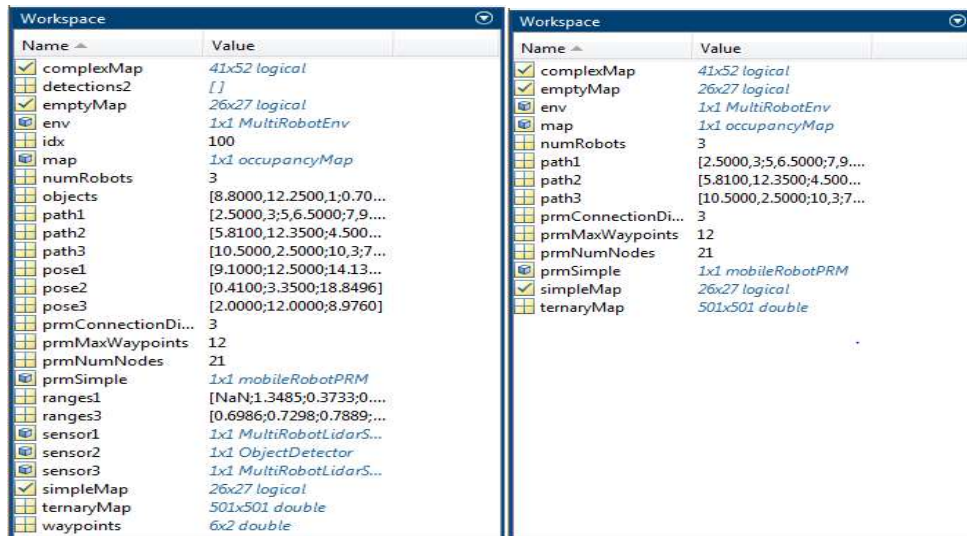


Figure 6.19: Workspace details depicted in Figure 6.18



The result of the simulation has shown that each robot has reached its target without collision with other robots, see Figure 6.20.

Figure 6.20: Robots successfully reach the target point

**Table 6.13:** Details of the multi-robot paths depicted in Figure 6.18

<b>Robots</b>	<b>Path waypoints</b>	<b>Position [x; y; <math>\theta</math>]</b>	<b>Object</b>
$Robot_1$	[5 6.5;7 9.5]	Pose1[2.5; 3.0 ; $\pi/2$ ]	[8.80, 12.25, 1.00]
$Robot_2$	[4.5 10;2.0 6.0]	Pose2[5.81; 12.35; $\pi$ ]	[0.7, 3.6, 2.00]
$Robot_3$	[10.0 3.0;7.0 6.50]	Pose3[10.5; 2.5; 0]	[2.18, 11.8, 3.00]

#### 6.4 Path planning Software by CA

We demonstrate a scenario with six obstacles, three starting points, and three target points in a workspace that has been generated. The next step is to generate Central Baseline (CB) that linking the starts and target points as highlighted in red, which is a key step for path planning. The obstacles that overlap with CB have been determined, as a result, the waypoints generated

from each vertex of each obstacle that intersects with CB, and calculated for each half of the map, the lines linking waypoints, and their intersection points (vertex of obstacle) are orthogonal to the original CB marked as dashed blue lines, as shown in Figure 6.21. Since there are two intersection points that lie along each CB path, each point having two waypoints are in each half of the map, so the total of four waypoints are computed, which are marked in red. Successive waypoints are connected with each other's to create multiple possible collision-free paths for robots around obstacles, see Figure 6.21

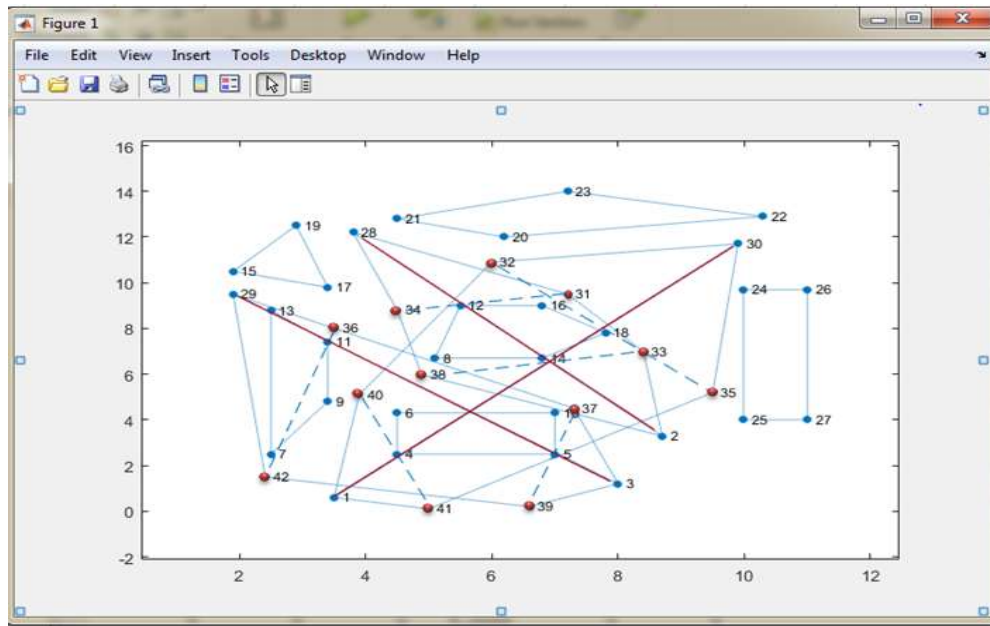


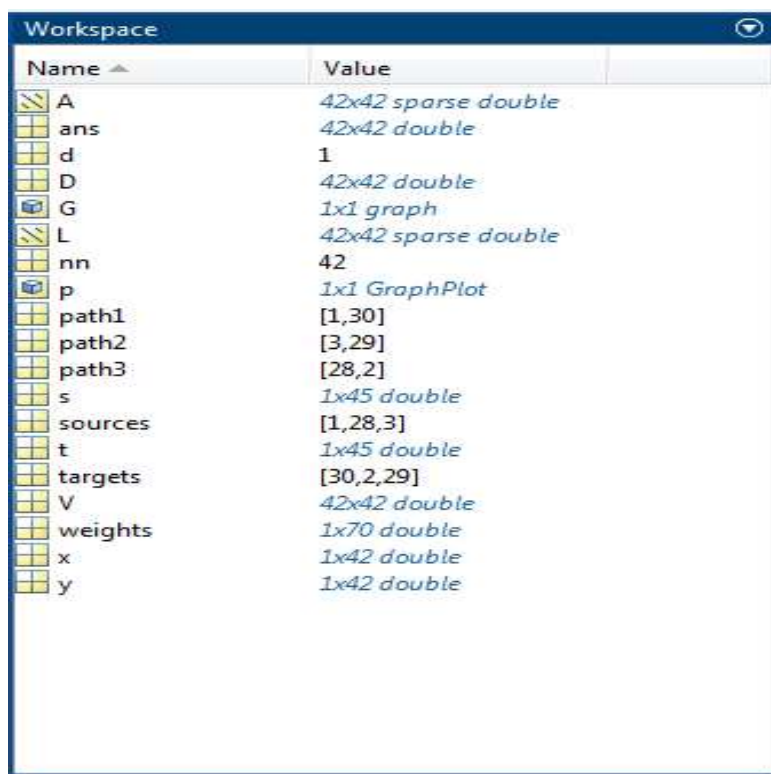
Figure 6.21: Application of the CA using MATLAB

**Table 6.14.** describes details of workspace information depicted in Figure 6.21.

```
>> C Algorithm

G = graph with properties:
    Edges: [45×1 table]
    Nodes: [42×0 table]
path1 = 1    30    d = 1
path2 = 3    29    d = 1
path3 = 28   2     d = 1

figure1 = Figure (1) with properties:
    Number: 1
    Name: ""
    Colour: [1 1 1]
    Position: [403 246 560 420]
    Units: 'pixels'
    Show all properties
```



Name	Value
A	42x42 sparse double
ans	42x42 double
d	1
D	42x42 double
G	1x1 graph
L	42x42 sparse double
nn	42
p	1x1 GraphPlot
path1	[1,30]
path2	[3,29]
path3	[28,2]
s	1x45 double
sources	[1,28,3]
t	1x45 double
targets	[30,2,29]
V	42x42 double
weights	1x70 double
x	1x42 double
y	1x42 double

Figure 6.22: Workspace details depicted in Figure 6.21

After all the required inputs are keyed in, the part of the visibility graph is generated by the CA. The visibility graph of the previous scenario, generated from the waypoints is shown in Figure 6.23. Displaying the part of the visibility graph is beneficial because it reflects the number of obstacles involved in the calculation of the paths.

Figure 6.23: Visibility graph by CA using MATLAB

**Table 6.15.** describes details of workspace information depicted in Figure 6.23.

```
>> Visibility graph by CA  
G = graph with properties:  
Edges: [70×2 table]  
Nodes: [42×0 table]
```

Figure 6.24: Workspace details depicted in Figure 6.23

Now, after collecting all the important information and creating the VG, Dijkstra's algorithm finds collision-free paths for each robot in a 2D workspace environment as shown in Figure 6.25

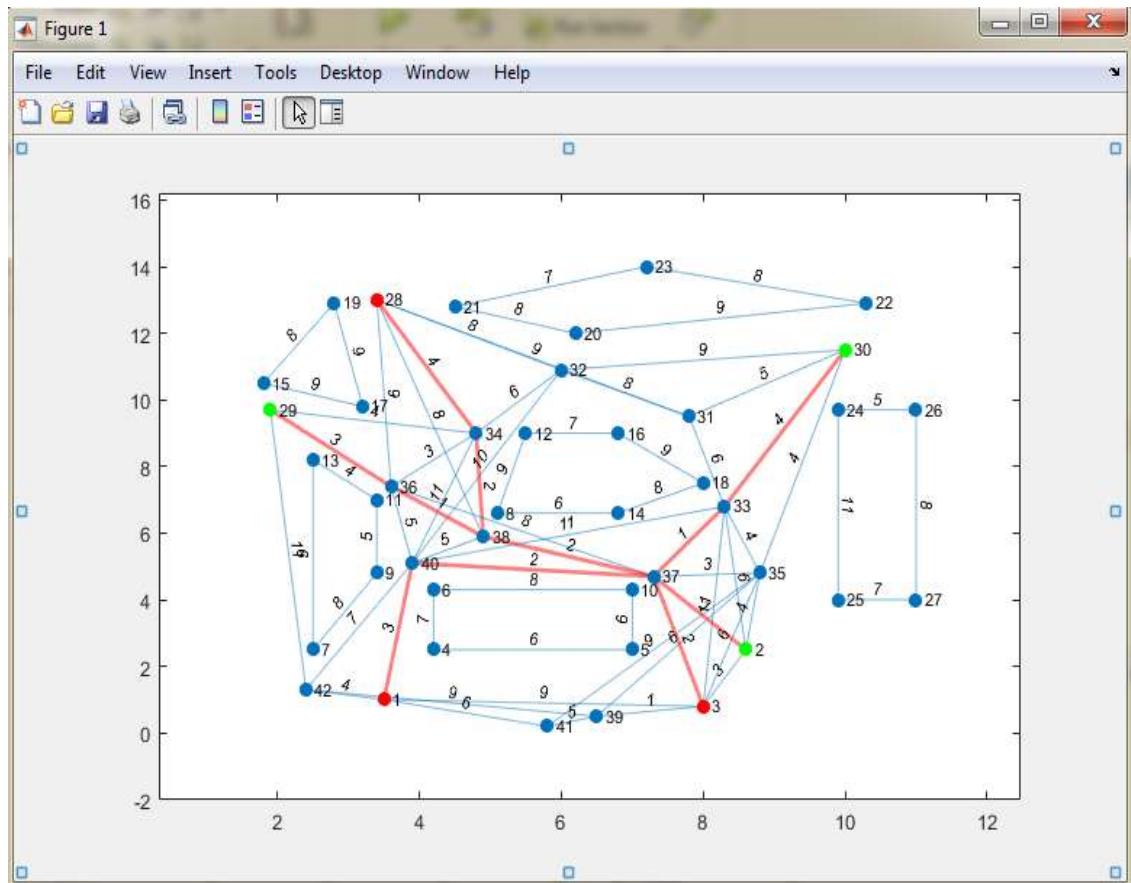


Figure 6.25: Paths planned by CA using MATLAB

**Table 6.16.** describes details of workspace information depicted in Figure 6.25. For example, the path3 for robot two from start point to the goal point is  $R_2 = v_{28} \rightarrow v_{34} \rightarrow v_{38} \rightarrow v_{37} \rightarrow v_2$ , and the distance from start point to the goal is  $d = 9$ .

```
>> Simulation Paths planned by CA
```

```
G = graph with properties:
```

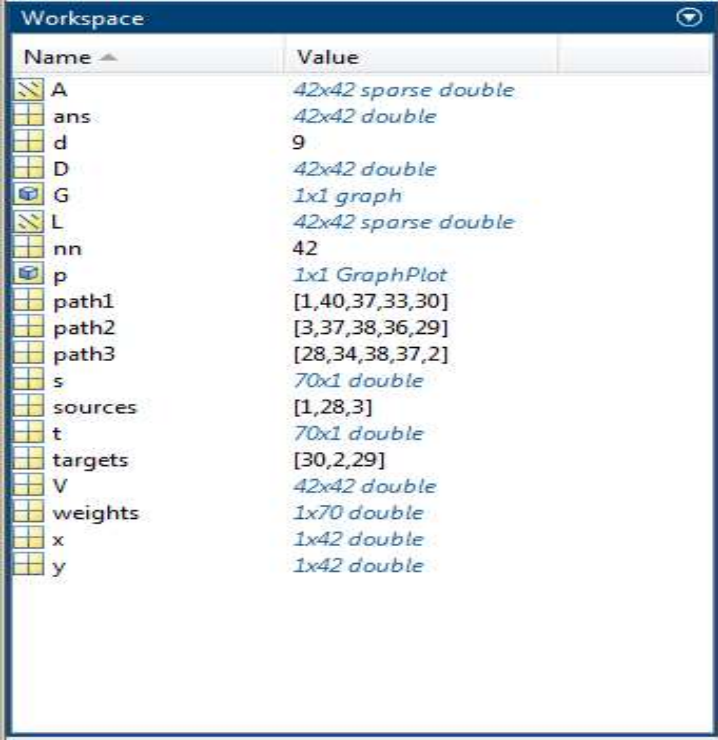
```
Edges: [70×2 table]
```

```
Nodes: [42×0 table]
```

```
path1 = 1    40    37    33    30           d = 10
```

```
path2 = 3    37    38    36    29           d = 8
```

```
path3 = 28   34   38   37    2           d = 9
```



Name	Value
A	42x42 sparse double
ans	42x42 double
d	9
D	42x42 double
G	1x1 graph
L	42x42 sparse double
nn	42
p	1x1 GraphPlot
path1	[1,40,37,33,30]
path2	[3,37,38,36,29]
path3	[28,34,38,37,2]
s	70x1 double
sources	[1,28,3]
t	70x1 double
targets	[30,2,29]
V	42x42 double
weights	1x70 double
x	1x42 double
y	1x42 double

Figure 6.26: Workspace details depicted in Figure 6.25

The result of the simulation shows that the robots succeeded to reach their targets, as shown in Figure 6.27.

Figure 6.27: Robots reach the target points

**Table 6.17.** describes details of workspace depicted in Figure 6.27. for example, Robot1 has the coordinate (3.000, 2.100).

```
>> Simulation Paths planned by CA
```

```
Robot1 = 3.000 2.1000
```

```
Robot2 = 4.5000 13.5000
```

```
Robot3 = 10.2000 2.3000
```

```
figure2 = Figure (2) with properties:
```

```
Number: 2
```

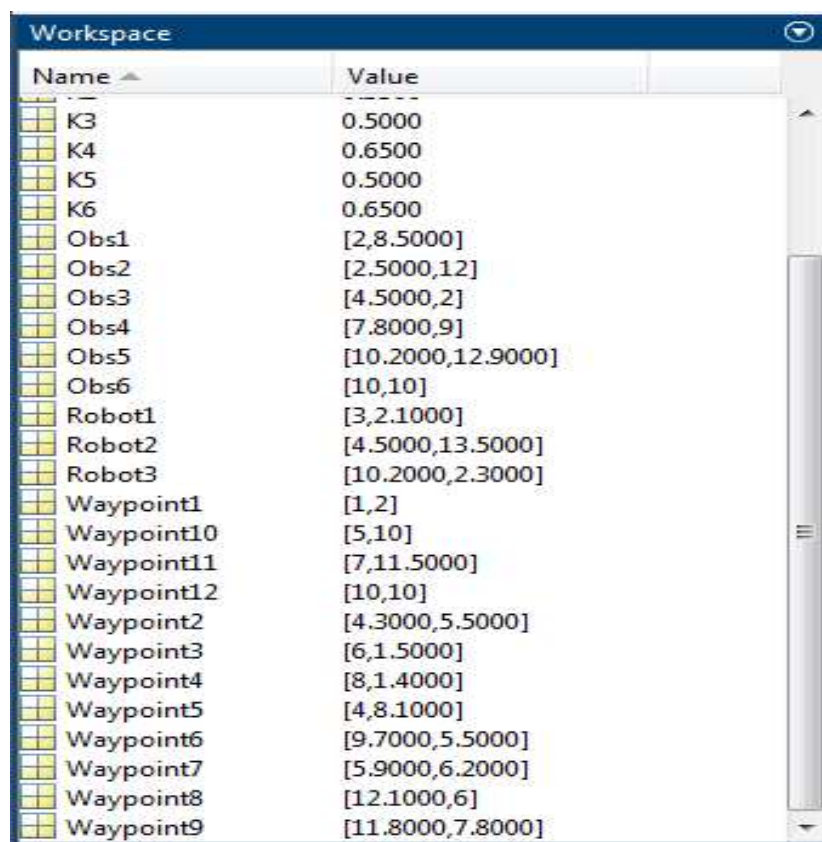
```
Name: ''
```

```
Colour: [1 1 1]
```

```
Position: [403 246 560 420]
```

```
Units: 'pixels'
```

```
Show all properties
```



Name	Value
K3	0.5000
K4	0.6500
K5	0.5000
K6	0.6500
Obs1	[2,8.5000]
Obs2	[2.5000,12]
Obs3	[4.5000,2]
Obs4	[7.8000,9]
Obs5	[10.2000,12.9000]
Obs6	[10,10]
Robot1	[3,2.1000]
Robot2	[4.5000,13.5000]
Robot3	[10.2000,2.3000]
Waypoint1	[1,2]
Waypoint10	[5,10]
Waypoint11	[7,11.5000]
Waypoint12	[10,10]
Waypoint2	[4.3000,5.5000]
Waypoint3	[6,1.5000]
Waypoint4	[8,1.4000]
Waypoint5	[4,8.1000]
Waypoint6	[9.7000,5.5000]
Waypoint7	[5.9000,6.2000]
Waypoint8	[12.1000,6]
Waypoint9	[11.8000,7.8000]

Figure 6.28: Workspace details depicted in Figure 6.27

Moreover, we have simulated a scenario of a simple workspace environment using the Central algorithm to illustrate how it worked and its impact on the different environments. The workspace has three robots marked with blue colour, while the green points are the goals that robots must reach. Also, we defined the obstacles that overlap with CB, and waypoints that are generated by the algorithm are highlighted with red, as shown in Figure 6.29.

Figure 6.29: Waypoints generated by CA

After created waypoints, the part of the visibility graph is generated. By ticking all the important information, Dijkstra's algorithm will find collision-free paths for each robot. The result of the simulation shows that the robots succeeded to reach their targets without collision with obstacles or with other robots, see Figure 6.30

Figure 6.30: Robots successfully reach the target points

**Table 6.18.** describes details of workspace depicted in Figure 6.30.

```
>> Simulators  
s = struct with fields:  
    RobotPose: [0 0 0]  
    LookaheadPoint: [1.7964 3.3458]
```



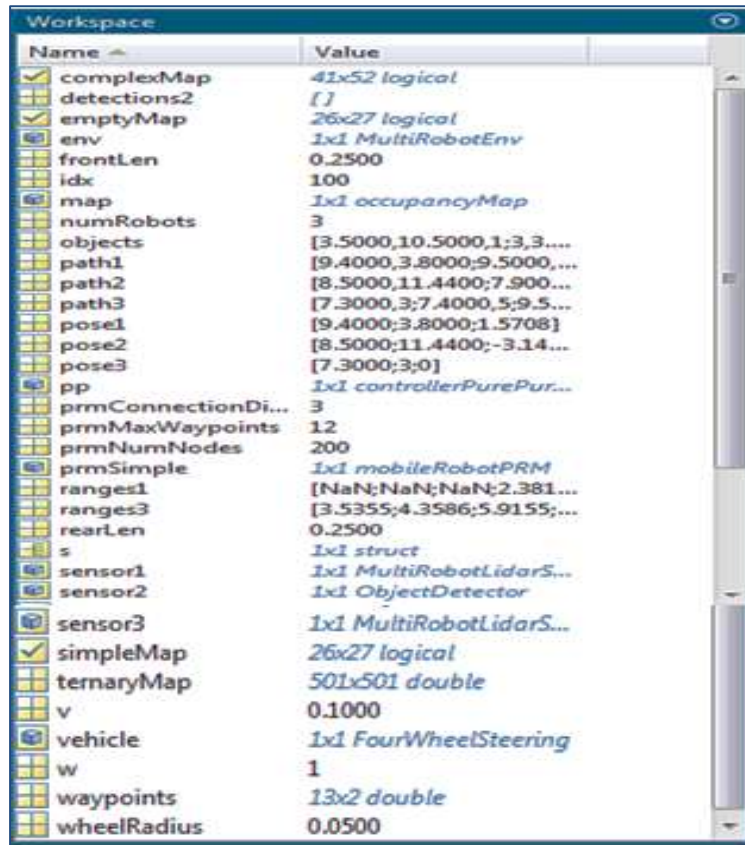


Figure 6.31: Workspace details depicted in Figures 6.30

**Table 6.19:** Details of paths depicted in Figures 6.30, and 6.31

Robots	Path waypoints	Position [x; y; $\theta$ ]	Object
$Robot_1$	[6.98 7.0; 4.7 8.78]	Pose1[10.5;4.5;pi/2]	[2.085, 10.64, 1.00]
$Robot_2$	[8.46 9.46; 4.0 6.0]	Pose2[9.72;10.44;-pi]	[0.81, 3.51, 2.00]
$Robot_3$	[9.62 6.6; 10.2 7.5]	Pose3[7.6;2.38;0]	[11.98, 11.29, 3.00]

## 6.5 Path planning Software by OCA

The package executes the OCA to find collisions-free paths in 2D environment. The algorithm considers the obstacles location with safe distance, the starting points and target points during the path calculation process, where the red points represent the starting positions whilst the green points denote the goals positions. See scenario of Figure 6.32.

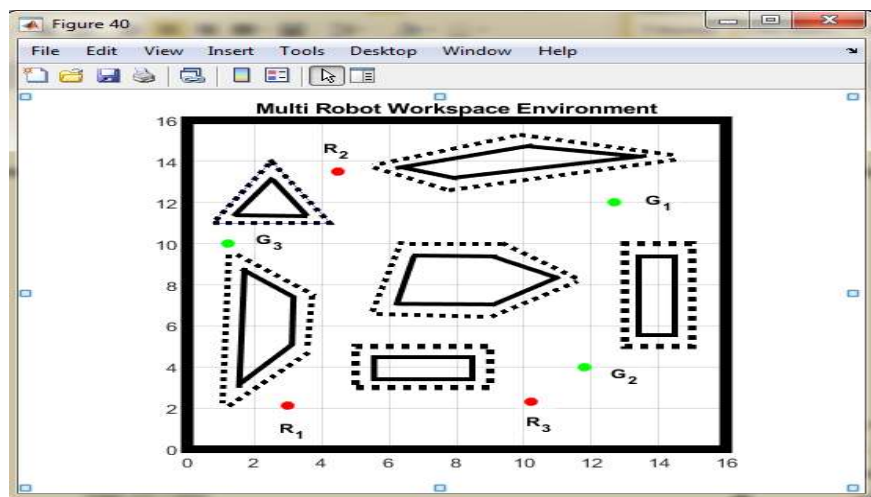


Figure 6.32: Application of the OCA algorithm using MATLAB

**Table 6.20.** describes details of workspace information depicted in Figure 6.32.

<code>&gt;&gt; OCA</code>	
<code>G = <a href="#">graph</a></code> with properties:	<code>figure1 = <a href="#">Figure</a> (1)</code> with
properties:	
Edges: [48×1 table]	Number: 1
Nodes: [54×0 table]	Name: "
	Colour: [1 1 1]
	Position: [403 246 560 420]
	Units: 'pixels'
	Show <a href="#">all properties</a>

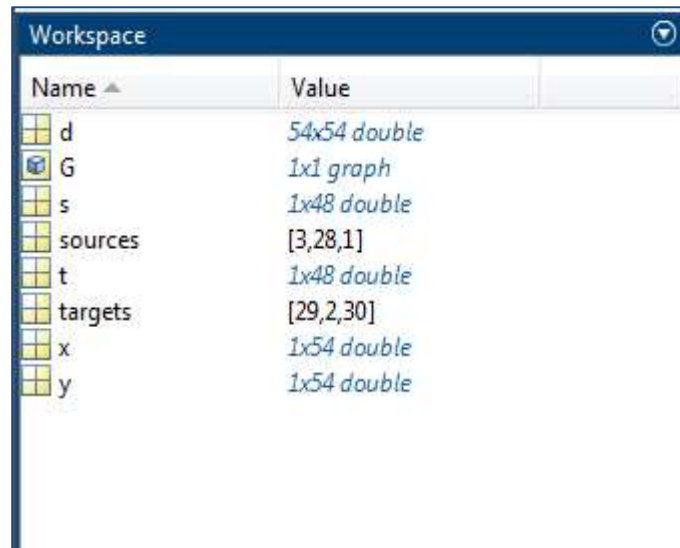


Figure 6.33: Workspace details depicted in Figure 6.32

The part of visibility graph generated after all the required inputs are keyed by using the OCA is depicted in Figure 6.34. The part of the visibility graph is useful because it determined the number of obstacles contributory in the calculation of the paths.

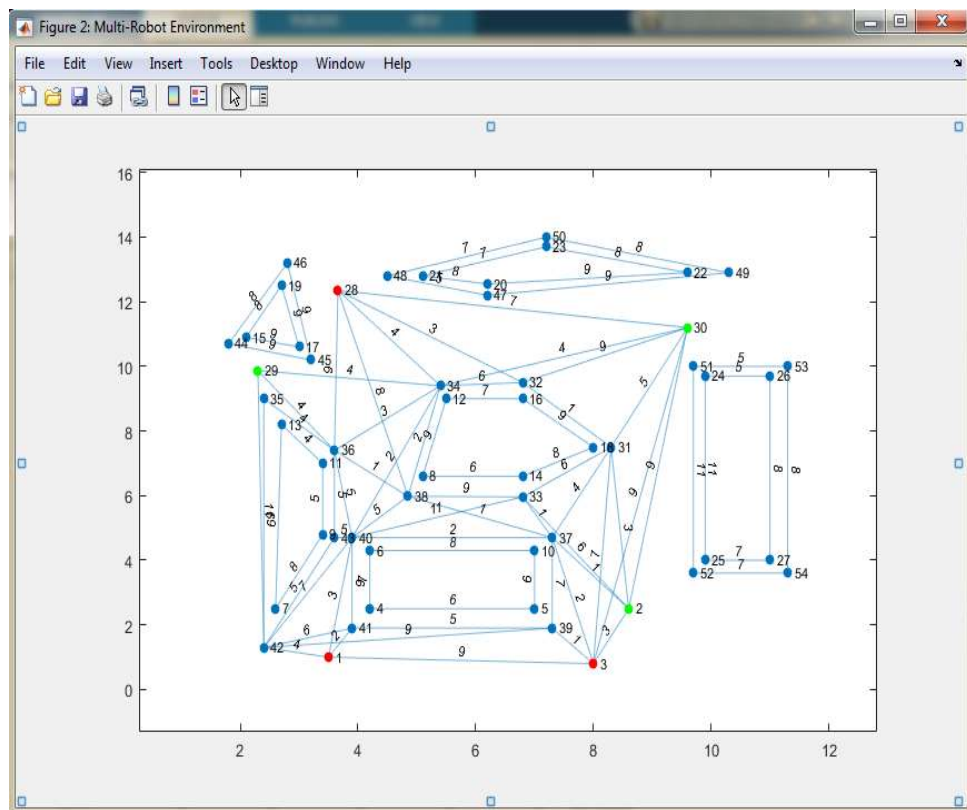


Figure 6.34: Visibility graph by OCA using MATLAB

**Table 6.21.** describes details of workspace information depicted in Figure 6.34.

```
>> VG by OCA
G = graph with properties:
  Edges: [85x1 table]
  Nodes: [54x0 table]
```

The path planning starts with calculating initial paths by using Dijkstra's algorithm and finds collision-free paths for each robot in a 2D workspace environment as shown in red in Figure 6.35.

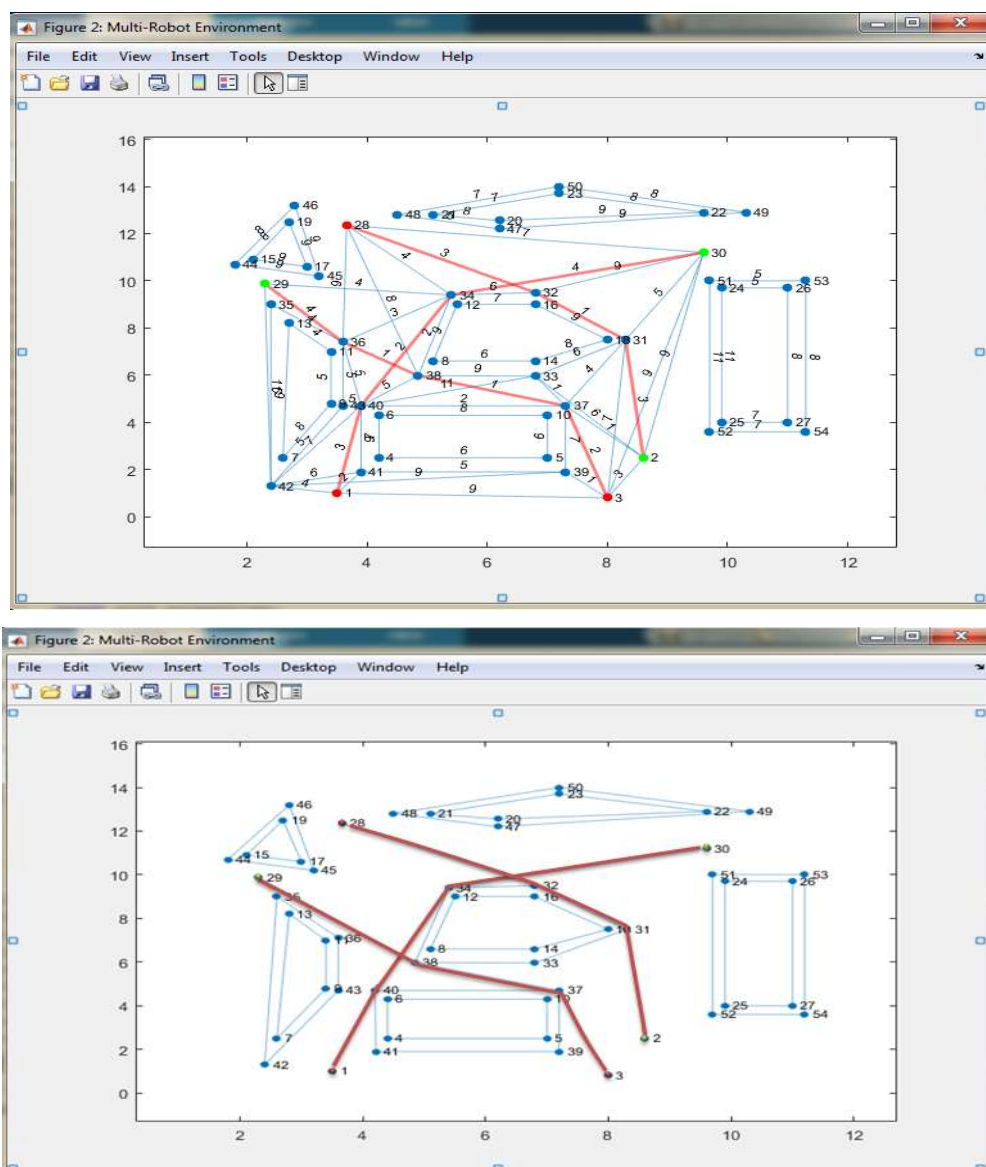


Figure 6.35: Paths planned by OCA using MATLAB

**Table 6.22.** describes details of workspace depicted in Figure 6.35, for example, the path3 for robot two from the start point to the goal point is  $R_2 = v_{28} \rightarrow v_{32} \rightarrow v_{31} \rightarrow v_2$  , and the distance from start point to the goal is  $d = 7$ .

```
>> Paths planned by OCA
G = graph with properties:
  Edges: [85×2 table]
  Nodes: [54×0 table]
path1 =   1  40  34  30          d =  9
path2 =   3  37  38  36  29      d =  8
path3 =  28  32  31   2          d =  7
```

Figure 6.36: Workspace details depicted in Figure 6.35

The result of the simulation shows that the robots succeeded to reach their targets, as shown in Figure 6.37.

Figure 6.37: Robots reach the target points

**Table 6.23.** describes details of workspace depicted in Figure 6.37, for example, Robot1 has the coordinate (3.000, 2.100).

```
>> Simulation Paths planned by OCA
```

```
Robot1 = 3.000 2.1000
```

```
Robot2= 4.5000 13.5000
```

```
Robot3 = 10.2000 2.3000
```

```
figure2 = Figure (2) with properties:
```

```
    Number: 2
```

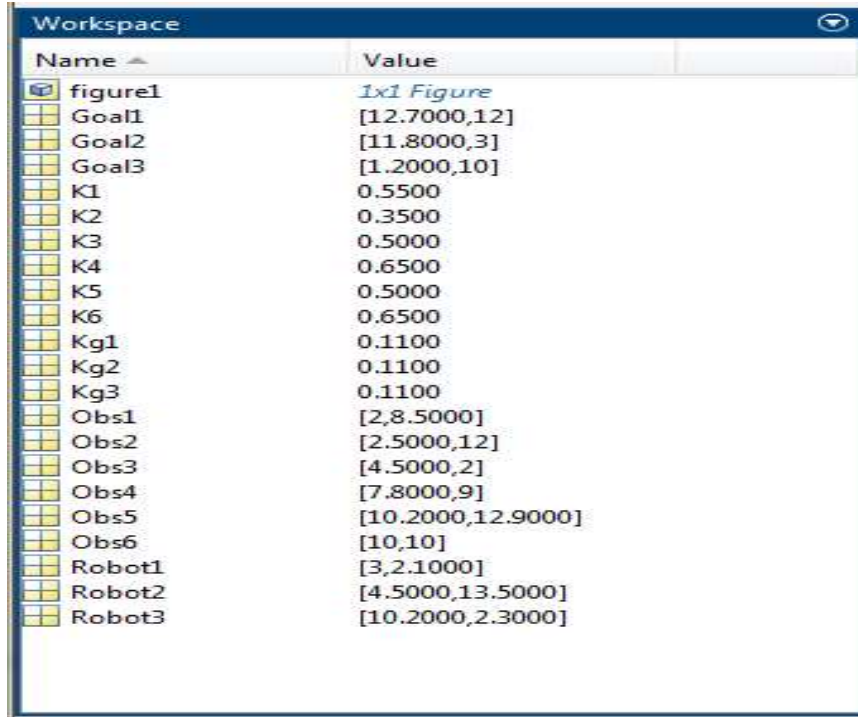
```
    Name: ''
```

```
    Colour: [1 1 1]
```

```
    Position: [403 246 560 420]
```

```
    Units: 'pixels'
```

```
Show all properties
```



Name	Value
figure1	1x1 Figure
Goal1	[12.7000,12]
Goal2	[11.8000,3]
Goal3	[1.2000,10]
K1	0.5500
K2	0.3500
K3	0.5000
K4	0.6500
K5	0.5000
K6	0.6500
Kg1	0.1100
Kg2	0.1100
Kg3	0.1100
Obs1	[2,8.5000]
Obs2	[2.5000,12]
Obs3	[4.5000,2]
Obs4	[7.8000,9]
Obs5	[10.2000,12.9000]
Obs6	[10,10]
Robot1	[3,2.1000]
Robot2	[4.5000,13.5000]
Robot3	[10.2000,2.3000]

Figure 6.38: Workspace details depicted in Figure 6.37

## 6.6 Conclusion

The software package has been designed and developed to perform path planning to facilitate the process of finding paths for teams of multi-robots. The proposed algorithms have made finding shortest paths based on the Visibility Graph method with Dijkstra's algorithm in the two-dimensional workspace in a simple way because the process of path planning is equipped with pre-calculated step-by-step instructions. The CA and OCA have been developed to address the drawback of the VG method. Both algorithms make pathfinding based on the VG easier as the path planning process has been done systematically. The purposes of the software package are to validate the effectiveness of the proposed algorithms and introduce them in an intuitive manner as the package was designed to be user easy. In addition, the software package has validated the effectiveness of the algorithms proposed through simulations experiments using the software/MATLAB. Simulations have been conducted to compare the performance of the presented algorithms, where we developed different experiments to test the algorithms. In each experiment, robots have successfully reached their target without collisions. This demonstrates the advantage of the method presented in this thesis comparing to the ones with classical methods in the literature. Besides, a set of experiments demonstrated the effect of changing the safe distance on the planned paths. The results of the experiments are promising, as they illustrated the effectiveness, computational efficiency, and adaptability of the presented approach.

In summary, the simulation results confirmed that the CA and OCA based on the visibility graph followed by Dijkstra's algorithm are computationally effective in creating collision-free paths for robots. Besides that, these paths were optimal (short and safe), complete, and have the lowest distance to reach the targets.



## **Chapter Seven**

## 7 Conclusion and Future work

### 7.1 Conclusion

Nowadays, multi-robot systems (MRS) have widely spread, especially with continually improving technology. MRS focus on the control of a team of robots to accomplish tasks more effectively and with more robustness and/or in less time. A large amount of effort has been devoted in the scientific community to the field of multi-robot systems because they exhibit better fault-tolerance, flexibility, performance and are able to share information among their members in a fast and reliable manner. Besides that, it can often deal with tasks that are difficult to achieve using a single robot. Despite recent technology used in a multi-robot system, several problems still need to be addressed, for example, the motion planning problem that was tackled in this thesis.

The main objective of this study is to demonstrate how path planning can be improved using graph theory. By using methods from graph theory, path planning can be done more efficiently and robustly, as shown in the previous chapters. Employing the graph of the potential paths makes the designed algorithms of robots' path planning correlated with the environment model, thus improving their application capability. The algorithms provide global optimality during path planning according to different given optimum criteria such as least paths length, safe paths, effectiveness, computational efficiency, etc. Briefly, path planning includes a problem of finding an optimal (short, and safe) path from a starting position to a goal position. In addition, there are three criteria for path planning which must be considered before designed any path planning algorithm, computational efficiency, path optimality and completeness. This chapter summarises the work done in the thesis on path planning for a multi-robot system, with consideration of all the above-mentioned criteria, also involving the developed path planning algorithms, path planning software packages, and possible extensions of the work that have been developed in this thesis.

#### 7.1.1 2D Path Planning Algorithm

We have proposed a new method to design a roadmap-based path planning algorithm in 2D static environments, named the Multi-Robot Path Planning Algorithm (MRPPA), which assumes a-priori knowledge of robots' positions, their goals' positions, and surrounding obstacles. The algorithm combines the visibility graph method with the algebraic connectivity (second smallest eigenvalue  $\lambda_2$ ) of graph Laplacian and the Dijkstra's algorithm. MRPPA provides robots collision avoidance because it is automatically planning safe paths that do not

intersect each other in the time-space continuum. The algorithm implies sequential path planning for each of robots (path by path) based on the value of algebraic connectivity of graph Laplacian, which controls the inter-robot's connectivity when it is not zero and has a predefined weight evaluation function (edge weights). When planning the path of each robot, all the paths that have been already planned took into consideration potential collisions avoidance. For this reason, the algorithm provides optimality of all planned paths because the paths depend on the order of planning, so the choice of the right sequence for path planning of robots has significant impact on performance of the team and avoids collisions. In addition, visibility graph (VG) is a path planning method that can produce optimal paths if they exist especially when combined with Dijkstra's algorithm. In addition, we have conducted several simulation experiments on different scenarios of workspaces to test performance of the algorithm. The results showed the influence of the algorithm in creating collision-free paths for robots, and visibility graph (VG) is capable of producing an optimal path if one exists. However, the paths planned by this VG method may not be collisions-free, because it is planning paths that passes through the vertices of obstacles, and it is also computationally expensive when the environments are obstacles-rich. Therefore, we have developed the Central Algorithm (CA) coupled with Optimisation Central Algorithm (OCA), which is based on the VG method. The main idea of these algorithms is that the obstacles associated with Central Baseline (CB) are only considered whilst the rest are discarded during the path calculation. Thus, it can create paths relatively fast and is convenient for path planning applications in obstacle-rich environments because it uses a small set of obstacles, whilst retaining the advantages of the VG.

### **7.1.2 2D Path Planning Algorithms Based on Visibility Graph Method**

Two path planning algorithms in two-dimensional (2D) workspace environments have been designed in Chapter 5. The first one, Central algorithm (CA) is utilised to find a path that passes through a set of obstacles in the workspace environment represented by the configuration space (C-space). A Central Baseline (CB), which is determined in the Central algorithm, is a straight line that connects the starting location (s) and goal location(g). Its purpose is to define a set of obstacles (O), which will be employed for paths computation. The algorithms are computationally efficient because the number of obstacles that are used for path computation is relatively small. This means the algorithms find paths by decreasing the number of obstacles (as well as edges) to be taken into consideration, which reduces the calculation time contrary to the VG method. On the other hand, the algorithms hold the completeness criterion as they will generate a path if one exists, hence they solve the problem of the conventional VG method,

and they hold the completeness criterion. The outcome of the developed path planning algorithms is optimal (shortest) and collision-free (safe) paths that direct robots safely away from obstacles. In addition, they reduce the computational complexity of roadmap approaches and are also able to produce general solutions for different environments. The algorithms plan safe paths for the robots in the C-space; so that they can traverse through the vertices of obstacles in different scenarios of workspaces without collisions, even with added new obstacles. It is also worth emphasising that the algorithms possess the criteria of path planning and may be capable of finding a globally optimal path if the knowledge of the environment is fully and accurately known. Note, the optimal paths here mean the safe and shortest paths.

#### **7.1.2.1 Central Algorithm (CA)**

The CA generates a set of waypoints in free C- space around obstacles, and these waypoints are generated from each vertex of each obstacle that intersects with CB. Waypoints are used to establish a part of the visibility graph network from a specific area of the configuration space. As CA contains a relatively small number of obstacles, VG can be established in a relatively short time. The central algorithm then plans paths based on the VG method using Dijkstra's algorithm. CA, however, provides paths that may not be collision-free because it just considers the obstacles that intersect with CB and neglects other obstacles that may be near the paths planned and causes collisions. Therefore, another algorithm called the Optimisation Central algorithm (OCA) had been proposed.

#### **7.1.2.2 Optimisation Central Algorithm (OCA)**

In fact, the Central algorithm is a part of OCA. To ensure the completeness of the OCA, an improvement to CA has been proposed to enhance its performance to generate safer paths and not too close to obstacles. Hence, to address this problem, the size of obstacles has been expanded by a certain distance before the paths are planned. This distance is called the Safety Distance, which could be used to optimise the drawback of the Central algorithm. Safety distance can produce general solutions and acceptable results for different workspaces because it generates waypoints around obstacles in the C-space. Thus, it is planning safe paths for the robots so that they can traverse through the vertices of obstacles in different scenarios of workspaces without collisions, even when new obstacles are added. After establishing the safety distance around obstacles by OCA, CA will be called to find CB from starting points to target (goal) points, which passes through a set of obstacles in the environment represented by the C-space. This procedure guarantees the use of a smaller number of obstacles and therefore reduces the computational complexity when calculating paths in the environment workspace

and provides a great extent of flexibility. Because of this, OCA will result in paths which are collision-free. OCA provides the shortest collision-free paths while maintaining a safe distance from obstacles, which makes it safer. Also, it employs a smaller number of obstacles, and this reduces the computational complexity of roadmap approaches, which means the calculation time decreases when calculating paths. All these advantages make OCA completely effective to produce optimal and complete paths, and it is more efficient than the visibility graph method.

### **7.1.3 Software Package for 2D Path Planning environment**

A path planning software package has been introduced and developed in this project for multi-robot path planning in Chapter 6. The main purpose of the package is to validate the effectiveness of the proposed control algorithms. Additional design intention includes execute and display the algorithms in an intuitive method and is designed to be user friendly using MATLAB. The software package contains two strategies, and each has its own objective. The first strategy is used to implement the Multi-robot Path Planning Proposed algorithm, while the second executes the CA and OCA. Simulations have been conducted to compare the performance of the presented algorithms, where we developed different experiments to test the algorithms. Simulation results demonstrated effectiveness of competed algorithms and confirmed that these algorithms are suitable to be implemented in creating collision-free paths, where the paths were optimal (short and safe), and complete. Also, the results illustrated the effectiveness, computational efficiency, and adaptability of the presented approach.

## **7.2 Future Work**

Although this thesis has demonstrated the potential of finding the optimal path in a workspace environment for a multi-robot system by advanced graph algorithms techniques, opportunities for extending the scope of this thesis remain. This section briefly presents the proposed future work.

The 2D path planning graph algorithms have been successfully used to find the optimal paths for a team of multi robots. Whilst these paths satisfy required standards such as path optimality, completeness, and computational efficiency, nevertheless, improvements are possible. Currently, the MRPPA, CA, and OCA (explained in Chapters 4 and 5, respectively) assume that the workspace environment of starting positions, targets, and obstacles' positions and sizes are accurately known in C-space. Future work must consider unknown environments, especially the positions and sizes of obstacles if applied in a real scenario in real robots' environments. Since the VG based method generates paths through the obstacle's vertices, the lack of accurate knowledge of the obstacles' positions will pose a challenge. In addition, the

algorithms proposed have assumed that all obstacles in workspace are static. However, the obstacles may navigate from one place to another in real scenarios. Thus, future work should consider moving obstacles in the design of path planning algorithms. Furthermore, the proposed 2D path planning algorithms consider a multi-robot system because the tasks performed by MRS achieve better and faster results when compared to those performed by a single robot, if they are applied in real-time. Therefore, the current study could be extended to other graph algorithms and tested to find a solution for the problem of path planning which provide a comparison between results. This is potentially an important area of development in future work. Further research could investigate the different methods of graph theory and the algebraic graph theory since they have an important influence on the results obtained.

Finally, the presented algorithms have shown promising characteristics, and generated better results to find optimal paths for a team of multi-robot in different experiments and scenarios. Therefore, further investigation is needed to improve the execution and lead to applications in the future. The presented work can also be further extended to be applied in different fields, especially since recent advances in robotics enabled more intensive interactions with people and support their daily activities [166]. Some possible environments for future work include warehouses, shopping malls, agricultural fields, and hospitals. Our work can be applied in shopping malls where each robot can interact with people to provide them information about shops and path guidance to facilitate the shopping process [166][167]. In [166] and [167], a robot has been developed to provide customers with shopping centre information, and it has shown its effectiveness in directing people and increasing their interest around the shopping centre [166][167][168]. In agriculture, agricultural robots can be developed to cooperate and perform missions in a simple and safe way rather than the heavy machinery and tractors that are used today. For example, a team of automated vehicles can operate to accomplish agricultural tasks under the supervision of farmers to shorten time and effort [169]. In warehouses, where the operation of warehouses is considered a complex mission that requires many staff to collect, sort, and deliver items, especially in large ones. A team of robots can be employed to perform this task quickly and efficiently [170]. In conjunction with what is happening today around the world from the spread of the coronavirus's disease (COVID-19), our work can be applied in hospitals. For example, in the Department of Communicable Diseases and Epidemics, where the team of robots can help and perform some tasks that may be difficult to carry out or dangerous to the medical staff, especially in the presence of spreads of diseases and epidemics. The environment of the department is known (static obstacles, start point, and target direction), robots can contact patients and provide some necessary things such

delivery of food instead of staff to keep staff safe and reduce contact with them. For example, robots have been used for logistics services in China to protect workers and medical staff from the risk of infection of Coronavirus's disease (Covid-19), such as medical transport, serving foods to patients, spray disinfectants and clean, transport medical samples, and perform diagnostics and thermal imaging [171].

## 8 Appendix [A] Important Concepts from Graph Theory

This Appendix provides an overview of the basic concepts of graph theory and algebraic graph theory used in the thesis and emphasises their importance in strengthening the communication network between robots. It also presents a brief overview of Dijkstra's algorithm to provide the reader with the basic knowledge and information on how it works to find the shortest path to tackle the so-called problem of motion planning. Most of those definitions are standard, so they are not in quotes, but there are references to the related sources.

### 8.1 Graph Theory

A graph is a pair  $G = (V, E)$  that consists of a set of vertices  $V = \{v_1, v_2, \dots, v_n\}$ , (sometimes referred to as nodes, or points), and a set of edges  $E \subseteq V \times V, E = \{e_1, e_2, \dots, e_n\}$  (also referred to as lines) between pairs of vertices. The quantities  $|V|$  (the number of vertices) and  $|E|$  (the number of edges) are, respectively, called the order and size of the graph [134].

The Figure 8.1 is example of graph with the vertex set  $V = \{A, B, C, D, E\}$  and edge set  $E = \{\{A, B\}, \{A, C\}, \{A, E\}, \{B, C\}, \{B, E\}, \{C, E\}, \{E, D\}\}$

Figure 8.1: Example of a graph. The labelled circles represent the vertices  $v_i$ ,  
while the lines between them represent the edges.

As we already mentioned in Chapter two a multi-robot system can be represented by a graph, the vertices are representing the robots (agents) and the edges represent the connections (possibility to communicate) between the robots. There are two different categories of graphs: directed graphs and undirected graphs.

**Directed Graph:**  $G = (V, E)$  consists of a vertex set (a finite set of elements)  $V = \{v_1, v_2, \dots, v_n\}$  and an edge set (a subset of ordered pairs of  $(v_i, v_j)$ ), the “2-element subsets” of  $V$ .  $V \times V = \{(v_i, v_j)\}, i = 1, \dots, n, j = 1, \dots, m, i \neq j, E \subseteq V \times V, (v_i, v_j) \in E \nRightarrow (v_j, v_i) \in E, v_i \neq v_j$ .



Here  $v_i$  is the initial vertex and  $v_j$  is the terminal vertex. The information interchange is unidirectional. These commonly exploited to model unidirectional communication between the robots, which possibly based on the pure sensing [71][144][155][172].

**Undirected Graph:**  $G = (V, E)$  consists of a vertex set (a finite set of elements)  $V = \{v_1, v_2, \dots, v_n\}$  and an edge set (a subset of unordered pairs of  $(v_i, v_j)$ , the “2-element subsets” of  $V$ ).  $V \times V = \{(v_i, v_j)\}, i = 1, \dots, n, j = 1, \dots, m, i \neq j, E \subseteq V \times V, (v_i, v_j) \in E \Leftrightarrow (v_j, v_i) \in E$ . The information interchange is bidirectional. Therefore, it often exploited to model bidirectional communication between the robots [142][173][174][175][176].

Figure 8.2: Example of the directed and undirected graph.

Two vertices are said to be adjacent and are neighbours if they are the endpoints of an edge in graph  $G = (V, E)$ . If edge  $e_i = \{v_i, v_j\} \in E(G)$ , then  $v_i$  and  $v_j$  are adjacent or neighbours.

- An independent set in a graph is a set of pairwise nonadjacent vertices.
- A vertex  $v$  is incident with an edge  $e$  if  $v \in e$ . The degree  $d_G(v)$  or  $(v)$  of a vertices  $v$  is the number  $|E(v)|$  of edges incident at  $v$ . The number  $\delta(G) := \min\{d(v) | v \in V\}$  is the minimum degree of  $G$ . The number  $\Delta(G) = \max\{d(v) | v \in V\}$  is the maximum degree of  $G$ , see Figure 8.3 [173][174][175][176].

Figure 8.3: Example of degree of graph

- A graph  $(G)$  is regular if  $\Delta(G) = \delta(G)$ , Graph  $(G)$  is  $k$ -regular if the degree is  $\deg(v) = k$  for all  $v \in G$ ; if graph  $(G)$  is 3-regular it is called cubic, see Figure 8.4.

Figure 8.4: Example of regular graph

- A subgraph of a graph  $G$  is a graph  $H$  such that:  $V(H) \subseteq V(G)$  and  $E(H) \subseteq E(G)$ , and the assignment of endpoints to edges in  $H$  is the same as in  $G$ . Figure 8.5 example of subgraph where:  $H_1$ ,  $H_2$ , and  $H_3$  are subgraphs of  $G$  [176].

Figure 8.5: Example of subgraph

- An induced subgraph: If  $G \subseteq G'$  and  $G'$  contains all the edges  $v_i v_j \in E$  with  $v_i v_j \in V'$   $V' \subset V$ , then  $G'$  is an induced subgraph of  $G$ . A set  $S$  of vertices is an independent set if and only if the subgraph induced by it has no edges, see Figure 8.6 [129][173][174][176].
- A spanning subgraph  $H$  of a graph  $G$  is a subgraph obtained by edge deletions only, in other words, a subgraph whose vertex set is the entire vertex set of  $G$  [117][173][174][176].

Figure 8.6: Example of induced subgraph and independent set

$G_1$  is isomorphic to  $G_2$ ,  $G_1 \cong G_2$  if there exists a bijection  $f: V(G_1) \rightarrow V(G_2)$  such that:  $[v_i v_j \in E(G_1) \Leftrightarrow f(v_i) f(v_j) \in E(G_2)]$  for all  $v_i, v_j \in V$ , see Figure 8.7, where  $\{f_1: w \rightarrow c, x \rightarrow b, y \rightarrow d, z \rightarrow a\}, [f_2: w \rightarrow a, x \rightarrow d, y \rightarrow b, z \rightarrow c]\}$ .

Figure 8.7: Example of isomorphic subgraph

- A path is a sequence or a list of adjacent vertices such that two consecutive vertices are adjacent [90][129][174][176].  
 $v = \{v_1, v_2, \dots, v_n\}$  and edge set  $E = \{\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_n, v_{n-1}\}\}$ .

- A simple path is a path where the vertex is not used more than once.
- A cycle is a simple closed path where the first vertex and last vertex are the same. See example in Figure 8.8 where: (a, d, c, b, e) is a path, and (a, d, c, b, e, a) is a cycle [90][129][174]176].
- The directed graph without cycles is called a directed acyclic graph [90][129][174]176].

Figure 8.8: Example of a path and a cycle in a graph

- A graph without cycles is called a forest, and a connected forest is called a tree.
- A tree  $T$  is a connected graph that has no cycles and often has a pyramid shape or a hierarchical structure [90][174][176][177].
- A spanning tree for a graph is a subgraph of  $G$  which is a tree that includes every vertex of  $G$  [90][174][175][176], see Figure 8.9.

Figure 8.9: Example of forest, tree and spanning tree graph.

- Loop: An edge whose endpoints are equal.
- Multigraphs are graphs that contain multiple edges.
- A simple graph is the graphs without loops or multiple edges [90][174][176][177].

Figure 8.10: Example of simple, multiple, and loop graph and cycle

- A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. A complete subgraph in a graph, called a clique, is a set of pairwise adjacent vertices.
- A weighted graph is defined as a graph  $G = (V, E)$  where  $V$  is a set of vertices and  $E$  is a set of edges  $E = \{(v_i, v_j) | v_i, v_j \in V\}$  associated with a weight function  $w: E \rightarrow R$ , where  $R$  denotes the set of all real numbers or weights that are assigned to each edge of the graph. Most of the times, the weight  $w_{ij}$  of the edge between nodes  $i$  and  $j$  indicates either distance (temporal geometric) or cost. In addition, the weight of the edge represents the relevance of the connection, and by that way, more information is linked to the graph  $G$  [90][174][176][177], see Figure 8.11.

Figure 8.11: Example of a weighted graph

A graph is called connected if any two vertices are linked by a path or there is an edge from each vertex to every other vertex in graph G.

- A maximal connected subgraph of G is called a component of G.
- The distance between  $v_i$  and  $v_j$  in G, denoted by  $dG(v_i, v_j)$ , is defined as the length of the shortest path between  $v_i$  and  $v_j$  contained in G; if no such path exists then  $dG(v_i, v_j) = \infty$ .
- A graph G is called disconnected if there are non-connected parts of vertices and edges [90][170][174][176][177].

Figure 8.12: Example of connected and disconnected graphs

- A graph G is bipartite if  $V(G)$  is the union of two disjoint independent sets called partite sets of G [90] [173][174][178].

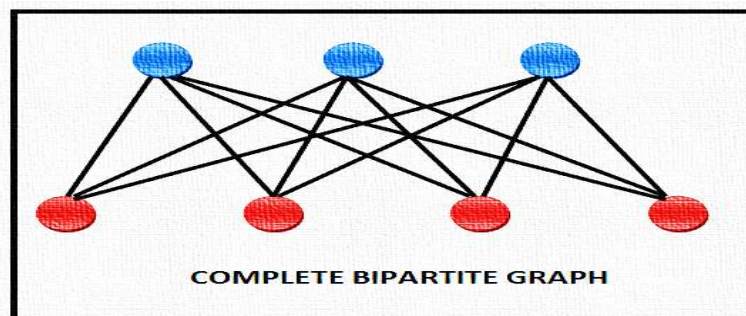


Figure 8.13: Example of Bipartite graphs

Weinstein, E. W. (2008). Complete Tripartite Graph. Available online at <http://mathworld.wolfram.com/CompleteTripartiteGraph.html>

- Planar graphs: a graph that can be drawn on a plane without any lines crossing, or if it can be drawn on the plane so no edges intersect with each other (other than the endpoint), such as a graph A in Figure 8.14.

- A non-planar graph is a graph that has many overlapping links, or its edges intersect each other's, such as graph B in Figure 8.14 [174][176][177].

Figure 8.14: Example of planar and nonplanar graphs

[https://transportgeography.org/?page\\_id=6003](https://transportgeography.org/?page_id=6003).

## 8.2 Graph Rigidity

Graph rigidity: a graph is called rigid if there is an embedding of a graph in a Euclidean space that is structurally rigid. Also, a graph is rigid if the structure formed by replacing the edges by rigid rods and the vertices by flexible hinges is rigid such a cycle graph  $C_3$ . Whilst a graph that is not rigid is called flexible, such as a cycle graph  $C_4$  [179].

On the other hand, "a framework (or graph) is rigid if the only allowed motions satisfying the constraints are those of the complete graph or if and only if the continuous motion of the points of the configuration maintaining the bar constraints comes from a family of motions of all Euclidean space which are distance-preserving" [179].

A **framework** is rigid if there exists a neighborhood  $Z \subset R^{Nd}$  of  $p$  such that:  $gG^{-1}(gG(p) \cap Z = gk^{-1}(gK(p)) \cap Z$  [176][178].

**Bar-and-joint framework:** A framework in  $R^d$  is a pair  $(G; p)$ , where  $G = (V, E)$  is a graph and  $P: V \rightarrow R^d$  is a map from  $V$  to  $R^d$ . We consider the framework to be a straight line embedding of  $G$  in  $R^d$  in which the length of an edge  $(v_i, v_j) \in E$  is given by the Euclidean distance between points  $p(v_i)$  and  $p(v_j)$ . Let  $(G, p_1)$  and  $(G, p_2)$  be frameworks then [178] [180][181]:

- **Framework equivalency:** two frameworks  $(G, p_1)$  and  $(G, p_2)$  are equivalent if they have the same edge lengths  $gG(p_1) = gG(p_2)$ .
- **Framework congruency:** two frameworks  $(G, p_1)$  and  $(G, p_2)$  are congruent if

$|p_1(v_1) - p_1(v_2)| = |p_2(v_1) - p_2(v_2)|$ , for all  $v_i, v_j \in V$  (the constraints are satisfied over all the possible edges [178] [180][181].

- $(G, p_1)$  is rigid if there exists an  $\varepsilon > 0$  such that every framework  $(G, p_1)$  which is equivalent to  $(G, p_2)$  and satisfies  $|p_1(v) - p_2(v)| < \varepsilon$  for all  $v \in V$ , is congruent to  $(G, p)$ .
- $(G, p_1)$  is globally rigid if every framework  $(G, p_2)$  which is equivalent to  $(G, p_1)$ , is congruent to  $(G, p_2)$  [178] [180].

**Graph is minimally rigid** if it is rigid and if no single edge can be removed without losing rigidity (minimally rigid= keep formation rigid with minimal number of edges), or a graph  $G$  is minimally rigid in  $R^d$  if  $G$  is rigid and  $G - e$  is not rigid for all  $e \in E$  [178] [180][181].

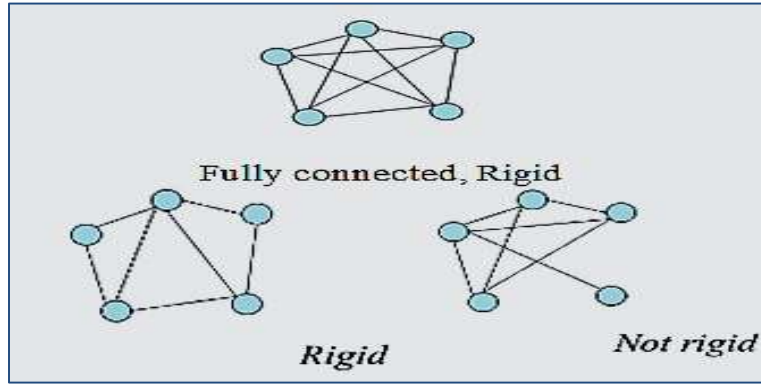


Figure 8.15: Example of a rigid and a non-rigid graph

<http://slideplayer.com/slide/5019665/>

### 8.3 Algebraic Graph Theory

There are several matrices that can be associated with graphs, and many graph properties can be deduced from the associated matrices. The following algebraic tools are fundamental for linking graph theory to the study of multi-robot systems when they are viewed as collections of dynamical systems. Given a graph  $G = (V, E)$  we can assign two possible directions to each edges of graph, thus we can define the Incidence matrix over the edge set. The Incidence matrix  $B(G) \in R^{n \times m}$  can be defined as:

$$B = (b_{ij}) = \begin{cases} -1 & \text{if } e_i = (v_i, v_j) \\ 1 & \text{if } e_i = (v_j, v_i) \\ 0 & \text{otherwise.} \end{cases} \quad (8.1)$$



Where  $M$  is the cardinality of the edge set and  $e_i$  is the  $i$ -th edge of  $G$ . When the orientation map is not defined, a random orientation can be chosen [49][82][175][182][183]. The incidence matrix corresponding to the graph in Figure 8.1 is given by:

$$B = \begin{bmatrix} -1 & 0 & 0 & -1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & 1 & -1 & 1 \end{bmatrix} \quad (8.2)$$

The communication graph can be described by means of the adjacency matrix  $A \in \mathbb{R}^{n \times n}$ . Each element  $a_{ij}$  is defined as the weight of the edge between the  $i$  and the  $j$  robot and is a positive number if  $j \in N_i$ , zero otherwise, [49][50][158][165].

$$A = (a_{ij}) = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases} \quad (8.3)$$

$$a_{ij} = a_{ji}, \quad a_{ii} = 0 \text{ for all } i.$$

It is square and symmetric i.e.,  $A = A^T$  (only for undirected graphs). Communications among robots  $i$  and  $j$  are possible if the  $(i, j)$  entry in  $A(a_{ij})$  has a value of one (1), however if the  $(i, j)$  entry in  $A(a_{ij})$  has a value of zero (0) then there are no connections between robot  $i$  and  $j$ . Here a graph is symmetric that means all connections exists in both directions and thus each robot  $i$  that can receive data from another robot  $j$ , can transmit the data back to that robot  $j$  [50][175][183][184]. The Adjacency matrix corresponding to Figure 8.1 is given by:

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix} \quad (8.4)$$

Notice that: the adjacency matrix can be computed numerically using MATLAB predefined codes.

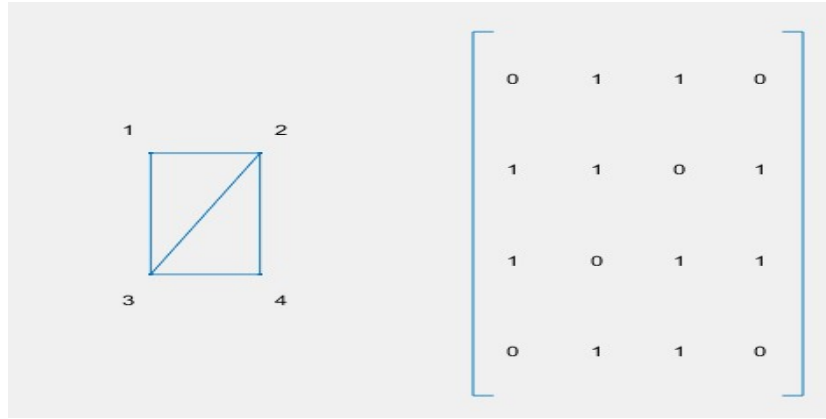


Figure 8.16: Example of adjacency matrix

The degree matrix  $D \in \mathbb{R}^{n \times n}$  is a Diagonal matrix showing a number of edges connected to each vertex  $v_i$ . the  $(i, i)$  entry in  $D$  ( $d_{ij}$ ) is given by:

$$d_i = \sum_{j=1}^n A_{ij} \quad (8.5)$$

As an example, the degree matrix corresponding to the graph in Figure 8.1 is given by:

$$D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4 \end{bmatrix} \quad (8.6)$$

A matrix that plays a central role in many graph-theoretic treatments of multi-robot systems (mobile robots) is the Laplacian matrix, which is defined as:

$$L = D - A \text{ or } L = BB^T \quad (8.7)$$

The Laplacian matrix corresponding to the graph in Figure 8.1 is given by:

$$L = \begin{bmatrix} 3 & -1 & -1 & 0 & -1 \\ -1 & 3 & -1 & 0 & -1 \\ -1 & -1 & 3 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & -1 & 4 \end{bmatrix} \quad (8.8)$$

Note that all diagonal entries are non-negative while all off-diagonal entries are non-positive [49] [50][184]. The Laplacian matrix  $L$  exhibits some remarkable properties:

1.  $L$  is positive semi-definite.

2. The eigenvalues of the Laplacian matrix are always non-negative. Moreover, they can always be ordered as follows [183][185]:

$$\text{eig}(L) = \{0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n\}$$

3. If  $\lambda_1$  is a simple eigenvalue (*i. e.*  $0 = \lambda_1 < \lambda_2$ ), then the graph is connected. Moreover, in this case,  $\text{null}(L) = \text{span}\{1\}$ , where  $\vec{1} = [1 \dots 1]^T$  and  $\vec{0} = [0 \dots 0]^T$  are vectors of  $N$  elements all equal to 1 and 0 respectively. This implies  $L1 = 0$ , [49][50][183][185][186].

## 8.4 The importance of graphs in networks analysis

To understand the basics of networks theory, how networks are composed, and what their relation to graphs is, we need to know the basic information about the basic elements and properties of different networks. All this can be achieved and realized via the concepts of graphs and their elements (vertices  $V$ , and edges  $E$ ), where the description of the network analysis is based on the graph theory. In the domain of geography and colloquial language, is often used the term network for all types of graphs such the directed weighted graphs are called networks [46][187][188].

### 8.4.1 The connectivity of the network (graph)

The connectivity degree for the graph is measured via Beta index ( $\beta$ ) that measures the communication density, which is given by:  $\beta = \frac{E}{V}$ , where  $V$  is the overall number of vertices in the network (graph) and  $E$  is the overall number of edges [142]. For example, in the Figure 8.17 the number of connecting edges is increasing gradually from four edges to ten edges whereas the number of the vertices is staying fixed. The connectivity between vertices rises with the increasing number of edges in the graph [142]. Also, the value of Beta index is changing by increasing the number of edges or vertices as follows: Graph (1) in Figure 8.17,  $\beta = \frac{4}{5} = 0.8$ , while Graph (6) in Figure 8.17,  $\beta = \frac{10}{5} = 2$

Figure 8.17: Example of Beta index calculated for different graphs

### 8.5 Connectivity in multi-robot networks

Let  $G(t) = (V, E(t), W(t))$  denote a graph on  $n$  vertices indexed by the set of mobile robots, where  $V = \{1, \dots, n\}$ ,  $E(t) \subset V \times V$  is the set of edges at time  $t$ , and  $W(t) = \{w_{ij}(t) \mid (i, j) \in V \times V\}$  is a set of weights so that  $w_{ij}(t) = 0$  if  $(i, j) \notin E(t)$  and  $w_{ij}(t) > 0$  otherwise. If  $w_{ij}(t) = w_{ji}(t)$  for all pairs of vertices  $(v_i, v_j)$ , then the weights are called symmetric, if a graph has symmetric weights, then it is also undirected; otherwise, it is called directed [89][90]. Also, the set of neighbours of node  $i \in V$  is  $N_i(t) = \{j \in V \mid (i, j) \in E(t)\}$ . Consider a set of  $n$  robots in  $R^m$  and let  $x_i(t) \in R^m$  denote the position of robot  $i$  at time  $t$ . The robots can be described by either single integrator models, or double integrator models. Single integrator is:

$$\dot{x}_i(t) = v_i(t) \quad n = 1, \dots, N \quad (8.9)$$

Where  $v_i(t) \in R^m$  refers to the control input to robot  $i$  at time  $t$ , or double integrator models is:

$$\dot{x}_i(t) = u_i(t) \quad i = 1, \dots, N \quad (8.10)$$

$$\dot{u}_i(t) = v_i(t) \quad i = 1, \dots, N \quad (8.11)$$

Where  $u_i(t) \in R^m$  refers to the speed of robot  $i$  at time  $t$  [89].

The weight function  $w \in R^m$  assigned to each edge  $e_{ij}$ , can be seen as a function of the distance between robots  $i$  and  $j$ ; such that

$$w_{ij}(t) = w_{ji}(t) = f(x_i(t), x_j(t)) = f(\|d_{ij}(t)\|) \quad (8.12)$$

Where  $w_{ij}(t)$  are positive edge weight functions,  $x_i(t)$  is the position corresponding to vertex  $v_i$  at time  $t$  (the state of  $i$ -th agent at time  $t$ ),  $d_{ij}$  is the Euclidean distance between vertices  $(v_i, v_j)$  given by  $d_{ij}(t) = \|x_i(t) - x_j(t)\|$ , and the function  $f$  to be a decreasing function of the inter-robot distanced  $\|d_{ij}(t)\|$ , [89][103] such that

$$\begin{aligned} 0 \leq f(\|d_{ij}(t)\|) \leq 1 & \quad \text{if } \|d_{ij}(t)\| \leq \rho_e \\ f(\|d_{ij}(t)\|) = 0 & \quad \text{if } \|d_{ij}(t)\| > \rho_e \end{aligned} \quad (8.13)$$

## 8.6 Algebraic Connectivity of Graph

The second smallest eigenvalue  $\lambda_2(L)$  of the Laplacian matrix  $L \in \mathbb{R}^{n \times n}$  or the Fiedler eigenvalue plays an important role for network analysis and connectivity, where the graph is connected if and only if  $\lambda_2(L) > 0$  [113]. Therefore,  $\lambda_2(L)$  also called algebraic connectivity value of network.  $\lambda_2(L)$  is a measure of the degree of connectivity in a graph. The larger its value, the “more connected” the graph is.

Figure 8.18: Examples of connected graph

In this work, we suppose that the communication topology in a team of multiple-robots can be described by an undirected weighted graph  $G = (V, E)$ , where  $v_i \in V$ ,  $i \in N$ ,  $N = \{1, \dots, n\}$  are vertices,  $e(v_i, v_j)$  is an edge between vertices  $v_i$  and  $v_j$  equipped with the weight function  $w: E \rightarrow \mathbb{R}$  that maps edges to real-valued weights  $w_{ij}$ . That means that if the  $i$ -th robot can obtain data from the  $j$ -th one, the  $j$ -th robot can obtain data from the  $i$ -th one as well [89][176][183]. The Neighbours subset of the  $i$ -th agent (robot) is defined as:

$$N_i = \{\forall v_j \in V : (v_i, v_j) \in E\},$$

Each robot is assumed to be able to interchange data with their neighbours, i.e. with all the agents that are in its neighbour subset. In many applications, the weight is a function of the distance between robots  $i$  and  $j$ . For the weight of the edge  $e(v_i, v_j)$ , we labelled the weight of

the edge  $e(v_i, v_j)$ , either as  $w(v_i, v_j)$  or  $w(e_{ij})$ . Interactions (e.g. communication links) among the robots are presented as weights of edges that can be defined by the Adjacency matrix  $A(t)$ , where if the robot  $i$  and robot  $j$  are adjacent or neighbours in  $A(a_{ij})$  then the communication links among robots are possible and have positive values, while if the robot  $i$  and robot  $j$  in  $A(a_{ij})$  are not adjacent, then there is no connection between them and it has a value of zero [183][189]. The Adjacency matrix  $A(t) \in \mathbb{R}^{n \times n}$  of the weighted graph  $G$  which is defined by:

$$[A(t)]_{ij} = w(t)_{ij} \quad (8.14)$$

While  $D(t) = [\text{diag} \sum_{j=1}^n w_{ij}(t)]$  denotes the Diagonal matrix of degrees of the weighted graph  $G$ .

The Laplacian matrix  $L(t) \in \mathbb{R}^{n \times n}$  of weighted graph  $G$  with entries is defined by:

$$[L(t)]_{ij} = \begin{cases} \sum_{j=1}^n w_{ij}(t) & \text{if } i = j \\ -w_{ij}(t) & \text{if } i \neq j \end{cases} \quad (8.15)$$

Or the Laplacian matrix can be written as:

$$L(t) = D(t) - A(t) \quad (8.16)$$

The Laplacian matrix of weighted graph  $G$  with symmetric weights is always a symmetric positive semi definite matrix. The connectivity between robots is measured by the second smallest eigenvalue  $\lambda_2(L)$  of the Laplacian matrix  $L(t)$ , where the robots have perfect connectivity if  $\lambda_2(L)$  has a great value [189].

## 8.7 Dijkstra's Algorithm

Dijkstra's algorithm is one of the most famous solutions for path planning problems. Many researchers have proposed different ways to develop the storage structure of the algorithm such as the vertex search way to improve the efficiency of the algorithm and an improvement to data storage structure. An approach to data storage structure has been applied to address the problem of large memory usage resulting from the adjacent matrix. The heap structure was adopted to store vertices based on the improved storage structure that can decrease the vertex finding time and reduce storage space. Furthermore, the improvement of conventional Dijkstra's algorithm has been considered in terms of storage structure and the search area in the planning of road networks methods [141][190]. On the other hand, using the Dijkstra algorithm it is possible to define the shortest distance (the lowest cost / or least effort) between a start vertex and any other vertex in a graph or from a single source vertex ( $S$ ) to all other vertices ( $v_i$ ) for a weighted graph with nonnegative edge path costs. Edge Cost ( $e$ ) is the distance between two vertices. If the edges have negative values, then the actual shortest path cannot be obtained. Besides, it

processes vertices in order of their distances from the source (which might be the starting point, or any other location indicated as a vertex) [191][192][193]. Moreover, it adopts to obtain a suitable path from source to reach the target and find out all the free lines that intersects the path at  $v_i$ , then connect vertices of obstacles corresponding to those lines, with initial and goal positions  $(s, g)$ . Through this method, we can get bounded area where the optimal path inside it. The path is a link of series of edges. The idea of the algorithm is to continuously compute the shortest distance starting from a beginning point and to exclude longer distances when making an update [141][190][191][192][193].

### 8.7.1 Description of Dijkstra's algorithm

**Firstly:** let start with the general description of the algorithm, suppose that we want to find the shortest path from a certain vertex  $v$  to other vertices in a network (one-to-all shortest path problem). Dijkstra's algorithm solves this problem in a following way: The algorithm finds the shortest path from a certain vertex  $v$  to all other vertices in the network, and it finds the next nearest vertex through preserving the new border vertices in a priority queue. The vertex  $v$  is called a starting or an initial vertex [70][72][142][190].

#### How does the algorithm operate?

Dijkstra's algorithm begins by assigning some initial (starting) values for the distances from vertex  $v$  and to each other vertex in the network. It is working on stages, where the algorithm improves the distance values in each stage. Also, at each stage, the shortest distance from vertex  $v$  to another vertex is determined [70][72] [191][194][195].

#### How is the algorithm achieving this?

The algorithm achieves this through maintaining the shortest distance of vertex  $v$  from the source in an array and the shortest distance from the source to itself is zero. In addition, the distance for all other vertices is set to infinity to refer to that those vertices are not processed yet. After the algorithm finishes processing the vertices, it will have the shortest distance of vertex  $v$  to  $w$ . Dijkstra's algorithm works to calculate the shortest path from the source  $s$  to vertices  $w$ , between source  $s$  and vertex  $w$  there will be a set called frontier of  $x$  vertices which consist of the closest vertices to the source  $s$ , and the vertices that are located outside frontier will be calculated and placed in a set called the new frontier. The algorithm finds a next nearest vertex by keeping the new frontier vertices in a primary queue. The two sets are preserved of frontier and new frontier that help in the processing of the algorithm. The frontier has  $x$  vertices

that are nearest to the source, the algorithm will have calculated the shortest distances to these vertices, for restricted paths up to  $x$  vertices. [70][146].

### **Secondly: the formal description of the algorithm**

Dijkstra's algorithm distinguishes each vertex via its case. The case of a vertex contains two features which are: case label of a vertex and distance value of a vertex. The case label of a vertex is an attribute that determines whether the distance value of a vertex is equal to the shortest distance to vertex  $v$  or not, and it is permanent if its distance value is equal to the shortest distance from vertex  $v$ . Otherwise, the case label of a vertex is temporary. Whereas the distance value of a vertex is a scalar that represents an estimate of its distance from vertex  $v$ . Furthermore, the algorithm maintains and step-by-step updates the cases of the vertices, where at each step, one vertex is set as a current vertex [70][193][191].

**Note that:** In the most studies, the general description of Dijkstra's algorithm always uses the terms intersection, road, and map just for the ease of understanding. For example, how we can find the shortest path between two locations on a town map, a starting point, and a destination. However, these terms officially described as a graph, vertex, and edge, respectively [70][181][191].

### **8.7.2 Runtime of Dijkstra's algorithm**

Normally, a runtime of the Dijkstra's algorithm relies on the data structure utilised to store the distance to each vertex. The maximum bound of the running time of Dijkstra's algorithm can be expressed on a graph as a function of  $|E|$  and  $|V|$ . There exist three commonly used choices of the runtime of the algorithm [70][185][191]. One simple choice of Dijkstra's algorithm is to store the distances for vertices of set  $S$  in an array or ordinarily connected list and extract the least (minimum) one from  $S$  which obviously represents a linear search via all vertices in  $S$ , where  $V = \{1, \dots, n\}$  and the input distance  $[i]$  stores the distance to vertex  $v$ . In this state, the running time is  $O(|E| + |V|^2) = O(|V|^2)$ . Also, point 8 in the pseudo-code of the algorithm in page 70 requires  $O(n)$  runtime every time it is performed. Since the while-loop runs  $n$  times, the runtime of Dijkstra's algorithm using an array is  $O(n^2)$  [70][142][191][196][197].

Another selection is the priority queue that also utilises a data structure, where the least element is stored at the top of a binary heap for easy reaches. The least element can be removed in this data structure from  $Q$  in  $O(\log n)$  time. However, the point 13 in the pseudo-code of the



algorithm in page 71, in order to maintain the priority queue correctly,  $O(\log n)$  time also requires changing the value of an input of the queue for each operation [70][142][191][196][197].

Note that there is a more progress data structure called a Fibonacci heap, where the runtime of Dijkstra's algorithm becomes  $O(n \log n + m)$ . This is not usually using in practice because of the complexity of the structure [70][142][191][196][197].

The graph  $G$ , its vertices, edges, edge weights, and the distance may have various meanings, as the path distance can be changed. Also, a priority queue can be maximum or minimum. As well as the relaxation operation can vary and require a various distance initialisation [198][199]. The algorithm above can be understood and explained completely by using the example in Figure 8.19, where it has shown an undirected Weighted Graph ( $G$ ) and a vertex ( $v$ ) to find the shortest path to all reachable vertices from the vertex ( $v$ ). On the other hand, for a given vertex like the initial vertex  $S$ , the algorithm finds the shortest path to all other vertices if it is reachable from the vertex. The undirected graph  $G = (V, E)$ , consists of six vertices and nine edges, where

$$v = \{1, 2, 3, 4, 5, 6\}, E = \{(1, 2), (1, 3), (1, 6), (2, 3), (2, 4), (3, 4), (3, 6), (4, 5), (5, 6)\}.$$

Notice that the graph should not contain a negative edge which means all edges must have a positive number in this graph, in the example of the graph ( $G$ ) in Figure 8.19 each step that is taken is explained below, how distance is measured, and how to obtain the shortest path from start vertex one 1 to goal vertex 5. Dijkstra's algorithm is used in routing, and the images below illustrate the working of this algorithm.

.

Figure 8.19: Example of steps of Dijkstra's Algorithm

<http://www.programminggeek.in/2013/08/java-implementation-of-dijkstra-shortest-path-algorithm-for-coursera-programming-assignment-5.html>.

Dijkstra's algorithm chooses the unvisited vertex with the least-distance, then calculates the distance through it to each unvisited neighbour, and updates the neighbour's distance if a new distance is lower. It marks visited vertices (set to red colour) when done with neighbours. In the second step in the example above, the algorithm picks vertex 3 because the distance from vertex 1 to vertex 3 is less than the distance to the vertex 2, this means  $\{[v_1 + v_3] = 9 < [v_1 + v_2 + v_3] = 7 + 10 = 17\}$ , then it chooses the vertex 6 instead of the vertex 4, because the distance from vertex 3 to vertex 6 is lower than the distance from vertex 3 to the vertex 4, which is equal to 11(eleven)  $\{[v_3 + v_6] = 9 + 2 = 11 < [v_3 + v_4] = 9 + 11 = 20\}$ . Finally, the algorithm picks the vertex 6 to reach the goal vertex 5, where  $\{v_6 + v_5\} = 11 + 9 = 20\}$ .

## 8.8 The shortest path

The shortest path between two vertices is a path with the shortest length (least number of edges), also called the link distance. In undirected weighted graph  $G = (V, E)$ , with the weight function  $w: E \rightarrow R$  mapping edges to real-valued weights  $w_{ij}$ , if the weight of the edge  $e = (v_i, v_j)$ , we write either  $w(v_i, v_j)$  for  $w(e_{ij})$ . The path of a robot can be described itself as a graph, and the length of a path  $p = \langle v_1, v_2, v_3, \dots, v_m \rangle$ , where  $v_i$  a position of robot along its path, would be the sum of the weights of its constituent edges expressed as: The length of a path  $(p) = \sum_1^m w(v_{i-1}, v_i)$ . The distance from  $v_i$  to  $v_j$ , denoted by  $\delta(v_i, v_j)$ , is the length of the minimum path if there exists a path from  $v_i$  to  $v_j$ ; and  $\infty$  is otherwise. To construct the shortest path: in each stage add a new one edge, corresponding to the built of the shortest path to the present new vertex. Where this is done in the following stages [72][131][132]:

- Initialisation: Every vertex has its associated “distance”, representing to the length of the path to it, and all other vertices with “distance” values are set to "infinite", that is

each vertex is assigned as unvisited (a distance value is  $\infty$ ), except the starting vertex  $s$  which is assigned a zero distance value (The case of vertex  $s$  is  $d[s] = 0$ , and the case of all other vertices is  $[v] = \infty$ ).

- Maintain an estimate  $[v]$  of the length  $(s, v)$  of the shortest path for each vertex  $v$  as follows.
  - Start with setting marks for the distance of the beginning vertex  $s$ , and label it as permanent, whereas all other vertices are labelling as temporarily.
  - Designate vertex  $s$  as the current vertex.
  - Assign an active start vertex, and compute the temporary distances of all neighbour vertices of the active vertex by summarizing the distance with the weights of the edges
  - If a computed distance of a vertex is smaller as the current one, update the distance and set the current vertex as predecessors. This stage is also called 'update', and its Dijkstra's central idea.
  - Set the vertex with the minimal temporary distance as active and mark its distance as permanent.
  - Repeat the steps until there no vertices are left with a permanent distance, where their neighbours still have temporary distances.
  - In each repetition, select the unvisited vertex with least distance indicated it as visited [72][131][132][142].

**Note that:** the algorithm finishes when the goal vertex is visited. Always  $d[v] \geq (s, v)$ , for all  $v \in V$  and  $[v] = \infty$  if we have no path so far. The processed vertex's estimate will be validated as being the least real distance,  $d[v] = (s, v)$ . The processing of the vertex  $v$  consists of finding a new path and updating  $[v]$  for all  $v \in adj[u]$  if needed. The process by which an estimate is updated is called relaxation. When all vertices are processed,  $d[v] = (s, v)$  for all  $v$ . Dijkstra's algorithm will find new paths when processing of the vertex  $v$ , after that it will test all vertices  $v \in adj[u]$ , for each vertex  $v \in adj[u]$ , a new path from  $s$  to  $v$  is found (i.e., a path from  $s$  to  $v$  + new edge). In addition, the algorithm will do the relaxation If a new path from  $s$  to  $v$  is the least or shorter than  $d[v]$ , then update  $d[v]$  to the length of this new path. Also, whenever we assign  $d[v]$  to a limited value, there is a path of this length. So  $d[v] = (s, v)$ . Furthermore, if  $d[v] = (s, v)$ , hence more relaxations cannot alter its value) [70][72][131][190][199].

**Observation 1:** The shortest path to the vertex does not pass twice the same vertex.

**Proof:** A path that passes twice the same vertex contains a cycle. Removing cycle gives a shorter path [191][194][200].

**Observation 2:** Any sub-path from the shortest path must be also the shortest path.

**Proof:** When we define the shortest path to the vertex  $v$ , then the paths that continue from  $v$  to each of its neighbours can be the shortest paths to each of those adjacent vertices. Which means, if the path  $(P) = (v_1, v_2, \dots, v_m)$  is the shortest path from  $v_1$  to  $v_m$  then the sub path from  $v_1$  to each of the vertices  $v_k \in p(v_1, v_2, \dots, v_k)$  is also the shortest path, where  $P = (v_1, v_2, \dots, v_k) \subset P = (v_1, v_2, \dots, v_m)$ . As we can see in the following example of the graph in Figure 8.20 the shortest path is  $(\langle A, B, C, E \rangle)$  and the sub path  $(\langle A, B, C \rangle)$  is the shortest path of the graph as well [70][191][201][202].

Figure 8.20: Example of the shortest path in graph

**Observation (3):** the shortest path from the vertex to any other vertex cannot contain any cycle.

**Proof:** Each edge in the graph has a positive weight [70][186].

In the example of the graph above in Figure 8.19 Dijkstra algorithm can be illustrated in how to find the shortest path between start vertex to all other vertices to get the goal destination, where Figure 8.19 depicts an undirected weighted graph  $G$ , this graph  $G$  consists of six vertices marked from one to six. The length between two adjacent vertices is indicated as edge cost which is the term used for the value between two vertices. For instance, the edge cost between vertices 1 and 2 is 7 [70][132][146]. As a result of the above considerations is shown in the example of in Figure 8.19, it has displayed how Dijkstra's algorithm works and how it is applied to the graph. What is more, how it has obtained the shortest path from the initial position to goal position as shown in Figure 8.21, whereas the algorithm starts with just the initial configuration (vertex one) in  $S$ . Then, it relaxes the neighbors of the set  $S$  by updating the shortest path if needed. After that, the configuration  $u$  in  $V$  with the shortest path is moved into  $S$ . This process continues even if there are no extra configurations inside the set  $V$  [69][132][141][146][201][202]. In this method, the algorithm not just finds the solution, but it

also finds the shortest path among the initial point and any other configuration to reach the target. The shortest path obtained from the example in Figure 8.19 above is visualised in Figure 8.21 is:  $\{ [v_1 + v_3 + v_6 + v_5] = 9 + 2 + 9 = 20 \}$ .

Figure 8.21: Example of shortest path in graph

## 9 Appendix [B]

### 9.1 Program 1:

This program for calculate the shortest paths using Visibility graph and Dijkstra's algorithm for multi-robots systems. This code is partially based on the codes:

```
Cornell University
% MAE 4180/5180: Autonomous Mobile Robots
% Final Competition
% Pu, Kenneth (kp295)
function [path,goal,gfound] = dijkstra(V,E,start,goals)

% DIJKSTRA: Takes as input a graph represented by a set of nodes V and edges E, a start
position, and a list of goal positions, then returns
% the shortest path between start node and the closest goal node.
%Uses
% Dijkstra's algorithm
% [PATH,GOAL,GFOUND] = DIJKSTRA(V,E,START,GOALS) returns
%the shortest path between a start and goal node given a set of nodes V and edges E
%INPUTS
    V Set of nodes in graph
    E Set of edges in graph
    start 1-by-2 array containing x/y coordinates of start node
    goals N-by-2 array containing x/y coordinates of goal nodes
%OUTPUTS
%path N-by-2 array containing a series of points representing the shortest path connecting
initial and closest goal points
% goal 1-by-2 array containing x/y coordinates of closest goal node
% gfound An integer denoting the number of goals found
Examples demonstrate how to find shortest paths for three robots
Example one
Uses
% Dijkstra's algorithm
% [PATH, GOAL, GFOUND] = DIJKSTRA (V, E, START, GOALS) returns
% Adjust Properties of GraphPlot Object
% Create a |GraphPlot| object, and then show how to adjust the properties
% of the object to affect the output display.
```

% Create and plot a graph.

% Copyright 2015 the MathWorks, Inc.

```
s=[1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 5 5 5 6 6 6 6 6 6 7 7 7 8 8
8 8 8 8 8 9 9 9 9 10 10 10 11 11 11 11 11 11 12 12 12 12 12 12 13 13 15 16 16 16 16 16
15 17 17 17 17 17 17 18 18 19 20 4 4 5 6 8 8 12 14 16 19 19 20 20 21 21 22 22 22 23
24 24 25 25 26 26]; t=[2 3 4 5 6 7 8 9 15 3 4 5 10 18 24 25 27 5 10 14 18 25 27 7 9 11 24
25 27 7 8 9 11 14 17 9 13 15 9 10 11 15 17 19 21 11 12 20 21 14 18 25 12 13 15 17 20 21
13 17 19 20 21 22 15 17 17 19 20 21 22 24 19 19 20 21 22 24 28 22 24 28 24 5 6 10 10 12
14 16 18 18 21 23 21 22 23 28 23 24 28 28 25 26 18 27 27 22];
weights = [8 9 10 7 5 9 11 8 9 7 9 7 4 9 11 11 9 6 5 5 11 5 8 9 10 8 7 8 7 9 8 7 3 9 11 8 9 8
7 5 10 7 6 8 3 5 7 10 12 8 9 7 8 4 7 6 11 5 8 6 5 7 9 7 3 6 9 10 7 6 9 7 8 9
8 7 9 8 6 10 3 8 9 6 7 6 8 9 6 7 8 9 7 7 8 9 7 8 8 5 4 9 11 5 3 7 8 9];
G = graph(s,t,weights),
Plot (G,'edge Label', G.Edges. Weight);
x=[2 8.7 11 4.5 7.8 4.5 2 5 3.3 7.8 3.3 5.2 2 7.5 1 7.5 3.2 9 2.5 6.2 4.5 10.8
7.2 10 10 11.5 11.5 5];
y=[1 0.3 1 2 2 4 2.5 6 4.8 4 7 9 8.5 6 10 9 9.5 7.5 13 12 12.6 12.9
14 10 5 10 5 16];
p = plot (G,'XData',x,'YData',y,'edgeLabel',G.Edges.Weight);
d = distances (G);
Sources = [1 2 3];
Targets = [28];
d = distances (G,sources,targets)
[path1,d] = shortestpath(G,2,28)
[path2,d] = shortestpath(G,3,28)
[path3,d] = shortestpath(G,1,28)
highlight(p,path1,'EdgeColor','r','LineWidth',2)
highlight(p,path2,'EdgeColor','r','LineWidth',2)
highlight(p,path3,'EdgeColor','r','LineWidth',2)
highlight(p,1,'NodeColor','red')
highlight(p,2,'NodeColor','red')
highlight(p,3,'NodeColor','red')
highlight(p,28,'NodeColor','g')
p.MarkerSize = 6;
```



```

A = adjacency(G)
nn = numnodes(G);
[s,t] = findedge(G);
A = A + A' - diag(diag(A));
full(A)
D = diag(sum(A))
full(D)
L = diag(sum(A)) - A
full(L)
[V,D] = eig( full(L))
Example two
%Uses
% Dijkstra's algorithm
% [PATH,GOAL,GFOUND] = DIJKSTRA(V,E,START,GOALS) returns
s=[1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3 4 4 4 4 5 6 6 6 6 6 6 6 7 7 7 7 8 8 8 8 8 8 8 8 9 9 9 10
11 11 11 11 11 11 12 12 12 12 12 12 12 12 13 13 13 14 14 15 15 15 15 16 16 16 16 16 17 17 17 17
18 18 19 19 19 20 20 21 21 22 22 22 22 23 23 24 24 24 25 26 26 27];
t=[3 4 5 6 7 9 3 10 14 18 25 27 4 5 10 18 24 25 5 6 7 9 10 7 8 9 10 11 14 17 21 9 13 15 29 9
10 11 12 14 17 21 19 28 11 12 21 14 12 13 15 17 21 16 17 19 20 21 22 28 15 17 29 18 25 17
19 28 29 18 20 21 22 24 19 21 28 29 24 25 21 23 28 21 22 23 28 23 24 30 28 30 25 26 30 27
27 30 30];
weights = [9 5 11 2 6 8 8 8 9 11 3 8 5 9 4 9 12 8 6 7 6 9 8 3 9 7 8 9 2 10 12 9 9 8 9 9 3 7 9 6 5
6 9 12 5 7 10 9 6 4 7 6 8 7 7 9 6 7 12 3 4 2 1 8 3 9 8 9 6 9 7 6 9 7 9 3 6 8 4 2 7 11 8 8 9 7 3 8
5 2 8 9 11 5 5 7 8 9 12];
G = graph(s,t,weights),
x=[3 11 8.7 4.5 7.8 4.5 2 5 3.3 7.8 3.3 5.2 2 7.5 1 7.5 3.2 9 2.5 6.2 4.5 10.8
7.2 10 10 11.5 11.5 3.4 0.5 12];
y=[1 1 0.3 2 2 4 2.5 6 4.8 4 7 9 8.5 6 10 9 9.5 7.5 13 12 12.6 12.9
14 10 5 10 5 16 9 12.7];Plot (G,'edge Label', G.Edges. Weight);
x = [3 11 8.7 4.5 7.8 4.5 2.0 5.0 3.3 7.8 3.3 5.3 2.0 7.5 1 7.5 3.2 9.0 2.90 6.0 4.50
10.8 7.0 10 10 12 12 5 0.5 12];
y = [1 1 0.3 2.0 2.0 4.0 2.5 6.0 5.0 4.0 7.0 9.0 8.5 6.0 10 9.0 9.5 7.5 12.9 12.3 12.6
12.9 14.0 10 5 10 5 16 9 12.7];
p = plot (G,'XData',x,'YData',y,'edgeLabel',G.Edges.Weight);

```

```

d = distances (G);
Sources = [1 3 28];
Targets = [30 29 2];
d = distances (G,sources,targets)
[path1,d] = shortestpath(G,1,30)
[path2,d] = shortestpath(G,3,29)
[path3,d] = shortestpath(G,28,2)
highlight(p,path1,'EdgeColor','r','LineWidth',2)
highlight(p,path2,'EdgeColor','r','LineWidth',2)
highlight(p,path3,'EdgeColor','r','LineWidth',2)
highlight(p,1,'NodeColor','red')
highlight(p,3,'NodeColor','red')
highlight(p,28,'NodeColor','red')
highlight(p,30,'NodeColor','g')
highlight(p,29,'NodeColor','g')
highlight(p,2,'NodeColor','g')
p.MarkerSize = 6;
A = adjacency(G)
nn = numnodes(G);
[s,t] = findedge(G);
A = sparse(s,t,G.Edges.Weight,nn,nn)
A = A + A.' - diag(diag(A));
full(A)
D = diag(sum(A))
full(D)
L = diag(sum(A)) - A
full(L)
[V,D] = eig( full(L))

```

Example **three**

```
%Uses
```

```
% Dijkstra's algorithm
```

```
% [PATH,GOAL,GFOUND] = DIJKSTRA(V,E,START,GOALS) returns
```

```

s=[1 1 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 5 5 6 6 7 7 7 7 7 8 8 9 9 9 10 10 10 10 10 10 10 10 11 11
11 12 12 12 12 12 12 12 13 13 13 14 14 14 14 15 16 16 16 16 17 17 18 19 19 20 21 22 22 22
23 24 24 24 26 27 27 28 28 29 29 30 30 31];
t=[2 3 4 5 8 4 5 32 4 5 9 5 6 8 6 7 8 13 15 7 8 9 11 13 15 17 10 17 11 13 17 12 15 16 19 20
22 24 28 13 15 25 14 16 19 20 22 24 28 15 21 25 28 30 31 32 17 18 19 20 22 19 23 26 20 23
23 25 24 26 28 25 26 27 28 27 28 29 30 31 30 31 31 32 32];
weights = [9 4 5 5 9 5 8 8 7 8 5 6 4 8 7 6 7 10 9 11 10 7 8 8 7 12 8 2 6 11 9 5 5 4 6 9 8 5 5 8 5
10 7 8 9 10 7 6 8 7 5 6 9 12 7 12 9 9 6 3 6 3 11 5 6 4 8 9 4 6 6 8 6 7 3 8 8 5 7 3 9 10 6 8 4];
G = graph(s,t,weights),
Plot (G,'edge Label', G.Edges. Weight);
x=[1.00 3.429 0.50 3.429 2.004 4.958 2.004 5.0 0.50 7.00 1.50 7.000 4.5 5.52 4.5 10.0
6.80 12.0 9.500 11.65 6.401 10 6.80 10.6 6.401 12 12.0 10 12.00 10.00 6.20 5.501];
y=[3.51 0.000 6.00 2.018 6.000 2.018 6.562 6.5 6.562 6.5 10.5 5.951 10.0 5.951 9.5 6.5
9.500 6.50 11.44 11.29 10.00 6 12.00 4 12.00 6 3.5 3.5 2.962 2.962 1.38 0.000];
p = plot (G,'XData',x,'YData',y,'edgeLabel',G.Edges.Weight);
d = distances (G);
Sources = [24 19 31];
Targets = [11 1 20];
d = distances (G,sources,targets)
[path1,d] = shortestpath(G,24,11)
[path2,d] = shortestpath(G,19,1)
[path3,d] = shortestpath(G,31,20)
highlight(p,path1,'EdgeColor','r','LineWidth',2)
highlight(p,path2,'EdgeColor','r','LineWidth',2)
highlight(p,path3,'EdgeColor','r','LineWidth',2)
highlight(p,24,'NodeColor','red')
highlight(p,19,'NodeColor','red')
highlight(p,31,'NodeColor','red')
highlight(p,1,'NodeColor','g')
highlight(p,20,'NodeColor','g')
highlight(p,11,'NodeColor','g')
p.MarkerSize = 5;
A = adjacency(G)
nn = numnodes(G);

```

```

[s,t] = findedge(G);
A = sparse(s,t,G.Edges.Weight,nn,nn)
A = A + A.' - diag(diag(A));
full(A)
D = diag(sum(A))
full(D)
L = diag(sum(A)) - A
full(L)
[V,D] = eig( full(L))

```

#### Example Four

%Uses

% Dijkstra's algorithm

% [PATH,GOAL,GFOUND] = DIJKSTRA(V,E,START,GOALS) returns

```

s=[1 1 1 1 2 2 2 2 3 3 3 3 7 7 9 11 15 15 17 4 4 5 6 8 8 12 14 16 20 20 21 22 24
24 25 26 28 28 28 28 28 29 29 29 30 30 30 30 31 31 32 32 33 33 33 34 34 34 35 35 35 36
36 36 37 37 38 39 39 40]; t=[3 40 41 42 3 33 35 37 33 35 37 39 9 13 11 13 17 19 19 5 6
10 10 12 14 16 18 18 21 22 23 23 25 26 27 27 31 32 34 36 38 34 36 42 31 32 33 35 32 33
34 40 35 37 40 36 38 40 37 39 41 37 38 40 38 40 40 41 42 42];
weights = [9 3 6 4 3 6 4 1 11 6 2 1 8 9 5 4 9 8 9 6 7 6 8 9 6 7 8 9 8 9 7 8 11 5 7 8 9 8 4 9 8 4 3
11 5 9 4 4 8 6 6 10 4 1 11 3 2 11 3 6 9 8 1 5 2 2 5 5 9 7];
G = graph(s,t,weights),
x=[3.5 8.6 8.0 4.2 7.0 4.2 2.5 5.1 3.4 7.0 3.4 5.5 2.5 6.8 1.8 6.8 3.2 8.0 2.8 6.2
4.5 10.3 7.2 9.9 9.9 11 11 3.4 1.9 10.0 7.8 6.00 8.3 4.8 8.8 3.6 7.3 4.90 6.5 3.9
5.8 2.4];
y=[1 2.5 0.8 2.5 2.5 4.3 2.5 6.6 4.8 4.3 7 9 8.2 6.6 10.5 9 9.8 7.5 12.9 12
12.8 12.9 14 9.7 4 9.7 4 13 9.7 11.5 9.5 10.9 6.8 9.0 4.8 7.4 4.7 5.90 0.5 5.1
0.2 1.3];
p = plot (G,'XData',x,'YData',y,'edgeLabel',G.Edges.Weight);

d = distances(G);
sources = [1 28 3];
targets = [30 2 29];
d = distances(G,sources,targets)
[path1,d] = shortestpath(G,1,30)
[path2,d] = shortestpath(G,3,29)
[path3,d] = shortestpath(G,28,2)
highlight(p,path1,'EdgeColor','r','LineWidth',2)
highlight(p,path2,'EdgeColor','r','LineWidth',2)
highlight(p,path3,'EdgeColor','r','LineWidth',2)
highlight(p,1,'NodeColor','red')
highlight(p,28,'NodeColor','red')
highlight(p,3,'NodeColor','red')
highlight(p,2,'NodeColor','g')
highlight(p,29,'NodeColor','g')

```

```
highlight(p,30,'NodeColor','g')
```

```
p.MarkerSize = 6;
A = adjacency(G)
nn = numnodes(G);
[s,t] = findedge(G);
A = A - diag(diag(A));
full(A)
D = diag(sum(A))
full(D)
L = diag(sum(A)) - A
full(L)
[V,D] = eig( full(L))
```

### Example Five

```
%Uses
```

```
% Dijkstra's algorithm
```

```
% [PATH,GOAL,GFOUND] = DIJKSTRA(V,E,START,GOALS) returns
```

```
s=[1 1 1 1 2 2 2 2 2 3 3 3 3 7 7 9 11 15 15 17 4 4 5 6 8 8 12 14 16 20 20 21 22
24 24 25 26 28 28 28 28 28 29 29 29 30 30 30 31 31 31 32 33 33 33 34 34 34 35 35 36 36
36 37 37 37 38 39 39 40 40 40 41 42 44 44 45 47 47 48 49 51 51 52 53]; t=[3 40 41 42 3 30
31 33 37 30 31 37 39 9 13 11 13 17 19 19 5 6 10 10 12 14 16 18 18 21 22 23 23 25 26 27
27 30 32 34 36 38 34 36 42 31 32 34 32 33 37 34 37 38 40 36 38 40 36 42 38 40 43 38 39
40 40 41 42 41 42 43 42 43 45 46 46 48 49 50 50 52 53 54 54];
weights = [9 3 2 4 3 9 3 6 1 9 7 2 1 8 9 5 4 9 8 9 6 7 6 8 9 6 7 8 9 8 9 7 8 11
5 7 8 7 3 4 9 8 4 4 11 5 9 4 1 6 4 6 1 9 11 3 2 2 4 9 1 5 5 1 7 2 5 5 9 6
7 5 6 5 9 8 9 5 9 7 8 11 5 7 8];
G = graph(s,t,weights),
x=[3.5 8.6 8.0 4.2 7.0 4.2 2.6 5.1 3.4 7.0 3.4 5.5 2.7 6.8 2.1 6.8 3.0 8.0 2.7 6.2
5.10 9.6 7.2 9.9 9.9 11 11 3.66 2.3 9.6 8.3 6.8 6.8 5.4 2.4 3.6 7.3 4.85 7.3 3.9 3.9
2.4 3.6 1.8 3.2 2.8 6.2 4.5 10.3 7.2 9.7 9.7 11.3 11.3];
y=[1 2.5 0.8 2.5 2.5 4.3 2.5 6.6 4.8 4.3 7 9 8.2 6.6 10.9 9 10.6 7.5 12.5 12.55
12.8 12.9 13.7 9.7 4.0 9.7 4 12.35 9.87 11.2 7.5 9.5 5.96 9.4 9.0 7.4 4.7 5.98 1.9 4.7
1.9 1.3 4.7 10.7 10.2 13.2 12.2 12.8 12.9 14.0 10.0 3.6 10 3.6];
p = plot (G,'XData',x,'YData',y,'edgeLabel',G.Edges.Weight);

sources = [1 28 3];
targets = [30 2 29];
d = distances(G,sources,targets)
[path1,d] = shortestpath(G,1,30)
[path2,d] = shortestpath(G,3,29)
[path3,d] = shortestpath(G,28,2)
highlight(p,path1,'EdgeColor','r','LineWidth',2)
highlight(p,path2,'EdgeColor','r','LineWidth',2)
highlight(p,path3,'EdgeColor','r','LineWidth',2)
highlight(p,1,'NodeColor','red')
highlight(p,28,'NodeColor','red')
```

```
highlight(p,3,'NodeColor','red')
```

```
highlight(p,2,'NodeColor','g')
```

```
highlight(p,29,'NodeColor','g')
```

```
highlight(p,30,'NodeColor','g')
```

```
p.MarkerSize = 5;
```

```
A = adjacency(G)
```

```
nn = numnodes(G);
```

```
[s,t] = findedge(G);
```

```
A = A - diag(diag(A));
```

```
full(A)
```

```
D = diag(sum(A))
```

```
full(D)
```

```
L = diag(sum(A)) - A
```

```
full(L)
```

```
[V,D] = eig( full(L))
```

<https://github.com/kennethpu/iRoombot/blob/172347c1be1d87b6e4138d1dfc848abff899af38/dijkstra.m>

<https://uk.mathworks.com/help/matlab/ref/graph.shortestpath.html>

## 10 Appendix [C]

### 10.1 Program 2:

This Program is to calculate Path following for Differential Drive robot.

This example demonstrates how to control a robot to follow the desired path using a Robot Simulator. The example uses the Pure Pursuit path following controller to drive a simulated robot along a predetermined path. The desired path is a set of waypoints defined explicitly or computed

using a path planner (refer to `<docid:robotics_examples.examplePathPlanningExample>`).

The Pure Pursuit path following controller for a simulated differential drive robot is created and computes the control commands to follow a given path. The computed control commands are used to drive the simulated robot along the desired trajectory to follow the desired path based on the Pure Pursuit controller.

Note: Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `|y = step (obj,x)|` and `|y = obj(x)|` perform equivalent operations.

% Copyright 2014-2016 The MathWorks, Inc.

Example one

load `exampleMaps.mat`

`prmSimple = mobileRobotPRM(map,32);`

`show(prmSimple)`

`path1 = [2.50 3;5 6.5;7 9.5;8.80 12.25];`

`path2 = [5.81 12.35;4.5 10;2.0 6.0;0.7 3.6];`

`path3 = [10.5 2.50;10.0 3.0;7.0 6.50; 2.18 11.8];`

`show(prmSimple)`

`figure (1)`

`title(' Paths planned by proposed algorithm')`

`hold on`

`path = findpath(prm, path1(:,1),path1(:,2),'k--d',path2(:,1),path2(:,2),'k--d',path3(:,1),`

`path3(:,2),'k--d')`

`hold of`

`%% Create sensors`

`sensor1 = MultiRobotLidarSensor;`

```

sensor1.robotIdx = 1;
sensor1.sensorOffset = [0,0];
sensor1.scanAngles = linspace(-pi/2,pi/2,9);
sensor1.maxRange = 3;
attachLidarSensor(env,sensor1);

sensor2 = ObjectDetector;
sensor2.fieldOfView = pi/4;
attachObjectDetector(env,2,sensor2);
sensor3 = MultiRobotLidarSensor;
sensor3.robotIdx = 3;
sensor3.sensorOffset = [0,0];
sensor3.scanAngles = linspace(-pi/4,pi/4,10);
sensor3.maxRange = 10;
attachLidarSensor(env,sensor3);
%% Define waypoints, objects, and initial poses
waypoints = [5.00 6.50;2.0 6.00;7.0 6.50;
             10.00 3.0; 7.00 9.50;4.50 10.00];
objects = [8.80, 12.25, 1;
           0.7, 3.6, 2.00;
           2.18, 11.8, 3.00];
env.objectColors = [0 1 0;0 1 0;0 1 0];
env.objectMarkers = 'so^';
pose1 = [2.5;3;pi/2];
pose2 = [5.81;12.35;pi];
pose3 = [10.5;2.5;0];
env.Poses = [pose1 pose2 pose3];

```



```

%% Now loop through the animation
for idx = 1:100
    % Step the sensors
    ranges1 = sensor1();
    detections2 = sensor2(pose2,objects);
    ranges3 = sensor3();
    % Step the visualizer
    % In multiple commands
    % env(1,pose1, waypoints, ranges1, objects);
    % env(2,pose2, waypoints, [], objects);
    % env(3,pose3, waypoints, ranges3, objects);
    % In single command
    env([1,2,3],[pose1,pose2,pose3],waypoints,{ranges1,[],ranges3},objects)
    % Update poses
    pose1 = pose1 + [0.066;0.095;pi/25];
    pose2 = pose2 + [-0.054;-0.09;pi/20];
    pose3 = pose3 + [-0.085;0.095;pi/35];
    figure(2)
    title('Multi-Robot path planned by proposed algorithm')
end

```

### Example two

```

load exampleMaps.mat
map = binaryOccupancyMap(simpleMap,2);
prmSimple = mobileRobotPRM(map,67);
show(prmSimple)
path1 = [10.5 4.5;8.8 5.7;6.12 7.64;4.03 9.33;2.085 10.64];
path2 = [9.72 10.44;8.1 9.11;6.12 7.64;3.69 5.75; 0.81 3.51];
path3 = [7.6 2.38;8.8 5.7;9.9 7.8; 10.8 9.6;11.89 11.29];
show(prmSimple)
figure (1)
title(' VG and Paths planned by CA algorithm')

```

```

hold on
path=findpath(prm,path1(:,1),path1(:,2),'k--d',path2(:,1),path2(:,2),'k--d',path3(:,1),
path3(:,2),'k--d')
hold of

%% Path planner parameters
prmConnectionDistance = 3;
prmNumNodes = 200;
prmMaxWaypoints = 12;

%create a multi robot environment
numRobots = 3;
env = MultiRobotEnv(numRobots);
env.robotRadius = [0,0,0];
env.showTrajectory = [true;true;true];
env.hasWaypoints = true;
load exampleMap
env.mapName = 'map';

%% Create sensors
sensor1 = MultiRobotLidarSensor;
sensor1.robotIdx = 1;
sensor1.sensorOffset = [0,0];
sensor1.scanAngles = linspace(-pi/2,pi/2,9);
sensor1.maxRange = 3;
attachLidarSensor(env,sensor1);

sensor2 = ObjectDetector;
sensor2.fieldOfView = pi/4;
attachObjectDetector(env,2,sensor2);

sensor3 = MultiRobotLidarSensor;
sensor3.robotIdx = 3;
sensor3.sensorOffset = [0,0];
sensor3.scanAngles = linspace(-pi/4,pi/4,10);
sensor3.maxRange = 10;
attachLidarSensor(env,sensor3);

```

```

%% Define waypoints, objects, and initial poses
waypoints = [6.98 7.0;4.7 8.78;5.501 7.000;11.8 4.99;11 4.80;4.3 10.4;
            8.46 9.46;6.3 9;4 6;7.4 5; 9.62 6.6;10.2 7.5];
objects = [ 2.085, 10.64, 1;
           0.81, 3.51, 2.00;
           11.89, 11.29, 3.00];
env.objectColors = [0 1 0;0 1 0;0 1 0];
env.objectMarkers = 'so^';
pose1 = [10.5;4.5;pi/2];
pose2 = [9.72;10.44;-pi];
pose3 = [7.6;2.38;0];
env.Poses = [pose1 pose2 pose3];

%% Now loop through the animation
for idx = 1:100
    % Step the sensors
    ranges1 = sensor1();
    detections2 = sensor2(pose2,objects);
    ranges3 = sensor3();
    % Step the visualizer
    % In multiple commands
    % env(1,pose1, waypoints, ranges1, objects);
    % env(2,pose2, waypoints, [], objects);
    % env(3,pose3, waypoints, ranges3, objects);
    % In single command
    env([1,2,3],[pose1,pose2,pose3],waypoints,{ranges1,[],ranges3},objects)
    % Update poses
    pose1 = pose1 + [-0.085;0.062;pi/85];
    pose2 = pose2 + [-0.09;-0.07;pi/85];
    pose3 = pose3 + [0.045;0.09;pi/75];
    figure(2)
    title('Multi-Robot path planning by CA algorithm')
end

```

%% Multi-Robot Sensor Example

% Copyright 2018-2019 The MathWorks, Inc.

<https://uk.mathworks.com/matlabcentral/fileexchange/66586-mobile-robotics-simulation-toolbox>

<https://github.com/mathworks-robotics/mobile-robotics-simulation-toolbox>

[https://viewer.mathworks.com/?viewer=plain\\_code&url=https%3A%2F%2Fuk.mathworks.com%2Fmatlabcentral%2Fmlc-downloads%2Fdownloads%2F33cfee76-0bb0-42ce-a1bc-46cc156d43f7%2F701dee20-fdf5-4012-aea5](https://viewer.mathworks.com/?viewer=plain_code&url=https%3A%2F%2Fuk.mathworks.com%2Fmatlabcentral%2Fmlc-downloads%2Fdownloads%2F33cfee76-0bb0-42ce-a1bc-46cc156d43f7%2F701dee20-fdf5-4012-aea5)

[b7b75748cfe0%2Ffiles%2Fexamples%2Fmatlab%2Fmultirobot%2FmrsMultiRobotSensors.m&embed=web](https://viewer.mathworks.com/?viewer=plain_code&url=https%3A%2F%2Fuk.mathworks.com%2Fmatlabcentral%2Fmlc-downloads%2Fdownloads%2F33cfee76-0bb0-42ce-a1bc-46cc156d43f7%2F701dee20-fdf5-4012-aea5b7b75748cfe0%2Ffiles%2Fexamples%2Fmatlab%2Fmultirobot%2FmrsMultiRobotSensors.m&embed=web)

## 11 Appendix [D]

### 11.1 Program 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Robot_Environment4 Code %%%%
%%% Alex Littler - 20035222 %%%%
%%% Edited from code provided %%%%
%%% by Dr. Lyuba Alboul 2014 %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Positions of obstacles and goals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clf;

%Obstacle 1
K1=0.55;
Obs1=[2 6.5];
%Obstacle 2
K2=0.35,
Obs2=[2,5 12];
close all;
clf;

%Obstacle 3
K3=0.50;
Obs3=[4.5 2];
%Obstacle 4
K4=0.65;
Obs4=[8.1 9];
close all;
clf;

%Obstacle 5
K5=0.50;
Obs3=[10.2 13.9];
%Obstacle 6
K6=0.65;
Obs4=[10 10];
```

```

%Goal 1
Kg1=0.11;
Goal1=[11.8 3];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial positions of robots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Robot 1
Robot1=[3 2.1];
%Robot 2
Robot2=[4.5 13.5];
%Robot 3
Robot3=[10.2 2.3];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% The Robot's starting environment
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% example of an environment
clf % clear
figure1 = figure('Color', 'w');
axis ([0 16 0 16])
rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on
rectangle ('Position',[5,3,4,2],'Curvature', [0,0], 'LineWidth', 4), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on
rectangle ('Position',[13,5,2,5],'Curvature', [0,0], 'LineWidth', 4), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'),hold on
rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on
rectangle ('Position',[ 0.9,3.7,2,5],'Curvature', [0,0], 'LineWidth', 4), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'),hold on
rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on
rectangle ('Position',[ 7.2,12.7,5,2],'Curvature', [0,0], 'LineWidth', 4), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'),hold on
plot(3,2.1,'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
plot(4.5,15.5, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
plot(10.2,2.3, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r')
plot(11.8, 4,'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r')

```

```

circleBlue(Obs2(1),Obs2(2),2,4); %plot obstacle 2

title('Multi Robot Workspace Environment')

hold on

axis([0 16 0 16]);

grid on;

Example two
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Position of the robot, obstacles and goals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all;
clf;

%Obstacle 1

K1=0.55;

Obs1=[2 6.5 ];

%Obstacle 2

K2=0.35;

Obs2=[2.5 12];

close all;
clf;

%Obstacle 3

K3=0.5;

Obs3=[4.5 2];

%Obstacle 4

K4=0.65;

Obs4=[8.1 9];

%Obstacle 5

K5=0.5;

```

```

Obs5=[10.2 13.9];

clf;

%Obstacle 6

K6=0.65;

Obs6=[10 10];

%Goal 1

Goal1=[12.7 12];

%Goal 2

Goal2=[11.8 3 ];

%Goal 3

Goal3=[1.2 10];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial positions of robots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Robot1

Robot1=[3 2.1];

%Robot2

Robot2=[4.5 13.5];

%Robot3

Robot3=[10.2 2.3];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Original Robot Environment
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure1 = figure('Color', 'w') ;

axis([0 16 0 16]);

rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on

rectangle ('Position',[5,3,4,2],'Curvature', [0,0], 'LineWidth', 4), hold on

```



```

rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 2), hold on
rectangle ('Position',[13,5,2,5],'Curvature', [0,0], 'LineWidth', 4), hold on
rectangle ('Position',[0.9,3.7,2,5],'Curvature', [0,0], 'LineWidth', 4), hold on
rectangle ('Position',[7.2,12.7,5,2],'Curvature', [0,0], 'LineWidth', 4), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
%rectangle ('Position', [3, 2, 1,1],'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
plot(3,2.1,'ro', 'MarkerSize', 6,'MarkerFaceColor', 'r')
plot(4.5,13.5,'ro', 'MarkerSize', 6,'MarkerFaceColor', 'r')
plot(10.2,2.3,'ro','MarkerSize', 6,'MarkerFaceColor', 'r')
plot(12.7,12,'go','MarkerSize', 6,'MarkerFaceColor', 'g')
plot(11.8,4,'go','MarkerSize', 6,'MarkerFaceColor', 'g')
plot(1.2,10,'go','MarkerSize', 6,'MarkerFaceColor', 'g')
circleBlue(Obs2(1),Obs2(2),2,4); %plot obstacle 2
title('Multi Robot Workspace Environment')
hold on
axis([0 16 0 16]);
grid on;

```

### Example three

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Positions of obstacles and goals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clf;

%Obstacle 1

K1=0.55;

```

Obs1=[2 8.5];

%Obstacle 2

K2=0.35;

Obs2=[2.5 12];

%Obstacle 3

K3=0.5;

Obs3=[4.5 2];

%Obstacle 4

K4=0.65;

Obs4=[7.8 9];

%Obstacle 5

K5=0.65;

Obs5=[10.2 12.9];

%Obstacle 6

K6=0.65;

Obs6=[10 10];

%Goal 1

Kg1=0.11;

Goal1=[12 12.7];

%Goal 2

Kg2=0.11;

Goal2=[12 4];

%Goal 3

Kg3=0.11;

Goal3=[1.0 10];

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial positions of robots
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Robot 1

Robot1=[3 2.1];

%Robot 2

Robot2=[4.5 13.5];

%Robot 3

Robot3=[10.2 2.3];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Initial positions of waypoints
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%waypoint 1

waypoint1=[1 2];

%waypoint 2

waypoint2=[4.3 5.5];

%waypoint 3

waypoint3=[6 1.5];

%waypoint 4

waypoint4=[8 1.4];

%waypoint 5

waypoint5=[4.8 1];

%waypoint 6

waypoint6=[9.7 5.5];

%waypoint 7

waypoint7=[9.5 6.2];

%waypoint 8

```

```

waypoint8=[12.1 6];

%waypoint 9

waypoint9=[11.8 7.8];

%waypoint 10

waypoint10=[5 10];

%waypoint 11

waypoint11=[7 11.5];

%waypoint 12

waypoint12=[10 10];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Original Robot Environment
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure1 = figure('Color', 'w') ;

axis([0 16 0 16]);

rectangle('Position',[0,0,16,16],'Curvature', [0,0], 'LineWidth', 6), hold on
rectangle ('Position',[5,3,4,2],'Curvature', [0,0], 'LineWidth', 4), hold on
rectangle ('Position',[0,0,16,16], 'Curvature', [0,0], 'LineWidth', 2), hold on
rectangle ('Position',[13,5,2,5], 'Curvature', [0,0], 'LineWidth', 4), hold on
rectangle ('Position',[0.9,3.7,2,5], 'Curvature', [0,0], 'LineWidth', 4), hold on
rectangle ('Position',[7.2,12.7,5,2], 'Curvature', [0,0], 'LineWidth', 4), hold on
% rectangle ('Position', [3, 2, 1,1], 'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
% rectangle ('Position', [3, 2, 1,1], 'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
%rectangle ('Position', [3, 2, 1,1], 'Curvature', [1,1], 'LineWidth', 3, 'EdgeColor', 'b'), hold on
plot(3,2,'ro', 'MarkerSize', 6,'MarkerFaceColor', 'r')
plot(4.5,13, 'ro', 'MarkerSize', 6,'MarkerFaceColor', 'r')
plot(10.2, 2.3, 'ro', 'MarkerSize', 6,'MarkerFaceColor', 'r')

```

```

plot(15,12.7, 'go', 'MarkerSize', 6,'MarkerFaceColor', 'g')
plot(12,4, 'go', 'MarkerSize', 6,'MarkerFaceColor', 'g')
plot(1.0,10, 'go', 'MarkerSize', 6,'MarkerFaceColor', 'g')
plot(1,2, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(4.3,5.5, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(6,1.5, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(8,1.4, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(4,8.1, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(9.7,5.5, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(5.9,6.2, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(12.1,6, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(11.8,7.8, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(5,10, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(7,11.5, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
plot(10,10, 'bo', 'MarkerSize', 6,'MarkerFaceColor', 'b')
circleBlue(Obs2(1),Obs2(2),2,4); %obstacle 2
title('Multi Robot Workspace Environment')
hold on
axis([0 16 0 16]);
grid on;

```

## 12 Bibliography

1. Quottrup, M. M. (2007). Towards Coordination and Control of Multi-robot Systems. Department of Electronic Systems, Automation Control, Aalborg University.
2. Swain, D. (2012). The role of the dynamics of relative motion in information passing in natural and engineered collective motion (Doctoral dissertation, Princeton University).
3. Arai, T., Pagello, E., & Parker, L. E. (2002). Advances in multi-robot systems. *IEEE Transactions on robotics and automation*, 18(5), 655-661.
4. Falconi, R. (2009). Coordinated control of robotic swarms in unknown environments (Doctoral dissertation, Alma).
5. Montijano, E. (2012). Distributed Consensus in Multi-Robot Systems with Visual Perception.
6. Yan, Z., Jouandeau, N., & Cherif, A. A. (2013). A survey and analysis of multi-robot coordination. *International Journal of Advanced Robotic Systems*, 10
7. Sugihara, K., & Suzuki, I. (1990, September). Distributed motion coordination of multiple mobile robots. In *Proceedings. 5th IEEE International Symposium on Intelligent Control 1990* (pp. 138-143). IEEE.
8. Jones, C. V. (2005). A Principled Design Methodology for Minimalist Multi-Robot System Controllers. University of Southern California.
9. Veloso, M. M., & Nardi, D. (2006). Special issue on multirobot systems. *Proceedings of the IEEE*, 94(7), 1253-1253.
10. Kolushev, F. A., & Bogdanov, A. A. (1999, July). Multi-agent optimal path planning for mobile robots in environment with obstacles. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics* (pp. 503-510). Springer, Berlin, Heidelberg.
11. Han, Z. M., Lin, Z. Y., Fu, M. Y., & Chen, Z. Y. (2015). Distributed coordination in multi-agent systems: a graph Laplacian perspective. *Frontiers of Information Technology & Electronic Engineering*, 16(6), 429-448.
12. Weiss, G. (Ed.). (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.
13. Dudek, G., Jenkin, M. R., Milios, E., & Wilkes, D. (1996). A taxonomy for multi-agent robotics. *Autonomous Robots*, 3(4), 375-397.

14. Dudek, G., Jenkin, M., Milios, E., & Wilkes, D. (1993, July). A taxonomy for swarm robots. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)* (Vol. 1, pp. 441-447). IEEE.
15. Cao, Y. U., Fukunaga, A. S., Kahng, A. B., & Meng, F. (1995, August). Cooperative mobile robotics: Antecedents and directions. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots* (Vol. 1, pp. 226-234). IEEE.
16. Couceiro, M. S., Vargas, P. A., Rocha, R. P., & Ferreira, N. M. (2014). Benchmark of swarm robotics distributed techniques in a search task. *Robotics and Autonomous Systems*, 62(2), 200-213.
17. Schranz, M., Umlauf, M., Send, M., & Elmenreich, W. (2020). Swarm robotic behaviours and current applications. *Frontiers in Robotics and AI*, 7, 36.
18. Tan, Y., & Zheng, Z. Y. (2013). Research advance in swarm robotics. *Defence Technology*, 9(1), 18-39.
19. McLurkin, J. (2008). Analysis and implementation of distributed algorithms for Multi-Robot systems (Doctoral dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science).
20. Michael, N., Zavlanos, M. M., & Kumar, V. (2009). Maintaining connectivity in mobile robot networks,” *Experimental Robotics*.
21. Omar, R. B. (2012). Path planning for unmanned aerial vehicles using visibility line-based methods (Doctoral dissertation, University of Leicester).
22. Farinelli, A., Iocchi, L., & Nardi, D. (2004). Multirobot systems: a classification focused on coordination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5), 2015-2028.
23. Alatis, M. B., & Hancke, G. P. (2020). A review on challenges of autonomous mobile robot and sensor fusion methods. *IEEE Access*, 8, 39830-39846.
24. EL KHAILI, M. (2014). Visibility graph for path planning in the presence of moving obstacles. *system*, 4(4).
25. Ismail, Z. H., & Sariff, N. (2018). A survey and analysis of cooperative multi-agent robot systems: Challenges and directions. In *Applications of Mobile Robots* (pp. 8-14). Intech Open.
26. Biggs, N., Lloyd, E. K., & Wilson, R. J. (1986). *Graph Theory, 1736-1936*. Oxford University Press.

27. The Origins of Graph Theory.<https://jlmartin.ku.edu/~jlmartin/courses/math410-S09/graphs.pdf>
28. Rahman, M. S. (2017). Basic graph theory. Springer International Publishing.
29. Poghosyan, A. (2010). The probabilistic method for upper bounds in domination theory (Doctoral dissertation, University of the West of England, Bristol).
30. Samanta,P. (2011). Introduction to Graph Theory. Department of Mathematics. Berhampur University. Berhampur-760 007
31. Griffin, C. Graph Theory: Penn State Math 485 Lecture.
32. Rantanen, M. (2014). Improving probabilistic roadmap methods for fast motion planning.
33. Elbanhawi, M., Simic, M., & Jazar, R. (2013). Autonomous robots path planning: An adaptive roadmap approach. In *Applied Mechanics and Materials* (Vol. 373, pp. 246-254). Trans Tech Publications Ltd.
34. Maschian, E., & Sedighizadeh, D. (2007). Classic and heuristic approaches in robot motion planning-a chronological review. *World Academy of Science, Engineering and Technology*, 23(5), 101-106.
35. Giesbrecht, J. (2004). Global path planning for unmanned ground vehicles. DEFENCE RESEARCH AND DEVELOPMENT SUFFIELD (ALBERTA).
36. Choset, H. (1996). Sensor based motion planning: The hierarchical generalized Voronoi graph (Doctoral dissertation, California Institute of Technology).
37. Canny, J. (1985, March). A Voronoi method for the piano-movers problem. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation* (Vol. 2, pp. 530-535). IEEE.
38. Takahashi, O., & Schilling, R. J. (1989). Motion planning in a plane using generalized Voronoi diagrams. *IEEE Transactions on robotics and automation*, 5(2), 143-150.
39. Lin, C. C., Chuang, W. J., & Liao, Y. D. (2012, August). Path planning based on Bezier curve for robot swarms. In *2012 Sixth International Conference on Genetic and Evolutionary Computing* (pp. 253-256). IEEE.
40. Asano, T., Asano, T., Guibas, L., Hershberger, J., & Imai, H. (1985, October). Visibility-polygon search and Euclidean shortest paths. In *26th annual symposium on foundations of computer science (SFCS 1985)* (pp. 155-164). IEEE.
41. Alexopoulos, C., & Griffin, P. M. (1992). Path planning for a mobile robot. *IEEE Transactions on systems, man, and cybernetics*, 22(2), 318-322.



42. Maekawa, T., Noda, T., Tamura, S., Ozaki, T., & Machida, K. I. (2010). Curvature continuous path generation for autonomous vehicle using B-spline curves. *Computer-Aided Design*, 42(4), 350-359.
43. Iocchi, L., Nardi, D., & Salerno, M. (2000, August). Reactivity and deliberation: a survey on multi-robot systems. In *Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems* (pp. 9-32). Springer, Berlin, Heidelberg.
44. Parker, L. E. (2008). *Handbook of Robotics Chapter 40: Multiple Mobile Robot Systems*.
45. Ren, W., & Atkins, E. (2007). Distributed multi-vehicle coordinated control via local information exchange. *International Journal of Robust and Nonlinear Control*, 17(10-11), 1002-1033.
46. Desai, J. P. (2002). A graph theoretic approach for modelling mobile robot team formations. *Journal of Robotic Systems*, 19(11), 511-525.
47. Coogan, S. (2010). Size switching in multi-agent formation control.
48. Parra-Gonzalez, E. F., & Ramírez-Torres, J. G. (2011). Object path planner for the box pushing problem. *Multi-Robot Systems, Trends and Development*, InTech Pub, 291-306.
49. Rahmani, A., Ji, M., Mesbahi, M., & Egerstedt, M. (2009). Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization*, 48(1), 162-186.
50. Ji, M., & Egerstedt, M. (2007, July). A graph-theoretic characterization of controllability for multi-agent systems. In *2007 American Control Conference* (pp. 4588-4593). IEEE.
51. Lemay, M., Michaud, F., Létourneau, D., & Valin, J. M. (2004, April). Autonomous initialization of robot formations. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 (Vol. 3, pp. 3018-3023)*. IEEE.
52. Ng, J. S. (2010). *An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments*,
53. Dong, W. J., Guo, Y., & Farrell, J. A. (2006, June). Formation control of nonholonomic mobile robots. In *2006 American Control Conference* (pp. 6-pp). IEEE.
54. Feddema, J. T., Parker, E. P., Wagner, J. S., & Schoenwald, D. A. (2004). Analysis and control of distributed cooperative systems (No. SAND2004-4763). Sandia National Laboratories.

55. Nebot, P., & Cervera, E. (2006). Acromovi architecture: A framework for the development of multirobot applications. *Mobile robotics, moving intelligence* () IntechOpen.
56. Verma, J. K., & Ranga, V. (2021). Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope. *Journal of Intelligent & Robotic Systems*, 102(1), 1-36.
57. Sarvepalli, S. S. K. (2015). Multi Robot System: The future of Scientific Learning.
58. Rocha, R., Dias, J., & Carvalho, A. (2005). Cooperative multi-robot systems: A study of vision-based 3-d mapping using information theory. *Robotics and Autonomous Systems*, 53(3-4), 282-311.
59. Fax, J. A., & Murray, R. M. (2004). Information flow and cooperative control of vehicle formations. *Automatic Control, IEEE Transactions on*, 49(9), 1465-1476.
60. Czarnecki, C. A. (1994). Multi-Robot Systems: A Perspective. *IFAC Proceedings Volumes*, 27(4), 201-205.
61. Cao, Y. U., Fukunaga, A. S., & Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1), 7-27.
62. Wang, Z., Tianfield, H., & Jiang, P. (2003, August). A framework for coordination in multi-robot systems. In *IEEE International Conference on Industrial Informatics, 2003. INDIN 2003. Proceedings.* (pp. 483-489). IEEE.
63. Tuci, E., & Trianni, V. (2014). On the evolution of homogeneous two-robot teams: clonal versus aclonal approaches. *Neural Computing and Applications*, 25(5), 1063-1076.
64. Pitla, S. K., Luck, J. D., & Shearer, S. A. (2010). Multi-robot system control architecture (MRSCA) for agricultural production. In *2010 Pittsburgh, Pennsylvania, June 20-June 23, 2010* (p. 1). American Society of Agricultural and Biological Engineers.
65. Parker, L. E. (2009). Path planning and motion coordination in multiple mobile robot teams. *Encyclopedia of complexity and system science*, 5783-5800.
66. Wang, X., Li, X., Cong, Y., Zeng, Z., & Zheng, Z. (2015). Multi-agent distributed coordination control: Developments and directions. *ArXiv Preprint arXiv:1505.02595*,
67. Ye, W., Vaughan, R. T., Sukhatme, G. S., Heidemann, J., Estrin, D., & Mataric, M. J. (2001). Evaluating control strategies for wireless-networked robots using an integrated robot and network simulation. Paper presented at the *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, , 3 2941-2947.

68. Huang, J., Fang, H., Dou, L., & Chen, J. (2014). An overview of distributed high-order multi-agent coordination. *IEEE/CAA Journal of Automatica Sinica*, 1(1), 1-9.
69. Ren, W., Beard, R. W., & Atkins, E. M. (2007). Information consensus in multivehicle cooperative control. *IEEE Control systems magazine*, 27(2), 71-82.
70. Lecture 10: Dijkstra's Shortest Path Algorithm CLRS 24.3. [https://home.cse.ust.hk › ~dekai › notes](https://home.cse.ust.hk/~dekai/notes).
71. De Abreu, N. M. M. (2007). Old and new results on algebraic connectivity of graphs. *Linear algebra and its applications*, 423(1), 53-73.
72. Kleiner, A., Nebel, B. Search Algorithms and Pathfinding Robot Motion Planning & Multi-Robot Planning. Introduction to Multi-Agent Programming. [gki.informatik.uni-freiburg.de › teaching › imap › 04\\_SearchAlgorithmsA](http://gki.informatik.uni-freiburg.de/teaching/imap/04_SearchAlgorithmsA).
73. Saez-Pons, J., Alboul, L., Penders, J., & Nomdedeu, L. (2010). Multi-robot team formation control in the GUARDIANS project. *Industrial Robot: An International Journal*, 37(4), 372-383.
74. Chen, Y. Q., & Wang, Z. (2005, August). Formation control: a review and a new consideration. In 2005 IEEE/RSJ International conference on intelligent robots and systems (pp. 3181-3186). IEEE.
75. Daigle, M. J., Koutsoukos, X. D., & Biswas, G. (2007). Distributed diagnosis in formations of mobile robots. *IEEE Transactions on Robotics*, 23(2), 353-369.
76. Barca, J. C., Sekercioglu, A., & Ford, A. (2013). Controlling formations of robots with graph theory. *Intelligent autonomous systems 12* (pp. 563-574) Springer.
77. Vehicle Formations. *IEEE Transactions on Automatic Control* pages 1465–1476.
78. Falconi, R., Sabattini, L., Secchi, C., Fantuzzi, C., & Melchiorri, C. (2015). Edge-weighted consensus-based formation control strategy with collision avoidance. *Robotica*, 33(2), 332-347.
79. Kanjanawanishkul, K. (2010). Coordinated path following control and formation control of mobile robots. Phd, Eberhard-Karls-Universität Tübingen,
80. Stare, O. (2012). Centralized multi-robot system. The Department of Software Engineering. Faculty of Mathematics and Physics. Charles University in Prague. Ph.D. Thesis.
81. Parker, L. E. (2000). Current state of the art in distributed autonomous mobile robotics. In *Distributed Autonomous Robotic Systems 4* (pp. 3-12). Springer, Tokyo.
82. Godsil, C., & Royle, G. F. (2001). Algebraic graph theory (Vol. 207). Graduate Texts in Mathematics.

83. He, Y., & Han, J. (2009). Multiple robots formation control based on receding horizon optimization. Paper presented at the Information and Automation, 2009. ICIA'09. International Conference on, 734-739.
84. Ji, M., & Egerstedt, M. (2007). Distributed coordination control of multiagent systems while preserving connectedness. *IEEE Transactions on Robotics*, 23(4), 693-703.
85. Sarkar, S., & Kar, I. N. (2015, July). Three-time scale behaviour analysis of the Leader Follower formation of multiple groups of nonholonomic robots. In 2015 American Control Conference (ACC) (pp. 44-49). IEEE.
86. Desai, J. P., Ostrowski, J. P., & Kumar, V. (2001). Modelling and control of formations of nonholonomic mobile robots.
87. Ren, W., & Atkins, E. (2007). Distributed multi-vehicle coordinated control via local information exchange. *International Journal of Robust and Nonlinear Control*, 17(10-11), 1002-1033.
88. Verret, S. (2005). Current State of the Art in Multi-robot Systems. Technical Memorandum. DRDC Suffield TM 241.
89. Zavlanos, M. M., Egerstedt, M. B., & Pappas, G. J. (2011). Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9), 1525-1540.
90. Zavlanos, M. M., & Pappas, G. J. (2013). Connectivity of dynamic graphs. *Encyclopedia of Systems and Control*, 1-8.
91. Bullo, F., Cortes, J., & Martinez, S. (2009). Distributed control of robotic networks: a mathematical approach to motion coordination algorithms (Vol. 27). Princeton University Press.
92. Schneider, F. (2013). Formation Navigation and Relative Localisation of Multi-Robot Systems.
93. Baca, J., Yerpes, A., Ferre, M., Escalera, J. A., & Aracil, R. (2008, September). Modelling of modular robot configurations using graph theory. In *International Workshop on Hybrid Artificial Intelligence Systems* (pp. 649-656). Springer, Berlin, Heidelberg.
94. Bullo, F., & Smith, S. L. (2016). Lectures on robotic planning and kinematics.version v0.91(e).
95. Solovey, K., Salzman, O., & Halperin, D. (2015). Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning. *Algorithmic foundations of robotics XI* (pp. 591-607) Springer.

96. Clark, C. M. (2005). Probabilistic road map sampling strategies for multi-robot motion planning. *Robotics and autonomous systems*, 53(3-4), 244-264.
97. Muntean, P. (2016). Mobile robot navigation on partially known maps using a fast a star algorithm version. *arXiv preprint arXiv:1604.08708*.
98. Yoshida, E., Yokoi, K., & Gergondet, P. (2010, October). Online replanning for reactive robot motion: Practical aspects. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 5927-5933). IEEE.
99. Naumov.V, Karova.M, Zhelyazkov.D, Todorova.M, Penev.I, Nikolov.V & Petkov.V.(2015). Robot Path Planning Algorithm.*International Journal of Computers and Communications*.Vol. 9.
100. Cai, C., Yang, C., Zhu, Q., & Liang, Y. (2007, August). Collision avoidance in multi-robot systems. In *2007 International Conference on Mechatronics and Automation* (pp. 2795-2800). IEEE.
101. Rashid, A. T., Ali, A. A., Frasca, M., & Fortuna, L. (2013). Path planning and obstacle avoidance based on shortest distance algorithm. *Robotics and Autonomous Systems*, 61(12), 1440-1449.
102. Sabattini, L. (2012). Nonlinear control strategies for cooperative control of multi-robot systems.
103. Cornejo Collado, A. (2012). Local distributed algorithms for multi-robot systems (Doctoral dissertation, Massachusetts Institute of Technology).
104. Gu, D. (2008). A differential game approach to formation control. *IEEE Transactions on Control Systems Technology*, 16(1), 85-93.
105. Toshiyuki Yasuda. *Multi-Robot Systems, Trends, and Development* [online]. 2011 <http://www.intechopen.com/books/multi-robot-systems-trends-and-development>
106. Burgard, W., Moors, M., Stachniss, C., & Schneider, F. E. (2005). Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3), 376-386.
107. Smith, Brian Stephen. (2009). "Automatic coordination and deployment of multi-robot systems." PhD diss., Georgia Institute of Technology.
108. Lin, Z., Broucke, M., & Francis, B. (2004). Local control strategies for groups of mobile autonomous agents. *IEEE Transactions on automatic control*, 49(4), 622-629.
109. Ren, W., & Beard, R. W. (2004, June). Consensus of information under dynamically changing interaction topologies. In *Proceedings of the 2004 American control conference* (Vol. 6, pp. 4939-4944). IEEE.

110. Fax, J. A., & Murray, R. M. (2002). Graph laplacians and stabilization of vehicle formations. *IFAC Proceedings Volumes*, 35(1), 55-60.
111. Jadbabaie, A., Lin, J., & Morse, A. S. (2003). Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on automatic control*, 48(6), 988-1001.
112. Reynolds, C. W. (1987, August). Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (pp. 25-34).
113. Falconi, R., Sabattini, L., Secchi, C., Fantuzzi, C., & Melchiorri, C. (2011). A graph-based collision-free distributed formation control strategy. *IFAC Proceedings Volumes*, 44(1), 6011-6016. (3), 376-386.
114. Zavlanos, M. M., & Pappas, G. J. (2007). Potential fields for maintaining connectivity of mobile networks. *IEEE Transactions on robotics*, 23(4), 812-816.
115. Kempton, L. (2018). *Distributed Control and Optimisation of Complex Networks via their Laplacian Spectra* (Doctoral dissertation, University of Bristol).
116. Griparić, K. (2021). Algebraic Connectivity Control in Distributed Networks by Using Multiple Communication Channels. *Sensors*, 21(15), 5014.
117. Sariff, N. B., & Buniyamin, N. (2010, October). Ant colony system for robot path planning in global static environment. In *9th WSEAS International Conference on System Science and Simulation in Engineering (ICOSSSE'10)* (pp. 192-197).
118. Omar, N. (2013). *Path Planning Algorithm for a Car-like Robot Based on Cell Decomposition Method* (Doctoral dissertation, Universiti Tun Hussein Onn Malaysia).
119. Khaluf, Y., Markarian, C., Simoens, P., & Reina, A. (2017, June). Scheduling access to shared space in multi-robot systems. In *International Conference on Practical Applications of Agents and Multi-Agent Systems* (pp. 144-156). Springer, Cham.
120. Reina, A., Gambardella, L. M., Dorigo, M., & Di Caro, G. A. (2014). zePPeLIN: Distributed path planning using an overhead camera network. *International Journal of Advanced Robotic Systems*, 11(8), 119.
121. Jamin, N. F. (2014). *Probabilistic roadmap-based path planning for mobile robots* (Doctoral dissertation, Universiti Tun Hussein Onn Malaysia).
122. Charalampous, K., Amanatiadis, A., & Gasteratos, A. (2012). Efficient robot path planning in the presence of dynamically expanding obstacles. In *Cellular Automata: 10th International Conference on Cellular Automata for Research and*

- Industry, ACRI 2012, Santorini Island, Greece, September 24-27, 2012. Proceedings 10 (pp. 330-339). Springer Berlin Heidelberg.
123. Charalampous, K., Kostavelis, I., & Gasteratos, A. (2015). Thorough robot navigation based on SVM local planning. *Robotics and Autonomous Systems*, 70, 166-180.
  124. Kostavelis, I., Nalpantidis, L., & Gasteratos, A. (2012). Collision risk assessment for autonomous robots by offline traversability learning. *Robotics and Autonomous Systems*, 60(11), 1367-1376.
  125. Kostavelis, I., Gasteratos, A., Boukas, E., & Nalpantidis, L. (2012, November). Learning the terrain and planning a collision-free trajectory for indoor post-disaster environments. In *2012 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* (pp. 1-6). IEEE.
  126. Charalampous, K., Kostavelis, I., Boukas, E., Amanatiadis, A., Nalpantidis, L., Emmanouilidis, C., & Gasteratos, A. (2015). Autonomous robot path planning techniques using cellular automata. *Robots and lattice automata*, 175-196.
  127. Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10), 560-570.
  128. Wooden, D. T. (2006). Graph-based path planning for mobile robots (doctoral dissertation, georgia institute of technology).
  129. Siméon, T, Laumond, J., & Nissoux, C. (2000). Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 477-493.
  130. Lingelbach, F. (2005). Path Planning using Probabilistic Cell Decomposition,
  131. Wager, M. L. (2000). Making roadmaps using voronoi diagrams. UWA Honours Thesis,
  132. Gonzalez-Garcia, H., & Ayala-Ramirez, V. ((2011, ). Two didactic tools for robot motion planning using visibility graphs.
  133. Abdulakareem, M. I., & Raheem, F. A. (2020). Development of path planning algorithm using probabilistic roadmap based on ant colony optimization. *Engineering and Technology Journal*, 38(3A), 343-351.
  134. Froese, V., & Renken, M. (2019). Advancing Through Terrains. *arXiv preprint arXiv:1904.08746*.
  135. Masehian, E., & Nejad, A. H. (2011). Graph-based Multi Robot Motion Planning: Feasibility and Structural Properties. In *Multi-Robot Systems, Trends and Development*. IntechOpen.

136. Froese, V., & Renken, M. (2021). A fast shortest path algorithm on terrain-like graphs. *Discrete & Computational Geometry*, 66(2), 737-750.
137. Hershberger, J. (1989). An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4(1-4), 141-155.
138. Thompson, A. M. (1977). The navigation system of the JPL robot.
139. Nilsson, N. (1969). A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of International Joint Conference on Artificial Intelligence*, (pp. 509-520).
140. Tokuta, A. (1998). Extending the VGRAPH algorithm for robot path planning.
141. Alija, A. S. (2015). Analysis of Dijkstra's and A\* Algorithm to Find the Shortest Path.
142. Flitter, H., & Grossmann, T. (2016, April). Accessibility (Network Analysis). Geographic Information Technology Training Alliance (GITTA). <http://www.gitta.info> - Version from: 28.4.2016.
143. Jaafar, H., Zabidi, M. H., Soh, A. C., Hoong, T. P., Shafie, S., & Ahmad, S. A. (2014). Intelligent Guidance Parking System Using Modified Dijkstra's Algorithm. *Journal of Engineering Science and Technology*, 9, 132-141.
144. Gupta, N., Mangla, K., Jha, A. K., & Umar, M. (2016). Applying Dijkstra's algorithm in routing process. *Int. J. New Technol. Res*, 2(5), 122-124
145. Reddy, H. (2013). Pathfinding—Dijkstra's and A\* Algorithm's. *International Journal in IT and Engineering*, 1-15.
146. Lecture 18 Solving Shortest Path Problem: Dijkstra's Algorithm. (2009, October). [www.ifp.illinois.edu/~angelia/ge330fall09\\_dijkstra\\_118](http://www.ifp.illinois.edu/~angelia/ge330fall09_dijkstra_118).
147. Van Den Berg, Jur P, & Overmars, M. H. (2005). Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5), 885-897.
148. Goyal, A., Mogha, P., Luthra, R., & Sangwan, N. (2014). Path finding: A\* or dijkstra's?. *International Journal in IT & Engineering*, 2(1), 1-15.
149. Bhagat, V., Agarwal, S., Pollock, J., & Austin, S. [A\\* Search and Dijkstra's Algorithm: A Comparative Analysis.https://cse442-17f.github.io/A-Star-Search-and-Dijkstras..](https://cse442-17f.github.io/A-Star-Search-and-Dijkstras..)
150. Solis, I., Motes, J., Sandström, R., & Amato, N. M. (2021). Representation-optimal multi-robot motion planning using conflict-based search. *IEEE Robotics and Automation Letters*, 6(3), 4608-4615. LaValle,
151. S. M. (2006). *Planning algorithms*. Cambridge university press.



152. Geraerts, R., & Overmars, M. H. (2004). A comparative study of probabilistic roadmap planners. *Algorithmic foundations of robotics V* (pp. 43-57) Springer.
153. Petrović, L. (2018). Motion planning in high-dimensional spaces. arXiv preprint arXiv:1806.07457.
154. González, D., Pérez, J., Milanés, V., & Nashashibi, F. (2015). A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17(4), 1135-1145.
155. Yan, M. (2014). Dijkstra's algorithm. Massachusetts Institute of Technology.Regexstr,
156. Hvězda, B. J. (2017). Comparison of path planning methods for a multi-robot team.
157. Toan, T. Q., Sorokin, A. A., & Trang, V. T. H. (2017, June). Using modification of visibility-graph in solving the problem of finding shortest path for robot. In 2017 International Siberian Conference on Control and Communications (SIBCON) (pp. 1-6). IEEE.
158. Tokuta, A.O. (2001). Extending the VGRAPH Algorithm for Robot Path Planning.
159. Burgard, W., Stachniss, C., Bennewitz, M., & Arras, K. (2011). Introduction to mobile robotics. University of Freiburg,
160. Roadmap Methods vs. Cell Decomposition in Robot Motion Planning.
161. Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., & Burgard, W. (2005). Principles of robot motion: theory, algorithms, and implementations. MIT press.
162. Choset, H., Lynch, K., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L., & Thrun, S. (2007). Principles of Robot Motion: Theory, Algorithms, and Implementation ERRATA!
163. Capelli, B., Fouad, H., Beltrame, G., & Sabattini, L. (2021). Decentralized Connectivity Maintenance with Time Delays using Control Barrier Functions. arXiv preprint arXiv:2103.12614.
164. Capelli, B., & Sabattini, L. (2020, May). Connectivity maintenance: Global and optimized approach through control barrier functions. In 2020 IEEE International Conference on Robotics and Automation (ICRA) (pp. 5590-5596). IEEE. //
165. De Berg, M., Van Kreveld, M., Overmars, M., & Schwarzkopf, O. C. (2000). Visibility graphs. *Computational geometry* (pp. 307-317) Springer.

166. Kanda, T., Shiomi, M., Miyashita, Z., Ishiguro, H., & Hagita, N. (2009, March). An affective guide robot in a shopping mall. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction* (pp. 173-180).
167. Kanda, T., Shiomi, M., Miyashita, Z., Ishiguro, H., & Hagita, N. (2010). A communication robot in a shopping mall. *IEEE Transactions on Robotics*, 26(5), 897-913.
168. Shiomi, M., Kanda, T., Glas, D. F., Satake, S., Ishiguro, H., & Hagita, N. (2011). A Network Robot System for Cooperative Guide Service in a shopping mall. *J. Robot. Soc. Jpn*, 29, 544-553.
169. Hameed, I. A. (2018, August). A coverage planner for multi-robot systems in agriculture. In *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)* (pp. 698-704). IEEE.
170. Tsang, K. F. E., Ni, Y., Wong, C. F. R., & Shi, L. (2018, November). A novel warehouse multi-robot automation system with semi-complete and computationally efficient path planning and adaptive genetic task allocation algorithms. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)* (pp. 1671-1676). IEEE.
171. Chen, B., Marvin, S., & While, A. (2020). Containing COVID-19 in China: AI and the robotic restructuring of future cities. *Dialogues in Human Geography*, 2043820620934267.
172. Colley, P., Lubiw, A., & Spinrad, J. (1997). Visibility graphs of towers. *Computational Geometry*, 7(3), 161-172.
173. Liang, Y., & Xu, L. (2009, June). Global path planning for mobile robot based genetic algorithm and modified simulated annealing algorithm. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation* (pp. 303-308). ACM.
174. Hogben, L. (2005). Spectral graph theory and the inverse eigenvalue problem of a graph. *Electronic Journal of Linear Algebra*, 14(1), 3.
175. Clemens, H & Stephan, W. (2009). Graph theory.
176. Fallat, S. M., & Hogben, L. (2007). The minimum rank of symmetric matrices described by a graph: A survey. *Linear Algebra and its Applications*, 426(2-3), 558-582.
177. Ogunsanya, A., & Ade, B. (1986). Graph theory in intra-urban traffic flow estimation. *Geojournal*, 12(3), 334-336

178. Giordano, P. R. (2015). Analysis and control of multi-robot systems. Elective in Robotics. Lecture: Formation Control of Multiple Robots.
179. Weisstein, E. W. (2014). Rigid Graph. <https://mathworld.wolfram.com/>.
180. Roth, B. (1981). Rigid and flexible frameworks. *The American Mathematical Monthly*, 88(1), 6-21.
181. Jackson, B. (2007, October). Notes on the rigidity of graphs. In *Levico Conference Notes (Vol. 4)*.
182. Shekhar, S. (2011-2012). Graph Theory: Penn State Math 485 Lecture Notes Version 1.4.2.1 Christopher Grin. Licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License. Thesis.
183. Sun, Z., Yu, C., & Anderson, B. (2013). Distributed estimation and control for preserving formation rigidity for mobile robot teams. *ArXiv Preprint arXiv:1309.4850*,
184. Godsil, C., & Royle, G. F. (2013). *Algebraic graph theory (Vol. 207)*. Springer Science & Business Media.
185. Kuijsters, W. (2015). Distributed connectivity-aware multirobot exploration.
186. Chung, F. R., & Oden, K. (2000). Weighted graph Laplacians and isoperimetric inequalities. *Pacific Journal of Mathematics*, 192(2), 257-273.
187. Han, Z., Lin, Z., Fu, M., & Chen, Z. (2015). Distributed coordination in multi-agent systems: A graph laplacian perspective. *Frontiers of Information Technology & Electronic Engineering*, 16(6), 429-448.
188. Bóta, A. (2015). *Methods for the Description and Analysis of Processes in Real-Life Networks*,
189. LEE, Geunho and CHONG, Nak Young (2008). A geometric approach to deploying robot swarms. *Annals of mathematics and artificial intelligence*, 52 (2-4), 257-280.
190. Michael, N., Zavlanos, M. M., & Kumar, V. (2009). Maintaining connectivity in mobile robot networks,” *experimental robotics*.
191. Herley, K. T. (2016, October). Lecture I: Shortest Path Algorithms. Department of Computer Science University College Cork.
192. Lecture 9: Dijkstra's Shortest Path Algorithm CLRS 24.3. [home.cse.ust.hk > faculty > golin > Notes > MyL09](http://home.cse.ust.hk/~faculty/golin/Notes/MyL09).
193. Fadzli, S. A., Abdulkadir, S. I., Makhtar, M., & Jamal, A. A. (2015, December). Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries.

- In 2015 2nd International Conference on Information Science and Security (ICISS) (pp. 1-4). IEEE.
194. Sharma, Y., Saini, S. C., & Bhandhari, M. (2012). Comparison of Dijkstra's shortest path algorithm with genetic algorithm for static and dynamic routing network. *International Journal of Electronics and Computer Science Engineering*, 1(2), 416-425.
  195. Park, J. (2015). Shortest path algorithms.
  196. BENAICHA, R., & TAIBI, M. (2013). Dijkstra algorithm implementation on fpga card for telecom calculations.
  197. Puthuparampil, M. (2007). Report Dijkstra's Algorithm. Unpublished Presentation, Computer Science Department, New York University. Available from: <http://www.Cs.nyu.Edu/courses/summer07> G, 22, 2340-001.
  198. Liang, Y., & Xu, L. (2009, June). Global path planning for mobile robot based genetic algorithm and modified simulated annealing algorithm. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation* (pp. 303-308). ACM.
  199. Greef, L. (2017). CSE 373: Data Structures and Algorithms. Lecture 16: Dijkstra's Algorithm (Graphs). <https://courses.cs.washington.edu> › lectures.
  200. De Berg, M., Van Kreveld, M., Overmars, M., & Schwarzkopf, O. C. (2000). Visibility graphs. *Computational geometry* (pp. 307-317) Springer.
  201. Goldberg, A. W., & Valley, S. (2000). Basic shortest path algorithms. DIKU Summer School on Shortest Paths. Microsoft Research. Silicon Valey.
  202. Madkour, A., Aref, W. G., Rehman, F. U., Rahman, M. A., & Basalamah, S. (2017). A survey of shortest-path algorithms. ArXiv preprint arXiv: 1705.02044.