Deformable Voronoi Diagrams for Robot Path Planning in Dynamic Environments

Tajudeen Adeleke Badmos

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

April 2021

Declaration

I hereby declare that:

1. I have not been enrolled for another award of the University, or other academic or professional organization, 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 26,858

Name	Deformable Voronoi Diagrams for Robot Path Planning in Dynamic Environments
Date	April 2021
Award	PhD
Faculty	Industry and Innovation Research Institute, MERI.
Director of Studies	Dr Lyuba Alboul

Acknowledgement

My gratitude first goes to my creator almighty Allah for seeing me through this research work leading to this thesis. Utmost thanks to my supervisory team led by my Director of Studies Lyuba Alboul PhD, the memories of late Prof Jacques Penders who passed away after my viva, and David O'Sullivan, but specifically Dr Lyuba Alboul who has been a motivating mentor and I offer her my deepest thanks. She provides an excellent environment and supports for me to accomplish my desires. She has made a lasting influence on me, particularly in the field of robotics. Our face-to-face discussions and other discussions such as Skype, phone calls, and WhatsApp chat with her have certainly helped shape this thesis and contributed enormously to my understanding of the course of study. I also appreciate all my fellow students in the MERI lab especially those of you in robotics.

I am also grateful to the previous and present rectors of Federal polytechnic, Nasarawa Prof. Shetima AbdulKadir and Alhassan Ahmed, PhD, for their support and agree with me to go for this study abroad.

I will be ungrateful if I did not express my sincere gratitude to the man that God used to save me from dropping out of this programme when the Nigerian government failed to pay my tuition fees after sending me to the UK. Thank you so much Alh. Engr. Raufu Olaniyan, the Deputy Governor of Oyo State, Nigeria. May Almighty Allah be pleased with every step you take.

Special thanks to my friends, Benedict Ayomanor PhD, AbdulWahab Muyideen, Prince Olaide Mohammed (my Boss), Mr Jerry O. Akimoh, Chief Dele AbdulSalam, Pharmacist AbdulAzeez Busari, Tope Adediji, Ogunbiyi Victor, Imrana Rufai, Bro Ahmed Abdullah and all staff of computer science department.

Finally, my special thanks to my loving wife Mrs Fausat Iyabo Badmos for being so understanding, to my children Aishat, AbdulRahman, Badmos Jnr and Fatima.

I say thank you for your sacrifices to you my brothers, Col W. A. Bakare, Alh. AbdulRasheed Giwa, AbdulGaniyu Yisau, Mr Tajudeen Giwa, am grateful for being supportive all the way to arrive at this point.

Dedication

I dedicate this dissertation to my darling wife**, Hajia Fausat Badmos,** and my dear children, **Aishat, AbdulRahman, Badmos Jnr** and **Fatima** who have sacrificed a lot to help me with this study and have been a source of inspiration and encouragement for me to achieve this goal.

Table of Contents

List of Figures

List of Tables

List of Notations

The following notations are used in this research work.

$\boldsymbol{\mathcal{A}} =$ denotes a robot

$\mathcal{W}$ = denotes robots' workspace

$\mathcal{WO}$ = denotes Obstacle in $\mathcal{W}$

$\boldsymbol{\mathcal{C}}$ = is the configuration space of $\boldsymbol{\mathcal{A}}$.

$\boldsymbol{q}$ = An element of the set $\boldsymbol{\mathcal{C}}$.

$\boldsymbol{\mathcal{A}}(\boldsymbol{q})$ = is the subset of $\mathcal{W}$ occupied by $\boldsymbol{\mathcal{A}}$

$\boldsymbol{\mathcal{C}}\mathcal{O}$ = is the configuration space obstacle region

$\boldsymbol{\mathcal{C}}_{\text{free}}$ = is the free configuration space and is the complement of $\boldsymbol{\mathcal{C}}\mathcal{O}$

VD = Voronoi Diagram

GVD = Generalized Voronoi Diagram

APF = Artificial Potential Field

Abstract

Path planning for mobile robots is a complex problem. However, it becomes more challenging when it comes to planning paths in dynamic environments. This is because the robot needs to reach an agreement between the need of having efficient and optimal paths and the need to deal with unexpected obstacles. The proposed algorithm for this work is based on two concepts, the Voronoi Diagram used for the environment representation and the Deformation Retracts which are integrated into the system to enable the path planner to deal with the effect of the moving obstacle by deforming the Voronoi Diagram. The fusion of the aforementioned two concepts, Voronoi Diagrams and Deformation Retracts, which are from two related mathematical disciplines (Computational geometry and Algebraic topology), has not yet been considered in robotics applications. The proposed system first extracts the collision-free space by computing a Generalised Voronoi Diagram (GVD) and generates a pre-planned robot path, then the deformation retract is applied on the free space of the Voronoi Diagram created after an interference due to a moving obstacle. The map is deformed, and the initial path is updated to an alternative path if it exists. One important feature of this algorithm is that it is complete because it generates a solution (path) and the dimension of the map has been reduced to one which represents the retracted free space in the environment. This makes the new system applicable to robot navigation in complex environments, and in other research areas such as computer games, virtual reality, and computational geometry to mention but a few. Simulation results of some environments demonstrate the effectiveness of the new algorithm. The findings of this work have shown that Voronoi Diagram and Deformation Retracts techniques are a good combination for solving path planning problem using Deformable Voronoi Diagram for mobile robot in a dynamic environment.

Chapter One

1.0 Introduction

1.1 Background and Motivation

Moving from one place to another is a simple task for humans to accomplish without any difficulties however, this trivial task can be challenging for a mobile robot to overcome. Therefore, it is now imperative to address the quest of planning path for mobile robots to solve this challenge of moving from one point (called starting point) to another (called goal or target point) without collision.

It is obvious that, be it in railway stations, airports, at carnivals, or at international conferences, humans can move through to their destinations with ease. Whereas for a robot even a simple basic task of motion represents a challenge, and navigating a crowded place could be even more complicated. This work focusses on how to equip a mobile robot to be able to navigate crowded places without difficulties.

Several researchers have conducted studies on the functions of autonomous mobile robots. For example, (Almanza-Ojeda & Ibarra-Manzano, 2011) presented a study on how an autonomous robot could navigate in outdoor environments using sensors, (Alves et al., 2011) showed that the main feature of a mobile robot is to perform functions ranging from cleaning tasks to the exploration of the universe without any risk to the robot and the user, and (Raol & Gopal, 2013) in their work were able to discover that for a mobile robot to perform a task, it had to possess some functionalities such as obstacle detection, obstacle avoidance and finding a safe path. An overview of initial approaches for path planning problems by various researchers can be found in (Latombe, 1991) and recently by some researchers like (Campbell et al., 2020).

In the beginning, robots were just simple mechanical arms and hence path planning techniques were applied solely to static environments. These types of robots are mostly found in industries and are mainly used by trained and qualified personnel.

However, due to technological innovations, fields like computer science, electrical and electronic engineering, mechanical engineering, and social sciences were used to manufacture different types of robots. Nowadays, many of these robots are used in different areas like in mining industries (Nanda, Dash, Acharya, & Moharana, 2010), nuclear energy companies (Cho & Woo, 2016), search and rescue for firefighters (Schneider & Wildermuth, 2017), etc. Other examples are Nano robots in medicine for monitoring human health, assistive robots in household environments, commercial robots such as Drones in delivery services (*https://retailminded.com/what-are-commercial-robots/#.YEKV6ej7TIU*), disaster response robots for dangerous tasks, robots in the educational sector, robots in the entertainment industry, exoskeleton robots for physical therapy, humanoid robots that are made to look like humans (Pearson & Beran, 2018) (*robot.net/robotics/types-of-robots*), and underwater walking robot by (J. Ayers, 2004) or robots for fast reaction to the dynamic (e.g. fish) and static (e.g. rocks) obstacles in the sea by (Williams, Pizarro, Mahon, & Johnson-Roberson, 2009). The work of (G.S. Virk, C. Herman, R. Bostelman, T. Heidegger, 2013) stated that robots would be used in every home in the nearest future. Few examples of these modern robots are in Figures 1.1, 1.2, and 1.3.

(Robot Hoover)

(robot dog)

Figure 1.1 Some Robots and areas of applications

https://th.bing.com

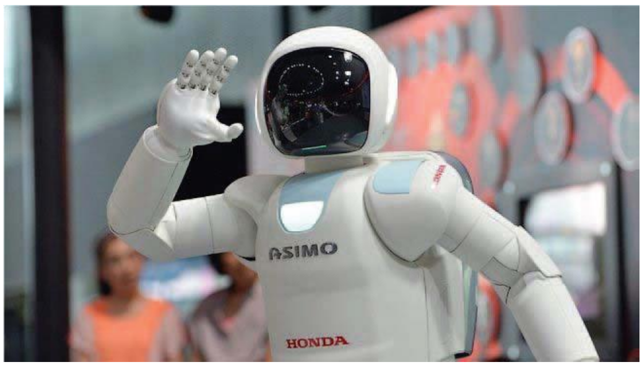


Figure 1.2 Robot for entertaining guest

https://www.thejakartapost.com/life/2018/06/28/end-of-the-line-for-asimo-japans-famed-robot.html

Figure 1.3 robot for guiding tourists

taiwannews.com.tw

But for a robot to perform a given task it has to be properly equipped with effective algorithms that are complete, optimal, robust against changing environment and uncertainties, able to deal with limitations, and comply to safety regulations. The algorithm has to address the path planning problem based on the robot's environment. Mobile robot's environment can be static or dynamic: a static environment is where the whole solution must be found before starting execution and a dynamic environment is characterized by the uncertainty of the environment that limits the ability of robot to make decisions where re-planning is needed at every stage with more time for planning update (A. Tuncer, M. Yildrim & K. Erkan, 2012). However, whether an environment is static or is dynamic, one common fundamental task is how can a mobile robot successfully navigate without any collision. Furthermore, whether it is a robotic manipulator or a mobile robot, obstacle detection and avoidance in the environment whilst performing a task is very essential. Therefore, since every robot is made for different purposes, the selection of suitable path planning approaches becomes very important. These path planning approaches are also classified into two categories based on the robot's environment: Global path planning and local path planning. The Global path planning is mostly applied to static environments whilst local path planning is mainly applied to dynamic environments.

1.2 Path planning approaches for mobile robot

An effective path planning approach is to enable a mobile robot to navigate from one position to another in order to perform the required function. An overview of several path planning approaches for mobile robot can be found in the works of (Jing, 2008) and (Kunchev et.al, 2006: Kazemi et.al, 2010).

Research on path planning approaches had gained attention already in the 70s with the advent of Artificial Intelligence, which led to the categorisation of path planning techniques into Classical approach and Reactive approach. In the work of (Patle et al., 2019), the classical approaches were said to be the only techniques used for solving path planning problems. Some of these classical approaches are complete i.e. they find a path if one exists or confirm that there is none found. Though despite its completeness, there are drawbacks such as high computational cost and failure to react to unexpected obstacles in the environment that affect the classical approaches. These drawbacks makes the traditional/classical approaches largely applied to static environments. However, the reactive approaches are mainly employed for path planning in dynamic environments (R. Brooks, 1986,

1990) because they have remedy for the uncertainties in the environment. Despite this advantage of reactive approaches over classical approaches, there are also some drawbacks such as high computational time, large memory requirement and, learning phase. Examples of Classical approaches are the Cell Decomposition method and Roadmap method whilst some examples of reactive approaches are fuzzy logic, particle swarm optimization, neural network, ant colony optimization, genetic algorithms (Patle, Pandey, Parhi, & Jagadeesh, 2019). Artificial Potential Field (APF) method is considered to belong to both approaches, i.e., to classical approaches, becausethe position of the target or direction to the target should be known, and to reactive approaches, because the position of the obstacles may not be known, and the robot reacts to them only when it 'senses' them.

The classical approaches have been proven to be less capable of dealing with unknown, partially understood, or dynamic settings, and they are known to be computationally costly. They are found to be dependent on the prior knowledge of the environment to generate a feasible path for robot whilst dynamic approaches can address these challenges due to their ability to handle unexpected or partially known environments.

Amongst the classical and reactive approaches lies a gap that needs to be bridged, which can be addressed by the deformation retracts technique. This (gap) problem emerges when an obstacle changes position after a path has been generated, the pre-planned path becomes inapplicable to avoid a collision. Therefore, an alternative path is needed. Since the classical approach cannot address such changes in the environment, the deformation retract may re-plan the environment, and update the connectivity in the free space so that an alternative or a new path is generated for robot execution. However, whether it is classical or reactive, researchers have shown that every approach has its limitations and also that one approach sometimes cannot solve certain problems. Therefore, researchers have come up with the fusion or integration of two or more approaches to solve path planning problems.

In this work, the fusion of the generalized Voronoi diagram (GVD) and deformation retracts is used to enhance the capabilities of a mobile with deformable diagram to deal with dynamic environment. This is because the changes in the environemnt required an update that leads to replanning. To finding the shortest route/path Dijkstra’s algorithm is employed. This approach addresses the problem of path planning using deformable diagram by combining the generalized Voronoi diagram and deformation retracts techniques.

1.3 Aim of the study

This study aims to complement the efforts of the past studies resulting from integrating two or more techniques for designing the path. The research is based on the evaluation of theoretical concepts and their application in simulation. The focus of the proposed research work is to design a path planning system for a mobile robot in a dynamic environment. To achieve this aim, a list of objectives is stated below.

1.4 Objectives of the study

1. To compute a generalized Voronoi diagram with static obstacles.
2. To plan a path for a mobile robot from a source point to a goal point without colliding with obstacles in a dynamic environment.
3. To design an algorithm for finding the shortest path on the free space by implementing a Voronoi diagram.
4. To introduce dynamic obstacles.
5. To describe the method of the deformation retracts and to analyse its procedures.
6. To build a simulation for testing and simulate several scenarios within the environment and discuss the results.

1.5 Contribution of the thesis

This work develops a technique based on the fusion of two fundamental concepts: Topology (Deformation retracts) and Computational geometry (Voronoi Diagrams). The novel contribution of this work is how the combination of these two established concepts: Deformation Retracts and Voronoi Diagrams, can be used to address path planning problems in a dynamic environment. Whilst these concepts (Voronoi Diagrams and Deformation Retracts) have been developed in related domains, their fusion has not yet been considered. The Generalized Voronoi diagram is used to reduce the workspace to one dimension by extracting the free space and the deformation retract is to re-plan and update the connectivity of the free space due to the movement of an obstacle in the environment and finally generate a new path. The combination of these two concepts results in the so-called Deformable Voronoi Diagrams. Using Deformable Voronoi Diagrams and also taking into consideration such factors as time, paths lengths and sizes of the Voronoi diagrams in simulation the effectiveness of the proposed algorithm is demonstrated. The proposed algorithm can be useful not only in robotics but also in other domains such as computer games, virtual reality, and computational geometry.

1.6 Outline of the Thesis

Chapter one presents the introduction, background and motivation, the aims and objective of the study, and the contribution of the thesis. Chapter two is the literature review about mobile robot path planning techniques in dynamic environments. Chapter three describes some theoretical concepts related to the subject matter. Chapter four describes the methodology in detail and initial results, and Chapter five presents simulations and a discussion of the results. Chapter six concludes the Thesis.

Chapter Two

2.0 Literature review

2.1 Background

Several researchers have been working tirelessly to find efficient solutions to path planning problems because robot motion is one of the major tasks of a robot whilst accomplishing its purpose. There are many works on robot navigation in both static and dynamic environments. Examples of some of these works can be found in (Blanco, Moreno, & Curto, 1998) where the design of a method for planning path for mobile robot in a dynamic environment, and specifically emphasized on the use of online method for dynamic environments for developing an optimal path planner, and in (Mahajan & Marbate, 2013) emphasis on how robots could socially interact with humans in the same environment without conflicts was made. However, path planning for robots in changing environments has been a focus for many researchers recently.

Considering the environment, there are two categories of path planning systems: the online planning system and the offline planning system. In online path planning, the total information about the environment is known in advance. Some of the approaches used for online path planning systems are Voronoi Diagram (Ó'Dúnlaing & Yap, 1985), and Visibility graph (Mitchell, 1988). The most applied in recent times are local path planning systems where obstacles motions cannot be determined in advance. In local path planning, the robots get information through sensors, a suitable approach used in local path planning is the Artificial potential field. This work contributes to the solutions of robot path planning problems and obstacle avoidance in a dynamic environment using deformable Voronoi Diagrams.

However, among the most widely used techniques for path planning problems are Visibility graph, Voronoi Diagrams, Cell Decomposition, and Artificial Potential Field (Campbell et al., 2020b). These approaches are faced with some drawbacks like the high cost of computation, and high time in execution of algorithms even if a small number of robots are implemented, and getting trapped in local minima (Bounini, Gingras, Pollart, & Gruyer, 2017; Lv & Feng, 2006) which becomes worse when the environment is changing. Heuristic or soft computing techniques such as Fuzzy logic (Mac, Copot, Tran, & De Keyser, 2016; Saffiotti, 1997), Ant Colony optimization (Bi, Yimin, & Yisan, 2009; Rashid et al., 2016), Genetics algorithms (Han, W., Baek, & Kuc, 1997; Lamini, Benhlima, & Elbekri, 2018), and Neural networks (Engedy & Horváth, 2009; Yu, Su, & Liao, 2020) are also used for robot path planning applications.

2.2 Path Planning Techniques

The research on path planning techniques has been in existence since in the 60s but this area of research did not get much attention until the work of (Lozano-Perez, 1990). He presented his work on spatial planning where he introduced a configuration approach, which was applied to selecting the motion of an object without collision. This was followed by many works on robot path planning techniques such like, a work on socially-aware trajectory planning was presented by (Kruse, Pandey, Alami, & Kirsch, 2013) which mainly deals with the behaviour of robot during navigation, also a work that divides path planners into a global planner and a local planner for robot navigation was presentented by (Chik et al., 2016), a comparative evaluation of Velocity Obstacle (VO) approaches for various agents was reported in (Douthwaite, Zhao, & Mihaylova, 2018). Different evaluation metrics to deal with the uncertainty produced by the robot's low-resolution sensors were also presented. The motion planning methods were categorized into classical and heuristic methods by (Cheng, Cheng, Meng, & Zhang, 2018). When the two approaches were compared, it was discovered that the heuristic technique performed better in online path planning and because of the application of mobile robots in many areas like health sectors, manufacturing companies, under-water operations, space explorations, nuclear energy plants. A good path planning technique allows autonomous mobile robot to traverse a path from the start location to the target location without any collision in the workspace, to minimize the danger to the robot, to take care of uncertainties, to find an optimal path in a short time, and to report to the user if there is none.

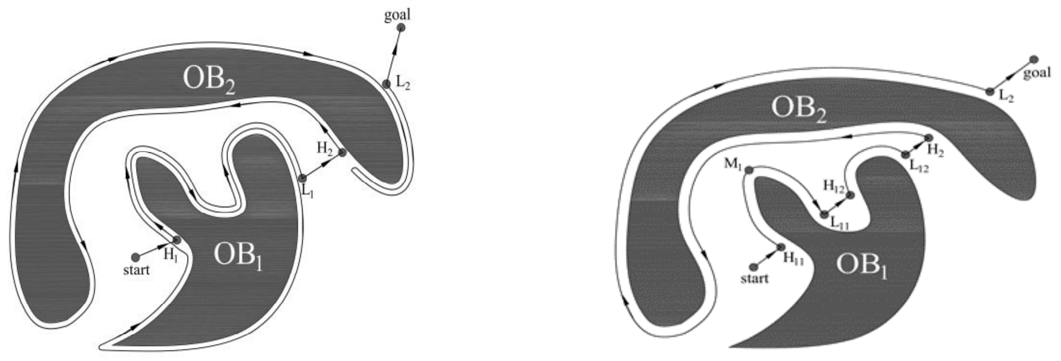
Today, there are numerous techniques for robot path planning. This can be divided into: Conventional and Heuristic techniques. Some of the Conventional techiques are Bug Algorithms, Roadmap approach, Cell Decomposition, Potential Fields methods, Sampling-based approach, Kalman filtering, and Heuristic approaches are Artificial Neural Network, Ant Colony Optimization, Genetic Algorithms etc. Each of these methods is effective for different path planning problems. The proposed algorithm is to compliment the efforts of the researchers in dealing with path planning problems with deformable Voronoi diagrams using the combination of generalized Voronoi diagrams and deformation retracts.

In what follows, more details are provided on the aforementioned techniques.

2.3 Conventional techniques

2.3.1 Bug algorithm

There are several types of Bug Algorithms, but the most widely implemented in path planning problems are Bug1, Bug2, VisBug, DistBug, and TangentBug (Choset Howie et al., 2005; Sariff & Buniyamin, 2006). The Bug's algorithms are simple path planning techniques with good assurance (Choset Howie et al., 2005) and the basic concept of Bug1 is that the robot continues to navigate towards the goal along the path from the start to the goal unless there is an obstacle, then the robot explores the alternative paths around the obstacle until the motion to the goal is available again (or concludes that there is no path) (Sankaranarayanan & Vidyasagar, 1990). Whilst in Bug2 the robot continuously follows the the straight line to the goal, and if an obstacle is encountered then it follows the edges of the obstacles until the line to the goal is discovered (Magid & Rivlin, 2004). Bug algorithms are good for online path planners with a few sensors (NGAH, Buniyamin, & Mohamad, 2010) (Behnke, 2003). However, these techniques have shown that they have complete solutions and the main weakness is at the cost of the length of the paths and time. This proposed work can also be seen in chapter 5 of the dissertation, Figure 5.34 and Figure 5.35 where ‘time’ is used as performance factor for evaluation where the time spent for the VD construction before and after deformation were considered.



Bug1 Bug2

Bug1 algorithm with H_1, H_2 are hit points whilst L_1, L_2 are leave points

Bug2 algorithm with H_{11}, H_1, and H_2 are hit points whilst L_{11}, L_{12}, L_2 are leave points

Figure 2. 1 *Mobile robot’s path (Nguyen & Le, 2016)*

2.3.2 Sampling-based approach

The sampling-based approach was presented in the 90s to solve the problem of deterministic path planning techniques for robots of six degrees of freedom under different constraints (Hsu, Kindel, Latombe, & Rock, 2002). The basic concept is to focus the search on the randomly explored configuration space instead of the whole space including obstacle spaces. This makes the design of the path planning algorithm less dependent on the geometric model of the environment. The sampling-based approach uses a collision detector as the only source of information. The free space has many samples that can be connected with free paths to get the path planning problem solved (Khaksar, Hong, Khaksar, & Motlagh, 2012). A sampling-based approach is simple, and it still works under many barriers or constraints. Some of the works on these constraints are:

(1) kinematic and/or dynamic motion constraints (Hsu et al., 2002),

(2) closure constraints (Han, L. & Amato, 2001),

(3) dynamic balance restrictions (Kuffner, Nishiwaki, Kagami, Inaba, & Inoue, 2001),

(4) re-configurable robots (Fitch, Butler, & Rus, 2003),

(5) manipulation constraints (Lamiraux & Kavraki, 2001),

(6) contact state constraints (Ji & Xiao, 2001),

(7) short inspection constraints (Danner & Kavraki, 2000).

It may be simple in a cluster environment where a sampling scheme may be applied, but the planning time is higher and makes it inefficient.

2.3.3 Artificial Potential Field approach

The Artificial Potential Field method in the past years has gained much attention for the obstacle avoidance problems in robotics. The main idea of this method is that artificial forces act on robots (Khatib, 1986). The Artificial forces comprise of the attractive force that attracts the robot to the goal and the repulsive forces that repulse the robot from obstacles. This approach can be used for global path planning with convex obstacles for a mobile robot using sensor data (Azariadis & Aspragathos, 2005). However, its simplicity makes it very popular among other approaches. In the work of (Krogh & Thorpe, 1986) they used potential field method for both offline and online path planning. The APF method has the problem of local minima, where a robot gets trapped and the goal not attainable (Ge & Cui, 2000). (Boukas, Kostavelis, Gasteratos, & Sirakoulis, 2014) in their work proposed an evacuation system using the Artificial Potential Field approach. However, the APF is not a retraction, but a robot in Potential Fields method is treated as a point represented in configuration space as a particle under the influence of an artificial potential field whose local variations reflect the structure of the free space.

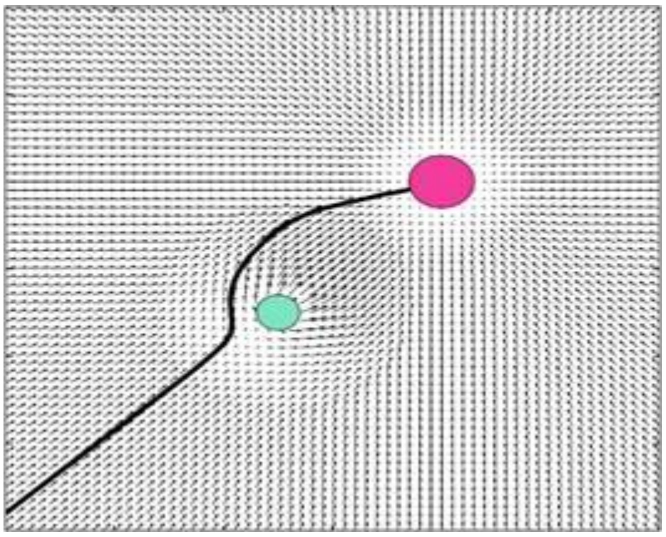


Figure 2. 2 Artificial Potential Field Approach[1]

2.3.4 The graph-based approaches

The Graph-based approaches are one of the oldest ways of building free space for the robot. These free spaces are connected through edges to generate free paths. The vertices can be considered as free spaces, and the network between the free spaces (vertices) and the lines (edges) are used by the graph-based approach to generate a collision-free path for the robot. This approach is applicable to both static and dynamic environments. The two well-known roadmap methods are the Visibility Graphs (VG) and Voronoi diagram (VD). They have achieved very good results with diagramatically different types of roads. A visibility graph is a graph that allows robot to come as close as possible to obstacles. As a result, the shortest path is found by applying this method. The path in VG touches obstacles at the vertices or edges which is dangerous for the robot. Contrarily, a Voronoi diagram generates a road that tends to maximize the distance between the robot and the obstacles. However, the solution paths based onVoronoi diagrams are not optimal with respect to path length (Mac et al., 2016).

a. Roadmap

*1.https://miro.medium.com/max/450/1*podzvpWd_ApSOo-SaYGw3w.jpeg*

The Roadmap approach is used to achieve the connectivity of the graph in the free space. (Tang, Khaksar, Ismail, & Ariffin, 2012) presented three stages to be followed whilst implementing the roadmap approach:

1) Navigation of robot from source to goal on the roadmap
2) Then moving from the target to another point and
3) Connecting the two points with lines in the roadmap.

The roadmap-based approach can also be used to solve computational difficulties in a complex environment. It can also make free space smaller using an undirected graph structure (Chia, Su, Guo, & Chung, 2010). The Visibility graphs and Voronoi Diagrams have computational geometry structures. Whilst Visibility graphs are designed for finding the shortest path, the Voronoi diagrams are implemented for maximum clearance paths. Though, the paths generated by the Visibility graph are shortest paths but not optimal, because the robots touch the obstacles whilst navigating which might not be too safe for robots. Therefore,with the introduction of the Voronoi diagrams, maximum clearance paths are created to make it safer for robot during navigation. This provides one of the solutions to the problem of Visibility graph. It was pointed out by (Šeda, 2007) that every region in a Voronoi diagram corresponds to a site which implies that all points in a region are closer to the site in that region than any site from other regions. One of the advantages of Voronoi-based path planning is that it reduces the dimension of the problem to one and this can be also refered to as Retraction.

b. Visibility graph

The idea of a Visibility graph is to form a network or graph of vertices of polygons (obstacles). (Saska, et al, 2006) indicated that for two vertices to be connected, they must be visible whilst Dijkstra’s algorithm is applied to search the shortest path. (Masehian & Sedighizadeh, 2007) showed in their work that a Visibility graph is a set of lines in the free space that connects the characteristic (node) of one obstacle to another and these are of polygonal shapes with $O(n^2)$ edges in the visibility graph.This approach is good for generating shortest path, though the paths may not be considered as safe because the robot touches the obstacles whilst navigating towards the goal. Since there are numerous complicated paths to be searched for, the efficiency of this method is negatively affected.

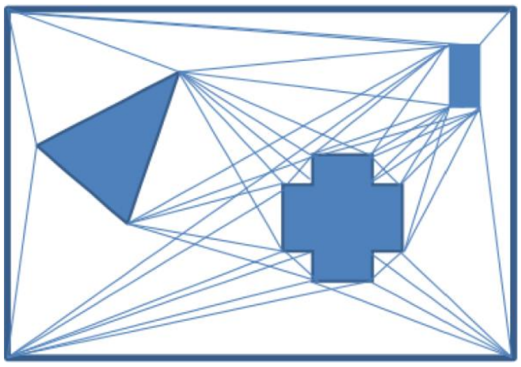


Figure 2. 3 Visibility graph[2]

c. Voronoi Diagrams and Generalised Voronoi Diagram

The Voronoi concept origin dates back to 17th century. Structures that looks like Voronoi diagrams can be traced back to the work of Rene Descartes in 1644, where he used a Voronoi-like diagram to refer to the location of matter within the solar system and since then researchers have started the development of numerous algorithms for the computation of Voronoi diagrams. The survey of some of these work can be found in (Aurenhammer & Klein, 2000), (De Berg, Van Kreveld, Overmars, & Schwarzkopf, 1997), and (Okabe, Boots, Sugihara, & Chiu, 2009) where numerous algorithms and their applications to Voronoi diagrams were discussed. (Canny & Donald, 1990) presented work on a Voronoi diagram as a tool in robot path planning where a search for a path in a particular space can be reduced to a search in one-dimensional space. The GVD has been used by many researchers as a basis for path planning for a long time as shown in the works of (Choset & Burdick, 1995), (Choset and Burdick, 1996), (ó'Dúnlaing, Sharir, & Yap, 1983), and (Wilmarth, Amato, & Stiller, 1999). However, the proposed

2. https://media.springernature.com/full/springer-static/image/art%3A10.1007%2Fs42154-019-00081-1

work differs from other work by deforming the Voronoi diagrams and still regenerates the robot path.

Several previous works have shown the computation of Voronoi diagrams such like (Fortune, 1987; Zavershynskyi & Papadopoulou, 2013) where a sweepline algorithm can be used to compute Voronoi diagram for n point sites in O(nlogn) time and an algorithm to construct order-k Voronoi diagrams with a sweepline technique in O(k2nlogn) time complexity and O(*nk*) space complexity respectively were presented. However, this work differ from the works mentioned above in the computation of Voronoi where a matlab function called '**voronoi**' is used for the computation of Voronoi Diagrams where point sites are used for computing Voronoi diagrams for polygonal obstacles.

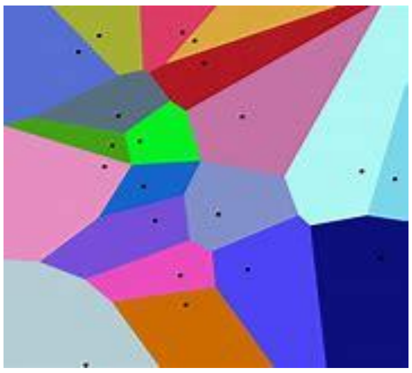


Figure 2. 4 Voronoi Diagram (http:/en.wikipedia.org/wiki/Voronoi_diagram)

The use of Voronoi Diagrams and deformation retracts to obtain a Deformable Voronoi diagram in solving path planning problem also makes this work different.

According to (Choset Howie et al., 2005), the generalized Voronoi diagram can be said to be a set of points in the workspace from different regions are equidistant from each other. Whilst (Mahkovic & Slivnik, 1998) and (Nagatani, Choset, & Thrun, 1998) developed a technique for the building of a generalized local Voronoi diagram and a Voronoi diagram in a dynamic environment respectively, none of these works combined VD with deformation retracts nor did they deform the VD in generating paths. Constructing an efficient generalized Voronoi diagram has always been not without drawbacks in path planning but some solutions have helped other works including this work in the construction of Voronoi diagrams. For instance, in works of Choset (Choset, 1995 and Choset, 1996) an algorithm solution, called the hierarchical generalized Voronoi graph, was designed for a robot to navigate a changing environment, and in (Steven et al., 1999) it was shown how the points on the diagram can be searched without constructing the whole map.

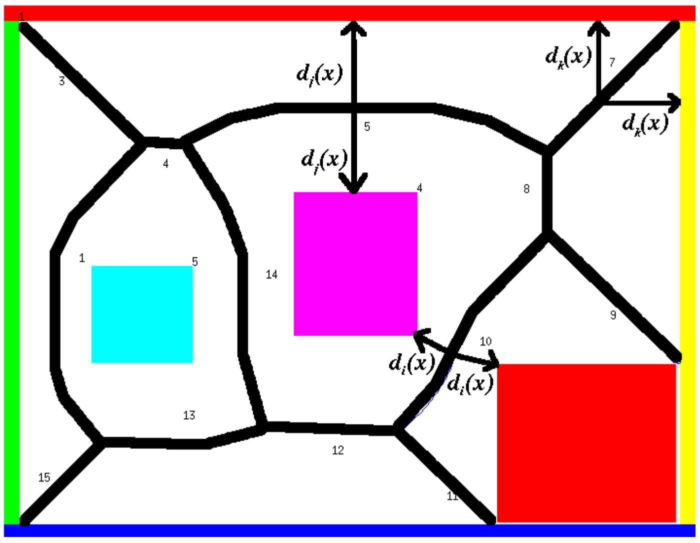


Figure 2. 5 Generalized Voronoi Diagram (https://th.bing.com/th/id/)

d. Deformation Retracts

The basic concept of deformation retracs is to smoothly deform a path or map without losing some of the properties of the original path or map. This is because deformation is about stretching or shrinking of a space or object but not crushing or cutting of a space or object. However, if a path is referred to as a sequence of vertices with the property that each vertex in the sequence is adjacent to the next vertex is smoothly deformed the number of vertices will still be the same with the original path before deformation. A similar work from (Gayle, Sud, Lin, & Manocha, 2007) was presented where an algorithm for motion planning that used deformable links and dynamically retracted to capture the connectivity of the free space for autonomous robots in a dynamic environment, and in the work of (Lamiraux & Bonnafous, 2002) a method to reactive obstacle avoidance for nonholonomic systems which was based on deformation of an initial motion generated by a path planner was presented.

Consider the figure 2.6, let the space $A = S^1$ is the unit circle

And $X = S^1 \times [0,1]$ is the unit cylinder

Then, A is a deformation retract of X.

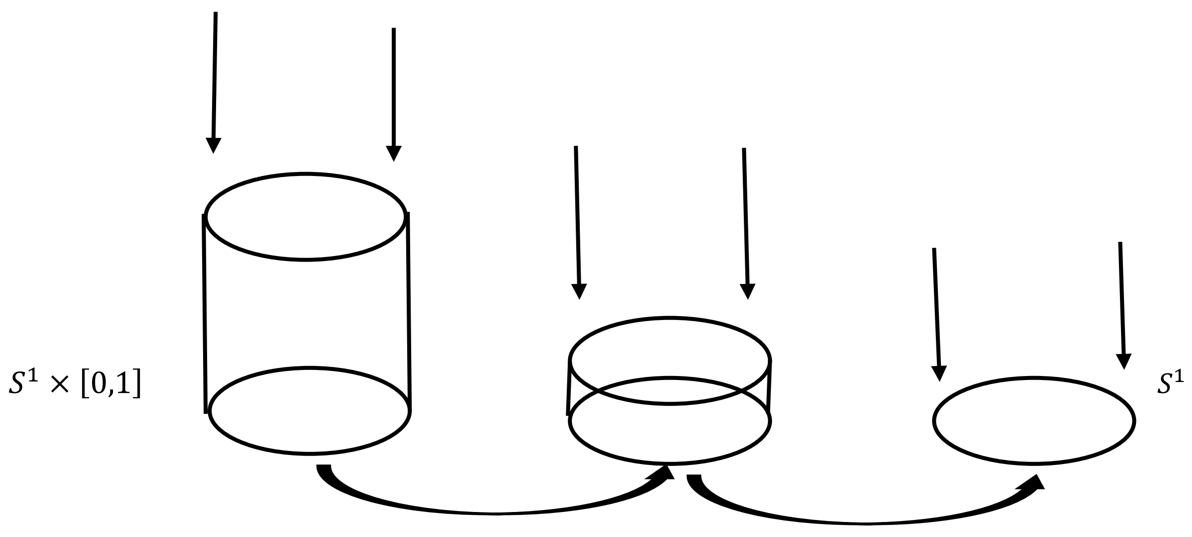


Figure 2.6 Example of deformation retract

2.4 Heuristic Approaches

The traditional techniques for path planning have problems of high time for algorithm execution and the presence of local minima that poses challenges to the efficiency of their implementation, but the heuristics approaches have some algorithmic solutions to address some of these problems. Therefore, there are many works on heuristic approaches, such like Neural Network (Zhu & Yang, 2006), Genetic Algorithms (Zhang, Sun, Xiao, & Tsang, 2007), Simulated Annealing (Manousakis, McAuley, Morera, & Baras, 2005; Mohamad, Taylor, & Dunnigan, 2006), Ant Colony Optimization (Mohamad, Taylor, & Dunnigan, 2006), PSO (Saska, Macas, Preucil, & Lhotska, 2006), Tabu Search (Masehian & Amin Naseri, 2004), and Fuzzy Logic (Lee & Wu, 2003). Most of these techniques both conventional and heuristic are combined to develop an efficient path planner (Charalampous, Kostavelis, & Gasteratos, 2015).

2.5 Path Planning Algorithms

The major characteristic of an algorithm for path planning is convergence. This implies that an algorithm for path planning must be able to discover a path, if one exists, otherwise to tell the user if such a path does not exist and then to stop. However, in the work of (Coenen & Steinbuch, 2012) we can find the characteristics of Convergent Algorithms as:

i) Length- the distance between the source and the goal should be the shortest. This implies that the algorithm should be able to find an optimal path.

ii) Time- the execution period needs to be as small as possible

iii) Robustness- this is the capacity of the algorithm to fault tolerance should be high. This implies that the approach should be able to deal with uncertainty, and this has been addresed by the proposed method, uncertainties such like the size of the obstacles. However, the new method still work correctly.

iv) Simplicity- the implementation of the algorithm should be as simple as possible.

2.6 Related Works on path planning using combined methods

Combinations of several path planning methods for the path planning problem can be found in (Guo et al., 2021; Masehian & Amin-Naseri, 2004) and (Dongbin & Jianqiang, 2006). However, few examples that include Voronoi Diagrams are reviewed below.

2.6.1 Voronoi Diagrams and Fast Marching

A method was presented by (Garrido, S. & Moreno, 2015; Garrido, Santiago, Moreno, Blanco, & Jurewicz, 2011) using the fusion of Voronoi diagrams and fast marching for mobile robot path planning. This method combines map-based and sensor-based techniques to generate a feasible motion plan, whereas it operates at the frequency of the sensor. The Voronoi diagram was used to reduce the configuration space into a unidimensional space and used fast marching to obtain the path from the collision-free areas of the Voronoi diagrams. This permits the usage of this method in complex environments where different Voronoi-based strategies will not work

2.6.2 Tube Skeletons structure and Fast Marching

(Garrido, Santiago, Moreno, Abderrahim, & Blanco, 2009) presented a similar to the work 2.6.1 but the safest areas in the environment are extracted by means of a tube skeleton like a Voronoi diagram but with tubular shape, then the fast marching obtained the collision-free path from the tube skeleton. This method is map-based and sensor-based, good for static environments. Its effective, it produces smooth trajectories and is characterized by non-holonomic restrictions.

2.6.3 Voronoi Diagrams and Genetic Algorithms

Researchers like (Li, Dong, Bikdash, & Song, 2005) and (Benavides, Tejera, Pedemonte, & Casella, 2011) developed methods for path planning based on Voronoi diagrams, where obstacles in the environment are considered as the generating points of the diagram, and a genetic algorithm is used to find a path without collision from the robot initial position to robot target position. For the optimal path, the fitness function was used. This method is good for static environments, efficiency is good, and has low execution time.

2.6.4 Ant Colony Optimization and Dynamic Voronoi Diagrams

This approach is presented by (Habib, Purwanto, & Soeprijanto, 2016) for mobile robot planning problem using the modified ant colony optimization algorithm based on the Voronoi diagram. The VD generates vertices that is assumed nodes in the ACO and the mobile robot is assumes as ant. This is robust for dynamic environments, attracts adaptability, and very good efficiency in path planning. It is also generates path that is more safe than the previous above.

2.6.5 Voronoi diagram using Parameter Clearance-based Shortest Path

(Bhattacharya & Gavrilova, 2008) presented a work that constructed a roadmap and the best path was obtained from the Voronoi diagram using clearance condition or constraint which would be set by the user initially and (Niu, Lu, Savvaris, & Tsourdos, 2018) combines the Voronoi-Visibility to allow Unmanned Surface Vehicles to avoid obstacles whilst at the same time using minimum amount of energy. The impact of parameters such as mission time, the USV speed and sea current state on the results were analysed and it shows that the proposed VV algorithm improves the quality of the Voronoi energy efficient path whilst keeping the same level of computational efficiency as that of the Voronoi energy efficient path planning algorithm. This assures the optimality of the path. This is used in a dynamic environment, more effective in speed quality, and generally very effective.

2.6.6 Voronoi Diagram and Probability Roadmap

(Bhattacharya & Gavrilova, 2007) designed a method based on sampling-based techniques that supported taking the problem into an imaginary environment. This is effective in dynamic environment, it is characterized by the time spent between the re-plan and deformation.

2.7 Obstacle Avoidance

Obstacle avoidance is a major problem in robot path planning simply because a robot needs to reach its destination without a collision with any obstacle. However, a path is said to be free if obstacles are avoided during robot navigation. (Khatib, 1986), in his work designed a unique real-time collision avoidance method for mobile robots and manipulators using the artificial potential fields to enable robot in a real-time dynamic environment, (Petres et al., 2007) designed a framework that was applied to path planning and obstacle avoidance for underwater vehicles for sonar purpose and (Choset Howie et al., 2005) applied an online technique to develop an algorithm for obstacle avoidance by means of reactive control during the robot motion. (Lamiraux & Kavraki, 2001) also presented an algorithm to address a path deformation for obstacle avoidance. Lastly on this, since one of the essential requirements for all intelligent machines is safety when it encountered an emergency during the task, it has made path planning and obstacle avoidance an interested area of research. In the work of (Liu, Li, Zhang, Zheng, & Yang, 2019) a dynamic obstacle avoidance and path planning problem of USV based on the Ant Colony Algorithm (ACA) and the Clustering Algorithm (CA) to construct an auto-obstacle avoidance method which is suitable for the complicated maritime environment was presented.

2.8 Conclusions

Several methods of representation have been reviewed and it is concluded that there is no method with absolute perfection. All the methods reviewed showed that they all have limitations. However, to resolve some of these limitations, some methods were combined, and the proposed method of this work is an example. This study proposes the combination of the Voronoi diagram and the Deformation retract method for the planner. This is a roadmap-based method that uses a computational geometry data structure. However, before designing a path planning method, three criteria need to be taken into consideration: path length, computational complexity, and completeness. In a dynamic environment, findings have shown that some methods like Visibility graphs produce a shortest path but they may be computationally intractable as they may run in non-polynomial time with respect to the number of obstacles, whilst considering the Voronoi Diagrams, the paths generated are relatively safe since the edges of the path are positioned far away from the obstacles, but may not be shortest, though these paths are not optimal in terms of path length but has the fastest computation time compared to other methods. For instance, the computation time of VG exponentially increases with respect to the number of obstacles, whilst, VD had a consistent increment in computation time as the number of obstacles increases. The difficulty of a motion planning problem is determined by the complexity of the obstacle space O and the configuration space dimension D. The finite number of nodes utilised to approximate a continuous space is indicated by N. For computational complexity, one needs to establish upper and lower limits on the time required by the most efficient solution to solve a given problem. However, an algorithm's complexity is assumed to represent its worst-case complexity, unless otherwise stated. To prove an upper limit T(n) on a problem's time complexity for a given number of inputs n, all that is required is to show that there is a specific method with a running time of at most T(n) and to show a lower limit of T(n) for a problem, one must demonstrate that no algorithm has a temporal complexity lower than T(n). Big O notation is used to express the upper bound or worst-case complexity, which hides constant factors and smaller words. This makes the limits independent of the specific details of the computational model used. For instance, if $T(n) = 7n^2 + 15n + 40$, in big O notation one would write T(n) = $O(n^2)$.

The reasons for the selection of this method are: because of its completeness, ability to create a maximum clearance path for the safety of robot that is not guarranteed with some methods and, the fastest time of computation which is in $O(nlogn)$ time (complexity) and lastly, the querying for a path in this method is faster than in other methods even though the quality of paths generated from the Voronoi diagrams are far from being optimal. Combining this method with another methods, the deformation retract enables the new system to react to unexpected changes in the environment by deforming and updating the map.

Chapter Three

3.0 Study Related Theory

3.1 Background

Robots are used in many areas such as factories, airports, train stations, offices, shopping malls, and international conferences. The objectives of robots to these areas are numerous: to save time, prevent risk, avoid hazards and reduce manpower also to improve productivity to mention but a few.

Figure 3. 1 A conference venue at Sheffield Hallam University

Figure 3.1 is a typical example of crowded and/or dynamic environments. This type of environment requires some resources (human) to perform the smallest task which could be boring, tedious and wasting the time of trained personnel. However, the application of robotic technology has addressed most of the aforementioned problems. For example, a mobile robot that can perform "a search and rescue" task could relieve/save fire brigade personnel from a dangerous (risk) task. Path planning is a very significant aspect of robotics, it is an integral part of many robotic applications, for example, in medical (endoscopic path planning) where a target is located in a lung by generating an endoscopic path to the target (Geiger, Kiraly, Naidich, & Novak, 2010), in graphics applications where the graphic-based path planning approach is used on ray casting and voxel models

(Tarbutton, Kurfess, & Tucker, 2010), in manufacturing and CAD, an autonomous system was used in place of painting manually (Chen, Fuhlbrigge, & Li, 2008). The task of finding or generating a path between two points (start and stop/goal/target) and avoiding collision is known as the path planning problem. The path planning problem faces numerous challenges such as sensor problems like giving error information, unexpected scenarios (open or closed door), real-time issues, and dynamic obstacles. However, finding the optimal path is mostly the objective of researchers working on path planning problems.

3.2 Path planning/Navigation system

Motion planners are algorithms that deal with the problems of motion planning and are characterized by the followings (Coenen & Steinbuch, 2012):

i. **Task**, which comprises navigation, environment coverage, and mapping (is a task related to an unknown environment)

ii. **Completeness**, a planner can be complete, if a solution is found or tell the user if no solution exists, *resolution complete* if a solution exists based on some features of space resolution, if a solution exists with probability of finding the solution tends to one whilst number of sample tends to infinity, then it is *probabilistic complete*, and *incomplete* if a planner is not capable of guaranteeing a solution.

iii. **Optimality**, which comprises optimal, conditional optimal and non-optimal

A path planning system transforms a high-level task into a low-level illustrations of a robot motion. For the robot motion to be planned, the robot needs the representation of the environment (map), and the representation of the environment is computed through perception with the aid of sensors, then the robot needs to identify its position (localization) after mapping the obstacles into the environment. Figure 3.2 illustrates the architecture of a navigation system, it shows how the robotic system performs its tasks/functions whilst moving towards the target.

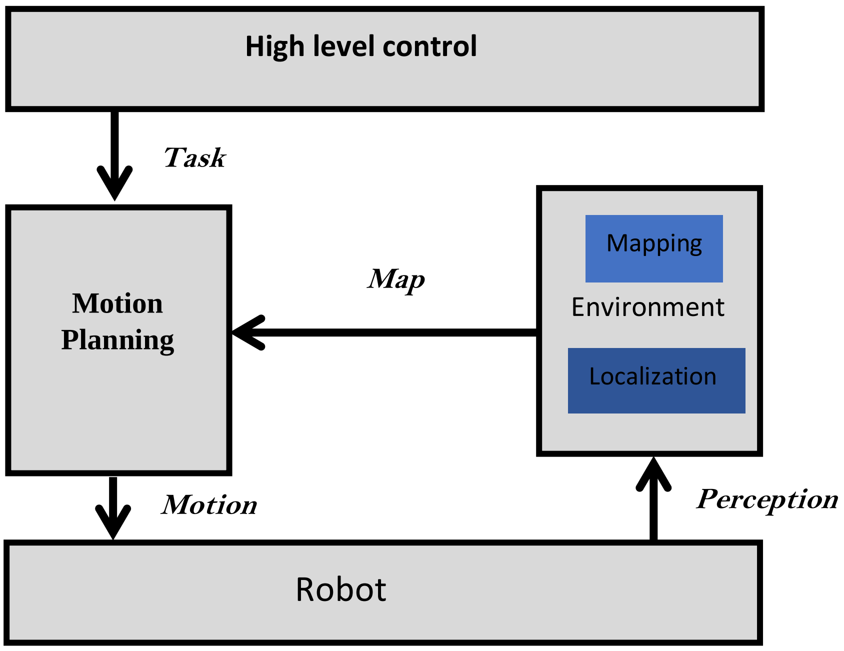


Figure 3. 2 Navigation system of a robot: Motion planning, Mapping, and localization[3]

3.2.1 Motion Planning Problem

We model a robot as a rigid body $\mathcal{A}$ moving in a workspace $\mathcal{W} = \mathbb{R}^d$ ($\mathbb{R}^d$ is Euclidean space) where d can be 2 or 3. The obstacle region $\mathcal{CO}_i$ is then the space occupied by obstacles $\mathcal{O}_i$ such that $i = 1, 2\ldots n$. The obstacle region is also referred to as the union of all obstacles. Since $\mathcal{A}$ and $\mathcal{WO}_i$ are subsets of $\mathcal{W}$, then the location and the size of $\mathcal{A}$ and $\mathcal{O}_i$ are known. Therefore, motion planning problem is referred to as: "given two points, the source and the goal of $\mathcal{A}$ in $\mathcal{W}$, find a path c that will not collide with $\mathcal{O}_i$ and will be used by $\mathcal{A}$ to move from the source to the goal and also indicate if such path does not exist*"* (Latombe, 1990).

3. *https://www.semanticscholar.org/paper/Motion-Planning-for-Mobile-Robots-A-Guide-Coenen-Steinbuch/6d6878f905d64ce8df9fd3d19ad2c11afbe4ee8e#extracted*

3.2.2 World Representation

To finding solution to path planning problem, identifying the position of the robot in regards to the workspace is necessary. This localization of the robot is considered whilst planning the motion so that no part of the robot will touch any obstacle when moving towards the goal. These considerations brought up the idea of configuration space (Lozano-Perez, 1983).

3.2.3 Configuration Space Vs Workspace

The real world where the robot performs its function or the space in which the robot works that can 2D or 3D is referred to as *Workspace* whilst the Configuration space or C-space of a robot is the space of possible positions robot may occupy. If the robot is taken to be polygonal, operating in a 2D environment, then the configuration of the robot will be specified by a translational vector, which can be represented by two coordinates (x, y). But if the robot changes its orientation by rotating, then an extra parameter θ is required for its orientation. Therefore (x, y, θ) can be used to represent the configuration space. The configuration of a robot is represented by the number of parameters which is equal to the number of degrees of freedom (DOF) of the robot. Generally, a robot translating in $\mathbb{R}^3$ will have three degrees of freedom because it can rotate about the (x, y, z) axes whilst a robot translating and rotating in $\mathbb{R}^3$ will have six degrees of freedom because it can move laterally about all the axes and also rotates about all the axes. The configuration space can therefore be thought of as the parameter space of a robot $\boldsymbol{\mathcal{A}}$, i.e. $(\mathcal{A})$.

3.2.4 Obstacle Configuration Space

Let $\mathcal{W}$ be the world that contains the robot and obstacles. For Euclidean space $\mathbb{R}^2$, let us consider that $\mathcal{W} \subset \mathbb{R}^2$ and $\mathcal{O} \subset \mathcal{W}$ is the obstacle region, that has a boundary. Then the complement $\mathcal{W} \setminus \mathcal{O}$ is taken as an open bounded set. And configuration space or $\mathcal{C} - space$ is the set of all rigid body transformations applied to the robot. Let $\mathcal{CO}$ be the part of $\mathcal{C}$ that part a robot cannot enter and let $\mathcal{A}\,(q) \subset \mathcal{W}$ be a closed set of points occupied by robot $\mathcal{A}$ when it transformed to configuration q. A configuration $q \in \mathcal{C}$ puts the robot into a collision if and only if $\mathcal{A}(q) \cap \mathcal{O} = \emptyset$, which implies that the robot and the obstacle intersect at least at one common point in $\mathcal{W}$, then the set of all non-colliding configurations is referred to as free space, $\mathcal{C}_{free}$. This is also described as:

$$\mathcal{C}_{free} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} = \emptyset\} \tag{3.1}$$

Whilst the compliment is referred to as the obstacle configuration space $\mathcal{CO}$ and is also defined as:

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}. \tag{3.2}$$

$$\mathcal{C}_{obs} = \mathcal{C}/\mathcal{C}_{free} \tag{3.3}$$

The **obstacle configuration space region** is the union of the obstacle configuration spaces:

$$\mathcal{CO} = \cup\mathcal{CO} \tag{3.4}$$

Then the compliment of the configuration obstacle space region is the **free configuration space**, $\mathcal{C}_{free}$:

$$\mathcal{C}_{free} = = \mathcal{W} \setminus \cup \cup_{i=1}^{i=n} C_{obs} \tag{3.5}$$

The motion planning problem is to find a path in $\mathcal{C}_{free}$ from a starting location q_0 to another location q_1.

Definition 3.1

A **path** is a continuous function c, which maps a parameter of path s to a curve in $\mathcal{C}$:

$$c : [0, 1] \rightarrow \mathcal{C} \tag{3.6}$$

such that $c(0) = q_0$, $c(1) = q_1$ and $c(s) \in \mathcal{C}$ for all $s \in [0, 1]$.

If $c(0)$ and $c(1)$ belong to the same $\mathcal{C}_{free}$, then a continuous function c is called a **free path**. When the robot touches the obstacle, the space that represents this configuration is referred to as **contact space**, $\mathcal{C}_{contact}$.

3.2.5 Configuration space representation

The problem of motion planning is described in the real world however, it is actually in an imaginary space. Because the configuration space transforms the problem of an imaginary shape object into a point. Then finding likely motions of that point in the configuration space, depends on the connectivity of the configuration of the free space. Roadmap, Cell decomposition, Sampling-based and Artificial Potential fields are methods that represent the connectivity of the free space in the configuration space in different forms. There are three prominent map concepts:

1) Topological concept,
2) Geometric concept,
3) The Grid concept.

1) The topological concept is about representing the environments in a graph structure, where nodes represent locations or objects, and the edges represent the relationship between the nodes.

2) The geometric concept is about using geometric primitives for representing the environment, by mapping the primitive's parameters with the sensors observation.

3) The grid concept is where the configuration is decomposed into a grid of cells with a predefined shape or size which are dominantly square or rectancular (Coenen & Steinbuch, 2012).

3.3 Roadmap

The roadmap is a type of a map in topology, it is a graph-like structure with nodes and edges. The nodes represent the object or location whilst the edges represent the relationship between two nodes. A roadmap can also be described as the network ofcurves in the free configuration space. An example of a roadmap is given in Figure 3.3.

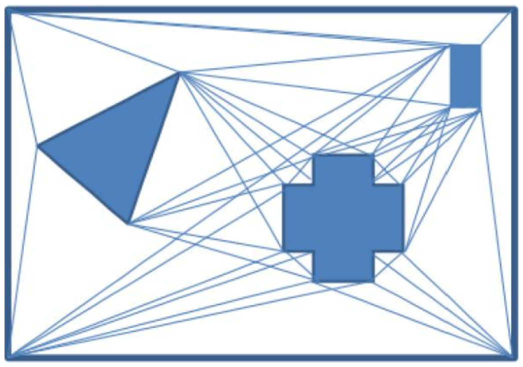


Figure 3. 3 Representation of an environment by the roadmap method

Definition3.2

An environment is a roadmap, R, if for all q_i and q_g in $\mathcal{C}_{\text{free}}$ that can be connected by a path, the following hold:

1. there is a path from the $q_i \in \mathcal{C}_{\text{free}}$ to some $q'_{\text{i}} \in$ R
2. there is a path from some $q'_{\text{g}} \in$ R to $q_g \in \mathcal{C}_{\text{free}}$
3. there exists a path in R between q'_{i} and q'_{g}

Some of the types of roadmaps are:

a. Deformation Retract is a function that maps a continuously shrinking or retracting a space into a subspace, this is similar to when a free space eroded into a skeleton shape, whilst this skeleton can be used for robot path planning.

b. Visibility graph technique is a non-directed graph, where the nodes corresponds to the vertices, and edges correspond to the edges in the polygon (Coenen & Steinbuch, 2012),

c. Retract-like structure is the union of one-dimensional structures (Choset Howie et al., 2005),

d. Piecewise retract is a graph for edge operating in the plane and

e. Silhouette's approaches have been proved to be complete for a number of dimensions with arbitrary obstacle structures. A brief description of Silhouette method can be found in (Choset Howie et al., 2005).

In this study, we restrict our attention to Deformation Retract.

3.3.1 Deformation Retract

Deformation Retract can simply be described by an analogy of a dissolving doughnut-shaped candy into a ring. The resulting ring is a subspace of the topological structure (space) of the candy. This ring that looks like a skeleton can be used for planning the robot's motion. We can deduct from this analogy that the doughnut-shaped candy to be the free space $\mathcal{C}_{\text{free}}$, and the ring or skeleton to be taking for a Deformation Retract.

For a set X, a retraction is a continuous function $f : X \rightarrow A$ such that $A \subset X$,

and $f(a) = a$ for all $a \in A$.

Then, the subset $A \subset X$ is the retract of X which implies that the dimension of X is greater than the dimension of A. Deformation is about space stretching or shrinking but not crushing or cutting, and retraction implies the continuous function of a space to a subspace. Deformation retract basic idea is to continuously squezing a space onto a space. So, a deformation retract is a geometrical structure obtained from a process of shrinking space to a subspace. It is the same as to eroding a free space into a subspace shaped like a skeleton, and this skeleton can be used for robot motion planning. However, the such a skeleton is mostly constructed by Voronoi diagram of free space.

Both the Generalized Voronoi Diagram and Deformation Retract are roadmap-based methods. Roadmap-based methods have two properties, the accessibility and the connectivity. However, the properties of both Deformation Retract and the GVD are completely the same because they can all be used to construct roadmap, both GVD and deformation retract follow from the connectivity and accessibility property of roamap method.

Characteristics of Deformation retract are:

Complete: the retraction technique is complete for any kind of search algorithm used for the roadmap because there exists always a path.

Optimal: The distance to obstacles and the computation cost of the diagram that allows maximum clearance from obstacles.

Complexity: the computation of the generalized Voronoi diagram depends on the number of edges, the more the number of edges the more point sites it involves.

From the point of view of algebraic topology, which is inspired by the work of (Rahul, 2016):

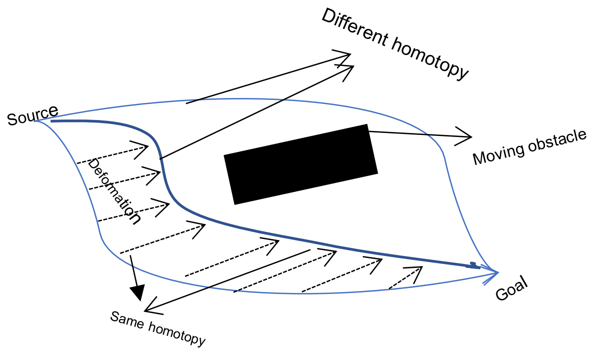


Figure 3. 4 Example of a Deformation

A retraction is called a deformation retraction if there is a continuous function, $\mathcal{R}$: [0, 1] x $X \rightarrow X$ such that $\mathcal{R}(0, .)$ is the identity map on X and $\mathcal{R}(1, .) = r$

Therefore, whenever there is a deformation retraction from X to Y, then we call Y a deformation retract of X.

If $\mathcal{C}_{\text{free}}$ describes the obstacle free space in an environment, the retraction function say, $\mathcal{R}$ builds a continuous subset where there is a path from starting point $\mathcal{S}$ and end point $\mathcal{G}$.

However, the problem of path planning for a robot is to find a path say, p from the source $\mathcal{S}$, to the goal $\mathcal{G}$.

With the prior information about $\mathcal{S}$ and $\mathcal{G} \in \mathcal{C}_{\text{free}}$.

Therefore the general state in the robot path is $p(s)$,

if $p : [0, 1] \rightarrow \mathcal{C}_{\text{free}}, \; p(0) = \mathcal{S}, \; p(1) = \mathcal{G}$.

From Figure 3.4, there are two paths of the same homotopic group, because one of the path can be deformed to the other with multiple small deformations that will lead to collision-free paths.

Let $\mathcal{p}_1 : [0, 1] \to \mathcal{C}_{\text{free}}$ and $\mathcal{p}_2 : [0, 1] \to \mathcal{C}_{\text{free}}$ be two paths,

with $\mathcal{p}_1(0) = \mathcal{p}_2(0) = \mathcal{S}$ and $\mathcal{p}_2(1) = \mathcal{p}_2(1) = \mathcal{G}$.

A deformation can be said to be:

$\mathcal{F} : \mathcal{F}_0 = \mathcal{p}_1, \ \mathcal{F}_1 = \mathcal{p}_2, \ \mathcal{F}_0 = \mathcal{S}, \ \mathcal{F}_1 = \mathcal{G}$

Therefore, two paths are said to be in the same homotopic group if such a continuous mapping $\mathcal{F}$ exists from the first path $\mathcal{F}_0 = \mathcal{p}_1$ to the second path

$\mathcal{F}_1 = \mathcal{p}_2$ and such that all intermediate paths $\mathcal{p}_t$ are collision-free from source to goal.

Definition 3.3

A Voronoi Diagram is the partitioning of a plane with n points into convex polygons such that each polygon consists of exactly one generating point and every point in a particular polygon is closer to its generating point than to any other.

Let $\mathcal{S}$ denote a set of point sites, n in the plane. i.e. = x, y, ..

Given two points $x = (x_1, x_2)$ and $a = (a_1, a_2)$, then let the Euclidean distance between the x and a be $d(x, a)$

$$= \sqrt[2]{(x_1 - a_1)^2 + (x_2 - a_2)^2} \tag{3.7}$$

Then, for $x, y \in \mathcal{S}$

let $P(x, y) = \left\{a \mid d(x, a) = d(y, a)\right\}$ be a perpendicular bisector of x and y.

$\mathcal{B}(x_1, x_2)$ is the perpendicular line through the segment, $\overline{x_1 x_2}$ that divides the half-plane

$Q(x, y) = \left\{a \mid d(x, a) < d(y, a)\right\}$ containing x from the half-plane $Q(y, x)$

containing y. Therefore, the Voronoi region of x with respect to $\mathcal{S}$ is written as,

$$\mathcal{VR}(x, \mathcal{S}) = \cap_{y \in \mathcal{S}, y \neq x} Q(x, y) \tag{3.8}$$

Then, the Voronoi region of $\mathcal{S}$ is now written as,

$$\mathcal{V}(\mathcal{S}) = \bigcup_{x,y\in\mathcal{S},x\neq y} \overline{\mathcal{VR}(x,\mathcal{S})} \cap \overline{\mathcal{VR}(y,\mathcal{S})} \tag{3.9}$$

a. Construction of Voronoi Diagram

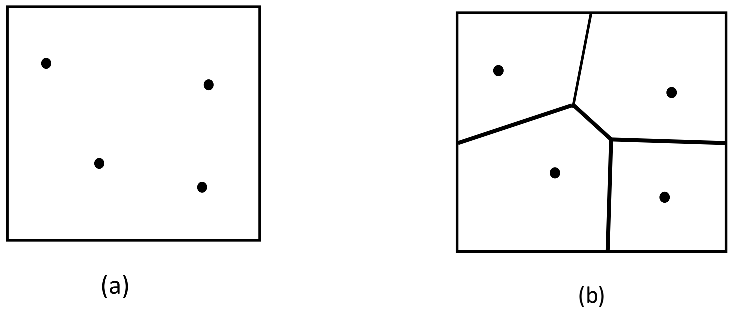


Figure 3. 5 Construction of Voronoi Diagram

Considering Figure 3.5(a), it shows four locations/points in an area/plane and Figure 3.5 (b) shows where these areas are divided into regions. Therefore, each location has a region, that consist of the area for which that location is the closest.

(a) Depicts a collection of points in a plane

(b) Divides the plane into regions, one region per generator and each region consist of those points that closer to the generator of the region than to any other generator.

(c) To construct a Voronoi Diagram, if we consider two points $\mathcal{X}$ and $\mathcal{Y}$ Figure 3.6, the region for point $\mathcal{X}$ is all the points closer to $\mathcal{X}$ than $\mathcal{Y}$.

The half-plane is shown in Figure 3.6 C(i) whilst the boundary is the perpendicular bisector of the edge $\overline{xy}$. Adding another point in Figure 3.6 C (ii) say $\mathcal{Z}$, we will get the region of $\mathcal{X}$ that consists of the points that are both closer to $\mathcal{X}$ than $\mathcal{Y}$, and also closer to $\mathcal{X}$ than $\mathcal{Z}$. And these intersections are $\mathcal{XY}$ and$\mathcal{XZ}$ half-planes. This goes on to produce the final diagram called the Voronoi Diagram in Figure 3.6 C(iii) that shows three perpendicular bisectors by deleting all the unnecessary parts.

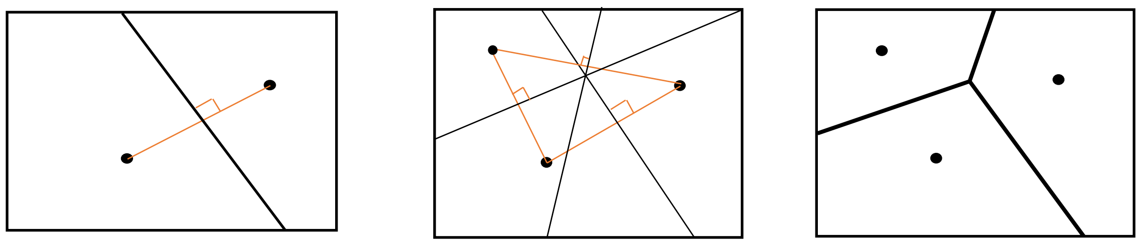


Figure 3. 6 Construction of VD

An **edge** of a Voronoi Diagram is the intersection of two Voronoi regions that is equidistant to the nearest sites. A **Voronoi cell** is the set of all points closer to a fixed site than any other site.

A **Voronoi vertex** is the intersection of at least three Voronoi edges, so a Voronoi vertex is equidistant from at least three sites.

Some Features of Voronoi Diagrams are:

1) the Voronoi Diagram of *n-1* parallel lines forms n cells if points Pi are collinear.

2) the edges of the Voronoi Diagram are segments if points P are not collinear.

3) Voronoi cells are convex.

Like the Voronoi Diagram, there is an important geometric structure called Delaunay triangulation. It is also referred to as the dual structure of the Voronoi diagram. The Delaunay triangulation is constructed by drawing lines between two sites whose Voronoi regions have the same edge. It also divides the convex hull into triangles.

There are numerous areas of application of the Voronoi Diagram (Mark, Maxim, Ming & Dinesh, 2001)

1. The Generalized Voronoi Diagram of the map (workspace) is used to bias sample generation in a randomized planner.

2. The path found from the Generalized Voronoi Diagram of the workspace is also used to provide intermediate points to serve as temporary attractive wells for a potential field planner.
3. The VD of the configuration can be searched simply for a path once the source and the target are connected.

In this study, the third application was used to generate the initial workspace path.

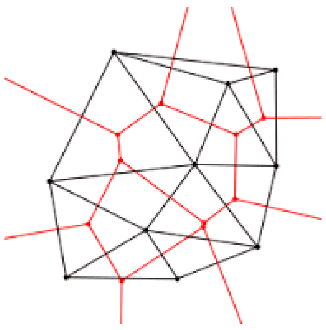
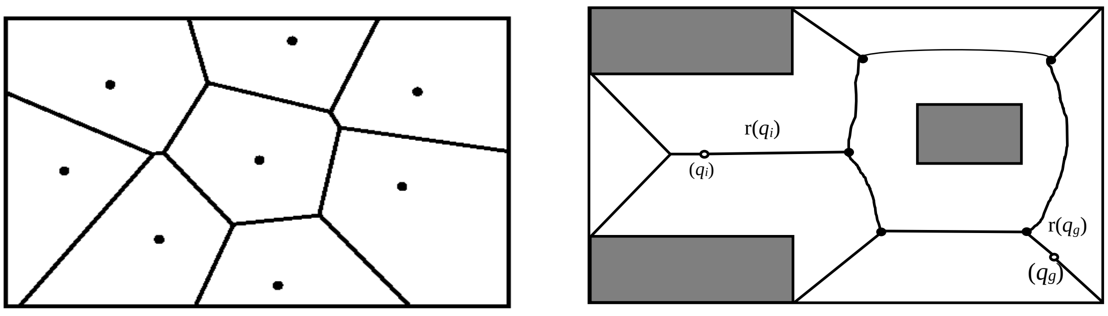


Figure 3.7 Sample of Delaunay triangulation[4]

b. Generalized Voronoi Diagram

Modelling obstacles as points, a Voronoi diagram is used to model the configuration space of a robot. In this model, the configuration space is the set of collection free spaces and the configuration space occupied $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{obs}$ respectively. A Voronoi diagram will help a robot to avoid obstacles with maximum clearance and a fast search is obtained because of the above assumption. However, in real life, obstacles are not points, but have shapes thus, generalized Voronoi diagrams were introduced by replacing points with objects. Generalized Voronoi diagrams (GVDs) can be used in high-dimensional spaces. (La Valle, 2006) suggested that the number of edges in the GVD would determine the computational complexity of GVD which is in $O(ElogE)$ time. Despite having straight line segments, it also contains arcs which make it visible. Here

4. https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Delaunay_Voronoi.svg/200px-Delaunay_Voronoi.svg.png the product of the retractions of q_i and q_g i.e. r(q_i) and r(q_g) with segment(path) between q_i and q_g is the path.



(a) Voronoi diagram (b) *(Coenen & Steinbuch, 2012)*

Figure 3.8 Deformation retract from (a)VD and (b)GVD

From Figure 3.8(a), considering a finite set of nodes in the plane. Each node is a Voronoi site, and its corresponding Voronoi cell consists of all points whose distance to this site is not greater than their distance to any other site. The edges of the VD are the points in the plane which are equidistant to the two nearest sites and the Voronoi nodes are e the points equidistant to three (or more) sites. The diagram becomes a generalised Voronoi diagram (GVD) for higher order site geometries, Figure 3.8 (b) illustrates this. The higher order character may be seen in the diagram, which includes arcs in addition to straight line segments. In the case of a GVD, the path is the product of retractions q_i and q_g on R, respectively $r(q_{i)}$ and $r(q_{i)}$, and a path between them in R.

A Generalized Voronoi Diagram is a Voronoi Diagram where point obstacles are replaced by objects, which implies that instead of regions around points, regions around objects are taken into account as shown in Figure 3.8. Path planning using a generalized Voronoi Diagram is obtained by using the follwing steps:

1. moving away from the closest point until getting to GVD.

2. navigate through the two equidistant towards the target.

3. then, from the GVD to the goal.

Since the Generalized Voronoi Diagram and Deformation Retract are both geometric structures that have the same properties in path planning it implies that both the Generalized Voronoi Diagram and Deformation Retract are examples of the roadmap approach.

3.3.2 Visibility Graph

Visibility graph is a method of inter-visible areas for set points and obstacles in the plane. Where vertex are locations and edge are connection between the locations. If no obstacle obstructs the line segment drawn to connect the nodes/locations, then an edge is drawn between the locations.

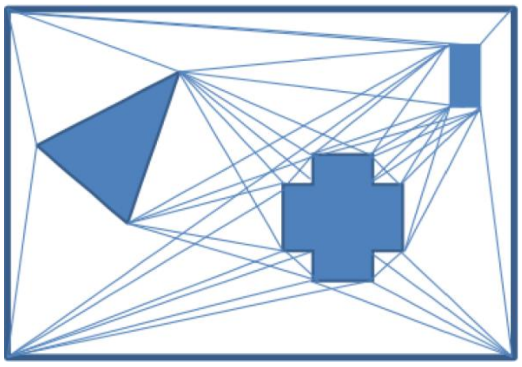


Figure 3.9 Visibility graph[5]

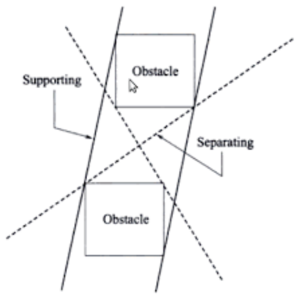


Figure 3.10 Supporting and separating line segments[6]

Using the Euclidean plane, the shortest path can be searched for with the visibility graph. Since the visibility graph consists of numerous edges, supporting lines and separating lines are used to reduce these numerous edges. A supporting line is a tangent to the obstacles where all obstacles belong to the same side of the line

5. https://media.springernature.com/full/springer-static/image/art%3A10.1007%2Fs42154-019-00081

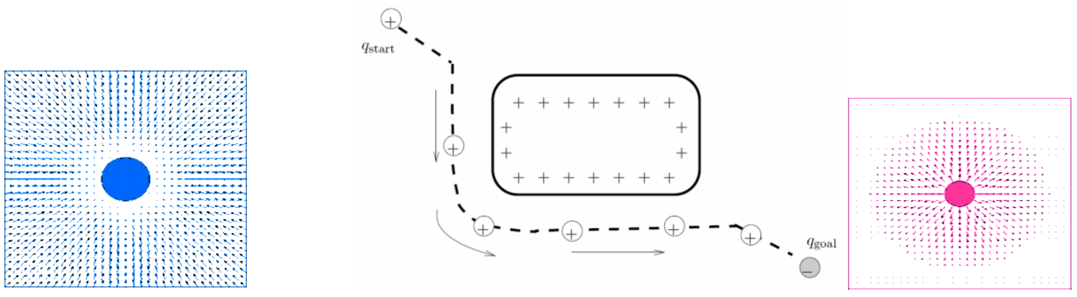
6. *https://image.slidesharecdn.com/visibilitygraphs-/visibility-graphs*

and a separating line is a tangent to two obstacles such that the obstacles belong to the opposites side of the line. Both the supporting line and separating line are used for the construction of what we called a *Reduced visibility graph*.

This implies that most of the lines in the visibility graph that are not part of the supporting and separating line will be deleted. This method is hardly used for problems with higher dimension (greater than 2D) because the method has a problem of optimality or completeness when applied to the high dimension. If a robot moves in a $\mathcal{W} = \mathbb{R}^3$ with a fixed translation, the paths generated may no longer be the shortest. However, a translating and rotating robot in $\mathcal{W} = \mathbb{R}^2$ with $\mathcal{C} = \mathbb{R}^2$ can be planned with a visibility graph method but it will be an incomplete solution. However, the problem of visibility graphs is that the generated paths touch or move too close to the objects' vertices and the edges. This may expose the robot to risk. Therefore, it reduces the chances of using of the method. However, Voronoi diagrams was designed to address this issue.

3.4 Artificial Potential fields

Researchers have shown that the Artificial Potential Field, APF method is mostly used because of its simplicity and mathematical sophistication however, it is good for static environment. The concept of the Artificial Potential field (APF) approach can be likened to the electric charge concept. In this method, the robot's environment (workspace) contains artificial forces, where obstacles are assigned with repulsive forces and goal point with the attractive force. The idea involves two types of forces; the attractive force generated by the goals and the repulsive force generated by the obstacles.



(a) Attractive Potential Field

(b) Concept of Potential field

Repulsive potential field

(c)

Figure 3.11 Artificial Potential field Method[7]

This method is good for implementation in static environments where both obstacles and the goal are stationary. However, in real-life scenarios where obstacles and environment are dynamic, the Artificial potential fields method is faced with local minima problem. Due to this, the Artificial potential field approach is not considered to be effective. But It is an efficient algorithm if compared to some other approaches, due to its simplicity, high safety and mathematical sophistication. However, it can applied for real-time scenarios because of low computation time (Sabudin, Omar, & Che Ku Melor, 2016). It is also applicable in workspaces greater than 2D. Another great advantage of the APF approach is that the robot still maintains its direction towards the goal even if there are changes in the environment. It models the robot as a point in a potential field U, which is the combination of attractive force used for goal attraction and the repulsive force used for obstacle repulsion. The Artificial potential field method is a local method and one of the features is iteration.

3.4.1 Attractive Potential

The sum of the attractive and repulsive forces is the artificial potential function, U.

$$U(q) = U_{att}(q) + U_{rep}(q) \tag{3.10}$$

7. *https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf*

where $U_{att}(q)$ is the function that attracts the robot towards the goal and $U_{rep}(q)$ is the repulsive function that repels the robot from the obstacle.

The sums of the two negative gradients vectors are,

$$\vec{F}_{att}(q) = -\vec{\nabla} Uatt(q) \text{ and} \tag{3.11}$$

$$\vec{F}_{rep} = -\vec{\nabla} rep(q) \tag{3.12}$$

These are the attractive and repulsive forces respectively.

A parabolic function is proportional to the square of the Euclidean distance $d(q, q_g)$ to the goal:

$$U_{att}(q) = \frac{1}{2}\zeta d^2(q, q_g) \tag{3.13}$$

where ζ is a positive scaling factor, a parameter use to scale the effect of the attractive potential, whilst $d^2(q, q_g)$ is the distance criteria that is selected as the Euclidean distance, $d(q, q_g)$. When attracting force $\nabla U_{att}(q)$ converges linearly to zero, the robot approaches the goal q_g and tends to infinity when $d(q, q_g)$ moves away from q_g.

$$\vec{F}att(q) = -\vec{\nabla} Uatt(q) \tag{3.14}$$

$$= -\zeta d(q) \nabla \mathrm{d(q)}$$

$$= -\zeta(q - q_g) \tag{3.15}$$

When there is an increase in velocity as a result of the q_i being far from the q_g, a combination of both quadratic and conic potentials is used. This implies to use the quadratic potential near goal and the conical potential farther away. Then the conic potential draws the robot from a far q_g whilst the quadratic potential also attracts the robot when it is close to q_g. The attractive potential is also called a conical function:

$$U_{att}(q) = \zeta d(q, q_g) \tag{3.16}$$

with equations 3.13 and 3.14, the attractive force is constant and indefinite in q_g. The basic idea of the attractive potential is that: $U_{att}(q)$ should increase as q moves away from q_g, for instance, potential energy increases as one moves away from the surface of the earth.

3.4.2 Repulsive Potential

The basic idea of the repulsive potential is to make sure that the robot moves away from obstacles. This is achieved by creating a barrier around the obstacles to prevent the robot from traversing through, whilst the motion of the robot is maintained without any influence when it is far away. Essentially, a robot $\mathcal{A}$ should be repelled from obstacles, not letting $\mathcal{A}$ hit an obstacle and if $\mathcal{A}$ is far from the obstacle, it is not desirable that an obstacle to affect $\mathcal{A}$'s motion. However, a potential function for convex obstacle that satisfies the two requirements is:

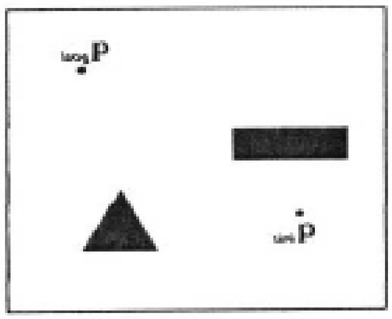
$$U_{rep,i}(q) = \begin{cases} \frac{\zeta_{r,i}}{2}\left(\frac{1}{d_{i(q)}} - \frac{1}{d_{0,i}}\right)^2 & for\ d_i\,(q) \le d_{0,i} \\ 0 & for\ d_i\,(q) > d_{0,i} \end{cases} \tag{3.17}$$

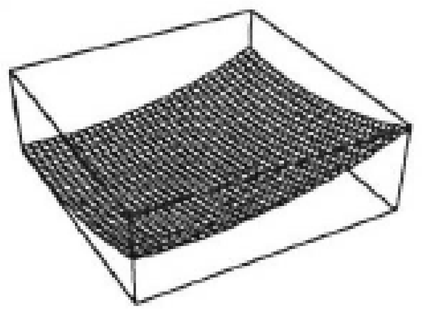
Where $\zeta_{r,i}$ = the positive scaling factor,

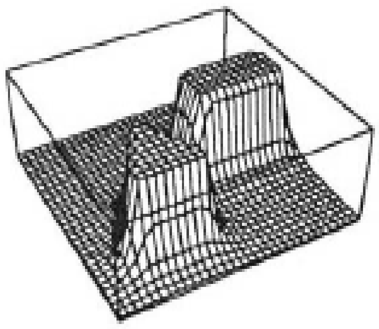
$d_{i(q)} = min_{q' \in CO} d_{(q,q')}$ = the minimal distance from obstacles to q:

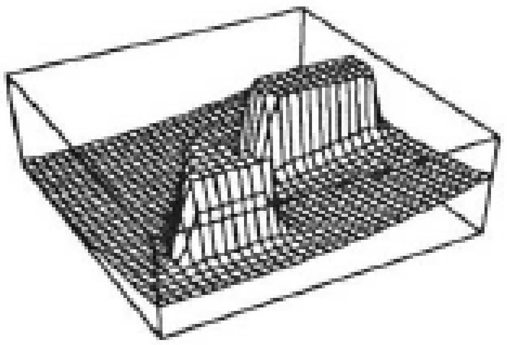
$d_{0,i}$ is the positive constant for the range at which obstacle influence the robot motion (i.e. the distance of influence): Then, as q approaches CO, $U_{rep,i}(q)$ approaches ∞. $\quad U_{rep}(q) = \sum_{i=1}^{n_0} U_{rep,i}(q)$ (3.18)

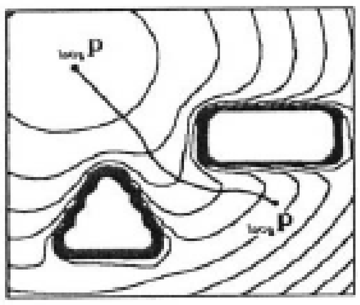
The above equation is the sum of the individual potentials associated with the convex components of CO. Figure 3.12 demonstrates how the potential field method is used to solve the motion planning problem.


(a)Two Obstacles


(b) U_{att} is attractive potential towards the goal


(c) Repulsive potential U_{rep} over the obstacles


(d) Total potℝ2ential function,


(e) The equipotential contours of the total potential and a path that is generated by the following the gradient of the combined potential.

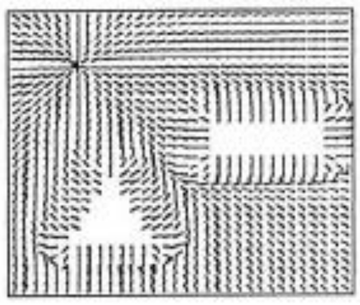

(f) The gradient vector orientation over the field $\vec{F}(c)$

Figure 3. 12 An example of motion planning problem by APF[8]

3.4.3 Local Minima

The key shortcoming of the potential field method is the problem of local minima. That is, when the attractive potential and the repulsive potential are equal or

8. 16-735, Howie Choset, with slides from Ji Yeong Lee, G. D. Hager and Z Dodds

nearly equal in the opposite way, the potential field is zero, and the robot is trapped. Therefore, no path is found and so the process is incomplete. This problem of local minima plagues all gradient (gradient of potential field function) descent algorithms. Gradient descent algorithm is a method used to optimize the problem by first starting from the current configuration, then moving a bit towards the opposite direction to the gradient which will produce a configuration different from the current configuration (Choset Howie et al., 2005). This is continued iteratively until the gradient approaches zero. Then a scalar $\propto (i)$ is used as a determinant factor for the step size at the iteration. This scalar $\propto (i)$ must be set to be smaller than the current distance to obstacle, $\propto (i)$ should be so small that it will prevent robot from collision with obstacle also so as not to overshoot the goal that may lead to high time in computation. However, when the robot reaches a point where the gradient descent converges, where $\vec{\nabla} U\big(q(i)\big) = 0$ such point $q(i)$ is a critical point of *i*. This point can be a minimum, a maximum or a saddle point. However, most potential field methods are incomplete, however, they are computationally fast.

3.4.4 Navigation Function

The navigation function is a potential field that is specified by functions that are free of local minima. And a function is referred to as a navigation function if it is infinitely differentiable with one minimum only at the goal configuration. This implies creating an artificial potential field that will have unique minima, so that when the robot moves, it is garranteed that it will reach the goal.

Definition 3.4

A function φ: $Q_{free} \rightarrow [0,1]$ is called a navigation function if it satisfies the following:

i. φ is smooth (or at least C^2)
ii. it has a unique minimum at q_g
iii. it is uniformly maximal on the boundary of the free space
iv. it is Morse (i.e. if every critical point is isolated).

3.5 Cell decomposition

The free space $\mathcal{C}$ can be represented using the cell decomposition method. The basic concept of this approach is that giving a point as the robot in a free space $\mathcal{C}$, first subdivide the free space into cells and make sure that the cells are not intersecting. Then construct a graph called *adjacency graph* since the cells have an adjacent relationship between them. These cells are the vertices of the graph whilst the edges are derived from the cells of common boundary. The planner then determines if the start and goal are contained in the cells before searching for a path in the adjacency graph. The cell decomposition method has some advantages over other methods of path planning because it can be used to achieve coverage since every cell has a simple shape that can be covered with a simple motion. Exact decomposition and approximate cell decomposition methods are the categories of cell decomposition method.

3.5.1 Exact cell decomposition method

The basic idea of Exact decomposition is that the shape and size of the cells depend on the dimension of the environment with the geometrical structure of the objects in the space. Using the dimension of the workspace and the location of the obstacle, there exist methods to decompose robot free space. to subdivide the robot's free space by decomposing free space into trapezoidal and triangular cells that are bounded with polygons, by drawing a non-overlapping line from each node of polygon internally to the vertex outside the boundary. A good example of these methods is the vertical cell decomposition or trapezoidal decomposition and also characterized as complete.

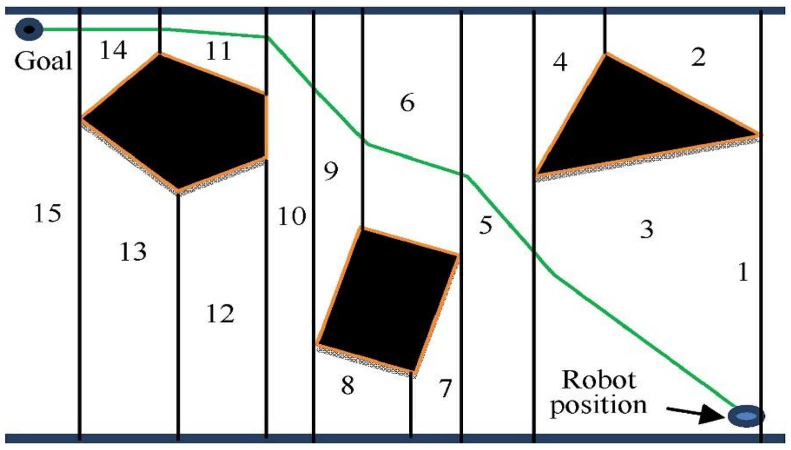


Figure 3.13 Vertical cell decomposition or trapezoidal decomposition [9]

(Patle, Pandey, Parhi, & Jagadeesh, 2019b)

3.5.2 Approximate cell decomposition

This type of cell decomposition is different from the exact cell decomposition because it has a recursive approach to subdividing the cells repeatedly until either the cells are completely within the obstacle region or an arbitrary limit resolution is reached. It is also referred to as quad-tree decomposition since every cell is subdivided into four smaller cells repeatedly until a free continuous path is discovered (Choset Howie et al., 2005).

3.6 Sampling-based approach

The basic concept of this approach is that rather than exhaustively exploring all possible spaces, a subset of the space is explored randomly and keeping track of the progress. The sampling-based method is very complex to use when the size of the configuration space increases. This method can also be described as searching for a collision-free path by sampling points. There are two categories of sampling-based method: the probabilistic roadmap (PRM) and the single-query planners (Coenen & Steinbuch, 2012).

9. *https://ars.els-cdn.com/content/image/1-s2.0-S2214914718305130-gr3.jpg*

3.6.1 Probabilistic Roadmap

A probabilistic roadmap (PRM) is a sample-based concept that also constructs a roadmap that results also into a network of edges and nodes in free space. The network between the nodes is used to search for a free path between the start and goal. Every sample is a node in the roadmap and robot configuration is represented by the sample. There are two phases involved in the construction of the probabilistic roadmap: the learning phase and the query phase.

a. Learning phase

The learning phase is where the configuration space is sampled and confirmed that the random configuration exists that is also free from collision, then the random configuration is added to the roadmap. Using a predefined distance, the random configuration space can be connected to a near sample using a straight line. This iteration continues until a predefined size of sampled nodes is achieved.

b. Query phase

This is where the connection between the initial configuration and goal configuration to the roadmap is done. However, a path search is only successful if there is a connection between the roadmap and both the initial and goal configurations. Otherwise, the planner returns to the query phase or failure if there is no success in trying to improve on the roadmap.

3.6.2 Single-query planner

The basic idea of a single-query planner is that a single path planning problem can be solved quickly without any pre-processing. The subset of free space relevant to the path planning problem is explored instead of representing exhaustively the free space of the roadmap. This is done with the use of a tree structure. In the probabilistic roadmap method, the random configuration is allowed to be added to the roadmap if it belongs to the configuration free space. However, in a single-query planner, it can only be added if there is a connection to the current configuration.

3.7 Search Algorithm

The space representation methods: Roadmap, APF, Cell Decomposition and Sampling-based method discussed earlier have solutions to path planning problems that need to be solved. These approaches convert the repeated process of searching for a path in free space into a process of searching a graph. The collection of nodes and edges is referred to as a graph, and if an edge connects two nodes it implies that there is a relationship between these two nodes called adjacent relationship.

This connection between two nodes is characterized by an 'adjacent relation'. There are also two types of graph: the 'directed graph' and 'undirected graph'. A directed graph is an environment where a robot can move in one direction only but if the robot can move in both directions on the edge it is referred to as an undirected graph. However, a graph is just like a tree, however, what differentiates a graph search from a tree search is that in graph search the track of the nodes visited are kept in an explored set whilst a tree search does not take note of the visited nodes and this creates unnecessary search loop. More about tree and graph can be found in Data structure[10]. For this work, we are going to consider the graph search approach which is in the next chapter. Graph Search can be categorized into the followings: Uniformed, Informed and Local search.

3.7.1 Uninformed search

Uninformed search is the type of algorithm that allows traversing through the graph without any prior information about the goal node and some examples of this search are Breadth-First search, Depth-first search and Dijkstra’s algorithm (Pathak, Patel, & Rami, 2018).

a. Breadth-first search

The idea behind this search is that the source or root node is first traversed followed by traversing all the nodes that are directly connected to the source node then traversing to the next layer of nodes. Every node is traversed and checked

10. *https://www.techgeekbuzz.com*

whether it is the goal node and if not it repeats the process until the goal node is discovered. All nodes of every layer are traversed before the nodes of the next layer is traversed, which implies first explore all the nodes of the current layer, then move to the next layer.

b. Depth-First search

The concept of the depth-first search is backtracking by a recursive algorithm. It traverses the deepest node of the tree. All the nodes are visited until the unvisited are visited then the next frontier is checked before backtracking.

c. Dijkstra' algorithm

This is the algorithm to find the shortest path from the root node to every other node in the graph. Both BFS, DFS and Dijkstra’s algorithm use the same function *f(n)* that determines the cost of expansion of node

$$f(n) = g(n)$$

however, in the case of breadth-first search, *g(n)* is determined by First-In-First-Out, (FIFO) queue whilst in Depth-first search the *g(n)* is determined by Last-In-First-Out, (LIFO) for the expansion of the stack.

3.7.2 Informed Search

This is a search that involves prior knowledge of the goal node. It requires a heuristic cost that will determine which node to explore next. A Euclidean distance to the goal node can be used as an example of a heuristic approach. This can also be used as a catalyst for a search. However, does not guarantee that the path obtained is the shortest. Two categories of informed search are *Greedy Best-First* and *A* search.*

a. Greedy Best-First search

In this search, the node closes the goal node is traversed first. The greedy best-first does not apply *g(n)* to determine the expansion of nodes, it uses only the heuristic to quickly found a path or solution. This search method cares less about the cost of the path but the cost to reach the goal node because even if the path length increases it keeps moving on.

***b.* A* Search**

This is another type of search for finding the shortest path. A* uses the combination of the cost function *g(n)* to reach a node and the cost function *h(n)* to get from the goal node. i.e. f(n) = *g(n)* + *h(n),* (Charalampous, Kostavelis, Amanatiadis, & Gasteratos, 2014).

c. D* Search

An extension of A* that addresses the problem of expensive re-planning when obstacles appear in the path of the robot, is known as D*. Unlike A*, D* starts from the goal vertex and can change the costs of parts of the path that include an obstacle. This allows D* to re-plan around an obstacle whilst maintaining most of the already calculated path.

3.7.3 Local search

The uninformed and informed search achieves their search by storing path(s) and also keeping a record of the visited node along the path. But local search does not bother about paths, instead of multiple paths it uses an existing node and traverse only to neighbouring nodes without keeping path(s) in memory.

3.8 Conclusions

The representation methods and search algorithms discussed in this chapter are purposely for the solution of path planning problems. The basic motion planning problem is to find a path for the robot from starting point to target without any collision, however, this path planning problem is now extended to path planning in a dynamic environment which is in the scope of this work. The focus is on motion planning with uncertainty and some constraints. To solve these extensions of the path planning problems both the representation method and the search algorithm can also be combined.

Chapter Four

4.0 Problem Definition and Method Used

4.1 Introduction

An environment is complex if there exist obstacles that are dynamic with uncertainties like open or closed doors. Most path planning techniques mentioned in Chapter Three are not effective in these kinds of environments. For instance, the Artificial potential field method results in the problem of local minima and this may occur quite often when there are several obstacles in the environment. However, among all the path planning techniques the roadmap-based technique is more effective and good for implementation because it is concise in representation i.e. the entire workspace is not needed to be discretized into small cells. A roadmap-based method called Generalized Voronoi Diagram (GVD) is applied in this work. Using the GVD, the robot is guaranteed a maximum clearance from the obstacles whilst moving through the environment. (Choset & Burdick, 1995) and (Choset & Burdick, 1996) works show that the generalized Voronoi Diagram has always been applied as a basis for motion planning algorithms for a long period.

Figure 4.1 shows an environment set up with different obstacles at different locations, having the robot starting position at the lower-left corner whilst the target at the upper right corner. The static obstacles are in pink colour and the moving obstacle in blue colour. This is a typical example of the proposed setting used for this work.

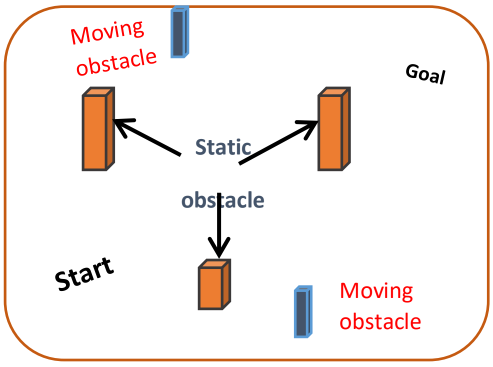


Figure 4.1 Proposed setting of the work.

However, Figure 4.2 is an example of a complex/dynamic environment where tables, walls, bins can be static obstacles and humans, robots are the moving obstacles in the workspace.

Figure 4.2 Train station, New York city[11]

4.2 Problem Definition

Assuming the robot is a circular object and with little information of the environment where the robot operates, i.e., workspace, $\mathcal{W}$. Let $\mathcal{W}$ be a subset of a two-dimensional (2D) plane. And if this workspace $\mathcal{W}$ contains polygonal obstacles, the problem now is, what is the minimal cost of path generation from where the robot is (starting point) to where the robot is asked to go (goal point).

To generate the path, it is assumed that the workspace contains two categories of objects: the obstacles (occupied space) and the free space. The obstacles are places where the robot cannot navigate through whilst the free space is where the

11. *https://traveltips.usatoday.com/new-york-train-routes*

robot can navigate. However, in this work, the workspace will be $\mathbb{R}^2$, for path generation purpose, the robot is modelled as a point, and the workspace as an arbitrary map which implies that the robot is reduced to a point whilst the workspace $\mathcal{W}$, is transformed to configuration space $\mathcal{C}$. This idea was presented by (Lozano-Perez, 1987).

The configuration space occupied by the obstacles is referred to as $\mathcal{C}_{obs}$ whilst the remaining part of the configuration space is free space where the robot is free to move is denoted by $\mathcal{C}_{free}$ (Latombe, 2012).

$$\mathcal{C}_{free} = \mathcal{W} \setminus \bigcup_{i=1}^{i=n} \mathcal{C}_{obs} \qquad (4.1)$$

4.3 Voronoi Diagram

The concept of Voronoi Diagram is to produce line segments that are equidistant to all points of the obstacle space whilst the meeting point of the line segments is referred to as nodes or vertices. Voronoi Diagram is a concept of computational geometry used to find the collision-free path and to build a map. Figures 4.3 shows a Voronoi Diagram, where a point in the yellow region is closer to another point in the same yellow region than any other point in another colour region, and Figure 4.4 shows a Generalized Voronoi Diagram of three polygonal obstacles. In Voronoi diagram, obstacles are assumed to be points so, VD is a graph with points and vertices. However, in a real world obstacles are not points, they are objects. Therefore, instead of regions around points, it is regions around objects which is GVD as seen in Figure 4.4.

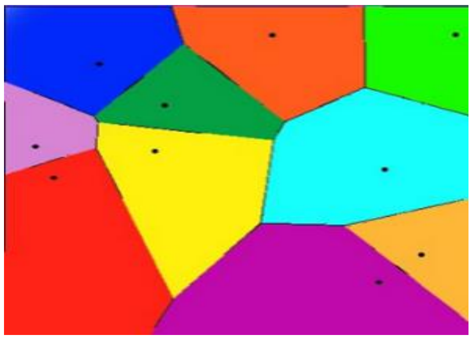


Figure 4.3 Example of a Voronoi Diagram[12]

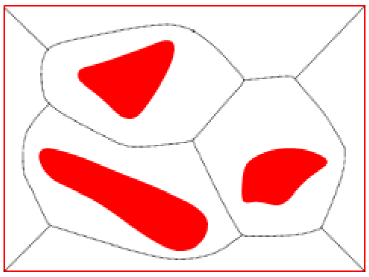


Figure 4.4 Voronoi Diagram (Nieuwenhuisen, Kamphuis, Mooijekind, & Overmars, 2004)

Let $d(\mathcal{P}_i, \mathcal{P}_j)$ be a distance from a point $\mathcal{P}_i = (x_i, y_i)$ to $\mathcal{P}_j = (x_j, y_j)$ in a plane, then

Let $\mathcal{P} = \{ \mathcal{P}_1, \mathcal{P}_2, \dots , \mathcal{P}_n\} \subset \Re^2$

be a set of points with the cartesian coordinates $(x_1, y_1), \dots, (x_n, y_n)$

where $2 < n < \infty$ and $\mathcal{P}_i \neq \mathcal{P}_j$ for $i \neq j$.

$$V(\mathcal{P}_i) = \{ x \in \Re^2 \mid d(x, \mathcal{P}_i) \leq d(x, \mathcal{P}_j) \text{ for } j \neq i\} \qquad (4.2)$$

We call $V(\mathcal{P}_i)$ the region Voronoi Diagram of $\mathcal{P}_i$.

Then, the $\mathcal{P}_i$ of $V(\mathcal{P}_i)$ is called the site or the generator of ith Voronoi polygon whilst the set $\mathcal{P} = \{ \mathcal{P}_1, \mathcal{P}_2, \dots , \mathcal{P}_n\}$ is called the generator set of Voronoi Diagram V.

$$V(\mathcal{P}) = \bigcup_{p_i \in p} v(p_i) \qquad (4.3)$$

$$= \bigcup_{p_i \in p} \{ x \in \Re^2 \mid d(x, \mathcal{P}i) \leq d(x, q) : \forall q \in (\mathcal{P} - \{\mathcal{P}i\}) \}$$

12. *en.wikipedia.org/wiki/Voronoi Diagram*

$$= \bigcup_{p_i \in p} \left[\bigcap_{q \in P - \{p_i\}} \{ X \in \Re^2 \mid d(x, p_i) \le d(x, q) \} \right] \qquad (4.4)$$

A Voronoi Diagram is a partitioning of a plane into regions based on the distance to points in a specific subset of the plane. The set of points (sites, or generators) is specified beforehand, and for each site, there is a corresponding region consisting of all points closer to that site than to any other. These regions are called **Voronoi cells**. The Voronoi Diagram of a set of points is dual to its **Delaunay triangulation**.

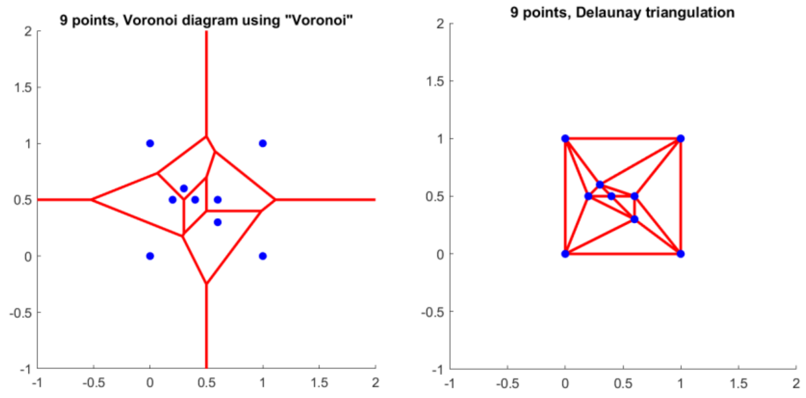


Figure 4.5 Voronoi Diagram V for 9 points and the Delaunay triagulation of the same VD Using 'voronoi' function in MATLAB

4.4 Generalized Voronoi Diagram

Since it is assumed that robot operates in a bounded and connected subset of free space $\mathcal{C}_{free}$,

therefore, this subset is also bounded by obstacles.

Mathematically, for every obstacle space $\mathcal{C}_{obs}$, the distance function:

$$d_i(x) = dist(C_{obs}, x) \qquad (4.5)$$

Then the Voronoi region is:

$$\mathcal{VR}_i = \left\{ x \,\middle|\, d_j(x) \le \ \forall j \ne i \right\} \qquad (4.6)$$

However, the Generalized Voronoi Diagram is the set of the Voronoi regions.

The Generalized Voronoi Diagram GVD basic concept in planning path for Robot is to maximize the distance from the robot to the obstacles. It consists of a point equidistant to two or more closest obstacles with straight and parabolic lines where obstacles are polygons.

Advantages of Voronoi Diagram over other methods

1. The Voronoi Diagram can be computed in $O(nlogn)$ time whilst it took $O(n^2)$ time for the fastest construction of the visibility graph.
2. Querying of a path in the Voronoi Diagram is faster than the querying in visibility graph since it has $O(n)$ edges.
3. It is difficult to construct the VD in higher dimensions or with non-polygonal subjects, approximation algorithms exist.
4. The VD is not good for heuristic, it is good for a known terrain where robot stays away from obstacles
5. Using robots that have a long sensor range, the Voronoi Diagram method has the advantage of execution over other obstacle avoidance techniques. In fact, following the Voronoi path results from maximizing the distance whilst maintaining equidistant from the surrounding objects, which can be done relatively easily with a good range finder.

4.5 The description of the proposed method

4.5.1 Part 1. Implementation of the method

This proposed method combines the GVD and Deformation Retracts. This is presented in a graphical user interface GUI. This section describes the new method's implementation. Figure 4.6 shows the GUI with three polygonal obstacles in a rectangular wall which is also considered to be an obstacle, with a starting point (green colour) and a goal point (red colour).

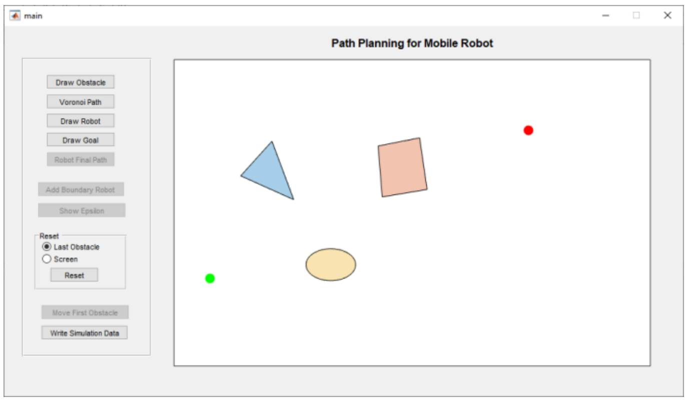


Figure 4.6 Representation of environment configuration

4.5.2 Roadmap generation

In 2D planar Euclidean space, when two Voronoi regions meet a Voronoi edge is obtained. And when two Voronoi edges meet we have a Voronoi vertex. The Voronoi edge can be a straight line or a curved segment. However, the straight line is the set of configuration that are closet to the same pair of obstacles' edges (edge/edge) or the same pair of obstacles' vertices (vertex/vertex) and a curved segment is the set of configurations that are closet to the same pair of an obstacles' edge and a vertex (edge/vertex).

To construct the Voronoi Diagram, the polygonal obstacles can be viewed as a set of line obstacles whilst each line obstacle can be also viewed as a set of point obstacles separated by a distance of $\mathcal{E}$. Then to draw the Voronoi Diagram in this configuration, the matlab function 'voronoi()' divides the polygons into some point sites using the parameter $\mathcal{E}$ to determine the distance between two consecutive point sites. If the distance between these consecutive point sites is small, there will be a greater number of vertices and this would lead to more spaced edges. Therefore, lesser $\mathcal{E}$ generates a smooth path or map but execution time will increase because of the increase in the number of points. Below are some examples of different samples of Voronoi Diagrams with different values of $\mathcal{E}$. Voronoi diagram for a line site can be generated by considering line as a linear array of point sites whilst the Voronoi diagran for polygonal ogject can be drawn by the polygon asa set of line segments.

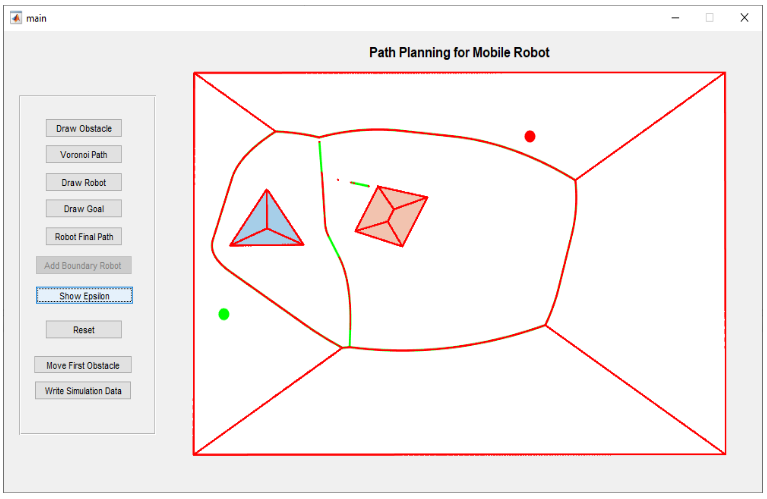


Figure 4.7 VD with $\mathcal{E}$ = 0.2

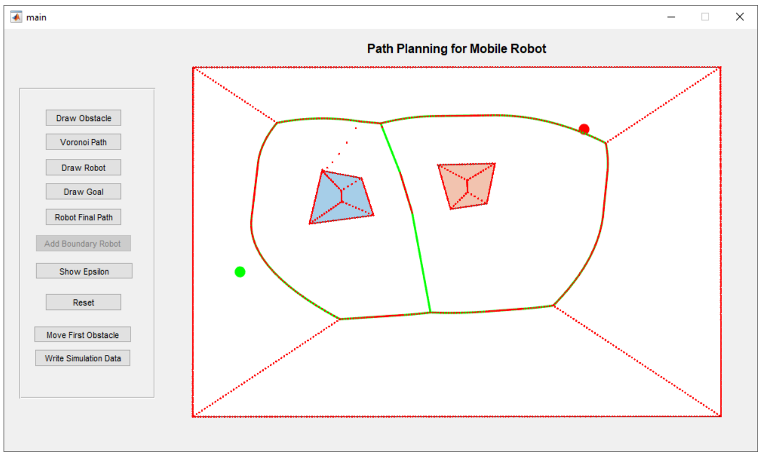


Figure 4.8 VD with $\mathcal{E}$ = 0.5

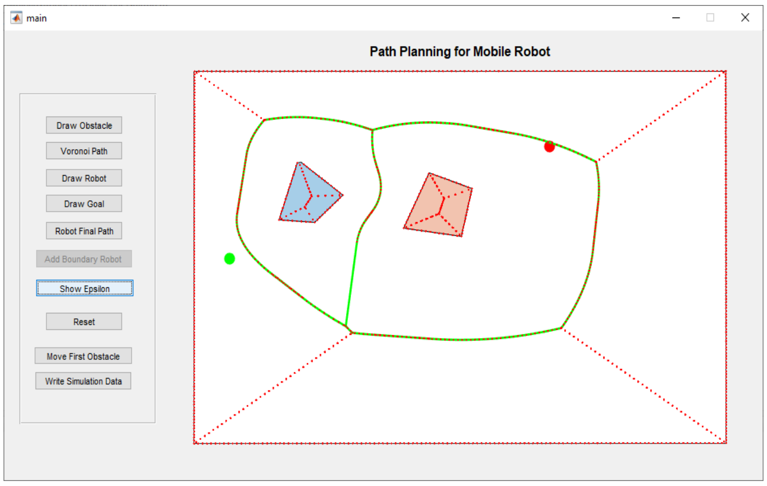


Figure 4.9 VD with $\mathcal{E}$ = 0.8

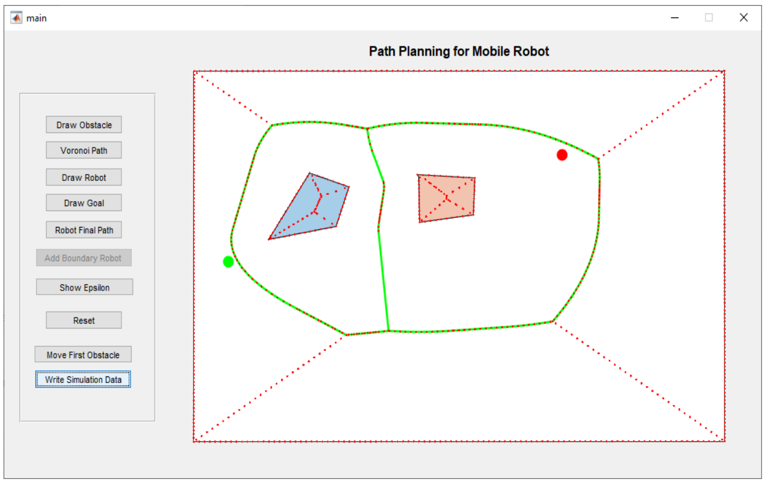


Figure 4.10 VD with $\mathcal{E}$ = 1

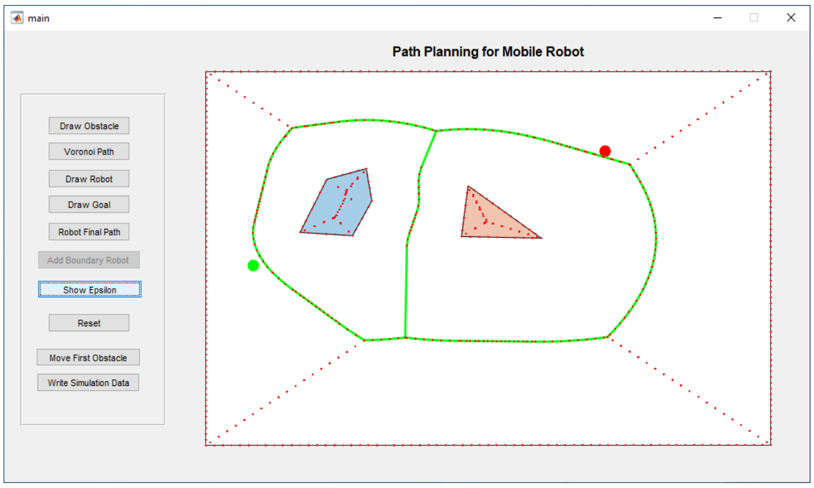


Figure 4.11 with $\mathcal{E}$ = 1.5

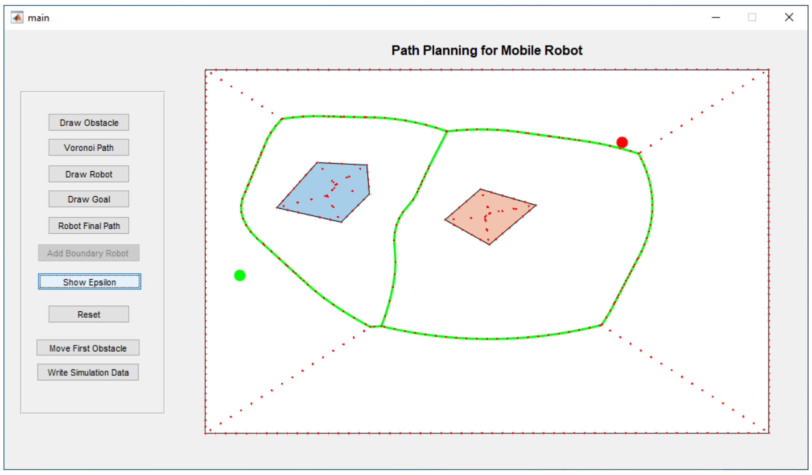


Figure 4.12 VD with $\mathcal{E}$ = 1.8

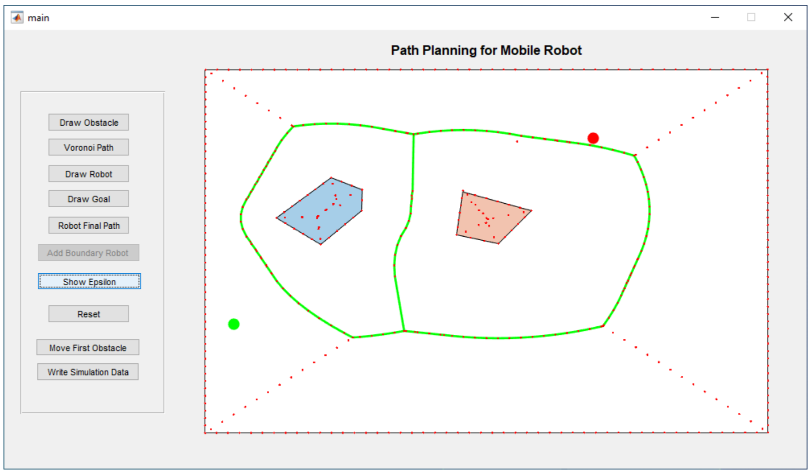


Figure 4.13 VD with $\mathcal{E}$ = 2

In Figures 4.7 to 4.13, it is shown that the higher the $\mathcal{E}$ the lesser the number of vertices and the lower the computation time for the construction of the Voronoi Diagram. It is also shown that there are set of points from the same objects and set of points from different objects that generate two different types of edges. However, it can also be observed that every Voronoi edge corresponds to two points and also a perpendicular bisector of the line to the points. The $\mathcal{E}$ = 1 is use to compute the Voronoi Diagram in order to reduce the number of vertices, and also lower computation time. Figure 4.10 shows Voronoi diagram for a given two obstacles configuration with $\mathcal{E}$ = 1. This generates two types of Voronoi edges: Voronoi edges formed by the same object's point sites and Voronoi edges generated by two separate object's point sites. Each Voronoi edge, on the other hand, corresponds to two point sites that are perpendicular to the segment connecting the points. The goal is to keep the second-category edges i.e. those formed from point sites of different objects whilst removing those that are formed by point sites of the same objects.   now Then, followed by the task of removing those edges that are generated by point sites of the obstacles whilst preserving those edges generated from two different objects. Since, Matlab does not directly provide information about the edges that relate to point sites. Then, a matlab function "*drawVoronoi*" is used as follows:

[Voro_Vertex, Voro_Cell] = drawVoronoi ([X_Total_points' Y_Total_points'])

Voro_Vertex holds an array of all Voronoi vertices, whilst Voro_Cell holds information about the edges that correspond to each point site. As a result, the information in Voro Cell can be used to separate the Voronoi edges of the second category. Figure 4.14 depicts the second category's separated Voronoi edges in green and this is the VD of the given obstacle configuration.

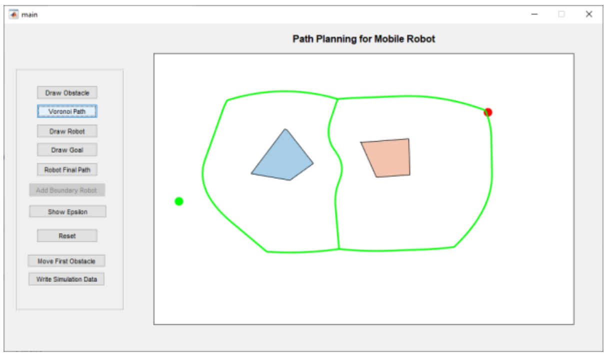


Figure 4.14 VD of a given obstacle configuration

To generate path, the Voronoi diagram for the environment needs to be computed to obtain the Voronoi edges which represent the maximum clearance between the nearest obstacles. Then, these associated Voronoi edges are followed to the goal. This implies that the series of Voronoi edges form the path for the robot to follow. However, whenever there is a change in the environment, the Voronoi diagram is deformed and the Voronoi edge is updated and new path is generated.

Next is to find the shortest collision-free path from the source to the goal. In some scenarios, both the starting point and goal point may not be on the Voronoi Diagram, the nearest vertex from the start and goal points are identified and referred to as *Start** and *Goal** respectively. Then, using Dijkstra' algorithm, the shortest path between the *Start** and *Goal** can be found. The final path from *Start to Goal* will be for instance, the combination of the paths *Start* (*Start* that do not lies on the VD) to *Start** (*Start** that lies on the VD), to *Goal** (*Goal** that lies on the VD), and to *Goal* (*Goal* that do not lies on the VD).
Then, the final path is shown in magenta colour in Figure 4.15.

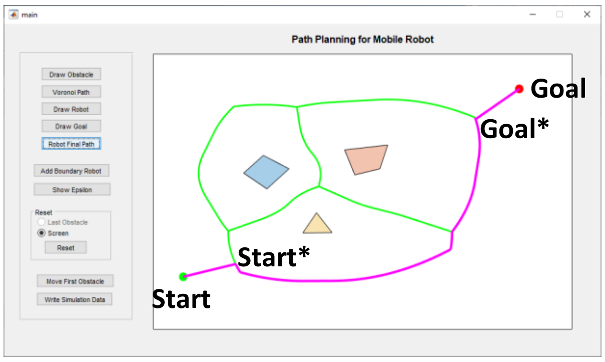


Figure 4.15 Final path (magenta) for robot

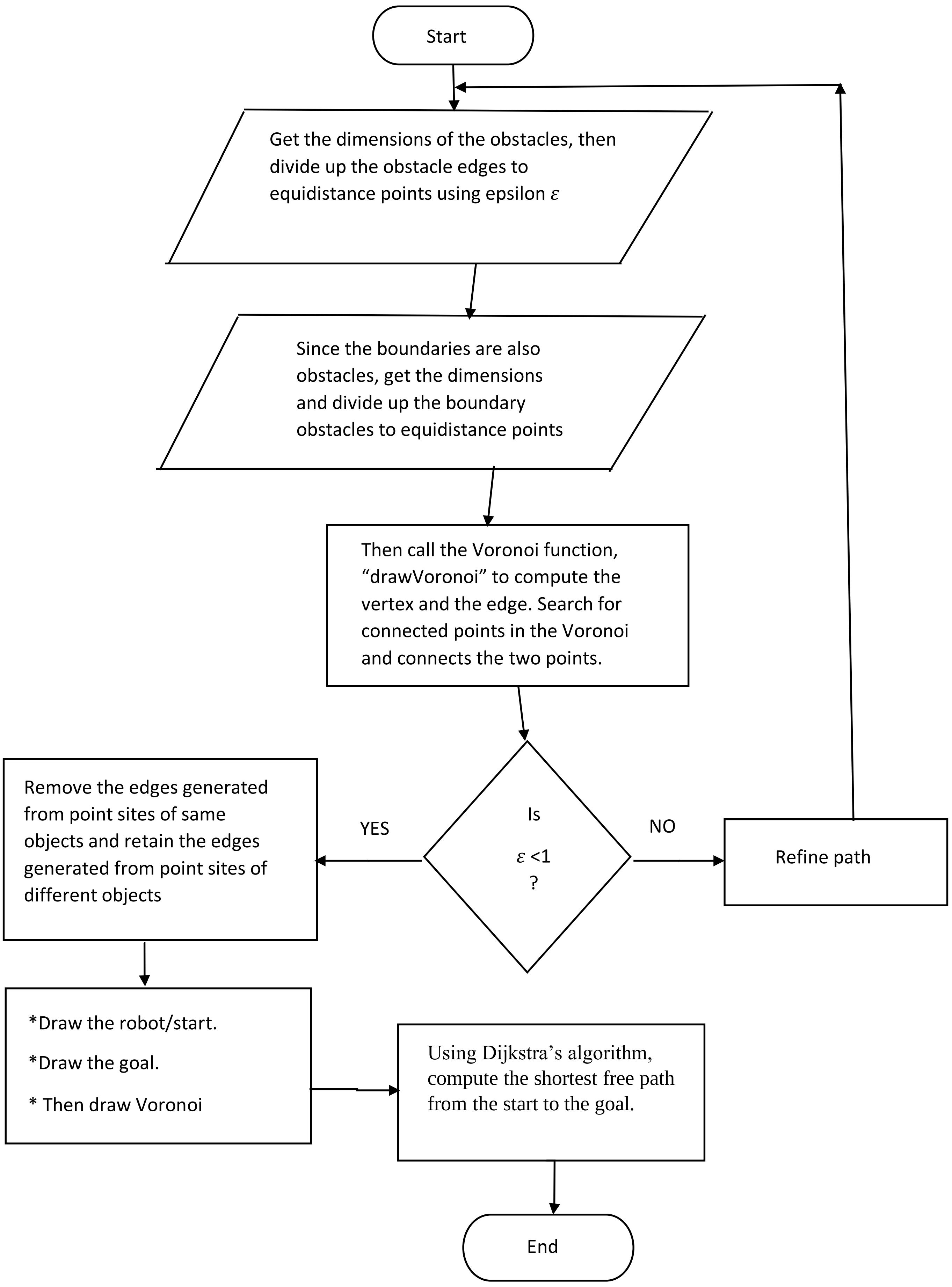

Start
Get the dimensions of the obstacles, then divide up the obstacle edges to equidistance points using epsilon ε
Since the boundaries are also obstacles, get the dimensions and divide up the boundary obstacles to equidistance points
Then call the Voronoi function, "drawVoronoi" to compute the vertex and the edge. Search for connected points in the Voronoi and connects the two points.
Is ε <1 ?
YES
NO
Refine path
Remove the edges generated from point sites of same objects and retain the edges generated from point sites of different objects
*Draw the robot/start.
*Draw the goal.
* Then draw Voronoi
Using Dijkstra's algorithm, compute the shortest free path from the start to the goal.
End

Figure 4.16 Flowchart of VD for the new system

In a real life situation, the proposed system would work for two different states; the execution and planning simultaneously. The environmental changes would be detected by sensing mechanisms and as soon as signals are received execution is done by updating the problem without stopping if updating is required. However, if no collision is anticipated, that is if the interference is insignificant, it would keep executing the path to avoid repetition of action by generating new path each time a change occurs.

Moreso, simulators are needed to drive this theoritical concept since many simulators have been implemented in Matlab, for instance (Corke & Khatib, 2011) presented a comprehensive set of *Matlab* and *simulink* scripts that deals with mobile robot navigation; motion planning, motion control, localization and mapping. Some of the functions of these simulators is to consider the physical characteristics of the robot i.e. size, weight and also speed that would definitely have impact on the performance evaluation.

4.6 Implementation of Deformable Voronoi Diagram

4.6.1 Deformation process

In a dynamic environment, when an obstacle moves or a new obstacle appears, a pre-planned path becomes inapplicable since the environment has changed, therefore the environment is deformed and updated to re-plan an alternative path for obstacle avoidance, which implies that every path in the configuration space must be continuously deformable to another path. This implies generating a new roadmap that will dynamically retract to capture the connectivity of the free space. The robot continues to follow the pre-planned path as long as it remains applicable unless the deformation of the environment due to the obstacle's motion.

4.6.2 Deformation distance

The procedure here is inspired by the work of (Yoshida & Kanehiro, 2011)

Let l_i be a repulsive distance.

If the distance between the robot and the obstacle is less than a repulsion distance l_i, then the deformation occurs.

Let X and Y be two closest points on the obstacle and the robot and let ln be a vector, where l is the distance between point X and point Y and n as the unit vector.

If there is a move from point X which makes l greater than the repulsion distance l_i and let the displacement be $\triangle X$.

If $\triangle X$ satisfies the equation below:

$$\triangle X \cdot n \geq l_i - l \qquad 4.7$$

Then if we use Jacobian matrix J_x at X to show the relationship between the small displacement $\triangle y$ and $\triangle X$ in the configuration space as:

$$\text{if } \triangle X = J_x \triangle y$$

$$\text{then, } J_x \triangle y \cdot n = \triangle y \cdot J_X^T n \geq l_i - l \qquad 4.8$$

using, $l_i - l$, and without inversing the J_x,

$\triangle y$ can be derived as:

$$= (l_i - l) \frac{J_X^T n}{\|J_X^T n\|^2}$$

This implies that the robot cannot move beyond the absolute value of $\triangle y$ because it will go outside the boundary.

The diagram below describes the concept of deformation.

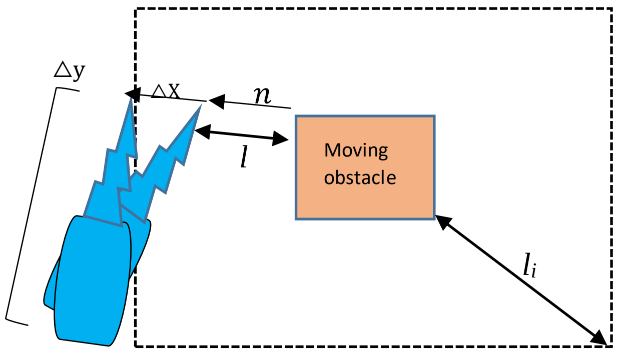


Figure 4.17 Path Deformation direction due to small displacement to avoid collision

The obstacle is not continuously moving, and our proposed system does not imply re-planning the Voronoi Diagram completely. So once there is interference due to environmental changes caused by the movement of the obstacle, the pre-planned path for the robot to execute will no longer be applicable and the robot will be in a state of “waiting time concept”. The problem is updated and check if an alternative path can be generated by updating the part of the VD affected by the obstacle motion. However, if an alternative path is found which implies that the deformation is successful then the alternative path can now be substituted with the pre-planned path and then continue the execution on the deformed Voronoi Diagram.

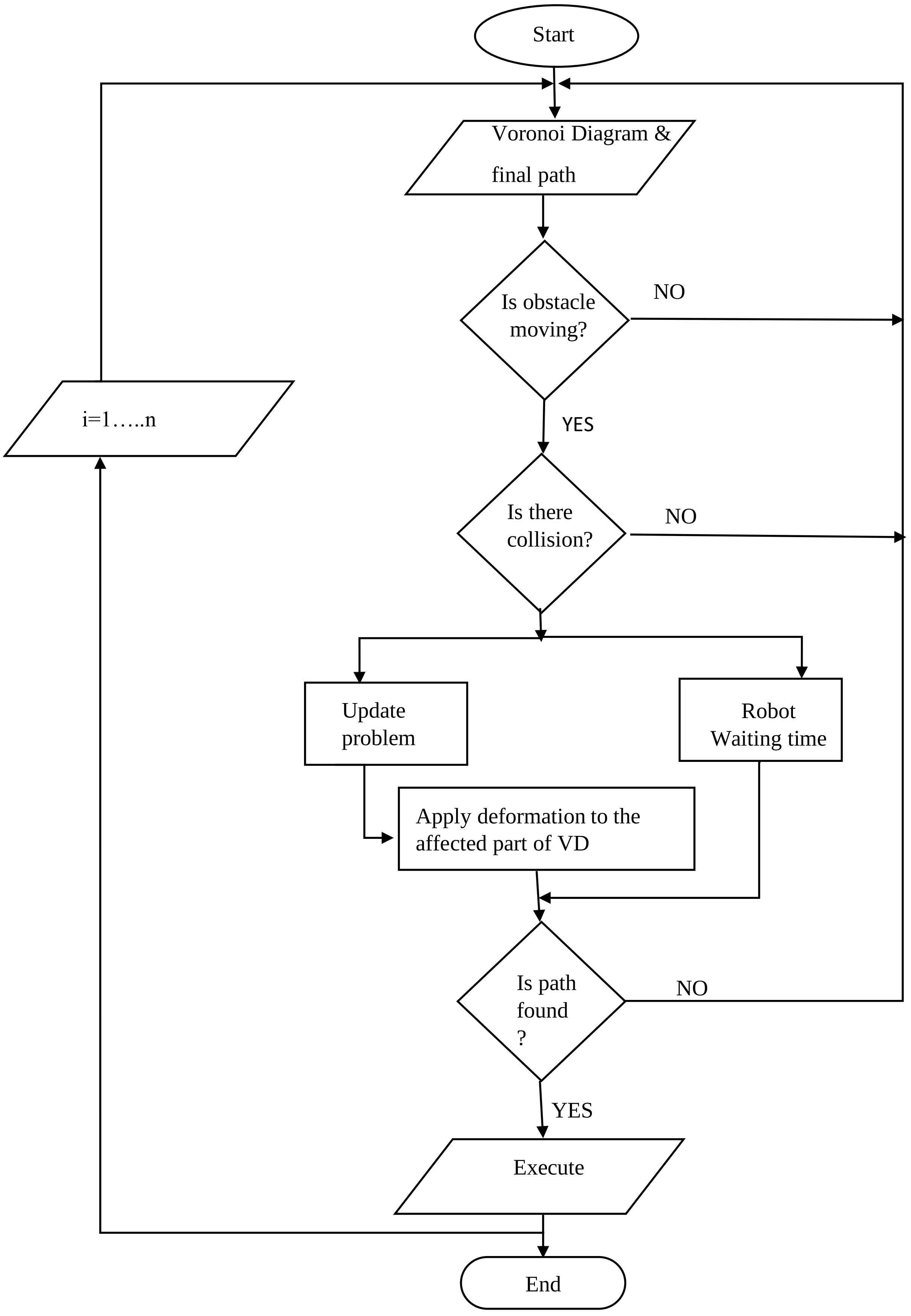


Figure 4.18 Flowchart for deformable Voronoi Diagram.

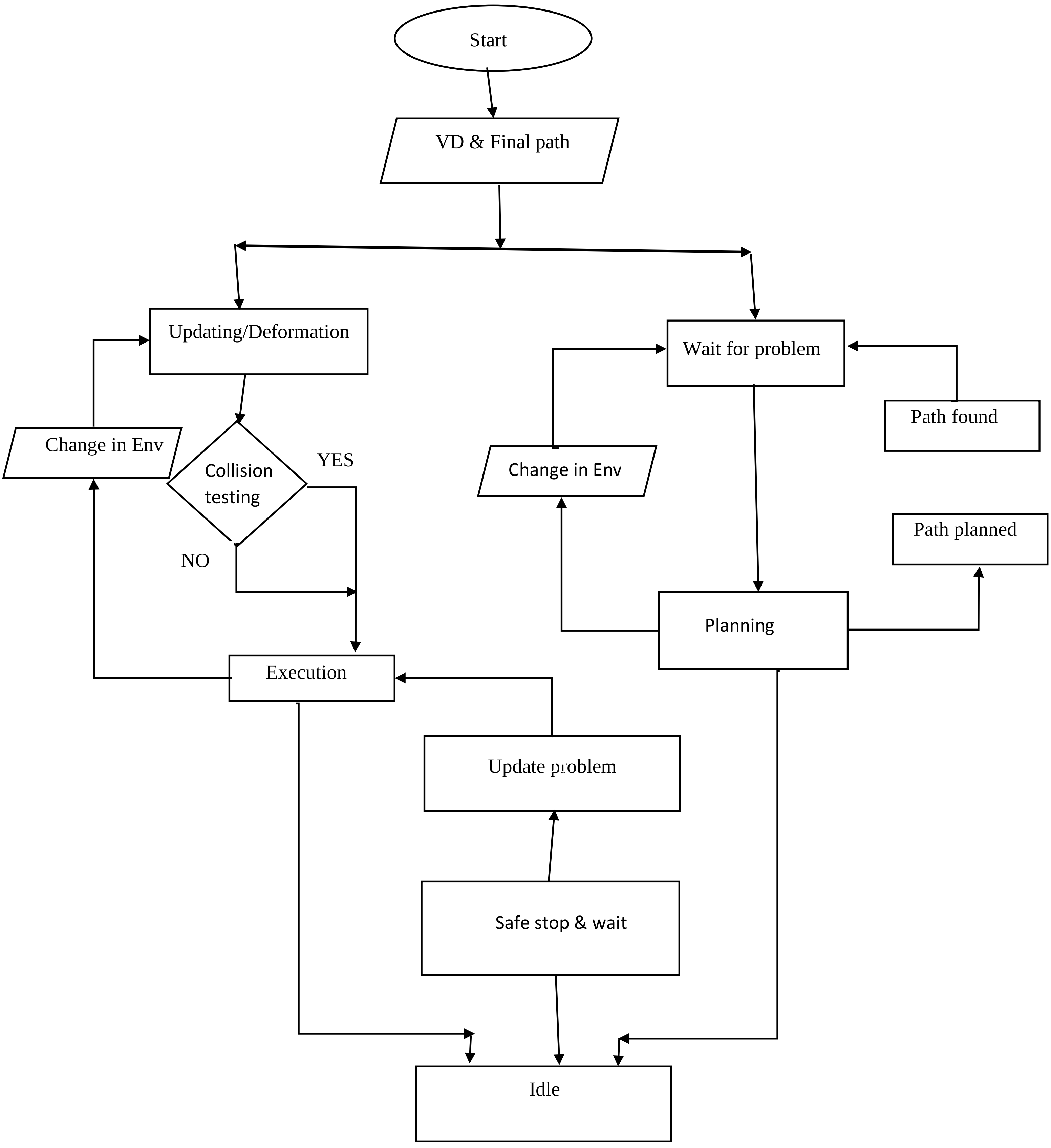


Figure 4.19 The flowchart of replanning and execution in a real life situation

4.7 The Graphical User Interface for the proposed system

Our new system presents a Graphical User Interface (GUI) that will allow any user to interact with the system using images instead of text commands. The GUI shows some graphical environments where some results are displayed. Space, where the robot operates, can be described as a Matlab axes object and there are push buttons for the user to perform actions required. The GUI which is shown in Fig.4.19 has a wide space/environment where the results are displayed which is called the workspace (*W*).

The GUI comprises four panels: the menu bar, the title bar, the workspace, and the path planning panel.

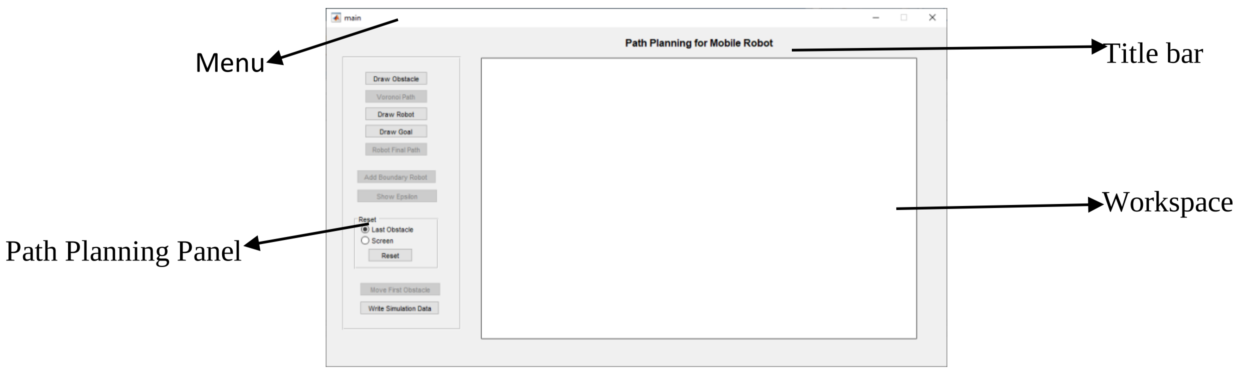


Figure 4.19 The main window of the setting.

(a) **The Menu Bar** is placed at the top of the window, horizontally displayed Menu. Sometimes called a file menu and helps in file handling in some cases.

(b) **The title bar** comes after the Menu bar on the top of the window: this describes the name of the GUI.

(c) **The workspace panel** is made up of a wide white space where obstacles are created, and the trajectory that is produced from the path planning algorithms displayed, also where the simulation are displayed.

(d) **Path planning panel** is placed vertically at the left side of the GUI which consists of eleven pushbuttons. Each of these pushbuttons is named for a particular purpose. The first pushbutton 'Draw Obstacle’ is used for drawing obstacles, the pushbutton ‘Draw Robot’ is to draw a robot and the pushbutton ’Voronoi Path’ is used to compute the GVD as the approach for planning the path. Clicking on ‘Robot Final Path' to obtain the shortest path for the robot. For each pushbutton on the panel, its label describes it is function.

Chapter Five

5.0 Results and Analysis

5.1 Background

This chapter discusses the simulation results and the evaluation of the performance of the proposed path planner using the simulation results. The Generalised Voronoi Diagram is used to extract the free space for a mobile robot and the Deformation Retract technique is applied to the extracted free space to generate alternative paths. The experimentation was done in MATLAB for different environmental settings. The graphical user interface (GUI) is designed for the path planning system to enable user-friendliness of the new system.

This is a path planning algorithm for a mobile robot with 2-dimensional complex environments and polygonal obstacles. However, the shortest paths are expected to be optimal, factors such as unevenness of the terrain, the path length, and the delay in executing the path (due to some constrained routes where robots need to navigate through some wave points) may produce shortest paths that do not end at the stated target location because it will be at the expence execution time.

5.2 Environments and Algorithms

To address the path planning problems, maps of the environments are required. These environments contain both static/dynamic obstacles, robots, and are represented in the GUI window (workspace). Both obstacles, robots and maps are created in the workspace and the simulation results are presented. The boundary of the workspace and robot are also regarded as obstacles. In this work, to compute the Voronoi Diagram using the parameter $\mathcal{E}$, the polygonal objects (obstacles) are divided into point sites and the map for all set of points is constructed. The distance between successive point sites is controlled by the parameter $\mathcal{E}$. If the value of parameter $\mathcal{E}$ is small, say, 0.5 or 0.8, then the number of nodes increases and this implies more spaced Voronoi edges.

The smaller the $\mathcal{E}$, the smoother the path generated at the cost of computation time, because the number of points increases. However, if a high value of $\mathcal{E}$ is employed, such an experiment will generate non-smooth and/or unconnected segments, because the points sites considered are not enough. Since the distance between the consecutive point sites is based on the parameter $\mathcal{E}$.

If the $\mathcal{E} \leq 2$, the smoother the path generated whilst the path generated will be non-smooth and/or unconnected segments if the $\mathcal{E} \geq 3$.

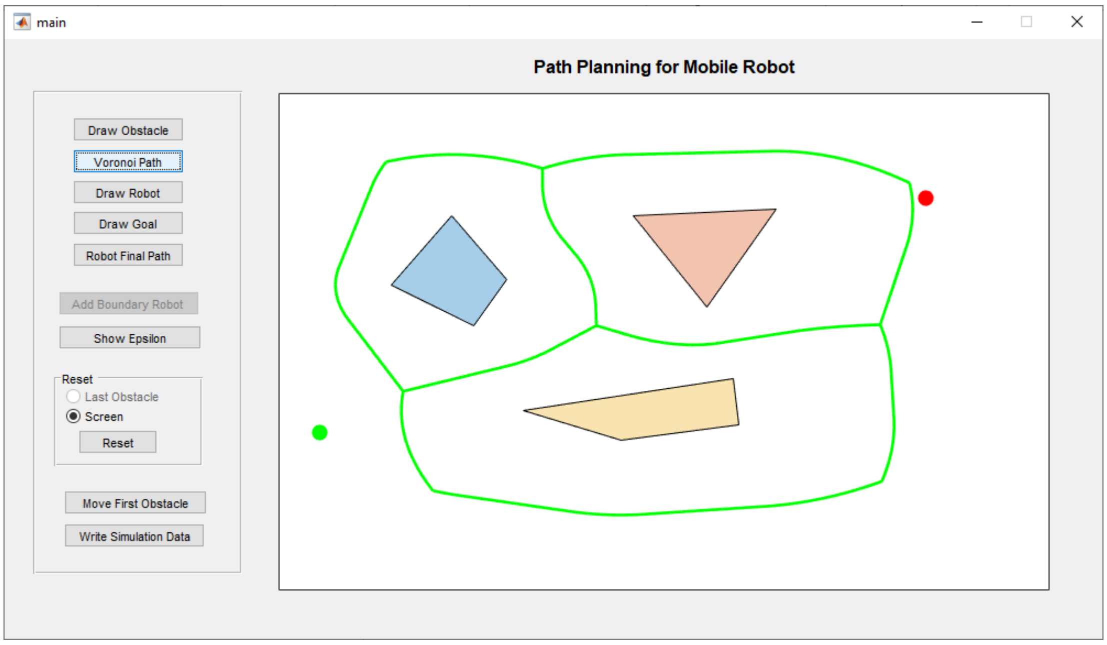


Figure 5.1 $\mathcal{E}$=1.5 with smoother path

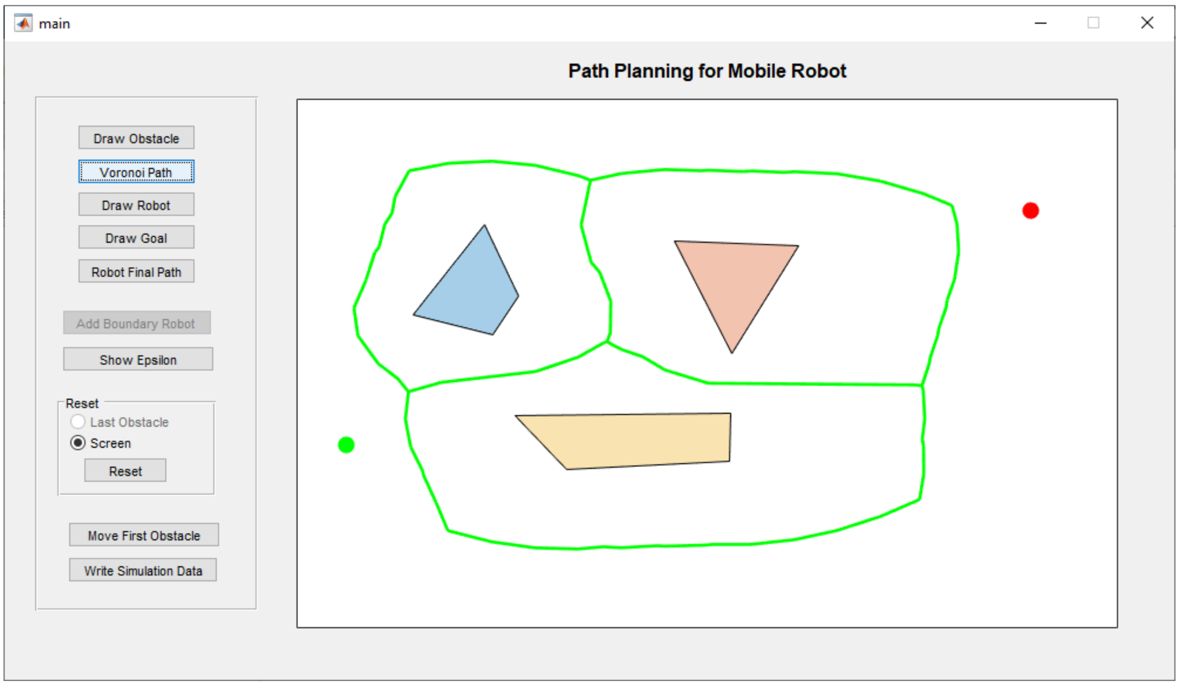


Figure 5.2 $\mathcal{E}$ = 5 with non-smooth but connected segments

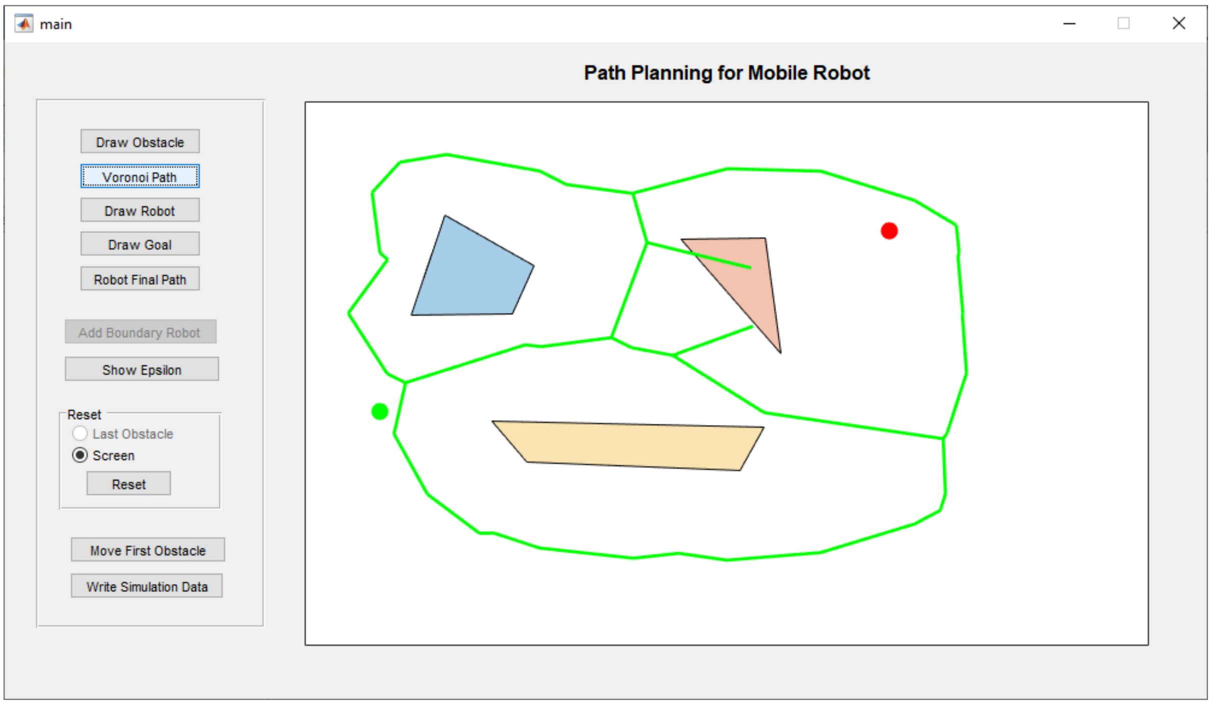


Figure 5.3 $\mathcal{E}$ = 10 with non-smooth and unconnected segments

The comparison variations of $\mathcal{E}$ and the number of edges and vertices in a static environment, i.e., before the deformation of the Voronoi Diagram can be illustrated in Figures 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, and Table 5.1.

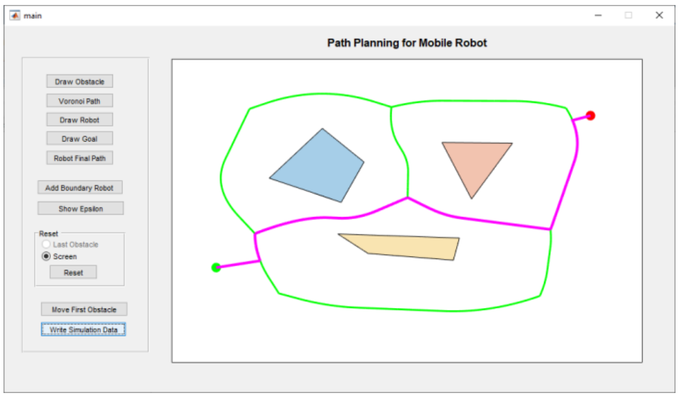


Figure 5.4 Env1 with $\mathcal{E} = 0.4$

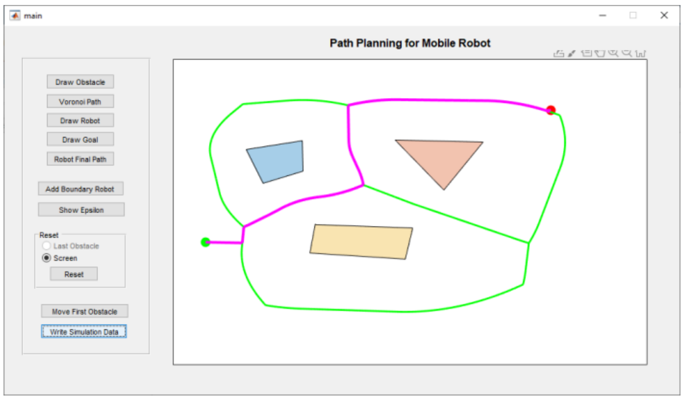


Figure 5.5 Env2 with $\mathcal{E} = 0.8$

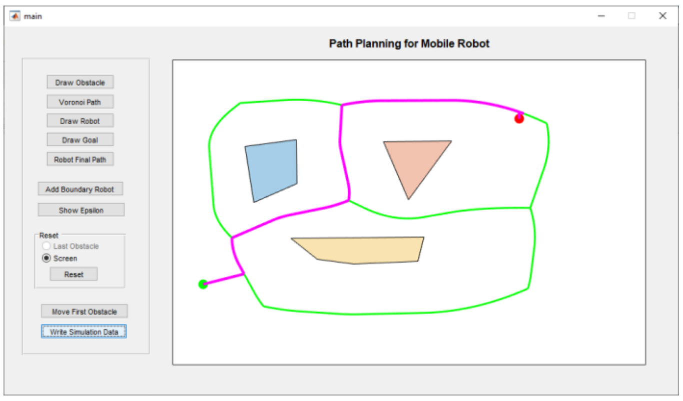


Figure 5.6 Env3 with $\mathcal{E} = 1.0$

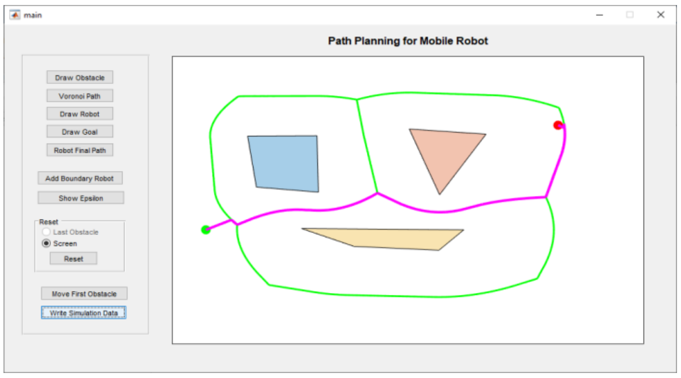


Figure 5.7 Env4 with $\mathcal{E} = 1.5$

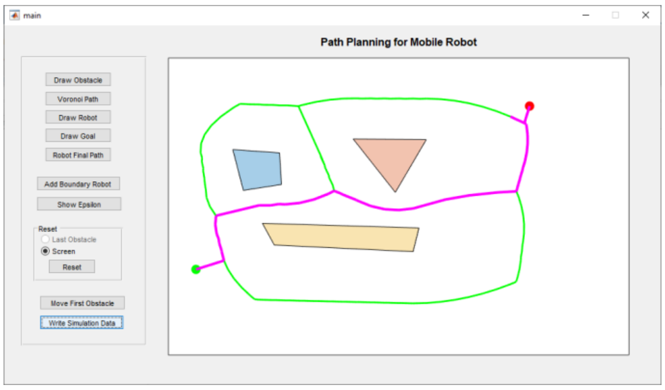


Figure 5.8 Env5 with $\mathcal{E} = 3$

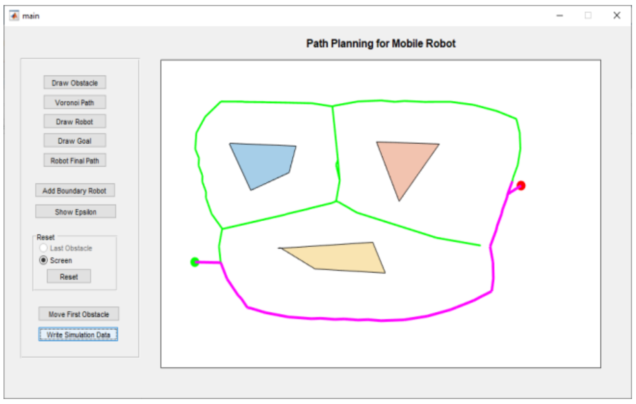


Figure 5.9 Env6 with $\mathcal{E} = 5$

Table 5.1 Comparing Variations of $\mathcal{E}$ and number of edges & vertices before VD deformation

Environments	**Number of obstacles**	**Epsilon, $\mathcal{E}$**	**Number of Vertices**	**Number of Edges**
Env1	3	0.4	1069	1381
Env2	3	0.8	505	648
Env3	3	1.0	421	547
Env4	3	1.5	302	394
Env5	3	3.0	145	181
Env6	3	5.0	85	103

However, the comparison of variations of $\mathcal{E}$ and the number of edges and vertices in a dynamic environment, i.e., after the deformation of the Voronoi Diagram can be illustrated in Figures 5.10, 5.11, 5.12, 5.13, 5.14, 5.15, and Table 5.2.

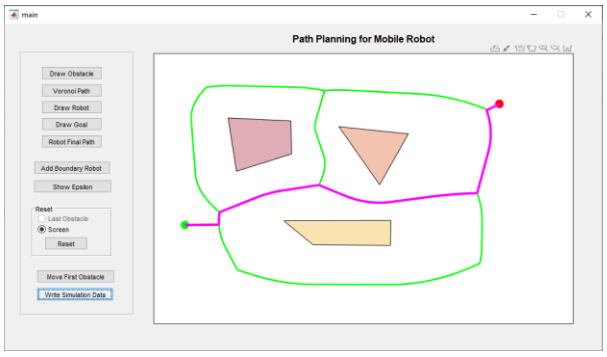


Figure 5.10 Env1 with $\mathcal{E} = 0.4$

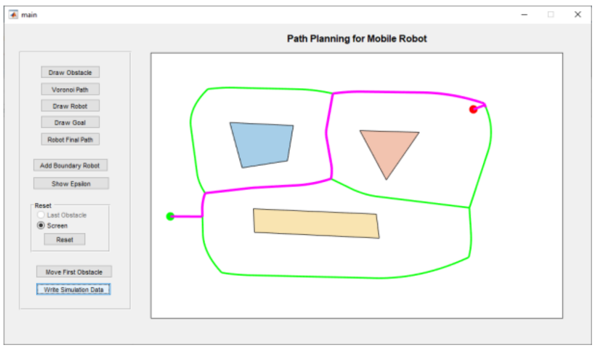


Figure 5.11 Env2 with $\mathcal{E} = 0.8$

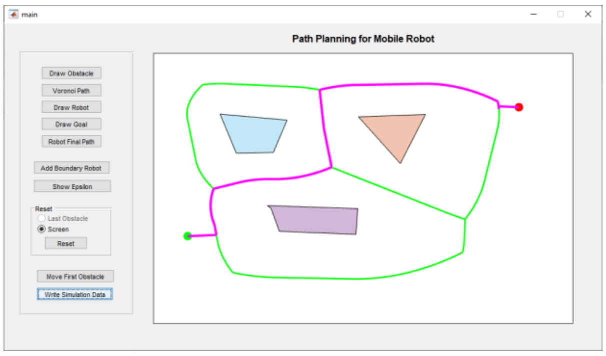


Figure 5.12 Env3 with $\mathcal{E} = 1.0$

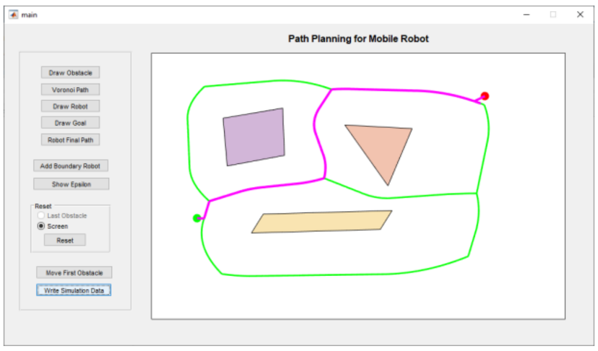


Figure 5.13 Env4 with $\mathcal{E} = 1.5$

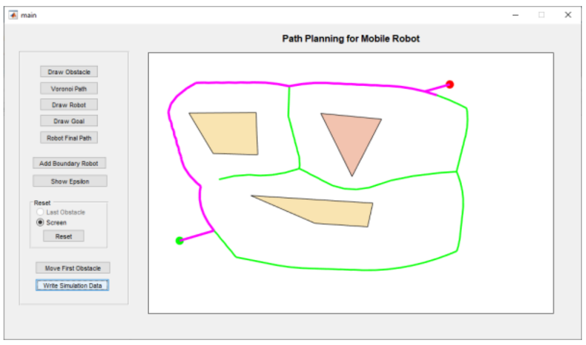


Figure 5.14 Env5 with $\mathcal{E} = 3.0$

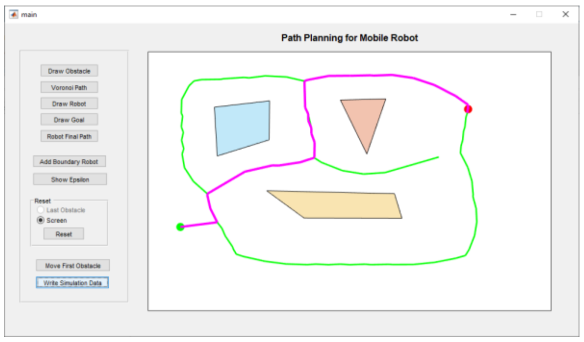


Figure 5.15 Env6 with $\mathcal{E} = 5.0$

Table 5. 2 Comparing Variations of $\mathcal{E}$ and number of edges & vertices after VD deformation

Environments	**Number of obstacles**	**Epsilon, $\mathcal{E}$**	**Number of Vertices**	**Number of Edges**
Env1	3	0.4	1088	1405
Env2	3	0.8	559	724
Env3	3	1.0	417	528
Env4	3	1.5	308	405
Env5	3	3.0	152	202
Env6	3	5.0	92	121

Six environments will also be used to make a comparison of the execution time for the Voronoi Diagram and the final path before and after deformation. Three obstacles are going to be used for these illustrations, one moving obstacle and two static obstacles. The first obstacle (i.e., upper left obstacle) is the moving obstacle as shown in Figure 5.16.

Considering the comparison of the execution time for computing the VD and the robot final path using six environments before deformation is illustrated in Figures 5.16 to 5.21.

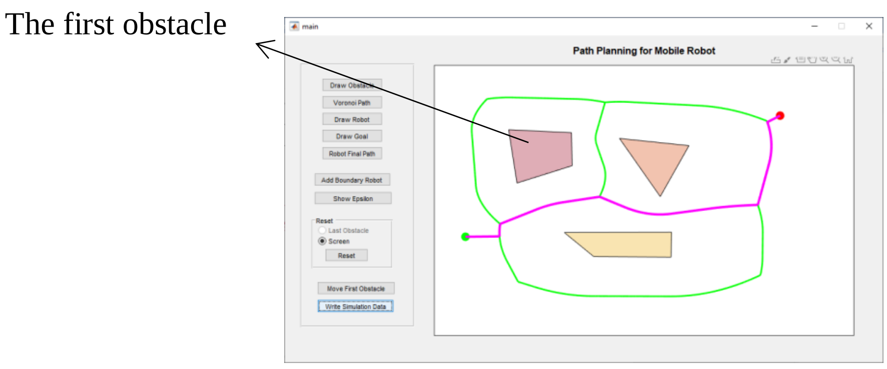


Figure 5.16 Env1 with $\mathcal{E} = 0.4$,

VD (t=6.94s), Final Path (t=26.89s)

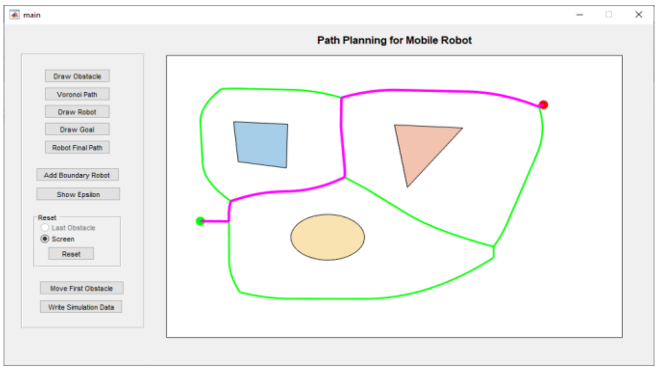


Figure 5.17 Env2 with $\varepsilon = 0.8$,

VD (t=2.07s), Final Path (t=7.11s)

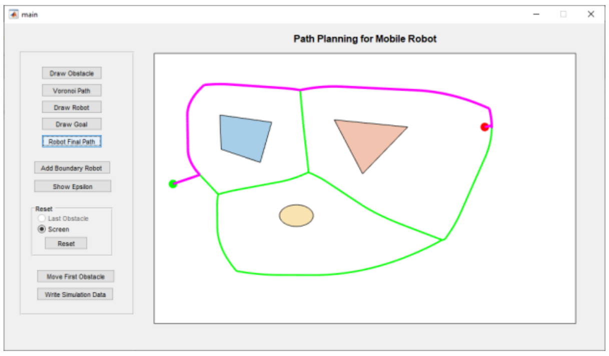


Figure 5.18 Env3 with $\varepsilon = 1.0$,

VD (t=1.30s), Final Path (t=6.25s)

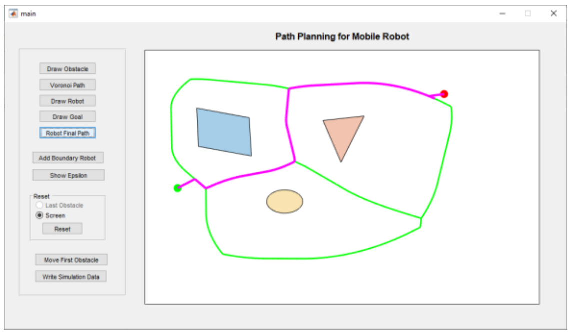


Figure 5.19 Env4 with $\varepsilon = 1.5$,

VD (t=0.77s), Final Path (t=1.70s)

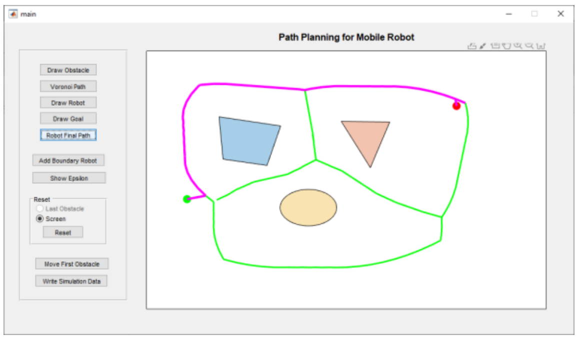


Figure 5.20 Env5 with $\mathcal{E} = 3.0$*,*

VD (t=0.35s), Final Path (t=1.56s)

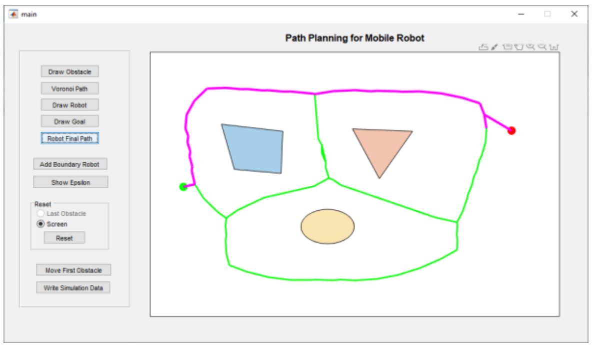


Figure 5.21 Env6 with $\mathcal{E} = 5.0$*,*

VD (t=0.32s), Final Path (t=0.91s)

This is also demonstrated in Table 5.3.

Table 5.3 Comparison of the execution time of getting VD and Final path before Deformation for six environments

Environments	**Number of Obstacles**	$\mathcal{E}$	**Voronoi Computation Time(s)**	**Final Path Execution time(s)**
Env1	3	0.4	6.94	26.89
Env2	3	0.8	2.07	7.11
Env3	3	1.0	1.30	6.25
Env4	3	1.5	0.77	1.70

Env5	3	3.0	0.35	1.56
Env6	3	5.0	0.32	0.91

However, the comparison of the execution time for computing the VD and the robot final path using the first obstacle (upper left 4-sided object) as the moving obstacle for the six environments after deformation is illustrated in Figures 5.22 to 5.27. The Voronoi computation time is proportional to the final path execution time.

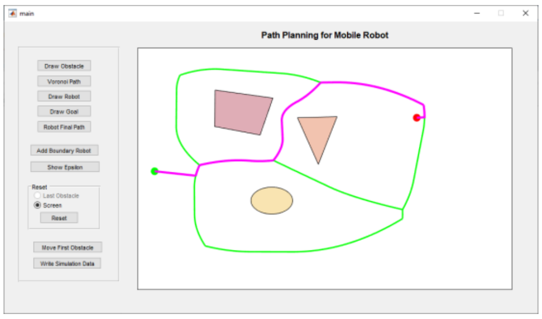


Figure 5.22 Env1 with $\mathcal{E} = 0.4$*,*

VD (t=6.26s), Final Path (t=20.32s)

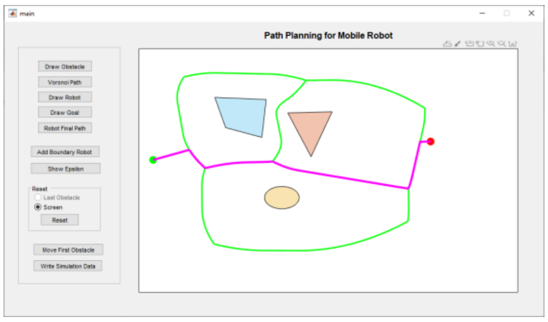


Figure 5.23 Env2 with $\mathcal{E} = 0.8$*,*

VD (t=1.86s), Final Path (t=6.27s)

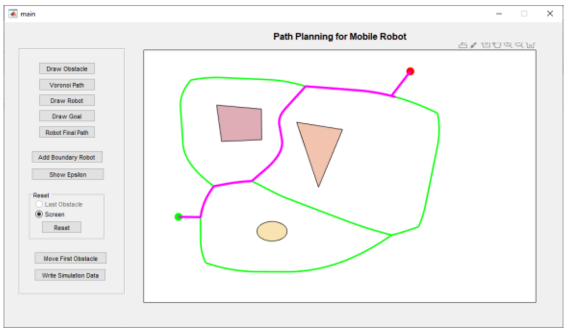


Figure 5.24 Env3 with $\mathcal{E} = 1.0$*,*

VD (t=1.83s), Final Path (t=3.82s)

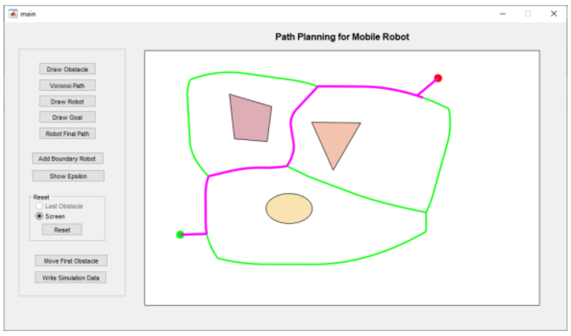


Figure 5.25 Env4 with $\mathcal{E} = 1.5$*,*

VD (t=0.83s), Final Path (t=1.60s)

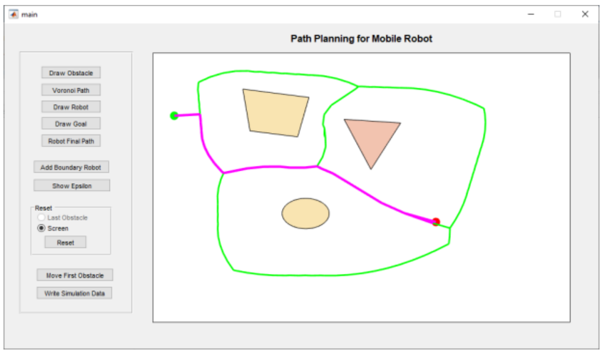


Figure 5.26 Env5 with $\mathcal{E} = 3.0$*,*

VD (t=0.31s), Final Path (t=0.74s)

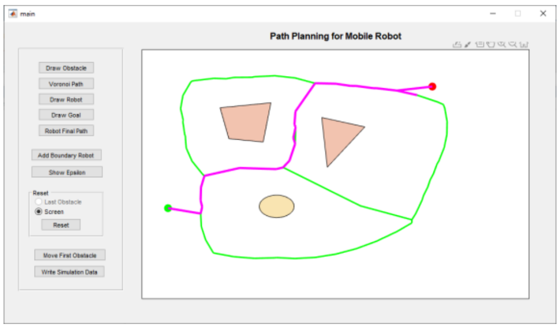


Figure 5.27 Env5 with $\mathcal{E} = 5.0$,

VD (t=0.17s), Final Path (t=0.70s)

A similar analysis for dynamic environments is described in table 5.4.

Table 5.4 Comparison of the execution time of getting VD and Final path after Deformation for six environments

Environments	**Number of Obstacles**	$\mathcal{E}$	**VD computation Time(s)**	**Final Path Execution time(s)**
Env1	3	0.4	6.26	20.32
Env2	3	0.8	1.86	4.16
Env3	3	1.0	1.83	3.82
Env4	3	1.5	0.83	1.60
Env5	3	3.0	0.31	0.74
Env6	3	5.0	0.17	0.70

5.3 Path Deformation process

To simulate the Voronoi Diagram deformation process, a scenario with three static and one dynamic obstacle was used. The robot is expected to traverse around these obstacles to reach the goal position. The first planned path has been generated with the assumption that all four obstacles are static. The point robot executes the first free path planned and successfully reaches the goal point since no obstacle moves and there is prior knowledge of the environment. However, in the dynamic environment, when the robot tries to execute the pre-planned path, interference occurs due to the moving obstacle (in this study the first obstacle). Furthermore, after the interference of the obstacle on the Voronoi, the robot observes a wait and check state whilst a new Voronoi Diagram is ccomputed and an alternative (collision-free) path is generated. Therefore, after the deformation of the map, the robot now executes the new path as shown in Figures 5.9, 5.10, 5.11, 5.12, 5.13 and 5.14. A planning result for the deformation process with the upper left object as the moving obstacle.

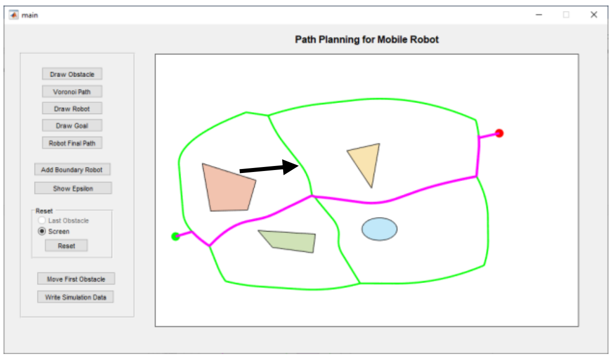


Figure 5.28 VD without interference (see the distance from the first obstacle and the final path, also to the segment in arrow)

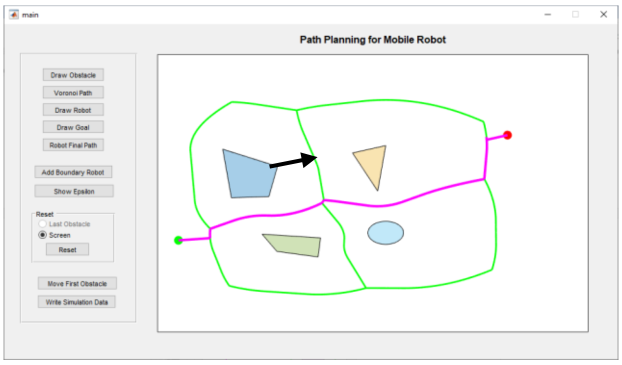


Figure 5.29 VD with minor interference (the first obstacle has moved away a bit from the magenta line and closer to the segment)

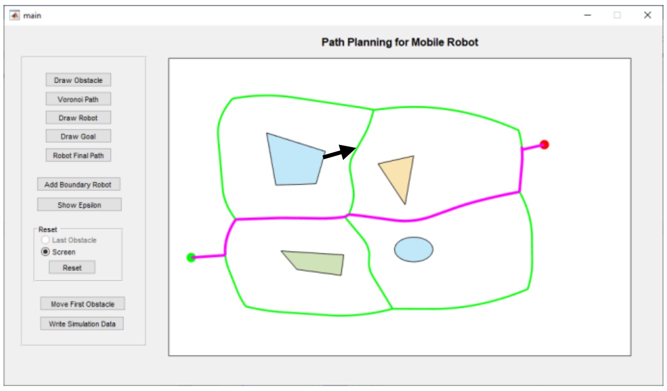


Figure 5.30 VD with less interference (the moving obstacle has again moved farther away from the final path and closer to segment)

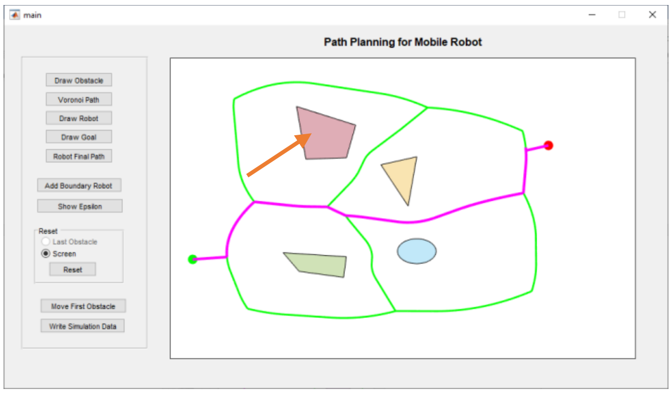


Figure 5.31 VD with noticeable interference (See the distance traversed by the moving obstacle as indicated by the red arrow)

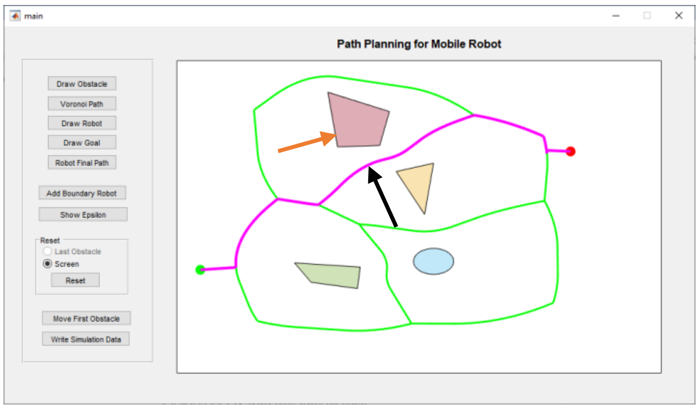


Figure 5.32 VD with high interference (the interference has made the final path to completely deformed to anther path in the VD as shown by the black arrow and also see the distance be the moving obstacle in red arrow)

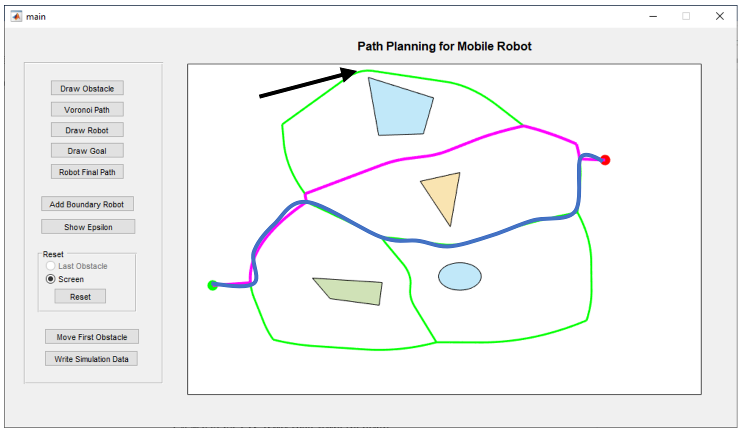


Figure 5.33 VD with higher interference (the final path has deformed due to the interference caused by the moving obstacle, see the pre-planned path in blue colour whilst the new path in magenta)

As the obstacle moves into the path of the robot, the distance between it and the robot (and hence, the minimum distance between the robot and any obstacle in the configuration space) becomes less than the repulsive distance, and the map is updated. The original path in 5.28 becomes inapplicable and as shown in Figure 5.33 it deforms to avoid the obstacle. This deformation continues as long as there is an obstacle. This goes through 5.30, 5.31, and 5.32 towards the goal position at 5.33 as the shortest path from the initial position. This process is repeated until the goal is reached. The Voronoi Diagram is updated whenever an obstacle moves and encounters interference.

To evaluate the efficiency of the new system, the performance was tested by comparing the execution of time spent computing the Voronoi Diagram before deformation (VD t1) with the execution time on Voronoi Diagram after deformation (VD t2) and it shows that there is no significant difference. Figure 5.34 uses different ε values for various Voronoi Diagram before and during deformation.

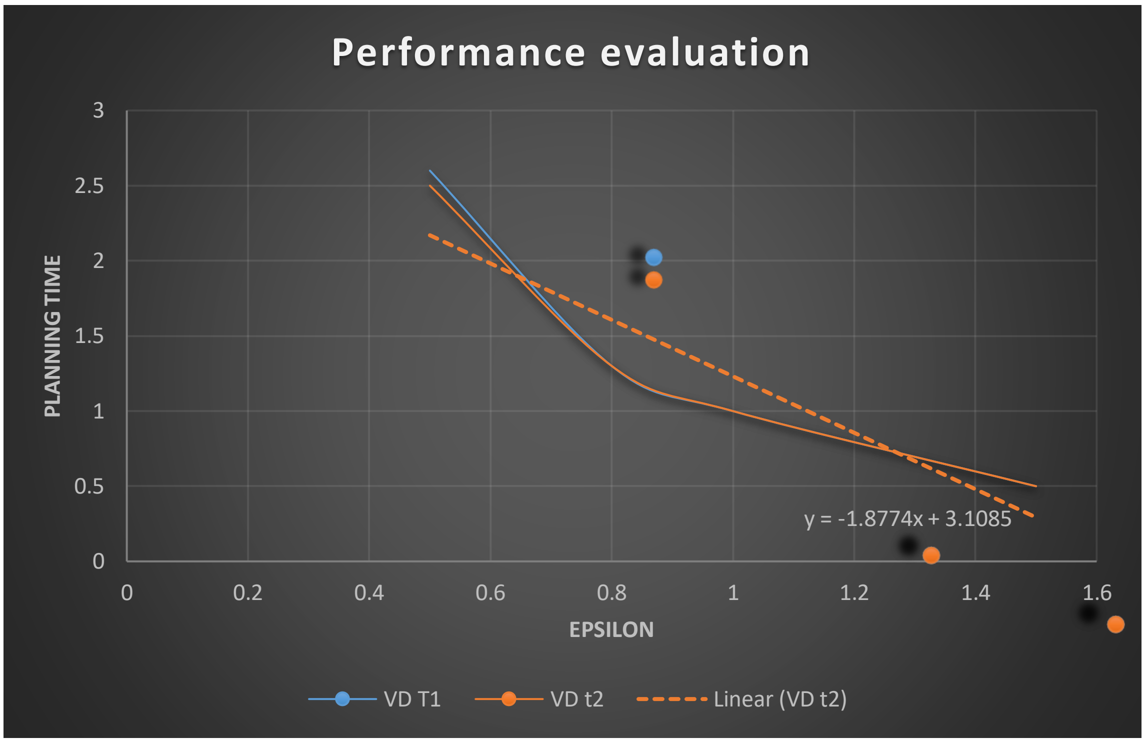


Figure 5.34 The comparison of computation time for VD before and during deformation

The comparison of the execution time for robot final path in a static environment with execution time for robot final path in a dynamic environment is considered. This shows that there is not much difference in the time spent on the final path before and after the deformation as it is illustrated in Figure 5.35.

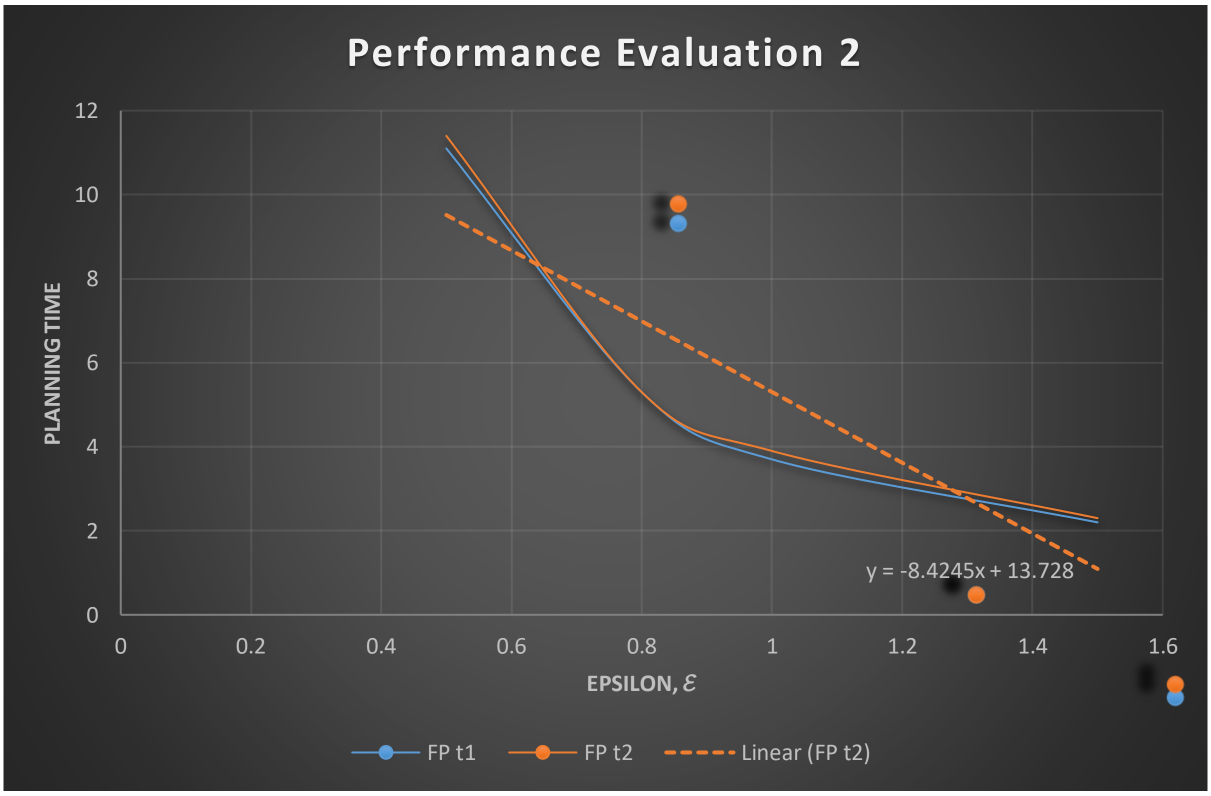


Figure 5.35 Comparison of execution time before and during deformation on robot final path.

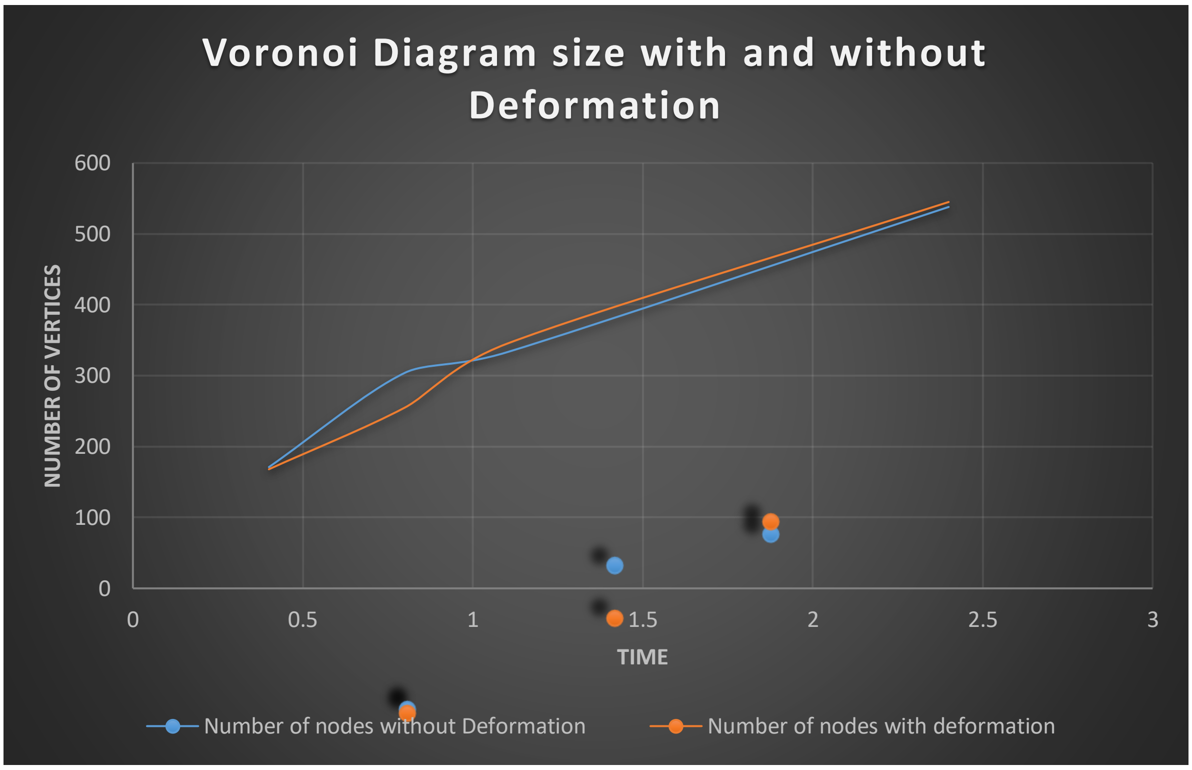


Figure 5.36 The comparison of VD size before and after deformation

Figures 5.28 to 5.33 show that the Voronoi Diagram size is almost the same with and without deformation. However, it is observe that the number of nodes visited between 0.5 and 1 time without deformation is greater than that with deformation, but later became almost the same.

From all the results, it is remarked that the combination of the Voronoi Diagram and Deformation Retract can also solve the problem of path planning effectively.

Chapter Six

6.0 Summary and Conclusion

This study aims to design a planner for a mobile robot in a dynamic environment using the fusion of the Voronoi Diagram method and deformation refracts. And for this to be achieved, the first task was how to generate the representation of the map using the Voronoi Diagram by extracting the safest areas in the environment based on parameter $\mathcal{E}$ and also to generate the final shortest free path for the given obstacle configuration and start and goal points. Then a deformation technique is introduced to the extracted free space based on a distance i.e. repulsive distance to the obstacle. This is aimed at dealing with the changes in the environment. The deformation occurs when the distance between a moving obstacle and a static or another moving obstacle is less than the repulsive distance. However, the first path generated becomes inapplicable since the initial map is deformed. This proposed method is a roadmap representation that retracts and updates as a function in the dynamic environment. It can only be used to plan the path of a single robot among dynamic obstacles.

Finally, a method is developed to allow the robot to reach its goal using the fusion of Generalised Voronoi Diagram and Deformation Retracts.

Our new system offers many advantages over existing methods. Most existing methods remove the edges as soon as they are invalidated but our edges retract based on the moving obstacle, they are not invalidated quite often. Therefore, the system updates and retracts the affected area of the map. Another advantage of the new path planner is that it uses the Voronoi Diagram that has the characteristics of maximizing the clearance between the robot and obstacle by generating the safest areas in the environment. The new path planning system will not only be useful in robotics to help robots in dynamic environments but also in other domains such as games theory, virtual reality, computational geometry to mention few.

The paths generated by the new system are smooth and safe and are free of any trap due to the integration of the deformation mechanism on the continuously updated map. A graphical user interface is used to implement the algorithm, to make the system user friendly. The new system can work also with a different type of shaped objects including circle. The algorithm is written in MATLAB, it is fast, works for dynamic obstacle, and can find a path if it exists.

The new method has some disadvantages because there are no assumptions on the motion of the moving obstacle, the path planned is likely to be influenced by other objects, it cannot also guarantee the optimality of the path generated. If the point robot is replaced with a robot with a shape, the new system needs to be updated for every robot shape. This system is designed only for point robot it cannot be applied to multiple robots.

The simulation results are produced to evaluate the effectiveness of the new system. Several types of environments were designed to evaluate the performance of the algorithm, and these simulation results confirmed that the lesser the value of $\mathcal{E}$ the geater number of vertices, and the smoother the output path but a higher cost of computation. Furthermore, the simulation results for a crowded environment with static and dynamic obstacles shows that the new system performance is effective because the computation cost before and after interference is almost the same.

The simulation results also showed that when the environment with two obstacles and another environment with three obstacles tested, and if the $\mathcal{E}$ value is the same, the new system performs at almost the same cost computation before and after the deformation.

The new method is very efficient because it deals properly with each scenario whilst applying the deformation and updating the environment in a timely way.
In this method, the deformation mechanism is used based on the distance observed to obstacle and it enhances the method to address the problem of environment changing due to obstacle movement through deformation. This

deformation is necessary when avoiding collisions, and based on the continuing update of the environment.

6.1 Future work

Deformable Voronoi diagram for robot path planning is a theoritical method that needs to be implemented in real life, there is also room for improvement in the number of moving obstacles used. Only one moving obstacle is used in this study. Though, this approach is promising the path deformation can be made more effective by removing extra work whenever there is little risk of anticipation of collision. The proposed system needs to be validated with complex robots in dynamic simulations. Also, hardware experiments need to be addressed in future by using the e-puck robot and also sensors.

References

Alves, S. F., Rosario, J. M., Ferasoli Filho, H., Rincon, L. K., Yamasaki, R. A., & Barrera, A. (2011). Conceptual bases of robot navigation modelling control and applications. *Advances in Robot Navigation,* , 26.

Aurenhammer, F., & Klein, R. (2000). Voronoi diagrams. *Handbook of Computational Geometry, 5*(10), 201-290.

Azariadis, P. N., & Aspragathos, N. A. (2005). Obstacle representation by bump-surfaces for optimal motion-planning. *Robotics and Autonomous Systems, 51*(2-3), 129-150.

Behnke, S. (2003). Local multiresolution path planning. Paper presented at the *Robot Soccer World Cup,* 332-343.

Benavides, F., Tejera, G., Pedemonte, M., & Casella, S. (2011). Real path planning based on genetic algorithm and voronoi diagrams. Paper presented at the *IX Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control, 2011 IEEE,* 1-6.

Bi, Z., Yimin, Y., & Yisan, X. (2009). Mobile robot navigation in unknown dynamic environment based on ant colony algorithm. Paper presented at the *2009 WRI Global Congress on Intelligent Systems, , 3* 98-102.

Blanco, F. J., Moreno, V., & Curto, B. (1998). Path planning method for mobile robots in changing environments. *IFAC Proceedings Volumes, 31*(2), 371-376. doi:10.1016/S1474-6670(17)44225-X

Boukas, E., Kostavelis, I., Gasteratos, A., & Sirakoulis, G. C. (2014). Robot guided crowd evacuation. *IEEE Transactions on Automation Science and Engineering, 12*(2), 739-751.

Bounini, F., Gingras, D., Pollart, H., & Gruyer, D. (2017). Modified artificial potential field method for online path planning applications. Paper presented at the *2017 IEEE Intelligent Vehicles Symposium (IV),* 180-185.

Campbell, S., O'Mahony, N., Carvalho, A., Krpalkova, L., Riordan, D., & Walsh, J. (2020). Path planning techniques for mobile robots a review. Paper presented at the *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE),* 12-16.

Campbell, S., O'Mahony, N., Carvalho, A., Krpalkova, L., Riordan, D., & Walsh, J. (2020). Path planning techniques for mobile robots a review. Paper presented at the *2020 6th International Conference on Mechatronics and Robotics Engineering (ICMRE),* 12-16.

Canny, J., & Donald, B. (1990). Simplified voronoi diagrams. *Autonomous robot vehicles* (pp. 272-289) Springer.

Charalampous, K., Kostavelis, I., Amanatiadis, A., & Gasteratos, A. (2014). Real-time robot path planning for dynamic obstacle avoidance. *Journal of Cellular Automata, 9*

Charalampous, K., Kostavelis, I., & Gasteratos, A. (2015). Thorough robot navigation based on SVM local planning. *Robotics and Autonomous Systems, 70*, 166-180.

Chen, H., Fuhlbrigge, T., & Li, X. (2008). Automated industrial robot path planning for spray painting process: A review. Paper presented at the *2008 IEEE International Conference on Automation Science and Engineering,* 522-527.

Cheng, J., Cheng, H., Meng, M. Q., & Zhang, H. (2018). Autonomous navigation by mobile robots in human environments: A survey. Paper presented at the *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO),* 1981-1986.

Chia, S., Su, K., Guo, J., & Chung, C. (2010). Ant colony system based mobile robot path planning. Paper presented at the *2010 Fourth International Conference on Genetic and Evolutionary Computing,* 210-213.

Chik, S. F., Yeong, C. F., Su, E., Lim, T. Y., Subramaniam, Y., & Chin, P. (2016). A review of social-aware navigation frameworks for service robot in dynamic human environments. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC), 8*(11), 41-50.

Cho, H. S., & Woo, T. H. (2016). Mechanical analysis of flying robot for nuclear safety and security control by radiological monitoring. *Annals of Nuclear Energy, 94,* 138-143.

Choset Howie, Hutchinson Seth, Lynch, K., Kantor George, Burgard Wolfram, Kavraki Lydia, . . . Thrun Sebastian. (2005). *Principles of robot motion: Theory, algorithms, and implementation* MIT press.

Choset, H., & Burdick, J. (1995). Sensor based planning. II. incremental construction of the generalized voronoi graph. Paper presented at the *Proceedings of 1995 IEEE International Conference on Robotics and Automation, , 2* 1643-1648.

Choset, H., & Burdick, J. (1996). Sensor based motion planning: The hierarchical generalized voronoi graph. *Algorithms for Robot Motion and Manipulation, ,* 47-61.

Coenen, S., & Steinbuch, M. M. (2012). Motion planning for mobile robots—A guide. *Control Systems Technology, ,* 79.

Corke, P. I., & Khatib, O. (2011). *Robotics, vision and control: Fundamental algorithms in MATLAB* Springer.

Danner, T., & Kavraki, L. E. (2000). Randomized planning for short inspection paths. Paper presented at the *Proceedings 2000 ICRA. Millennium*

Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. no. 00CH37065), , 2 971-976.

De Berg, M., Van Kreveld, M., Overmars, M., & Schwarzkopf, O. (1997). Computational geometry. *Computational geometry* (pp. 1-17) Springer.

Dongbin, Z., & Jianqiang, Y. (2006). Robot planning with ant colony optimization algorithms. Paper presented at the *2006 Chinese Control Conference,* 1460-1465.

Douthwaite, J. A., Zhao, S., & Mihaylova, L. S. (2018). A comparative study of velocity obstacle approaches for multi-agent systems. Paper presented at the *2018 UKACC 12th International Conference on Control (CONTROL),* 289-294.

Engedy, I., & Horváth, G. (2009). Artificial neural network based mobile robot navigation. Paper presented at the *2009 IEEE International Symposium on Intelligent Signal Processing,* 241-246.

Fitch, R., Butler, Z., & Rus, D. (2003). Reconfiguration planning for heterogeneous self-reconfiguring robots. Paper presented at the *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. no. 03CH37453), , 3* 2460-2467.

Fortune, S. (1987). A sweepline algorithm for voronoi diagrams. *Algorithmica, 2*(1), 153-174.

Garrido, S., & Moreno, L. (2015). Mobile robot path planning using voronoi diagram and fast marching. *Robotics, automation, and control in industrial and service settings* (pp. 92-108) IGI Global.

Garrido, S., Moreno, L., Abderrahim, M., & Blanco, D. (2009). Robot navigation using tube skeletons and fast marching. Paper presented at the *2009 International Conference on Advanced Robotics,* 1-7.

Garrido, S., Moreno, L., Blanco, D., & Jurewicz, P. (2011). Path planning for mobile robot navigation using voronoi diagram and fast marching. *Int.J.Robot.Autom, 2*(1), 42-64.

Gayle, R., Sud, A., Lin, M. C., & Manocha, D. (2007). Reactive deformation roadmaps: Motion planning of multiple robots in dynamic environments. Paper presented at the *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 3777-3783.

Ge, S. S., & Cui, Y. J. (2000). New potential functions for mobile robot path planning. *IEEE Transactions on Robotics and Automation, 16*(5), 615-620.

Geiger, B., Kiraly, A. P., Naidich, D. P., & Novak, C. L. (2010). No title. *System and Method for Endoscopic Path Planning,*

Guo, N., Li, C., Gao, T., Liu, G., Li, Y., & Wang, D. (2021). A fusion method of local path planning for mobile robots based on LSTM neural network and reinforcement learning. *Mathematical Problems in Engineering, 2021*

Habib, N., Purwanto, D., & Soeprijanto, A. (2016). Mobile robot motion planning by point to point based on modified ant colony optimization and voronoi diagram. Paper presented at the *2016 International Seminar on Intelligent Technology and its Applications (ISITIA),* 613-618.

Han, L., & Amato, N. M. (2001). A kinematics-based probabilistic roadmap method for closed chain systems: Li han, texas A nancy M. amato, texas A. *Algorithmic and computational robotics* (pp. 243-251) AK Peters/CRC Press.

Han, W., Baek, S., & Kuc, T. (1997). Genetic algorithm based path planning and dynamic obstacle avoidance of mobile robots. Paper presented at the *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, , 3* 2747-2751.

Hsu, D., Kindel, R., Latombe, J., & Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research, 21*(3), 233-255. doi:10.1177/027836402320556421

Ji, X., & Xiao, J. (2001). Planning motions compliant to complex contact states. *The International Journal of Robotics Research, 20*(6), 446-465.

Khaksar, W., Hong, T. S., Khaksar, M., & Motlagh, O. R. E. (2012). Sampling-based tabu search approach for online path planning. *Advanced Robotics, 26*(8-9), 1013-1034. doi:10.1163/156855312X632166

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. *Autonomous robot vehicles* (pp. 396-404) Springer.

Krogh, B., & Thorpe, C. (1986). Integrated path planning and dynamic steering control for autonomous vehicles. Paper presented at the *Proceedings. 1986 IEEE International Conference on Robotics and Automation, , 3* 1664-1669.

Kruse, T., Pandey, A. K., Alami, R., & Kirsch, A. (2013). Human-aware robot navigation: A survey. *Robotics and Autonomous Systems, 61*(12), 1726-1743.

Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M., & Inoue, H. (2001). Motion planning for humanoid robots under obstacle and dynamic balance constraints. Paper presented at the *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. no. 01CH37164), , 1* 692-698.

Lamini, C., Benhlima, S., & Elbekri, A. (2018). Genetic algorithm based approach for autonomous mobile robot path planning. *Procedia Computer Science, 127*, 180-189.

Lamiraux, F., & Bonnafous, D. (2002). Reactive trajectory deformation for nonholonomic systems: Application to mobile robots. Paper presented at the *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. no. 02CH37292), , 3* 3099-3104.

Lamiraux, F., & Kavraki, L. E. (2001). Planning paths for elastic objects under manipulation constraints. *The International Journal of Robotics Research, 20*(3), 188-208.

Li, Y., Dong, T., Bikdash, M., & Song, Y. (2005). Path planning for unmanned vehicles using ant colony optimization on a dynamic voronoi diagram. Paper presented at the *Ic-Ai,* 716-721.

Liu, X., Li, Y., Zhang, J., Zheng, J., & Yang, C. (2019). Self-adaptive dynamic obstacle avoidance and path planning for USV under complex maritime environment. *IEEE Access, 7,* 114945-114954.

Lozano-Perez, T. (1987). A simple motion-planning algorithm for general robot manipulators. *IEEE Journal on Robotics and Automation, 3*(3), 224-238.

Lozano-Perez, T. (1990). Spatial planning: A configuration space approach. *Autonomous robot vehicles* (pp. 259-271) Springer.

Lv, N., & Feng, Z. (2006). Numerical potential field and ant colony optimization based path planning in dynamic environment. Paper presented at the *2006 6th World Congress on Intelligent Control and Automation, , 2* 8966-8970.

Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2016). Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems, 86,* 13-28.

Magid, E., & Rivlin, E. (2004). CautiousBug: A competitive algorithm for sensory-based robot navigation. Paper presented at the *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. no. 04CH37566), , 3* 2757-2762.

Mahajan, P. B., & Marbate, P. (2013). Literature review on path planning in dynamic environment. *International Journal of Computer Science and Network, 2*(1), 115-118.

Mahkovic, R., & Slivnik, T. (1998). Generalized local voronoi diagram of visible region. Paper presented at the *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. no. 98CH36146), , 1* 349-355.

Manousakis, K., McAuley, T., Morera, R., & Baras, J. (2005). Using multi-objective domain optimization for routing in hierarchical networks. Paper presented at the *2005 International Conference on Wireless Networks, Communications and Mobile Computing, , 2* 1460-1465.

Masehian, E., & Amin-Naseri, M. R. (2004). A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Robotic Systems, 21*(6), 275-300.

Masehian, 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.

Mitchell, J. S. (1988). An algorithmic approach to some problems in terrain navigation. *Artificial Intelligence, 37*(1-3), 171-201.

Mohamad, M. M., Taylor, N. K., & Dunnigan, M. W. (2006). Articulated robot motion planning using ant colony optimisation. Paper presented at the *2006 3rd International IEEE Conference Intelligent Systems,* 690-695.

Nagatani, K., Choset, H., & Thrun, S. (1998). Towards exact localization without explicit localization with the generalized voronoi graph. Paper presented at the *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. no. 98CH36146), , 1* 342-348.

Nanda, S. K., Dash, A. K., Acharya, S., & Moharana, A. (2010). Application of robotics in mining industry: A critical review. *The Indian Mining & Engineering Journal, 8*, 108-112.

NGAH, W., Buniyamin, N., & Mohamad, Z. (2010). Point to point sensor based path planning algorithm for autonomous mobile robots. Paper presented at the *Proceedings of the 9th WSEAS International Conference on System Science and Simulation in Engineering,* 186-191.

Nguyen, H. T., & Le, H. X. (2016). Path planning and obstacle avoidance approaches for mobile robot. *arXiv Preprint arXiv:1609.01935,*

Nieuwenhuisen, D., Kamphuis, A., Mooijekind, M., & Overmars, M. H. (2004). Automatic construction of roadmaps for path planning in games. Paper presented at the *International Conference on Computer Games: Artificial Intelligence, Design and Education,* 285-292.

Niu, H., Lu, Y., Savvaris, A., & Tsourdos, A. (2018). An energy-efficient path planning algorithm for unmanned surface vehicles. *Ocean Engineering, 161*, 308-321.

ó'Dúnlaing, C., Sharir, M., & Yap, C. K. (1983). Retraction: A new approach to motion-planning. Paper presented at the *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing,* 207-220.

Ó'Dúnlaing, C., & Yap, C. K. (1985). A “retraction” method for planning the motion of a disc. *Journal of Algorithms, 6*(1), 104-111.

Okabe, A., Boots, B., Sugihara, K., & Chiu, S. N. (2009). *Spatial tessellations: Concepts and applications of voronoi diagrams* John Wiley & Sons.

Pathak, M. J., Patel, R. L., & Rami, S. P. (2018). Comparative analysis of search algorithms. *International Journal of Computer Applications, 179*(50), 40-43.

Patle, B. K., Pandey, A., Parhi, D., & Jagadeesh, A. (2019). A review: On path planning strategies for navigation of mobile robot. *Defence Technology, 15*(4), 582-606.

Pearson, J. R., & Beran, T. N. (2018). The future is now: Using humanoid robots in child life practice.

Petres, C., Pailhas, Y., Patron, P., Petillot, Y., Evans, J., & Lane, D. (2007). Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics, 23*(2), 331-341.

Rashid, R., Perumal, N., Elamvazuthi, I., Tageldeen, M. K., Khan, M. A., & Parasuraman, S. (2016). Mobile robot path planning using ant colony optimization. Paper presented at the *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA),* 1-6.

Sabudin, E. N., Omar, R., & Che Ku Melor, C. (2016). Potential field methods and their inherent approaches for path planning. *ARPN Journal of Engineering and Applied Sciences, 11*(18), 10801-10805.

Saffiotti, A. (1997). The uses of fuzzy logic in autonomous robot navigation. *Soft Computing, 1*(4), 180-197.

Sankaranarayanan, A., & Vidyasagar, M. (1990). A new path planning algorithm for moving a point object amidst unknown obstacles in a plane. Paper presented at the *Proceedings., IEEE International Conference on Robotics and Automation,* 1930-1936.

Sariff, N., & Buniyamin, N. (2006). An overview of autonomous mobile robot path planning algorithms. Paper presented at the *2006 4th Student Conference on Research and Development,* 183-188.

Saska, M., Macas, M., Preucil, L., & Lhotska, L. (2006). Robot path planning using particle swarm optimization of ferguson splines. Paper presented at the *2006 IEEE Conference on Emerging Technologies and Factory Automation,* 833-839.

Schneider, F. E., & Wildermuth, D. (2017). Using robots for firefighters and first responders: Scenario specification and exemplary system description. Paper presented at the *2017 18th International Carpathian Control Conference (ICCC),* 216-221.

Šeda, M. (2007). Roadmap methods vs. cell decomposition in robot motion planning. Paper presented at the *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation,* 127-132.

Tarbutton, J. A., Kurfess, T. R., & Tucker, T. M. (2010). Graphics based path planning for multi-axis machine tools. *Computer-Aided Design and Applications, 7*(6), 835-845.

Williams, S. B., Pizarro, O., Mahon, I., & Johnson-Roberson, M. (2009). Simultaneous localisation and mapping and dense stereoscopic seafloor reconstruction using an AUV. Paper presented at the *Experimental Robotics,* 407-416.

Wilmarth, S. A., Amato, N. M., & Stiller, P. F. (1999). MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. Paper presented at the *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. no. 99CH36288C), , 2* 1024-1031.

Yoshida, E., & Kanehiro, F. (2011). Reactive robot motion using path replanning and deformation. Paper presented at the *2011 IEEE International Conference on Robotics and Automation,* 5456-5462.

Yu, J., Su, Y., & Liao, Y. (2020). The path planning of mobile robot by neural networks and hierarchical reinforcement learning. *Frontiers in Neurorobotics, 14*

Zavershynskyi, M., & Papadopoulou, E. (2013). A sweepline algorithm for higher order voronoi diagrams. Paper presented at the *2013 10th International Symposium on Voronoi Diagrams in Science and Engineering,* 16-22.

Zhang, Q., Sun, J., Xiao, G., & Tsang, E. (2007). Evolutionary algorithms refining a heuristic: A hybrid method for shared-path protections in WDM networks under SRLG constraints. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 37*(1), 51-61.

taiwannews.com.tw
https://th.bing.com
https://www.thejakartapost.com/life/2018/06/28/end-of-the-line-for-asimojapans-famed-robot.html
*https://miro.medium.com/max/450/1*podzvpWd_ApSOo-SaYGw3w.jpeg*

https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Delaunay_Voronoi.svg/200px-Delaunay_Voronoi.svg.png

https://media.springernature.com/full/springer-static/image/art%3A10.1007%2Fs42154-019-00081

https://image.slidesharecdn.com/visibilitygraphs-/visibility-graphs

Appendix A (MATLAB Code)

Main Program

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Code and Documentation

% By Tajudeen Adeleke Badmos

% PhD MERI Path planning for mobile robot

% Sheffield Hallam University

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clear the data and Figure opened, then call function 'Main'

```
    clear:
    clc


    main
```

%% MAIN MATLAB code for main.fig

```
function varargout = main(varargin)
% Begin initialization code
gui_Singleton = 1:
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @main_OpeningFcn, ...
    'gui_OutputFcn',  @main_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',   []):
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1}):
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:}):
else
    gui_mainfcn(gui_State, varargin{:}):
end

end
% End initialization code

% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to Figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main (see VARARGIN)

    [handles] = initialize(handles);

    % Choose default command line output for main
    handles.output = hObject;

    % Update handles structure
    guidata(hObject, handles);

    % UIWAIT makes main wait for user response (see UIRESUME)
    % uiwait(handles.Figure1);
end

% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
    % varargout  cell array for returning output args (see VARARGOUT);
    % hObject    handle to Figure
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles and user data (see GUIDATA)

    % Get default command line output from handles structure
    varargout{1} = handles.output;
end

% --- Executes on button press in pushbuttonDrawObstacle.
function pushbuttonDrawObstacle_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonDrawObstacle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

   handles = drawObstacle(handles);

   if handles.Num_Object > 1
       %
       set(handles.pushbuttonVoronoiPath, 'Enable', 'on');
       %set(handles.pushbuttonRobotFinalPath, 'Enable', 'off');
   end

    % Save the handles structure.
    guidata(hObject,handles)
end


% --- Executes on button press in pushbuttonDrawRobot.
function pushbuttonDrawRobot_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonDrawRobot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    handles = drawRobot(handles);

    %
    if ~isempty(handles.Robot) && ~isempty(handles.Goal) &&
~isempty(handles.Edge_X1)
        set(handles.pushbuttonRobotFinalPath, 'Enable', 'on');
    end
% Save the handles structure.
    guidata(hObject,handles)
end
```

```
% --- Executes on button press in pushbuttonDrawGoal.
function pushbuttonDrawGoal_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonDrawGoal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    handles = drawGoal(handles);

    %
    if ~isempty(handles.Robot) && ~isempty(handles.Goal) &&
~isempty(handles.Edge_X1)
        set(handles.pushbuttonRobotFinalPath, 'Enable', 'on');
    end
    % Save the handles structure.
    guidata(hObject,handles)
end

% --- Executes on button press in pushbuttonVoronoiPath.
function pushbuttonVoronoiPath_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonVoronoiPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    X = ('MenuStart: Voronoi ');
    disp(X);
    tstart = tic;

    % create point obstacles at the edges
    [handles]= createObstaclePoints(handles);
    [handles]= drawVoronoi(handles);

    X = ['MenuEnd: Voronoi time= ', num2str(toc(tstart)) ];
    disp(X);

    set(handles.pushbuttonDrawRobot, 'Enable', 'on');
    set(handles.pushbuttonDrawGoal, 'Enable', 'on');
    %
    if ~isempty(handles.Robot) && ~isempty(handles.Goal) &&
~isempty(handles.Edge_X1)
        set(handles.pushbuttonRobotFinalPath, 'Enable', 'on');
    end

    set(handles.pushbuttonMoveFirstObstacle, 'Enable', 'on');
    set(handles.pushbuttonShowEpsilon, 'Enable', 'on');

    handles.radiobuttonLastObstacle.Value = 0;
    set(handles.radiobuttonLastObstacle, 'Enable', 'off');
    handles.radiobuttonScreen.Value = 1;


    % Save the handles structure.
    guidata(hObject,handles)
end

% --- Executes on button press in pushbuttonRobotFinalPath.
function pushbuttonRobotFinalPath_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonRobotFinalPath (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    X = ('MenuStart: Robot Final Path ');
    disp(X);
    tstart = tic;

    handles = drawRobotFinalPath(handles);

    X = ['MenuEnd: Robot Final Path time= ', num2str(toc(tstart)) ];
    disp(X);
    %
    if ~isempty(handles.Path)
        set(handles.pushbuttonBoundaryRobot, 'Enable', 'on');
    end

    % Save the handles structure.
    guidata(hObject,handles)
end

% --- Executes on button press in pushbuttonBoundaryRobot.
function pushbuttonBoundaryRobot_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonBoundaryRobot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% hObject    handle to pushbuttonDrawGoal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    handles = addBoundaryRobot(handles);

    % Save the handles structure.
    guidata(hObject,handles)
end

% --- Executes on button press in pushbuttonReset.
function pushbuttonReset_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonReset (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    if handles.radiobuttonLastObstacle.Value == 1
        handles = removeObstacle(handles);
    else
        handles = resetWorkspace(handles);
    end


    if handles.Num_Object < 2
        %
        set(handles.pushbuttonVoronoiPath, 'Enable', 'off');
        %set(handles.pushbuttonRobotFinalPath, 'Enable', 'off');
    end

    % Save the handles structure.
    guidata(hObject,handles)
end


% --- Executes on button press in pushbuttonMoveFirstObstacle.
function pushbuttonMoveFirstObstacle_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonMoveFirstObstacle (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Save the handles structure.
```

```
    set(handles.pushbuttonMoveFirstObstacle, 'Enable', 'off');

    X = ('MenuStart: Move First Obstacle');
    disp(X);
    tstart = tic;

    handles = movingFirstObstacle(handles);

    X = ['MenuEnd: Move First Obstacle time= ', num2str(toc(tstart)) ];
    disp(X);

    %
    set(handles.pushbuttonMoveFirstObstacle, 'Enable', 'on');

    guidata(hObject,handles)

end



% --- Executes on button press in pushbuttonWriteSimulationData.
function pushbuttonWriteSimulationData_Callback(hObject, eventdata,
handles)
% hObject    handle to pushbuttonWriteSimulationData (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%     T_Edges = table(handles.Edge_X1', handles.Edge_Y1', handles.Edge_X2',
handles.Edge_Y2', 'VariableNames',{'X1','Y1','X2','Y2'});
%     writetable(T_Edges, 'T_Edges.xlsx')

    Vertex_Cord = table(handles.Vertex_Cord(:,1), handles.Vertex_Cord(:,1),
'VariableNames',{'X','Y'});
    writetable(Vertex_Cord, '_Vertex_Cord.xlsx')

    %calculate voronoi path distance
    handles.VoronoiPath_distance = 0;
    for i=1:size(handles.VoronoiPath,1)
        a = handles.VoronoiPath(i,1:2);
        b = handles.VoronoiPath(i,3:4);
        d = norm(b-a);
        handles.VoronoiPath_distance = handles.VoronoiPath_distance + d;
    end
    TVoronoiPath =
table(handles.VoronoiPath(:,1),handles.VoronoiPath(:,2),handles.VoronoiPath
(:,3),handles.VoronoiPath(:,4),'VariableNames',{'X1','Y1','X2','Y2'});
    writetable(TVoronoiPath, '_VoronoiPath.xlsx')

    %calculate robot path distance
    handles.RobotPath_distance = 0;
    for i=1:size(handles.RobotPath,1)
        a = handles.RobotPath(i,1:2);
        b = handles.RobotPath(i,3:4);
        d = norm(b-a);
        handles.RobotPath_distance = handles.RobotPath_distance + d;
    end

    TRobotPath =
table(handles.RobotPath(:,1),handles.RobotPath(:,2),handles.RobotPath(:,3),
handles.RobotPath(:,4),'VariableNames',{'X1','Y1','X2','Y2'});
    writetable(TRobotPath, '_RobotPath.xlsx')

    %
    Edges_N = size(handles.VoronoiPath,1);
    Vertex_N = size(handles.Vertex_Cord,1);
    Robot_N = size(handles.RobotPath,1);
    T = table(handles.Epsilon, Vertex_N, Edges_N,
handles.VoronoiPath_distance, Robot_N, handles.RobotPath_distance, ...
        'VariableNames',{'Epsilon', 'Vertex_No','Edgdes_No',
'VoronoiPath_D', 'Robot_Edges_No', 'RobotPath_D'});
    writetable(T, '_Summary.xlsx')


end


% --- Executes on button press in pushbuttonShowEpsilon.
function pushbuttonShowEpsilon_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonShowEpsilon (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

    plot(handles.X_Total_points, handles.Y_Total_points, 'r.');
    plot(handles.Voro_Vertex(:,1), handles.Voro_Vertex(:,2), 'r.');
end
```

%% Initializing

```
function [handles] = initialize(handles)

    handles.W = [0 100 0 100];
    handles.D = 2;
    axis(handles.W);
    handles.i = 0;
    handles.Cx = 0;
    handles.Cy = 1;
    handles.xstep = 2;
    handles.ystep = 2;

    handles.X1 = [];
    handles.X1{1} = [];

    handles.Poly = [];
    handles.Poly{1} = [];
    handles.PPoly = [];
    handles.PPoly{1} = [];
    handles.showCircleRegion = 0;
    handles.PolyCircle = [];
    handles.PolyCircle{1} = [];
    handles.PPolyC = [];
    handles.PPolyC{1} =[];

    handles.Epsilon = 1;
    handles.Num_Object=0;

    handles.isBoundary = 0;

    handles.Robot = [];
    handles.Goal = [];
    handles.DynObstacle = [];
    handles.AllowMultipleRobot = 0;
    handles.AllowMultipleGoal = 0;
    handles.AllowMultipleDynObstacle = 0;

    %Total points
    handles.X_Total_points = 0;
    handles.Y_Total_points = 0;
    handles.All_cells_Number = 0;
    handles.Cell_start = 1;

    %
    handles.Voro_Vertex = [];
    handles.Voro_Cell = [];
    handles.Temp_Edge = [];
    handles.Path = [];
    handles.Vertex_Cord = [];

    handles.Edge_X1 = [];
    handles.Edge_X2 = [];
    handles.Edge_Y1 = [];
    handles.Edge_Y2 = [];

    handles.VoronoiPath = [];
    handles.RobotPath = [];
    handles.VoronoiPath_distance = 0;
    handles.RobotPath_distance = 0;
    %
    hold on;

    %
    set(handles.pushbuttonDrawRobot, 'Enable', 'on');
    set(handles.pushbuttonDrawGoal, 'Enable', 'on');
    set(handles.pushbuttonVoronoiPath, 'Enable', 'off');
    set(handles.pushbuttonRobotFinalPath, 'Enable', 'off');
    set(handles.pushbuttonBoundaryRobot, 'Enable', 'off');


    set(handles.pushbuttonMoveFirstObstacle, 'Enable', 'off');
    set(handles.pushbuttonShowEpsilon, 'Enable', 'off');

    %
    handles.radiobuttonLastObstacle.Value = 1;
    set(handles.radiobuttonLastObstacle, 'Enable', 'on');
    handles.radiobuttonScreen.Value = 0;

end
```

%% Drawing Obstacles

```
function [handles] = drawObstacle(handles)
%DRAW_OBSTACLE Summary of this function goes here
%   Detailed explanation goes here
     % get points
    p = 1;
    x=[];
    y=[];
    button = 1;
    while button == 1
        [px, py, button] = ginput(1);
        if button == 1 && p < 31
```

```
            x(p)=px; %point(1,1);
            y(p)=py; %point(1,2);
            if p > 1
                line([x(p-1), x(p)], [y(p-1), y(p)], 'Color', 'blue');
            end
            p = p+1;
        else
            k=0;
        end
    end

    [x,y]=fixPoints(x,y);

    % patch(x,y,'b');
    [handles] = getPoints(handles,[x',y'],1);
end
```

%% Drawing Robot (point)

```
function [handles] = drawRobot(handles)
%DRAW_ROBOT Summary of this function goes here
%   Detailed explanation goes here
    if isempty(handles.Robot) || handles.AllowMultipleRobot == 1
        %
        %Code for taking handles.Robot and End point as input
        handles.Robot = ginput(1);
        plot(handles.Robot(1),handles.Robot(2),'--
go','MarkerSize',10,'MarkerFaceColor','g');
        % drawnow;
    end
end
```

%% Drawing the goal

```
function [handles] = drawGoal(handles)
%DRAW_GOAL Summary of this function goes here
%   Detailed explanation goes here
    if isempty(handles.Goal) || handles.AllowMultipleGoal == 1
        handles.Goal  = ginput(1);
        plot(handles.Goal(1),handles.Goal(2),'--
ro','MarkerSize',10,'MarkerFaceColor','r');
        %drawnow;
    end
end
```

%% Drawing the Voronoi Diagram

```
function [handles]= drawVoronoi(handles)
%DRAWVORONOI Summary of this function goes here
%   Detailed explanation goes here
%set(handles.pushbuttonRobotFinalPath, 'Enable', 'off');
    % timetimer
    tstart = tic;

    % handles.Num_Object=length(handles.X1);
    handles.Num_Object = handles.i;

    %Getting Parameters of Voronoi Diagram
    [handles.Voro_Vertex, handles.Voro_Cell] =
voronoin([handles.X_Total_points' handles.Y_Total_points']);

    k=1;
    temp=0;
    for i=1:length(handles.All_cells_Number)
        L=length(handles.Voro_Cell{i});
      for j=1:L
          a=handles.Voro_Cell{i}(j);
          if(j==L)
              b=handles.Voro_Cell{i}(1);
          else
              b=handles.Voro_Cell{i}(j+1);
          end
          for l=1:handles.Num_Object
              if(temp==1)
                  temp=0;
                  break;
              end
              if (l==handles.All_cells_Number(i))
                  continue;
              end
              for m=handles.Cell_start(l):handles.Cell_start(l+1)-1
                  if((~isempty(find(handles.Voro_Cell{m}==a, 1))) &&
(~isempty(find(handles.Voro_Cell{m}==b, 1))))
                      handles.Temp_Edge(k,:)=[a b];
                      k=k+1;
                      temp=1;
                      break;
                  end
              end
          end
      end
    end

    handles.Temp_Edge=unique(handles.Temp_Edge,'rows');

    for i=1:length(handles.Temp_Edge)
        handles.Edge_X1(i)=handles.Voro_Vertex(handles.Temp_Edge(i,1),1);
        handles.Edge_X2(i)=handles.Voro_Vertex(handles.Temp_Edge(i,2),1);
        handles.Edge_Y1(i)=handles.Voro_Vertex(handles.Temp_Edge(i,1),2);
        handles.Edge_Y2(i)=handles.Voro_Vertex(handles.Temp_Edge(i,2),2);
    end

    % draw voronoi
    [g1, g2] = inObstacle(handles);
    for i=1:length(g1)
        handles.Edge_X1(i)= g1(i,1);
        handles.Edge_X2(i)= g2(i,1);
        handles.Edge_Y1(i)= g1(i,2);
        handles.Edge_Y2(i)= g2(i,2);
        plot([handles.Edge_X1(i) handles.Edge_X2(i)],[handles.Edge_Y1(i)
handles.Edge_Y2(i)],'color','g','LineWidth',2);

        handles.VoronoiPath(i,:) = [[handles.Edge_X1(i)
handles.Edge_Y1(i)],[handles.Edge_X2(i) handles.Edge_Y2(i)]];
    end

     X = ['Voronoi time ', num2str(toc(tstart)) ];
     disp(X);
```

```
end
```

%% Drawing the Collision free path for robot

```
function [handles] = drawRobotFinalPath(handles)
%DRAWROBOTFINALPATH Summary of this function goes here
%   Detailed explanation goes here
    tstart =tic:

    Vertex = unique(handles.Temp_Edge):
    N = length(Vertex):
    M = length(handles.Temp_Edge):

    for i=1:N
        handles.Vertex_Cord(i,:)= handles.Voro_Vertex(Vertex(i),:):
        Robot_distance(i)= norm(handles.Robot-handles.Vertex_Cord(i,:)):
        Goal_distance(i)= norm(handles.Goal-handles.Vertex_Cord(i,:)):
    end


    Voro_Graph = inf*ones(N):

    %Figure:
%     axis([0 100 0 100]):
%     hold on:

    for i = 1:M
        a= find(Vertex==handles.Temp_Edge(i,1)):
        b= find(Vertex==handles.Temp_Edge(i,2)):
        Distance = norm(handles.Vertex_Cord(a,:)-handles.Vertex_Cord(b,:)):
        Voro_Graph(a,b)=Distance:
        Voro_Graph(b,a)=Distance:
    end

    [~, Index_Robot]= min(Robot_distance):
    [~, Index_Goal]= min(Goal_distance):

    [handles.Path totalCost] = dijkstra(Voro_Graph,Index_Robot,Index_Goal):

    k = 0:
    % draw final path
    x=[handles.Robot(1) handles.Vertex_Cord(handles.Path(1),1)]:
    y=[handles.Robot(2) handles.Vertex_Cord(handles.Path(1),2)]:
    k=k+1:
    handles.RobotPath(k,:) = [[x(1) y(1)],[x(2) y(2)]]:
    plot(x,y,'-','color','m','LineWidth',3):
    drawnow:

    for i=1:length(handles.Path)-1
        x=[handles.Vertex_Cord(handles.Path(i),1)
handles.Vertex_Cord(handles.Path(i+1),1)]:
        y=[handles.Vertex_Cord(handles.Path(i),2)
handles.Vertex_Cord(handles.Path(i+1),2)]:
        plot(x,y,'-','color','m','LineWidth',3):
        k=k+1:
        handles.RobotPath(k,:) = [[x(1) y(1)],[x(2) y(2)]]:
        drawnow:
        hold on:
    end

    x=[handles.Vertex_Cord(handles.Path(i),1) handles.Goal(1)]:
```

```
    y=[handles.Vertex_Cord(handles.Path(i),2) handles.Goal(2)];
    plot(x,y,'-','color','m','LineWidth',3);
    k=k+1;
    handles.RobotPath(k,:) = [[x(1) y(1)],[x(2) y(2)]];
    drawnow;

    X = ['Robot final path time ', num2str(toc(tstart)) ];
    disp(X);

end
```

%% Moving Obstacle

```
function [handles] = movingFirstObstacle(handles)
%MOVINGFIRSTOBSTACLE Summary of this function goes here
%   Detailed explanation goes here
    % get an obstacle
    tstart = tic;

    p = 1;
    poly1 = handles.Poly{p};
    h1 = handles.PPoly{p};

    % up
    x = handles.Cx;
    y = handles.Cy;
%     handles.xstep=2;
%     handles.ystep=2;

    moves =1;
    for k=1:moves
        x = x + handles.xstep;
        y = y + handles.ystep;

        poly1m = translate(poly1,x,y);

        [handles, poly1m, h1] = redrawObstacle(handles, h1, poly1m, p);

        inOO = 0;

        % check if intercept with obstacles
        for i=1:handles.Num_Object-1
            if p ~= i
                poly2 = handles.Poly{i};
                % check if in or out
                [inF, inP, out] = collides(poly1m, poly2);
                if out
                    polyB = handles.Poly{handles.Num_Object};
                    [inF2, inP2, out2] = collides(poly1m, polyB);
                    if inF2
                       %[handles, poly1, h1]=
moveObstacleAndRedraw(handles, h1, poly1m, p);
                        inOO=0;
                    else
                        inOO =1;
                        [poly1m,handles.xstep,handles.ystep] =
obstacleCollision(poly1m,x,y,handles.xstep,handles.ystep);
                        break;
                    end
                else
                    inOO =1;
                    [poly1m,handles.xstep,handles.ystep] =
obstacleCollision(poly1m,x,y,handles.xstep,handles.ystep);
                    break;
                end
            end
        end

%          if inOO == 0
            [handles, poly1, h1]= moveObstacleAndRedraw(handles, h1,
poly1m, p);
%          end
        if inOO == 1
            handles.xstep = handles.xstep * -1;
            handles.ystep = handles.ystep * -1;
        end
    end
    %
    handles.PPoly{p} = h1;
    handles.Poly{p} = poly1;

    X = ['Move First Obstacle time ', num2str(toc(tstart)) ];
    disp(X);

end

function [poly1,xstep,ystep] = obstacleCollision(poly1,x,y,xstep,ystep)
    poly1 = translate(poly1,(x.*-2),(y.*-2));
%     [x1,y1]= changeDirection(x,y);
%     xstep = xstep * x1;
%     ystep = ystep * y1;


end

function [handles, poly1, h1] = redrawObstacle(handles, h1, poly1m, p)
    delete(h1);

    % move
    poly1 = poly1m;

    %
    if length(handles.PolyCircle) < p || isempty(handles.PolyCircle{p})
       h1 = plot(poly1);
    else

        points = poly1.Vertices;
        [cx, cy]= getCirclePoints(points);
        handles.PolyCircle{p} = polyshape(cx,cy);

        if handles.showCircleRegion == 1
            delete(handles.PPolyC{p});
            handles.PPolyC{p} = plot(poly1);
        end

        h1 = plot(handles.PolyCircle{p});
    end

    hold on
    drawnow
end
```

```
function [handles, poly1, h1]= moveObstacleAndRedraw(handles, h1, poly1m,
p)
    [handles, poly1, h1] = redrawObstacle(handles, h1, poly1m, p):

    % redraw voronoi
    [handles]=redraw_voronoi(handles, poly1, p):

    %redraw final path
    [handles]=redrawRobotFinalPath(handles):
end

function [handles]= redraw_voronoi(handles, poly1, p)
    hVor = findobj('type','Line','color','g','LineWidth',2):

    handles.X1{p} = []:
    handles.Poly{p} = poly1:
    handles.X1{p} = poly1.Vertices:

    % initialize for new voronoi

    %Total points

    handles.X_Total_points = 0:
    handles.Y_Total_points = 0:
    handles.All_cells_Number = 0:
    handles.Cell_start = 1:

    %
    handles.Voro_Vertex = []:
    handles.Voro_Cell = []:
    handles.Temp_Edge = []:
    handles.Path = []:

    handles.Edge_X1 = []:
    handles.Edge_X2 = []:
    handles.Edge_Y1 = []:
    handles.Edge_Y2 = []:

    for obs_i=1:handles.Num_Object
        [handles]= setObstaclePoints(handles, obs_i):
    end

    [handles]= drawVoronoi(handles):

    %pause(2):
    for h=1:length(hVor)
        delete(hVor(h))
    end

%     drawnow()
%     hold on
end

function [handles] = redrawRobotFinalPath(handles)
    hVor = findobj('type','Line','color','m','LineWidth',3):

%     [handles] = drawRobotFinalPath(handles):

    for h=1:length(hVor)
        delete(hVor(h))
    end
    [handles] = drawRobotFinalPath(handles):

    drawnow
    hold on
end
```

%% Reset workspace

```
function [handles] = resetWorkspace(handles)
%RESETWORKSPACE Summary of this function goes here
%   Detailed explanation goes here
    hold off
    cla(handles.axesMap):

    [handles] = initialize(handles):
end
```

Appendix B: How to use the GUI

We have made some screenshots below from our proposed algorithm on how to use the GUI for computing the GVD.

When the program starts, a window pops up as in fig.4.20

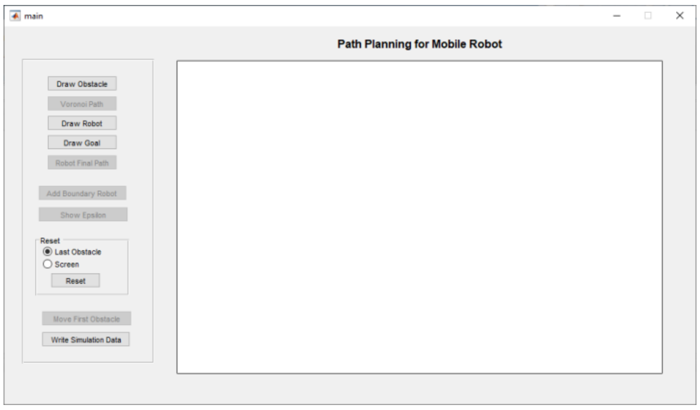


Figure 7. 1 GUI window

Push the “Draw Obstacle” to draw the obstacle whilst using the left mouse button to set the vertices of each obstacle in the workspace (the white space in the GUI), and using the right mouse button to end the process. Also, do the same procedure to draw the robot and to draw the goal by clicking the pushbutton “Draw Robot” and “Draw Goal” respectively. To compute the GVD, click on the pushbutton “Voronoi Path”.

To generate the shortest collision-free path, click the pushbutton “Robot Final Path”, then the path is shown from the robot to the goal. The pushbutton “Add Boundary Robot” can be used to describe the motion of a robot or an obstacle coming towards the configuration from a different configuration. The pushbutton “Reset” to start another set application on the GUI. To show the motion of a moving obstacle, push the button “Move First Obstacle” and lastly, the pushbutton “Write Simulation Data” is to generate some data on an excel sheet.

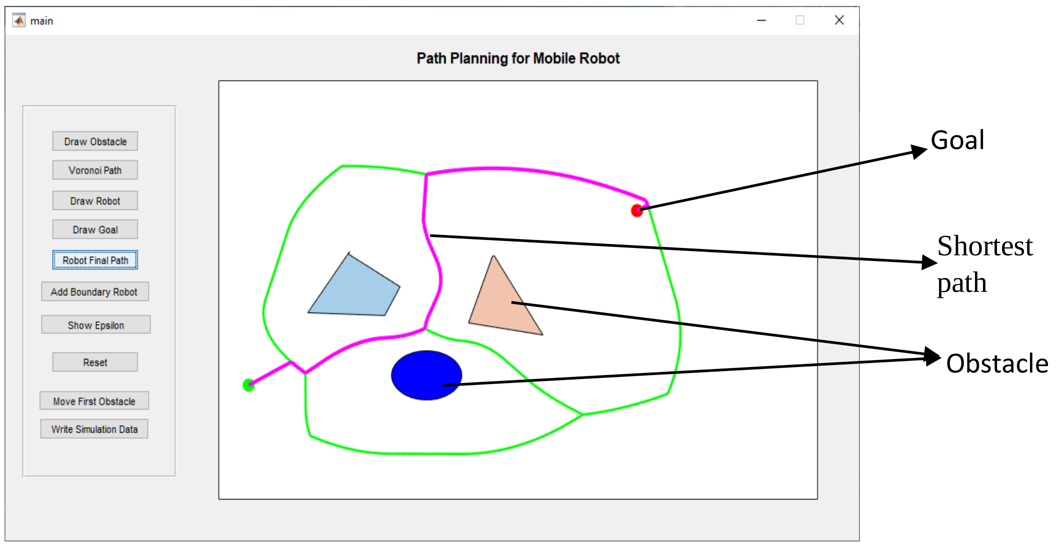


Figure 7.2 Using GUI buttons

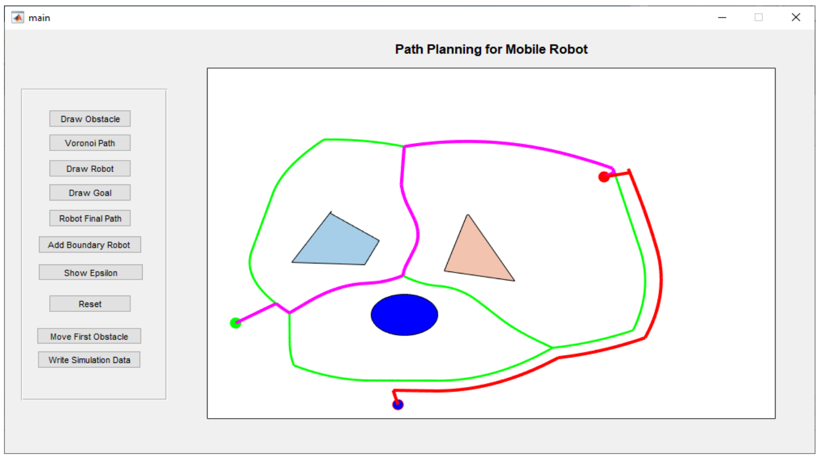


Figure 7.3 GUI buttons for "Add Boundary Robot"

Appendix C: Description of the Matlab Files

1. startUp
    a. clears the environment variables for a clean start
    b. calls the ***main*** program
2. main
    a. Creates the GUI
    b. Calls the ***initialize*** function.
    c. Open/display the GUI
    d. Controls and handles the GUI events, such as when buttons (Draw Obstacle) are clicked.
    e. Can call the following functions when their buttons are enabled
        i. drawObstacle
        ii. drawVoronoi
        iii. drawRobot
        iv. drawGoal
        v. drawRobotFinalPath
        vi. addBoundaryRobot
        vii. resetWorkspace
        viii. movingFirstObstacle
3. Initialize
    a. Initializes the GUI and program global variables
    b. Set Epsilon to determine point obstacle size
    c. Enables and disables GUI controls
    d. Disabled all buttons except the Draw Obstacle, Draw Goal and Draw Robot

4. drawObstacle
    a. Use to add one obstacle to the workspace.
    b. Click on the **|Draw Obstacle|** button.
    c. Obstacles are drawn using the mouse left click on the workspace.

   d. Mouse right-click ends obstacle drawing and connect the last point to the first point to form a solid polygon shape. For example to draw a rectangle left-click the four (4) points and end with a right-click to close the shape.
   e. Repeat the above step to add more obstacles to the workspace.
   f. Calls ***fixPoints*** functions
   g. Calls ***getPoints*** functions
   h. After adding more than one obstacle to the workspace, the button command to draw Voronoi is enabled.
5. fixPoints
   a. makes sure the points are not too smooth. Smooth point prevents epsilon division of edges.
6. getPoints
   a. Removes the obstacle drawing cursor lines.
   b. Draw the obstacle shape and fill it with a solid colour.
   c. Calls the ***setObstaclePoints*** function
7. setObstaclePoints
   a. Convert the obstacle edges to point obstacles using the Epsilon with the default value of 1.
   b. Stores the points with other obstacles points.
8. drawVoronoi
   a. Draws the boundary obstacle if not drawn already.
   b. Call the MATLAB in-built ***voronoin*** function with all the point obstacles.
   c. Store the return value of ***voronoin***, possible Voronoi vertices and cells for the point obstacles.
   d. Find the valid Voronoi diagram for each obstacle polygon from the obstacles points that areVoronoi vertices.
   e. Call ***inObstacle*** function to fix missing edges.
   f. Draws connecting edges of the Voronoi in green colour.

    g. Enable the **|Draw Robot|, |Draw Goal|** and **|Move First Obstacle|**
    h. Enable the **|Robot Final Path|** if Robot, Goal and Voronoi are in place.

9. inObstacle
    a. Removes all connecting Voronoi points in obstacles.
    b. Find unique connecting edges representing the Voronoi of the obstacles on the workspace.
10. drawRobot
    a. Use to draw the robot on the Workspace
    b. Click on the **|Draw Robot|** to add the robot to the Workspace.
    c. Click anywhere on the workspace using the left mouse button.
    d. A point dot of green colour will be added to the Workspace.
    e. Only one robot is allowed
    f. Enable the **|Robot Final Path|** if Robot, Goal and Voronoi are in place.
11. drawGoal
    a. Use to draw the goal or target on the Workspace
    b. Click on the **|Draw Goal|** to add the goal to the workspace.
    c. Click anywhere on the workspace using the left mouse button.
    d. A point dot of red colour will be added to the workspace.
    e. Only one goal (destination) allowed.
    f. Enable the **|Robot Final Path|** if Robot, Goal and Voronoi are in place.
12. drawRobotFinalPath
    a. Use to calculate and draw the shortest feasible distance between robot and goal.
    b. Calculate distance between each vertex and Robot, *dRi*
    c. Calculate distance between each vertex and Goal, dGi
    d. Calculate distance between each vertex to each other, dVi and stored in a square matrix.

e. Find the minimum distance dRi, dGi
f. Call a utility function Dijkstra to get the shortest path, using all the values of dVi and the minimum values of dRi and dGi.
g. Draws the shortest distance or path in magenta colour.
h. Enable the **|Add Boundary Robot|** button.

13. addBoundaryRobot
    a. Use to calculate and draw the shortest distance from the boundary robot and the goal parallel to the boundary.
    b. Click on the **|Add Boundary Robot|** button.
    c. Click anywhere on the workspace outside the boundary of the Voronoi Diagram.
    d. Add a new Robot.
    e. Calculate distance between each vertex and Robot, *dR2i*
    f. Calculate distance between each vertex and Goal, dGi
    g. Get the boundary vertices.
    h. Calculate distance between each boundary vertex to each other, dVi and stored in a square matrix.
    i. Find the minimum distance dR2i, dGi
    j. Call a utility function Dijkstra to get the shortest path, using all the values of dVi and the minimum values of dR2i and dGi.
    k. Compute a parallel path to the shortest distance.
    l. Draws the parallel shortest distance or path in red colour
    m.
14. resetWorkspace
    a. Use to rest the workspace to the initial state.
    b. Click on the **|Reset|** button.
    c. All the variables are cleared and reinitialized.
15. movingFirstObstacle
    a. Use to calculate and redraw the Voronoi path when the first obstacle becomes dynamic.

b. Click on the **| Move First Obstacle |** button.
c. Disable the **| Move First Obstacle |** button.
d. Set the step value for the x and y direction distance, *mstep*.
e. Set the number of steps the first obstacle will take before stopping, default = 10.
f. Get the first obstacle polygon.
g. Move the obstacle in a new x and y-direction
h. Check if the new position intercepts any other obstacle
i. And if Yes and if the minimum distance between the robot and the obstacle is less than the repulsive distance
j. Then apply deformation
k. If a new path found
l. Delete the old position and redraw at a new position.
m. And change direction and move in the new direction.
n. Repeat “step f” to i, for the default number of steps.