Generation of Non-Intersecting Continuous Paths using Bezier Curves for Multiple Robots

A Thesis submitted in partial fulfillment of requirement for the degree of Master of Computer Science and Engineering in the Department of Computer Science & Engineering, Jadavpur University

Submitted by

Utsa Roy

Registration Number: 140743 of 2017-18

Class Roll Number: 001710502002

Examination Roll No: M4CSE19001

Under the Supervision of

Dr. Chintan Kumar Mandal

Assistant Professor,
Dept. of Computer Science & Engineering

Faculty of Engineering and Technology

Jadavpur University

2019

Department of Computer Science & Engineering

Faculty of Engineering & Technology

Jadavpur University

To Whom It May Concern,

This is to certify that UTSA ROY, Registration Number: 140743 of 2017-18, Class Roll Number: 001710502002, Examination Roll Number: *M4CSE19001*, a student of MCSE, from the Department of Computer Science & Engineering, under the Faculty of Engineering and Technology, Jadavpur University has done a thesis report under my supervision, entitled as "Generation of Non-Intersecting Continuous Paths using Bezier Curves for Multiple Robots". The thesis is approved for submission towards the fulfillment of the requirements for the degree of Master of *Computer Science &Engineering*, from the Department of Computer Science & Engineering, Jadavpur University for the session 2018-19.

____

Dr. Chintan Kumar Mandal
(Supervisor)
Assistant Professor
Department of Computer Science and Engineering
Jadavpur University

____

Prof. Mahantapas Kundu
(Head of the Department)
Professor
Department of Computer Science and Engineering
Jadavpur University

____

Prof. Chiranjib Bhattacharjee
(Dean)
Professor
Faculty of Engineering and Technology
Jadavpur University

Department of Computer Science & Engineering

Faculty of Engineering & Technology

Jadavpur University

<u>Certificate of Approval</u>

(Only in case the thesis report is approved)

The forgoing thesis is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve this thesis only for the purpose for which it is submitted.

______________________________	______________________________
Signature of the Examiner	Signature of the Examiner
Date: _________________________	Date: _________________________

<u>Declaration of Originality &</u>

<u>Compliance of Academics Ethics</u>

I hereby declare that this thesis contains literature survey and original research work done by me, as part of my MCSE studies. All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

Name**: *Utsa Roy***

Registration No:*140743 of 2017-18*

Class Roll No: *001710502002*

Examination Roll No:*M4CSE19001*

Thesis Report Title: *Generation of Non-Intersecting Continuous Paths using Bezier Curves for Multiple Robots*

Utsa Roy

<u>Acknowledgement</u>

I would like to express my deepest gratitude to my advisor and guide Dr. Chintan Kumar Mandal, for his excellent guidance, caring, patience, and providing me with an excellent atmosphere for doing research. I strongly believe that I got a lot of encouragement and inspiration from him throughout the project. With his invaluable guidance, this work is a successful one. I am equally grateful to Dr. Mahantapas Kundu, Head of The Department, Computer Science & Engineering, Jadavpur University, for his support towards our department. Last but not least, I would like to thank my parents and all respected teachers for their valuable suggestions and helpful discussions.

Regards,

Utsa Roy
Department of Computer Science & Engineering
MCSE
Jadavpur University

Contents

List of Figures

List of Algorithms

Abstract

This thesis presents an algorithm for generating non-intersecting continuous shortest path for multiple robots. We consider an 2D-Euclidean environment with convex polygonal obstacles.

For solving the problem, we consider all robots assigned with an unique source and destination points. This approach of the algorithm sequential and it generates paths one after another based on an priority based on their Euclidean distance between their source and destination. In the beginning, a visibility graph generates all possible paths between the given source, destinations and vertices of the given polygonal obstacles. Based on the visibility graph, a shortest path (using Dijkstra's Algorithm) is found between each robot's given source to its corresponding destination. After that, the path gets modified and converted into a continuous path while updating the visibility graph in such a way such that the newly generated path act as an obstacle for subsequent robots. The proposed algorithm uses a classification method to decide the possibility of finding a non-intersecting path for a given pair. Based on the classification, the algorithm discards search for the shortest path for that pair after a certain number of attempts.

We draw conclusions of the proposed algorithm from graphs using the number of successful pairs and the number of obstacles. The obstacles are regular geometric polygons, like triangles, squares and hexagons kept in a grid array. The source and destinations are also kept in a regular patterns, to make the results consistent.

Chapter 1

Introduction

Robots are mechanical devices which are designed to perform one or more specific tasks. These robots can operate automatically or semi-automatically. Based on their mobility we can divide them into two categories - static robots and mobile robots.

The static robots are mainly used in industrial assembly line. These type of robots operates from a fixed position and have limited operating range/mobility. These robots can be used for the tasks like as welding, drilling, assembling, painting and packaging.

The mobile robots are those robots which can move from one position to another, while performing one or more tasks. These type of robots can be used in various fields like transportation/logistics, military, mining etc, farming.

Mobile robots can operate either automatically or with human intervention, known as semi-autonomous robots. Fully autonomous robots use data from sensors, odometers to take decisions while it circumvents in the environment. The locomotion or trying to find paths through the obstacles between source and destination using a set of instructions/algorithms are known as "path planning algorithms".

Path planning algorithms can be divided into two categories based on the knowledge of the given environment.

In the first category, when a map of the environment is not available or partially available. In this type of path planning the main target is to drive the robot from its source to its destination avoiding all the obstacles on the path, depending on its sensor inputs such as sonar range finder, Compass, IR range finder, Gyro, Positioning system, odometers etc.

In the second category, the map of the environment is completely known and path planning algorithms generates a predefined path based on that map. The robot follows the path to reach its destination. The objective of these path planning algorithms is to generate an optimal collision free path in minimal time.

There are several methods to plan a collision-free path in a completely known environment, such as cell decomposition method[15], visibility graph[5] and Potential Field method[18].

In most of the available literature, these paths are joined by discrete lines. However, these paths are not feasible for car like robots, as these paths contain sharp turns at points where two line segments meet. Due to these sharp turns of the path, these car like robots have to stop at turning points and after reorienting, follows the path further. This problem is related to the piano mover's problem [5][7][17][13]. To solve this issue, the paths are made continuous using Bezier[5], clothoid[2] and B-Spline[6] curves to name a few. The technique of converting a pre-planned discrete path to a continuous path is known as the **"path smoothing method"**.

There are two approach to solve this problem of path planning for multiple robots. In the first approach, the path planning algorithm generates a path for each robot independently without considering the paths of other robots and then at every intersection points of the paths control the velocity of the robots to avoid the collision. In second approach the algorithm generates a set of paths which does not intersect each other. We are focusing on the second approach to solve the problem of path planning for multiple robots.

1.1 Description of Keywords

In this section, we describe some important terms and concepts used in this thesis.

1.1.1 Visibility graph

A Visibility Graph[5] of a set of obstacles/polygons is a graph of inter-visible vertices. Every node in this graph is a vertex of obstacles/polygons and every edge in this graph represents a visible connection between two vertices. A visible connection implies that the straight line between two vertices does not intersect the interior of any obstacle/polygon. The edge between two visible vertices are called visibility edge.

The time complexity of a standard algorithm to implement a visibility graph from a set of obstacles is $O(E^2 \log E)$ and its naive version takes run-time $O(E^3)$. where E is the total number of obstacle edges. An example of visibility graph shown in Figure 1.1

1.1.2 Dijkstra's Algorithm

Dijkstra's Algorithm[4] is a single source shortest path algorithm. Given a graph and a source vertex in that graph this algorithm finds shortest paths from source to all vertices in that given graph.

This algorithm has many variants and one of them is given a graph and two vertices the algorithm finds the shortest path between those two vertices. We use this variant in our algorithm.

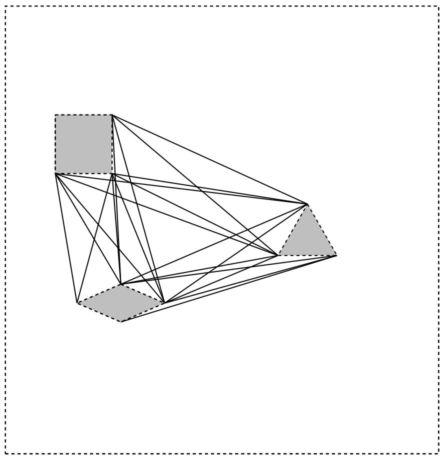


Figure 1.1: An example of visibility graph

The time complexity of this algorithm is $O(E + V \log E)$ where$E$ is the number of edges in the graph and V is the number of vertices in the graph.

1.1.3 Convex hull

The convex hull[4] of a set of points (P) denoted by CH(P) is the smallest convex polygon for which each point in set P is either on the boundary of the polygon or in its interior. An example shown in Figure 1.2. the thick lined polygon represents the convex hull of the given set of points.

1.1.4 Bezier Curve

A Bezier curve[16] is a parametric curve determined by a defining polygon, as shown in Figure 2.6. The degree of the polynomial defining the curve segment is one less than the number of defining polygon points. The curve generally follows the shape of the defining polygon. The first and last point on the curve are coincident with the first and last point of the defining polygon.

Mathematically a Bezier curve is defined by -

$$P(t) = \sum_{i=0}^{n} B_i J_{n,i}(t) \tag{1.1}$$

Where,

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{1.2}$$

and $0 \leq t \leq 1$

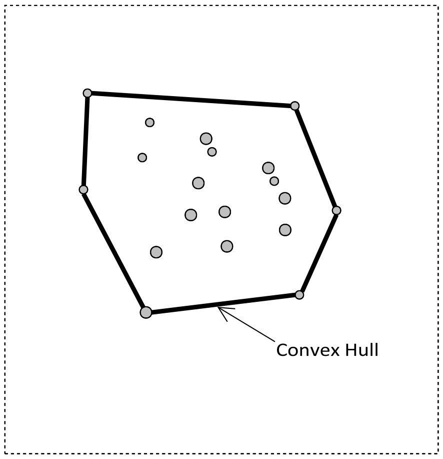


Figure 1.2: An example of Convex Hull

The Bezier curve which has triangle as defining polygon that is 2^{nd} degree of defining polynomial is known as quadratic Bezier curve [Figure 1.3b]. Mathematically,

$$P(t) = ((1-t)^2)p_0 + 2t(1-t)p_1 + t^2(p_2) \tag{1.3}$$

Where, p_0 is the start point p_1 is the control point and p_2 is the end point.

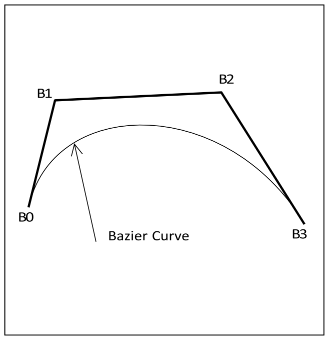
1.1.5 Turning Radius

The turning Radius[19] of a wheel based car like robot is the radius of the smallest circular turn that the vehicle is capable of making.

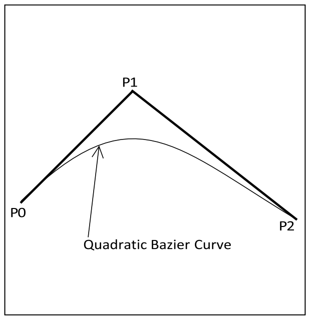
1.2 Related Works

A significant amount of work has been done in the area of path planning for multiple robots and path smoothing. However, in most cases the collision avoidance at the intersection points of the paths is done by controlling the velocity of the robots.

An alternative approach to the problem of path planning for multiple robots is introduced by Kamath and Yang[8]. In this paper they present a heuristic algorithm for generating non-intersecting paths for mobile robots in a multi-robot environment with polygonal obstacles. This algorithm works on two stage heuristic strategy to reach the solution. The outer level heuristic function works for a global optimum solution while the inner level heuristic function determines the best path for a robot at a given state of configuration. The algorithm uses a matrix to keep it informed of the status of the environment as new paths are formed. Paths generated by this algorithm are not continuous that is they are



(a) Bezier Curve and its defining Polygon .



(b) Quadratic Bezier curve.

Figure 1.3: An example of Bezier Curve.

not feasible for car like robots. This paths cannot be converted into continuous paths by using path smoothing methods as they are uses the vertices of the obstacles.

A Potential Fields based path planning for multiple robots is introduced by Warren [20]. This work describes a technique for coordinating the paths of multiple robots in the presence of obstacles using artificial potential fields. In this algorithm a path that avoids only the stationary obstacles is planned for the highest priority robot. Then, a trajectory for the next lower priority robot is planned so that it avoids both the stationary obstacles and the higher priority robot which is treated as a moving obstacle. This process is continued until trajectories for all of the robots have been planned. This algorithm works in a sequential manner and prefer the robots with higher priority first.

LaValle and Hutchinson[11] present a paper on Optimal Motion Planning for Multiple Robots. In the paper they addressed the problem in which the task is to simultaneously bring each of two or more robots from an initial configuration to a goal configuration. In addition to ensuring collision avoidance, each robot has a real-valued performance measure (or loss functional) to be optimized. This algorithm first generate path independently for each robots and to avoid the collision at the junction points a coordination diagram is used to plan a collision-free trajectory along the paths.

Another approach to the multi-robot path planning problem presented by the Svestka, Overmars [23]. Rather than the usual decoupled planning, they use a coordinated approach as a result the method is probabilistically complete, that is, any solvable problem will be solved within a finite amount of time. A data-structure storing multi-robot motions is built in two steps. First, a roadmap is constructed for just one robot. For this they use a Probabilistic Path Planner, which guarantees that the approach can be easily applied to different types robots.In the second step, a number of these simple roadmaps are combined into a roadmap for the multiple robots.

Bennewitzt et.al.[1] present an approach to optimize the priorities for decoupled and prioritized path planning methods for groups of mobile robots. This approach is a randomized method which repeatedly reorders the robots to find a sequence for which a plan can be computed and to minimize the overall path lengths. It is an any-time algorithm since it can be stopped at any point in time and can always return its currently best estimate.

Above mentioned paper are mainly describing the problem of path planning for multiple robots without considering the smoothness of the path. Now we discuss some path planning method where path smoothing is put under consideration. A Piecewise Bezier Curves based path planning is presented by Choi et.al.[3]. In this paper they present a practical path planning algorithm based on Bezier curves for autonomous vehicles operating under waypoints and corridor constraints. This algorithm generates the piecewise-Bezier-curves path such that the curves segments are joined smoothly with C^2 constraint which leads to continuous curvature along the path.

A research paper on continuous curvature path smoothing using cubic BÃĽzier spiral curves for non-holonomic robots presented by Yang et.al.[22]. This paper presents a path-smoothing algorithm over the piecewise linear path for non-holonomic robots. Based on the upper-bounded continuous curvature path-smoothing algorithm, three algorithms are proposed to enhance the path smoothing performance. First, an interactive algorithm, which fully utilizes extra distance margins of linear path, is suggested. Second, a bisection algorithm is proposed to relieve the violation of the maximum curvature constraints. Finally, an interpolating path-smoothing algorithm which passes intermediate points is suggested.

Latip and Omar[10] present a paper on the Feasible Path Generation Using Bezier Curves for Car-Like Vehicle. In this paper they show a way to generate continuous path by joining the path segments of a discrete path using Bezier Curve. This algorithm generates continuous path considering the vehicle wheelbase, maximum steering angle and maximum speed to ensure the path for the autonomous robot or vehicle is feasible.

In this present work, we are proposing an algorithm to find a set of non-intersecting path for a set of robots such that each path in that set is a continuous and optimal path for their corresponding robots.

1.3 Thesis Layout

The followings chapters are as follows : chapter 2 discusses the problem, assumptions and details of the proposed algorithm. Chapter 3 discusses the results of the proposed algorithm and Chapter 4 concludes the thesis with future works and challenges arising from this work.

Chapter 2

Concepts and Details of the proposed Algorithm

In this chapter, we discuss the proposed algorithm, which generates a set of non-intersecting continuous path for multiple robots.

2.1 Problem Statement

We consider a map constituting of a single or multiple number of convex polygons, which represent **static obstacles** in the 2D-Euclidean space.

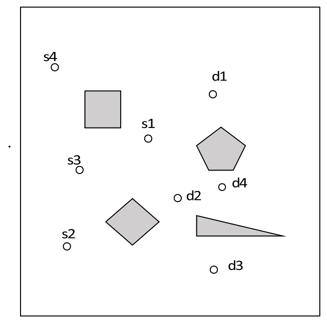
In this chapter, we propose an algorithm for the problem of finding the shortest non-intersecting continuous paths for a given *noRobots* robots with their corresponding source-destination pairs.

In Figure 2.1a, we give an example with a map. Given a map with obstacles(gray shaded areas) and four robots at sources s_1, s_2, s_3, s_4 and their corresponding destinations d_1, d_2, d_3, d_4 respectively. A possible solution of this problem is shown in Figure 2.1b

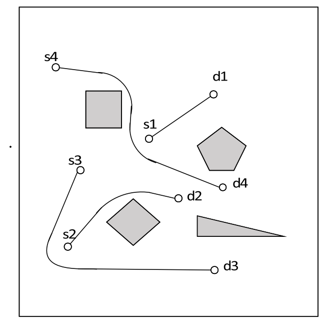
2.2 Assumptions

We state the assumptions for our proposed algorithm -

- We consider the environment in the 2D-Euclidean space.
- All robots will be considered as point robots, i.e. they will be represented as a co-ordinate in the 2D-Euclidean space instead of a polygon /robot shape.
- All the obstacles in the map are convex polygon and the map is reconstructed with Configuration Space using Minkowski sum. [5]



(a) A given map with obstacles and four robots with their corresponding source and destinations



(b) A possible solution for the problem.

Figure 2.1: An example a problem and its probable solution.

- The map of the environment will be known and it will be represented as a graph $G(V, E)$ where, V is the set of vertices of the obstacles and E is the set of all edges of the obstacles respectively.
- All source-destination pairs are point locations in the given map.

2.3 Details of the Algorithm

In this section, we discuss the working principle of the proposed algorithm, nonIntesectingPathPlanner [Algorithm 1].

The goal of this algorithm is to generate a set of shortest non-intersecting continuous paths for multiple robots.This algorithm works in two stages: the *first stage* generates non-intersecting discrete paths for all or maximum number of robots and *second stage* converts all the discrete paths into continuous paths.

2.3.1 Stage I

At first, a visibility graph [5][14][9] is generated using the convex polygonal obstacles and the given source-destination pairs. The function VisibilityGraph[1] [Algorithm 1] generates the visibility graph which is represented as $G_v(totalV, E_v)$, where $totalV$ is the union of vertices of the obstacles($obsV$) and the vertices of the source($Srce$) and destination($Dest$) pairs and E_v is the set of

[1]This function can be a standard visibility graph algorithm, as described in [5][21] having time complexity $O(N^2 \log N)$, where N is the number of edges of the obstacles.

Algorithm 1 nonIntesectingPathPlanner

```
Input: $obstacleMap(obsV, obsE)$, $noRobots$, $maxAttempts$,$tRadius$, $sourceDest(Srce, Dest)$
                    ▷ $noRobots$ also implies the total number of pairs of source-destination
                                                                                  ▷ Stage I
 1: $pathList \leftarrow \phi$                          ▷ $pathList$ is a list of paths for all Robots
 2: $totalV \leftarrow obsV \cup Srce \cup Dest$
 3: $E_v \leftarrow$VisibilityGraph($totalV$ ,$obsE$)
                          ▷ $E_v$ is a visibility edge matrix between all $v_i \in totalV$
 4: $wtMatrix \leftarrow$ CreateWtMatrix($totalV$, $E_v$)
 5: $SortedSrceDestList \leftarrow$ EuclideanDistSort($Srce$, $Dest$)
 6: $cHull \leftarrow$ convexHull($obsV$)
 7: for $j = 1$ to $noRobots$ do
 8:     $noIntersectCHull[j] \leftarrow$ IntersectConvexHull($cHull$, $Srce[j]$, $Dest[j]$)
 9: end for
10: for $i = 1$ to $noRobots$ do
11:     if SuccessClass($Srce[i]$, $Dest[i]$, $noIntersectCHull$) $\equiv$ $HIGH$ then
12:         $noAttempts \leftarrow maxAttempts$
13:     else if SuccessClass($Srce[i]$, $Dest[i]$, $cHull$, $noIntersectCHull$) $\equiv$ $MEDIUM$ then
14:         $noAttempts \leftarrow maxAttempts/2$
15:     else
16:         $noAttempts \leftarrow maxAttempts/4$
17:     end if
18:     $pathList \leftarrow pathList \cup$ Path($totalV$,$obsE$,$wtMatrix$,$tRadius$,
         $noAttempts$, $pathList$,$Srce[i]$,$Dest[i]$)
19: end for
                                                                                  ▷ Stage I
20: $contPath \leftarrow$ContinuousPath($pathList$,$tRadius$)
21: return $ContPath$
```

edges from v_i to v_j ($v_i, v_j \in totalV$), if and only if v_i and v_j are visible to each other. Two points v_i, v_j are visible to each other means that the line segment between them do not intersect the interior of any obstacle.

A weighted matrix $wtMatrix$ is calculated from the visibility graph edge set E_v by using the function CreateWtMatrix [Algorithm 2], where the weight is the Euclidean Distance between the two vertices $v_i(x_i, y_i)$ and $v_j(x_j, y_j)$:

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.1}$$

Algorithm 2 CreateWtMatrix

```
 1: function CreateWtMatrix
Input: $totalV, E_v$
Output: $wtMatrix$
 2:     for $i = 1$ to $|totalV|$ do
 3:         for $j = 1$ to $|totalV|$ do
 4:             if $E_v[i][j] \equiv 1$ then
 5:                 $WtMatrix[i][j] \leftarrow$ EuclideanDistance($totalV[i]$, $totalV[j]$)
 6:             else
 7:                 $WtMatrix[i][j] \leftarrow \infty$
 8:             end if
 9:         end for
10:     end for
11:     return $wtMatrix$
12: end function
```

The source-destination pairs are sorted [Algorithm 1: Line 5] into a priority list, $SortedSrceDestList$ based on their Euclidean distance in an ascending order. This priority list helps in deciding the priority of the source-destination pair for which the path will be generated first. The path, which is created first will act as a new obstacle for subsequent source-destination pairs in the priority list thus forcing the algorithm to generate non-intersecting paths.

Algorithm 3 SuccessClass

```
 1: function SuccessClass
Input: $Srce[i]$, $Dest[i]$, $noIntersectCHull$
Output: $LOW$ or $MEDIUM$ or $HIGH$          ▷ The classification of source-destination pair
 2:     $lineParameters \leftarrow$ lineSegmentEquation($Srce[i]$, $Dest[i]$)
                    ▷ lineParameters is a list of parameters of Line equation $ax + by + c = 0$
 3:     for $j = 1$ to $(i - 1)$ do
 4:         if $noIntersectCHull[j] > 1$ then
 5:             $prevLineParameters \leftarrow$lineSegmentEquation(Srce[j],Dest[j])
 6:             if Intersect($line$, $prevLineParameters$) then
 7:                 return $LOW$
 8:             end if
 9:         end if
10:     end for
11:     for $k = 1$ to $(i - 1)$ do
12:         if $noIntersectCHull[k] \equiv 1$ then
13:             $prevLineParameters \leftarrow$lineSegmentEquation(Srce[j],Dest[j])
14:             if Intersect($line$, $prevLineParameters$) then
15:                 return $MEDIUM$
16:             end if
17:         end if
18:     end for
19:     return $HIGH$
20: end function
```

The function SuccessClass as called in Algorithm 1: Line 10 [Algorithm 3] classifies a given pair of source-destination into three types of classes namely $HIGH$, $MEDIUM$ and LOW. These classes indicate the possibility of successfully finding a non-intersecting path for a given pair in that class.

The success class is found by the function SuccessClass of a given source-destination pair depends on the previous source-destination pairs. If the input is the i^{th} pair of source-destination of priority list(*SortedSrceDestList*) then its success class depends on the all $(i - 1)$ source-destination pairs in priority list (*SortedSrceDestList*).If any of the straight lines joining the $(i - 1)$ source destination pairs partition the convex hull into two regions and the given source and destination belong to separate regions then the success class of that pair will be LOW. In case there is no complete partition of the convex hull into two regions then it searches for lines which partially divide the convex hull that is lines which intersect the convex hull only once. If any of these lines joining the $(i-1)$ source-destination pairs intersect the line between given source-destination pair than the success class of that given pair will be $MEDIUM$. Except above two conditions all other pairs will be in success class $HIGH$.

The function IntersectConvexHull[Line 8 : Algorithm 1] takes a source-destination pair and the convex hull of the obstacles as input. This function returns the number of intersection between the line joining the given source-destination pair and the convex hull. The list *noIntersectCHull* store this numbers which indicate that the given pair partition the convex hull completely or partially. If the number is greater than 1 then the given pair partition the

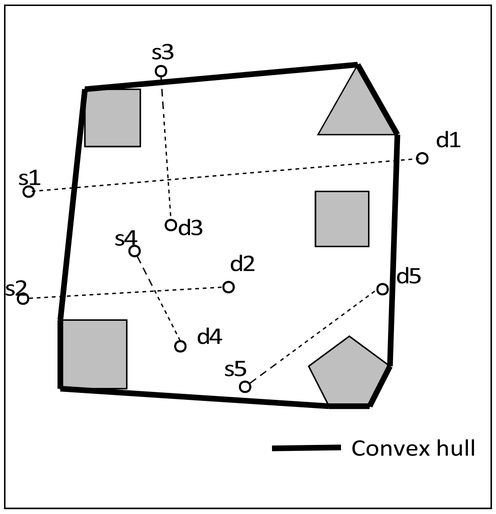


Figure 2.2: An example of classification of source-destination pairs

convex hull completely and if it is equal to 1 then the given pair partition the convex hull partially. otherwise, that is in case of zero the given pair dose not partition the convex hull.

The function SuccessClass is illustrated in Figure 2.2. The thick line is the convex hull of the obstacles. The source-destination are taken sequentially from the priority list *SortedSrceDestList* as $s_1 - d_1$, $s_2 - d_2$, $s_3 - d_3$, $s_4 - d_4$ and $s_5 - d_5$. The first pair $s_1 - d_1$ is classified as $HIGH$, as it is the first pair in priority list *SortedSrceDestList*. The pair $s_1 - d_1$ also partitions the convex hull into two regions. For the second pair, $s_2 - d_2$ as their source and destinations do not belong to separate regions of the convex hull, $s_2 - d_2$ will be classified as $HIGH$. For $s_3 - d_3$: the given source s_3 and destination d_3 belong to separate regions of the convex hull divided by the line between $s_1 - d_1$ [Figure 2.2], thus classifying it as LOW. Considering the fourth pair $s_4 - d_4$: s_4 and d_4 do not fall in the different regions partitioned convex hull. However, the pairs $s_2 - d_2$ and $s_3 - d_3$ partially divides the convex hull and the line joining $s_4 - d_4$ intersects $s_2 - d_2$. For this reason, the pair of $s_4 - d_4$ will be classified as $MEDIUM$. For the last pair $s_5 - d_5$, which neither intersects lines $s_2 - d_2$, $s_3 - d_3$ or $s_4 - d_4$ nor its source and destination belong to separate regions of the partition of the convex hull, $s_5 - d_5$ will be classified as $HIGH$.

Depending on the classification by SuccessClass for a given source-destination pair, the function Path [Algorithm 4] tries to find a non-intersecting path within a specified number of attempts, else it *discards* that pair. An attempt of $HIGH$ indicates the maximum number of attempts for which a search will continue for finding a successful non-intersecting path; $MEDIUM$ and LOW consequently

Algorithm 4 PATH

```
1: function PATH
Input: $totalV$, $obsE$, $wtMatrix$, $tRadius$, $noAttempts$, $pathList$, $Srce[i]$, $Dest[i]$
Output: $path$

2:     $refPath$ ←DIJKSTRA($wtMatrix$, $Srce[i]$, $Dest[i]$)
3:     $path$ ←PATHSHIFTING($refPath$, $tRadius$)
4:     $collidingEdge[2]$ ← COLLISION($path$, $totalV$, $obsE$, $pathList$)
                 ▷ collidingEdge[1] is the row number and collidingEdge[2] is the column
        number of $wtMatrix$ respectively
5:     if $collidingEdge[1] \equiv -1$ then
6:         for k = 2 to (NOVERTEXINPATH($path$) -1) do
7:             for $row$ = 1 to $|totalV|$ do
8:                 for $col$ = 1 to $|totalV|$ do
9:                     if INCIDENT($refPath[k], wtMatrix[row][col]$) then
10:                        $wtMatrix[row][col] \leftarrow \infty$
11:                    end if
12:                end for
13:            end for
14:        end for
15:        for k = 1 to NOEDGEINPATH($path$) do
16:            for $row$ = 1 to $|totalV|$ do
17:                for $col$ = 1 to $|totalV|$ do
18:                    if INTERSECT($pathEgde[k]$, $wtMatrix[row][col]$) then
19:                        $wtMatrix[row][col] \leftarrow \infty$
20:                    end if
21:                end for
22:            end for
23:        end for
24:        return $path$
25:    else
26:        $wtMatrix[collidingEdge[1]][collidingEdge[2]] \leftarrow \infty$
27:        $noAttempts \leftarrow noAttempts - 1$
28:        if $noAttempts \equiv 0$ then
29:            return $\Phi$
30:        else
31:            PATH($totalV$, $obsE$, $wtMatrix$, $tRadius$, $noAttempts$, $pathList$,
        $Srce[i]$, $Dest[i]$)
32:        end if
33:    end if
34: end function
```

will be lower fractions of $HIGH$. For our function, SUCCESSCLASS, we have chosen $MEDIUM$ as half of the $HIGH$, LOW one-fourth of the $HIGH$ and $HIGH$ will be an high integer value.

The Algorithm NONINTESECTINGPATHPLANNER [Algorithm 1] takes source-destination pairs one by one from a sorted list *SortedSrceDestList* and call the function PATH with the following inputs: vertex set of the visibility graph ($totalV$), obstacle edge set ($obsE$), weighted matrix of visibility graph edge set ($wtMatrix$), turning radius[2] of robots ($tRadius$), number of attempts($noAttempts$), a list of previously generated paths($pathList$), and source-destination ($Srce[i]$,$Dest[i]$) pair.

The function PATH [Algorithm 4] finds the shortest path from the given source $Srce[i]$ to its destination $Dest[i]$ by using the function DIJKSTRA [4], [3] from the $wtMatrix$ and the source-destination ($Srce[i]$,$Dest[i]$) pair. This path will be refer as "*Reference Path*" henceforth.

As the *Reference Path* consists of discrete lines, it is made continuous by joining the line segments using Bezier curves. This path smoothing process is done by the function CONTINUOUSPATH [Algorithm 6]]. However, the *Reference Path* cannot be used directly in the path smoothing process, as it is formed from the vertices and edges of the obstacles. The continuity of the discrete lines uses approximation curve such as Bezier curves [function CONTINUOUSPATH in Algorithm 6]. These curves does not go through the intermediate control points, which results in the continuous path intersecting the obstacles as shown in Figure 2.3. To avoid this, the discrete lines of the path are shifted to a certain distance. This path shifting is processed by the function PATHSHIFTING [Algorithm 5], which shifts the path segments depending on the angles at the turning points and turning radius($tRadius$) of the robot.

The shifting distance d is determine by a formula which is mentioned in stage II.

To shift the paths, the function PATHSHIFTING [Algorithm 5] first determines the equation of the line which bisect the angle θ at the turning point t_1 as shown in Figure 2.4. Let, L_1 and L_2 be the two-line segments at the turning point $t1$ given as

$$L_1 \equiv \quad a_1x + b_1y + c_1 = \quad 0 \tag{2.2}$$
$$L_2 \equiv \quad a_2x + b_2y + c_2 = \quad 0 \tag{2.3}$$

The equation of the line L_A, which bisects angle $\angle\theta$ at turning point t_1 is :

$$L_A \equiv \frac{a_1x + b_1y + c_1}{\sqrt{a_1^2 + b_1^2}} = \frac{a_2x + b_2y + c_2}{\sqrt{a_2^2 + b_2^2}} \tag{2.4}$$

[2]The turning radius of a robot is the radius of the smallest circular turn that the robot is capable of making.

[3]The function DIJKSTRA is the Dijkstra's algorithm with time complexity $O(|E_v| \log |totalV|)$.

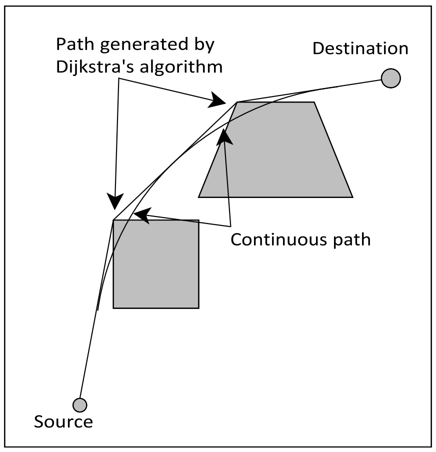


Figure 2.3: An example of continuous path intersecting an obstacle.

Algorithm 5 PathShifting

```
 1: function PathShifting
Input: $refPath$, $tRadius$
Output: $shiftedPoints$
 2:     $shiftedPoints \leftarrow \Phi$
 3:     $shiftedPoints \leftarrow shiftedPoints \cup refPath[0]$
 4:     for i=2 to (NoVertexInPath($Path$)-1) do
 5:         $pVertex \leftarrow refPath[i-1]$                      ▷ pVertex is Previous Vertex
 6:         $tVertex \leftarrow refPath[i]$                        ▷ tVertex is target Vertex
 7:         $nVertex \leftarrow refPath[i+1]$;                     ▷ nVertex is next Vertex
 8:         $line_1Parameters \leftarrow$ LineSegmentEquation($pVertex$, $tVertex$)
 9:         $line_2Parameters \leftarrow$ LineSegmentEquation( $tVertex$, $nVertex$)
10:         $lineAParameters \leftarrow$ AngleBisectingLine($line_1Parameters$, $line_2Parameters$)
                        ▷ LineA which bisected the angle between $line_1$ and $line_2$
11:         $\theta \leftarrow$ AngleBetweenLines($line_1Parameters$, $line_2Parameters$)
12:         $d \leftarrow \frac{tRadius*\cos(\frac{\theta}{2})}{\tan(\frac{\theta}{2})}$
13:         $shiftedPoint \leftarrow shiftedPoint \cup$ pointOnLine($lineA$, $tVertex$, $d$)
14:     end for
15:     $shiftedPoints \leftarrow shiftedPoints \cup refPath[i+1]$
16:     return $shiftedPoints$
17: end function
```

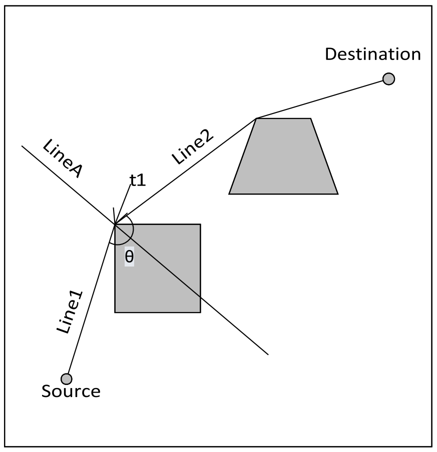


Figure 2.4: L_A bisecting the $\angle\ \theta$ at turning point t_1.

On this line (L_A) the function PathShifting determine a shifted point$(sp)(x_{sp}, y_{sp})$, which is d distance away from the turning point t_1 and in opposite direction to the obstacle [Figure 2.5a] using the following method -

Let the coordinate of point t_1 is(x_{t_1}, y_{t_1}) and the equation of line L_A is-

$$L_A \equiv a_A x + b_A y + c_A = 0 \tag{2.5}$$

If (x_{sp}, y_{sp}) is the point d distance away from the point t_1 on the line L_A then

$$d = \sqrt{(x_{t_1} - x_{sp})^2 + (y_{t_1} - y_{sp})^2} \tag{2.6}$$

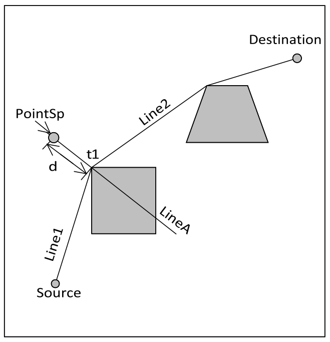
and,

$$a_A x_{sp} + b_A y_{sp} + c_A = 0 \tag{2.7}$$

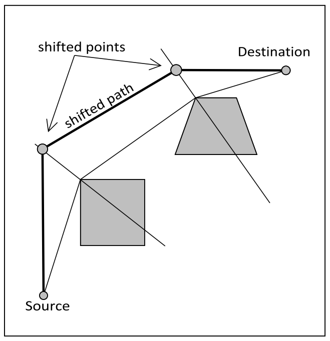
Solving equations 2.6 and 2.7 we get two solutions for (x_{sp}, y_{sp}). One point is toward the obstacle from t_1 and another is in opposite direction to obstacle from t_1.We choose the coordinate of point sp which will be the point in opposite direction to obstacle from t_1.

For all turning points the function PathShifting will determine shifted points. Then it joins them to generate the shifted path as [Figure 2.5b].

The function Collision [Line 4 : Algorithm 4] check for collisions amongst the shifted path, obstacles and previously generated paths. The collisions are checked by checking line intersections among the line segments of the shifted path, obstacle edges and previously generated paths. If line collision/intersection is detected, then the edge number of the shifted path, for which collision occurs is noted. This edge number of the shifted path is same as its corresponding edge number in *Reference Path* for which that edge in the shifted path get generated. The function Collision returns a vector, *collidingEdge*, the row and column



(a) *sp* is a point which is d distance away from t1 on the L_A and opposite direction to the obstacle.



(b) Shifted path generated by joining shifted points.

Figure 2.5: An example of PATHSHIFTING [Algorithm 5]

number of the colliding edge in $wtMatrix$. If no collision is detected, then the vector will be -1.

On detection of a collision, the colliding edge will be deleted from $wtMatrix$ and *noAttempt* is reduced by 1. If *noAttempts* become 0 then the function PATH [Algorithm 4] returns a null list, Φ indicating that no path is found between source-destination. But if *noAttempts* $\neq 0$, then it makes a recursive call to the function PATH with updated $wtMatrix$ and *noAttempts*.

In case of no collision detected by function COLLISION the function PATH [Algorithm 4] deletes all the edges from $wtMatrix$ which are incident to the vertices of *Reference Path*. An edge incident to a vertex or not that determines by the function INCIDENT[Line 9 : Algorithm 4] which takes a vertex and an edge as input and returns true if the edge incident to that vertex and if not then false.

The function PATH [Algorithm 4] also deletes all the edges from $wtMatrix$ which intersect any edge of the shifted path. The function INTERSECT[Line 18 : Algorithm 4] determines the intersecting edges of $wtMatrix$. After these two updates of the $wtMatrix$ the function PATH returns the shifted path to Algorithm 1. which is added to the *pathList*.

The former updates of $wtMatrix$ deletes edges of the visibility graph whose end vertices are became not visible to each other after generating a path.

After generating non-intersecting paths for maximum number of robots the Algorithm 1 enters in the Stage II.

2.3.2 Stage II

In stage II, the function CONTINUOUSPATH [Algorithm 6] converts all the paths which, generated from the Stage I into continuous paths using quadratic Bezier curve based path smoothing method.

Algorithm 6 ContinuousPath

```
1: function ContinuousPath
Input: $pathList$, $tRadius$
Output: $contPath$
2:     $contPath \leftarrow \phi$
3:     for $i = 1$ to NoPath($PathList[i]$) do
4:         $sPath \leftarrow \phi$
5:         if noTurn($pathList[i]$) $\equiv 0$ then
6:             $sPath \leftarrow pathList[i]$
7:         else
8:             $sPath \leftarrow$VertexAt($PathList[i]$, $first$)
9:             for $j = 1$ to NoTurnigPoints($pathList[i]$) do
10:                $\theta \leftarrow$ angleOfturning($j$)
11:                $r \leftarrow \frac{turningRadius}{\tan(\frac{\theta}{2})}$
12:                $c \leftarrow$ CircleEquation($turningPoint, r$)
                           ▷ c is a list parameters of the circle equation $(x-h)^2 + (y-k)^2 = r^2$
13:                $intersectingPoints \leftarrow$ CircleIntersectPathSegment($c, pathList[i]$)
14:                $sPath \leftarrow$Bezier($intersectingPoint_1$, $turningPoint_j$, $intersectingPoint_2$)
15:            end for
16:            $sPath \leftarrow$VertexAt($PathList[i]$, $last$)
17:        end if
18:        $contPath \leftarrow contPath \cup sPath$
19:    end for
20:    return $contPath$
21: end function
```

The quadratic Bezier curve is a parametric curve represented as Equation 2.8 -

$$P(t_b) = ((1-t_b)^2)p_0 + 2t_b(1-t_b)p_1 + t_b^2(p_2) \tag{2.8}$$

where $P(t_b)$ is a point on the curve, p_0 is the starting point, p_1 is the control point, p_2 is the end point and $0 < t_b < 1$. An example of quadratic Bezier curve shown in Figure 2.6

To make a path continuous the function ContinuousPath [Algorithm 6] apply quadratic Bezier curve to each turning points on that path.

We illustrate the function ContinuousPath in the Figure 2.7, the circle is the turning circle of the robots. A turning circle of a robot or vehicle is the smallest circular turn that the robot or vehicle is capable of making. Therefore, To make the path feasible for the robots no curve on the path should be smaller than the curve of that turning circle. That implies that, the radius of any curve on a path must always be greater or equal to the turning radius. As shown in the figure 2.7 paths segments are tangent to the turning circle.The tangent points are p_s and p_e. To make the continues path feasible for the robots the Bezier curve must be starts from a point which is not in between point p_s and point t. That is the distance between starting point of the Bezier curve and and the point t must be greater or equal to the distance between p_s and t. The same condition applied for the end point of the Bezier curve. As the path segments are tangent to the same circle from same point the length of p_s to t and t to p_e will be same.

Now to find the distance between p_s and t consider a line joining t and centre of the turning circle c. The triangle Δ ctp_s is a right triangle and the angle at point t is θ.

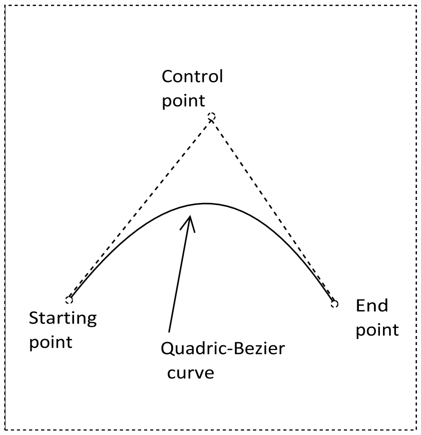


Figure 2.6: Quadratic Bezier curve

The distance between p_s and t will be $r = tRadius * tan\frac{\theta}{2}$ where, $tRadius$ is the turning radius of the robots

As mentioned earlier quadratic Bezier curve require three control points. To find these three control points,the function ContinuousPath [Algorithm 6] generates a circle at the turning point(t) taking radius r. This circle will intersect the path at two points p_0 and p_1 as shown in Figure 2.8a. These two points p_0 and p_1 act as the starting and ending point for the quadratic Bezier curve and the turning point (t) acts as the control point.Using these three control points the function ContinuousPath [Algorithm 6] generates the quadratic Bezier curve for that turning point.

This method is applied to all the turning points of the given path resulting in a continuous path. An example given is in Figure 2.8b. After repeating the function for all paths in the $pathList$, the funtion ContinuousPath [Algorithm 6] constructs a set of continuous path for all ($noRobots$) or a maximum number of source destination paths .

As mentioned earlier in Stage I that the function PathShifting [Algorithm 5] will shift the path d distance away from the Reference Path. Now we describe the formula to determine the shifting distance d.

Consider the Figure 2.9, p_s, t and p_e are the three control points of the Bezier curve(Shown in Figure 2.9 by dotted curve). The distance between p_s to t and t to p_e is r ($r = tRadius * tan\frac{\theta}{2}$).

Let a and c be the middle points of the line segments $p_s - t$ and $t - p_e$ respectively. Now according to the properties of Bezier curve the peek point b of the Bezier curve belongs to the straight line $a - c$. The Triangle Δabt is the right triangle and the angle at the point t is θ.

Therefore, the distance between point b and point t is $d = \frac{r}{2} * \cos(\frac{\theta}{2})$. If the path get shifted by this distance d then the Bezier curve will go through the point t that is the resulting continuous path will not intersect the obstacle.

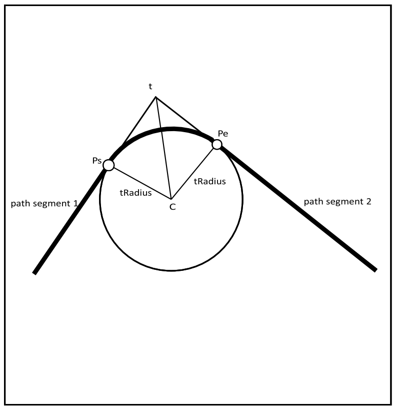
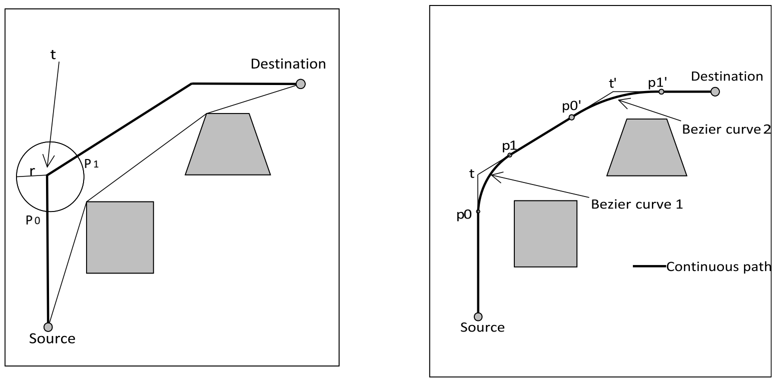


Figure 2.7: Relation between Bezier curve and turning radius of the robot.



(a) Circle with radius r and centre t intersect the path at two points p_0 and p_1.

(b) The Bezier curves are drawn using their respective control points for all turning points and continuous path is generated.

Figure 2.8: An example of the path smoothing method.

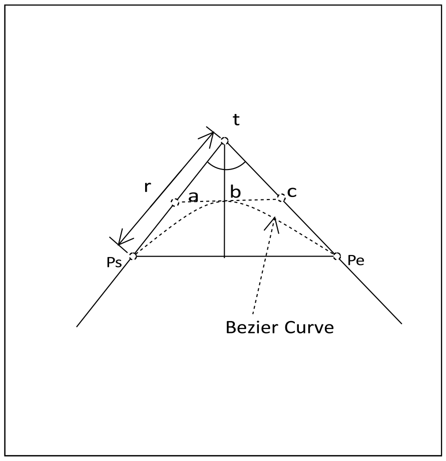


Figure 2.9: Relation between Bezier curve and its control points.

Chapter 3

Discussion of Results

In this chapter we discuss the results of the simulation of the proposed algorithm.

The proposed Algorithm is simulated in javafx using the IDE IntelliJ IDEA. This algorithm is tested with three types of obstacle maps namely triangle, square and Hexagon.These three types of maps have variable number of obstacles. Variation of the obstacle number is in following order $(4, 9, 16, 25, 36, 49, 64, 81)$. Therefore, For each type there are eight maps available.

3.1 Input Patterns for Source-Destination of the Robots

The source-destination pairs are given in two types of patterns, a linear and other is zigzag as Shown in Figure 3.1a and Figure 3.1b respectively. The source and destination points in each pattern are organized in four types of formation. This formations are illustrated in Figure 3.2.

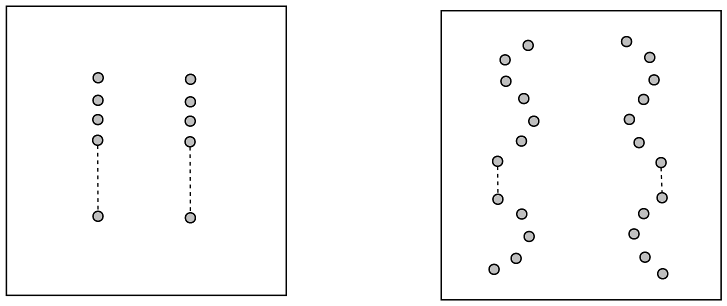
3.2 Results

The results of the simulation given below in form of graphs with *Number of Obstacle* Vs *Number of Successful Non-intersecting Paths.* **Number of obstacles also can be interpreted as the number of vertices into the obstacles.**

The graphs are plotted using the software "Origin Pro 8".

3.3 Discussion of results

From the graphs in Figure 3.3 to Figure 3.26 we can conclude that for formation 1 and 2 in both pattern 1 and pattern 2 the number of successful pairs will reduce if the number of obstacles increase. The number of obstacle(x) and number of successful(y) related by the function $y = a * x^b$.The values of b's in this function $y = a * x^b$ always less than 0. Therefore we can substitute the term x^b by $1/x$.

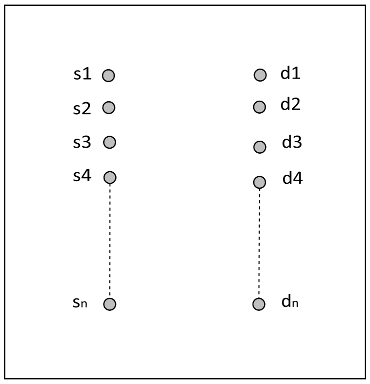


(a) Pattern 1 (b) Pattern 2

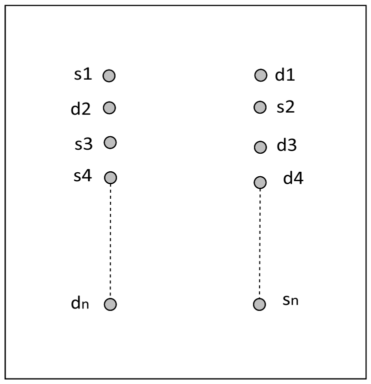
Figure 3.1: The Linear and Zigzag pattern of source and destination.

That implies the number of successful pairs inversely proposal to the number of obstacles.That is in these types of source-destination arrangement the path intersection occurs due to obstacle avoidance. Therefore, higher the number of obstacles lesser the number of successful pairs.

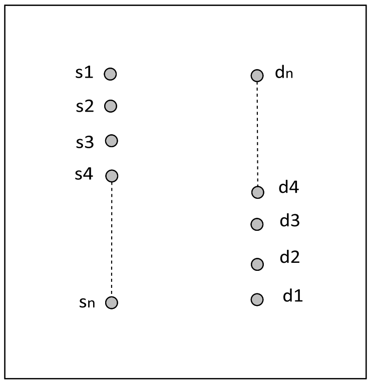
In case of formation 3 and 4 in both pattern 1 and 2 the intersection between paths occurs because of the arrangement of source-destination points.Due to their arrangement these paths will intersect each other if there is no obstacles. The number of obstacles does not effect the success rate compare to formation 1 and 2. As it can observed from the graphs that, for the formation 3 and 4 the relation between number of obstacle(x) and number of success(y) is linear define by the function $y = A + B * x$ and approximately constant that is value of B approximately 0.



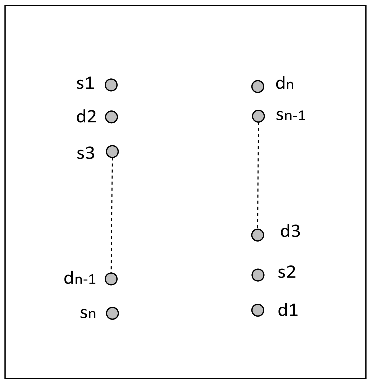
(a) Formation 1



(b) Formation 2

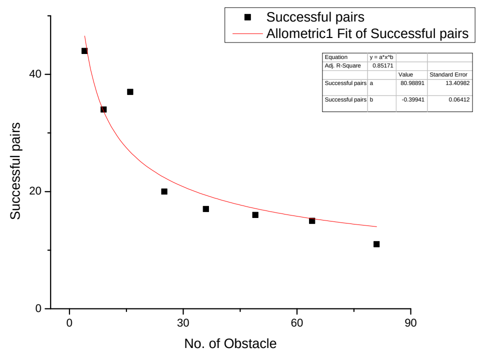


(c) Formation 3



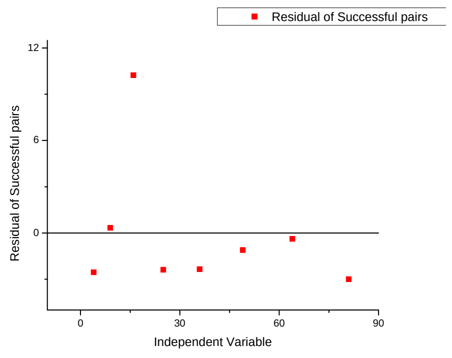
(d) Formation 4

Figure 3.2: Formations of source and destination.



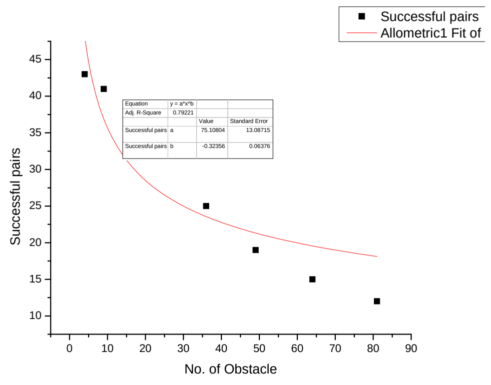
Equation	y = a*x^b		
Adj. R-Square	0.85171		
		Value	Standard Error
Successful pairs	a	80.98891	13.40982
Successful pairs	b	-0.39941	0.06412

(a) Triangular obstacle map with source-destination pattern 1 formation 1. Function of the curve is $y = a * x^b$ and $a = 80.98$, $b = -0.39941$



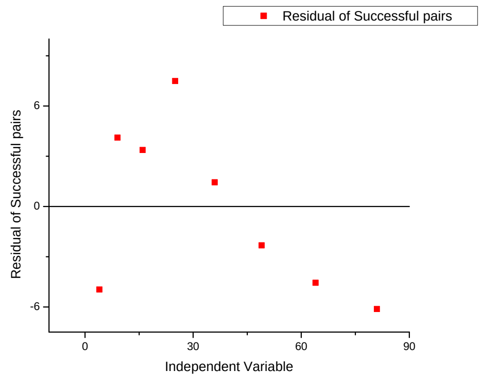
(b) Residual graph of the given graph is 3.3a

Figure 3.3: Pattern 1, Formation 1 for Obstacle $\triangle$



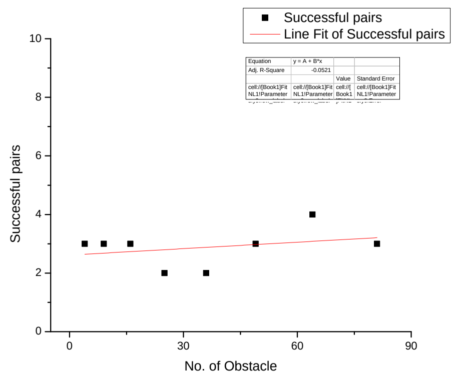
Equation	y = a*x^b		
Adj. R-Square	0.79221		
		Value	Standard Error
Successful pairs	a	75.10804	13.08715
Successful pairs	b	-0.32356	0.06376

(a) Triangular obstacle map with source-destination pattern 1 formation 2. Function of the curve is $y = a * x^b$ and $a = 75.10$, $b = -0.32356$

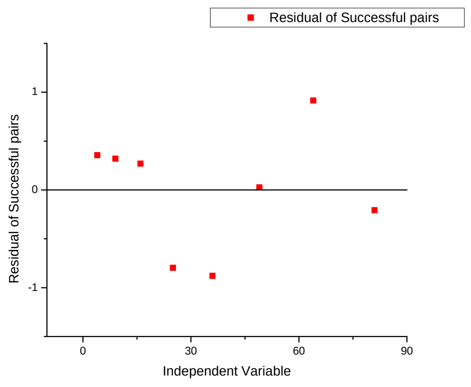


(b) Residual graph of the given graph is 3.4a

Figure 3.4: Pattern 1, Formation 2 for Obstacle $\triangle$

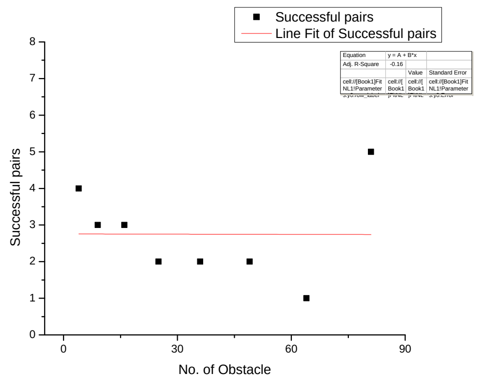


(a) Triangular obstacle map with source-destination pattern 1 formation 3. Function of the curve is $y = A + B * x$ and $a = 2.61467$, $b = 0.00733$

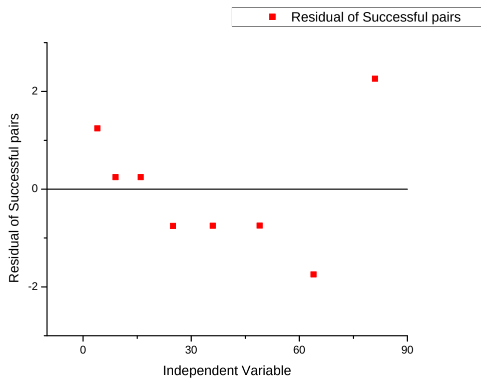


(b) Residual graph of the given graph is 3.5a

Figure 3.5: Pattern 1, Formation 3 for Obstacle △

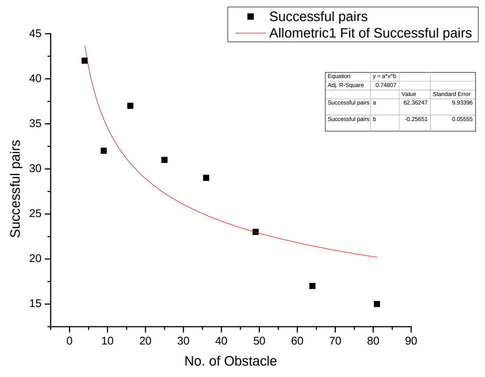


(a) Triangular obstacle map with source-destination pattern 1 formation 4. Function of the curve is $y = A + B * x$ and $a = 2.75676$, $b = 0.0001904$



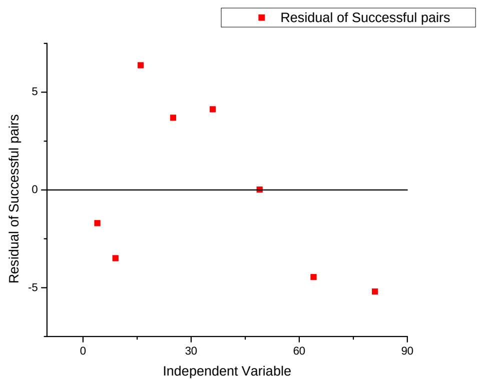
(b) Residual graph of the given graph is 3.6a

Figure 3.6: Pattern 1, Formation 4 for Obstacle △



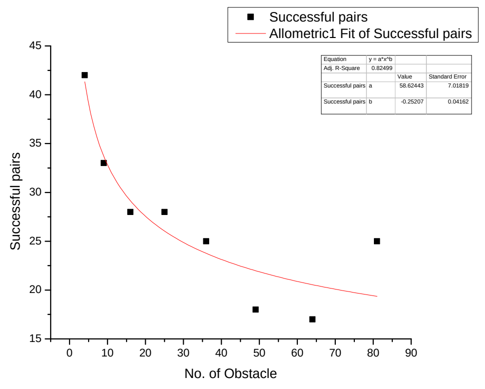
Equation	y = a*x^b		
Adj. R-Square	0.74807		
		Value	Standard Error
Successful pairs	a	62.36247	9.93396
Successful pairs	b	-0.25651	0.05555

(a) Square obstacle map with source-destination pattern 1 formation 1. Function of the curve is $y = a * x^b$ and $a = 62.36247$, $b = -0.25651$



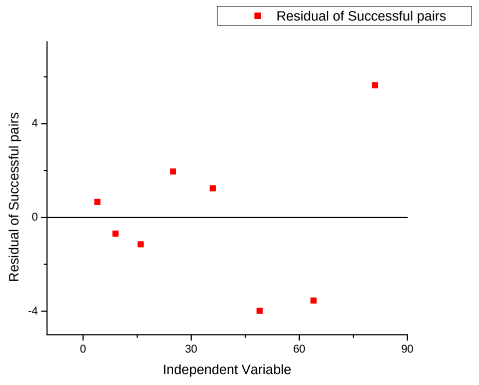
(b) Residual graph of the given graph is 3.7a

Figure 3.7: Pattern 1, Formation 1 for Obstacle □



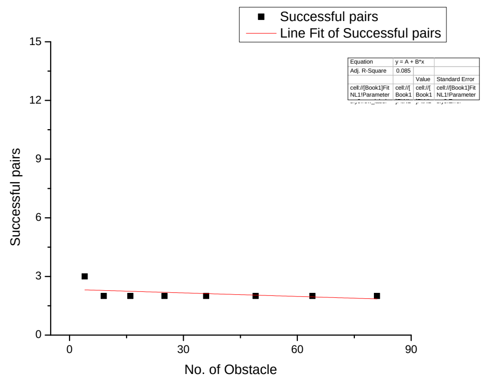
Equation	y = a*x^b		
Adj. R-Square	0.82499		
		Value	Standard Error
Successful pairs	a	58.62443	7.01819
Successful pairs	b	-0.25207	0.04162

(a) Square obstacle map with source-destination pattern 1 formation 2. Function of the curve is $y = a * x^b$ and $a = 58.62443$, $b = -0.25207$

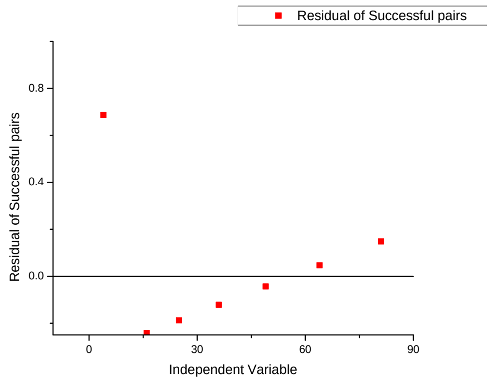


(b) Residual graph of the given graph is 3.8a

Figure 3.8: Pattern 1, Formation 2 for Obstacle □

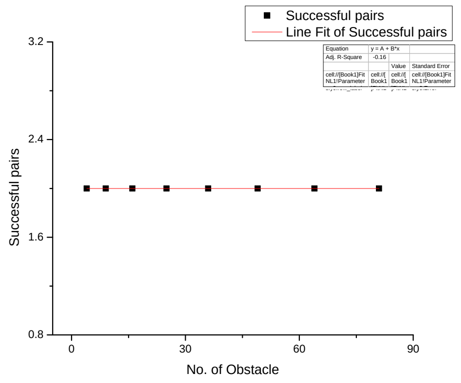


(a) Square obstacle map with source-destination pattern 1 formation 3. Function of the curve is $y = A + B * x$ and $a = 2.338$, $b = -0.006$

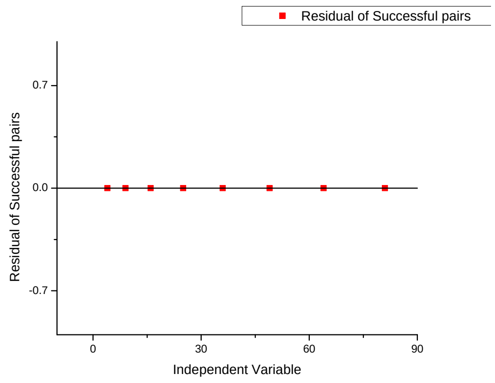


(b) Residual graph of the given graph is 3.9a

Figure 3.9: Pattern 1, Formation 3 for Obstacle □

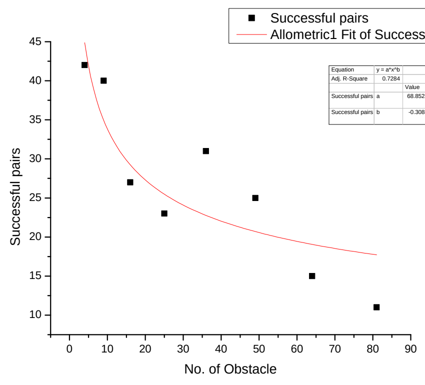


(a) Square obstacle map with source-destination pattern 1 formation 4. Function of the curve is $y = A + B * x$ and $a = 2$, $b = 0$



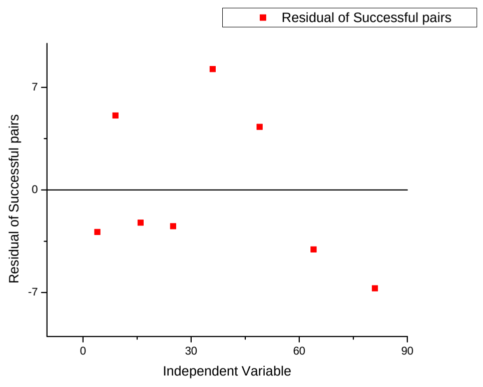
(b) Residual graph of the given graph is 3.10a

Figure 3.10: Pattern 1, Formation 4 for Obstacle □



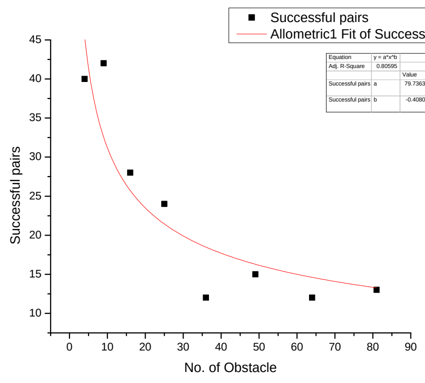
Equation	y = a*x^b		
Adj. R-Square	0.7284		
		Value	Standard Error
Successful pairs	a	68.85241	13.34152
Successful pairs	b	-0.30887	0.07014

(a) Hexagonal obstacle map with source-destination pattern 1 formation 1. Function of the curve is $y = a * x^b$ and $a = 68.85241$, $b = -0.30887$



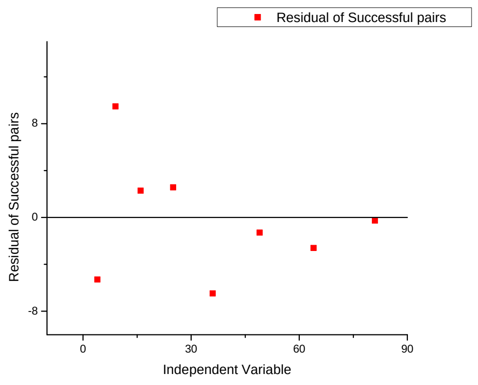
(b) Residual graph of the given graph is 3.11a

Figure 3.11: Pattern 1, Formation 1 for Obstacle ⬡



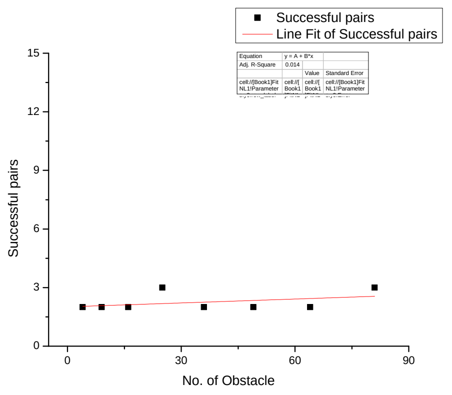
Equation	y = a*x^b		
Adj. R-Square	0.80595		
		Value	Standard Error
Successful pairs	a	79.73639	15.87661
Successful pairs	b	-0.40808	0.07762

(a) Hexagonal obstacle map with source-destination pattern 1 formation 2. Function of the curve is $y = a * x^b$ and $a = 79.73639$, $b = -0.40808$

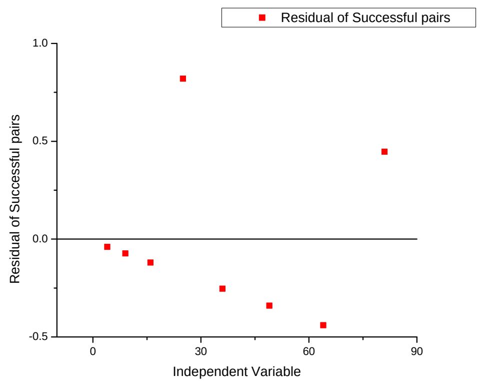


(b) Residual graph of the given graph is 3.12a

Figure 3.12: Pattern 1, Formation 2 for Obstacle ⬡

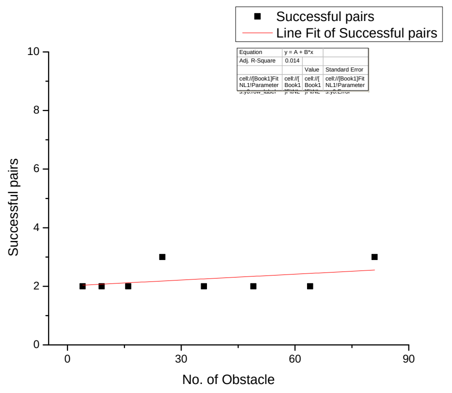


(a) Hexagonal obstacle map with source-destination pattern 1 formation 3. Function of the curve is $y = A + B * x$ and $a = 2.01333$, $b = 0.00667$

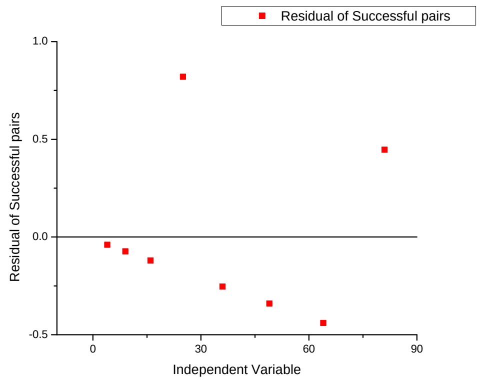


(b) Residual graph of the given graph is 3.13a

Figure 3.13: Pattern 1, Formation 3 for Obstacle ⬡

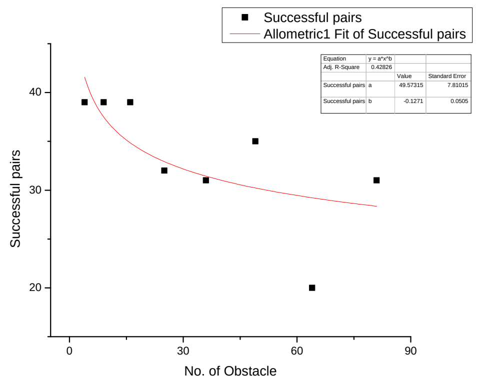


(a) Hexagonal obstacle map with source-destination pattern 1 formation 4. Function of the curve is $y = A + B * x$ and $a = 2.01333$, $b = 0.00667$



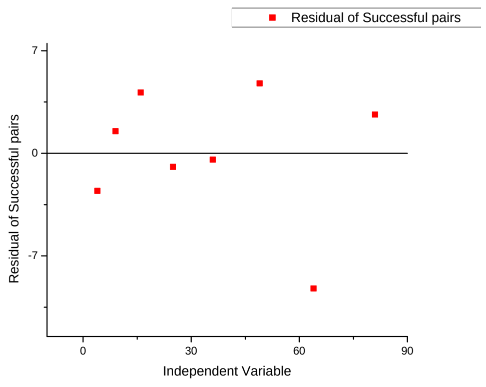
(b) Residual graph of the given graph is 3.14a

Figure 3.14: Pattern 1, Formation 4 for Obstacle ⬡



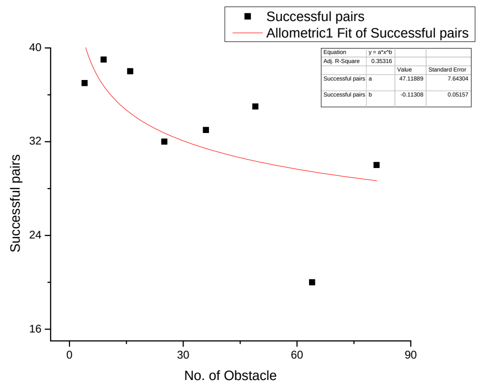
Equation	y = a*x^b		
Adj. R-Square	0.42826		
		Value	Standard Error
Successful pairs	a	49.57315	7.81015
Successful pairs	b	-0.1271	0.0505

(a) Triangular obstacle map with source-destination pattern 2 formation 1. Function of the curve is $y = a * x^b$ and $a = 49.57315$, $b = -0.1271$



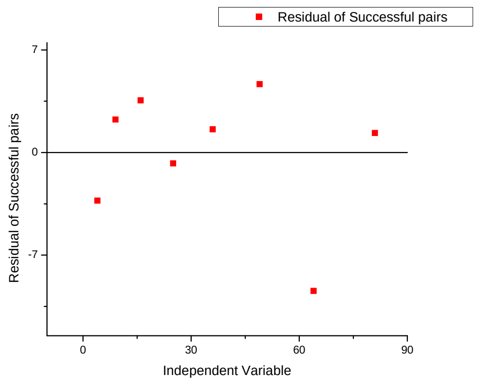
(b) Residual graph of the given graph is 3.15a

Figure 3.15: Pattern 2, Formation 1 for Obstacle △



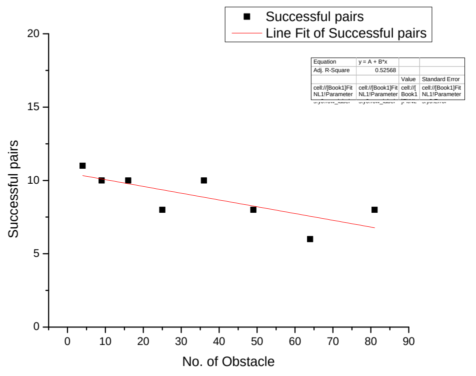
Equation	y = a*x^b		
Adj. R-Square	0.35316		
		Value	Standard Error
Successful pairs	a	47.11889	7.64304
Successful pairs	b	-0.11308	0.05157

(a) Triangular obstacle map with source-destination pattern 2 formation 2. Function of the curve is $y = a * x^b$ and $a = 47.11889$, $b = -0.11308$

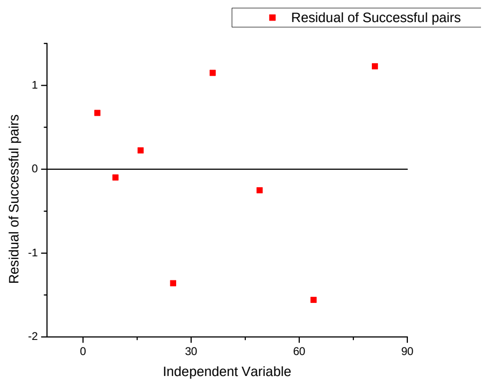


(b) Residual graph of the given graph is 3.16a

Figure 3.16: Pattern 2, Formation 2 for Obstacle △

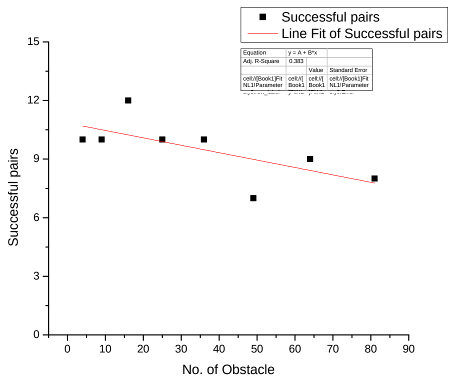


(a) Triangular obstacle map with source-destination pattern 2 formation 3. Function of the curve is $y = A + B * x$ and $a = 10.51476$, $b = -0.04619$

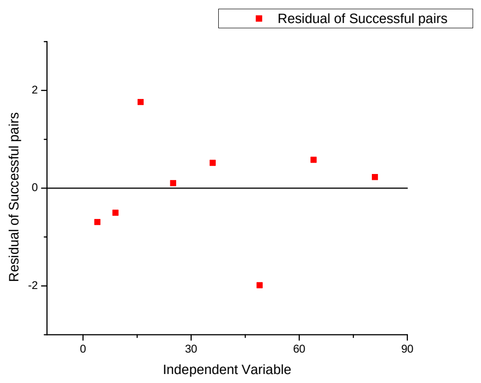


(b) Residual graph of the given graph is 3.17a

Figure 3.17: Pattern 2, Formation 3 for Obstacle $\triangle$

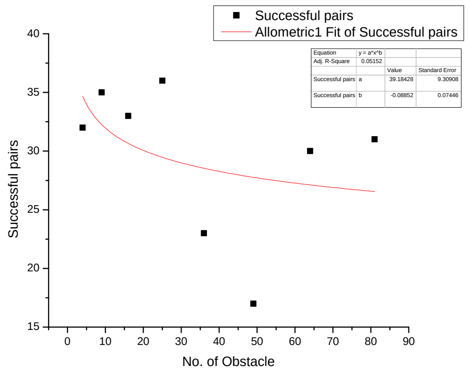


(a) Triangular obstacle map with source-destination pattern 2 formation 4. Function of the curve is $y = A + B * x$ and $a = 10.84562$, $b = -0.0379$



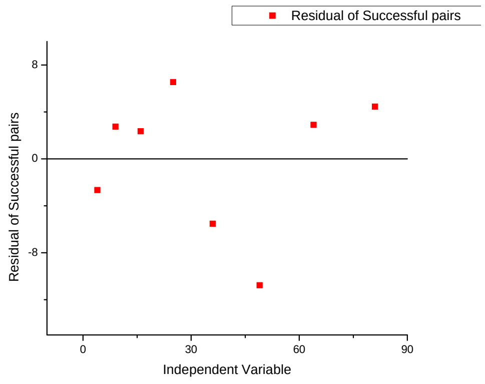
(b) Residual graph of the given graph is 3.18a

Figure 3.18: Pattern 2, Formation 4 for Obstacle $\triangle$



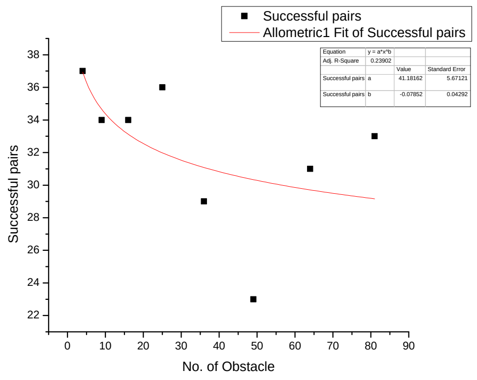
Equation	y = a*x^b		
Adj. R-Square	0.05152		
		Value	Standard Error
Successful pairs	a	39.18428	9.30908
Successful pairs	b	-0.08852	0.07446

(a) Square obstacle map with source-destination pattern 2 formation 1. Function of the curve is $y = a * x^b$ and $a = 39.18428$, $b = -0.08852$



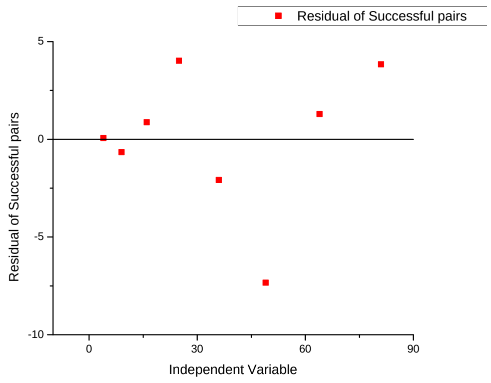
(b) Residual graph of the given graph is 3.19a

Figure 3.19: Pattern 2, Formation 1 for Obstacle □



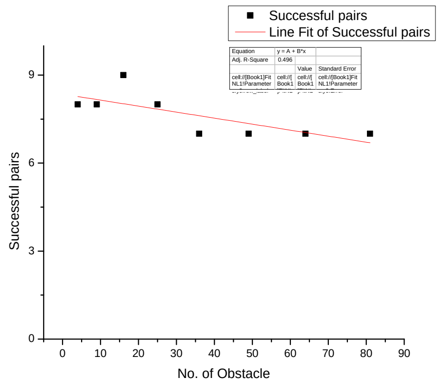
Equation	y = a*x^b		
Adj. R-Square	0.23902		
		Value	Standard Error
Successful pairs	a	41.18162	5.67121
Successful pairs	b	-0.07852	0.04292

(a) Square obstacle map with source-destination pattern 2 formation 2. Function of the curve is $y = a * x^b$ and $a = 41.18162$, $b = -0.07852$

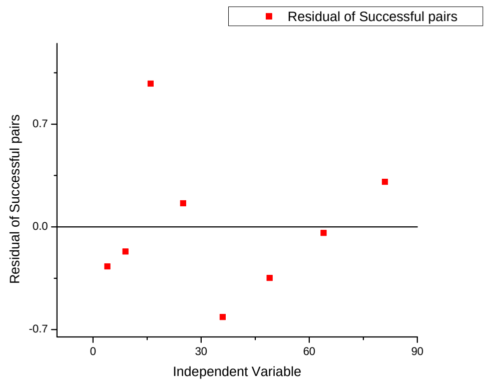


(b) Residual graph of the given graph is 3.20a

Figure 3.20: Pattern 2, Formation 2 for Obstacle □

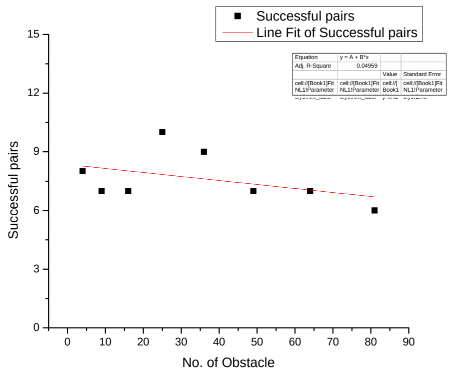


(a) Square obstacle map with source-destination pattern 2 formation 3. Function of the curve is $y = A + B * x$ and $a = 8.3519$, $b = -0.02048$

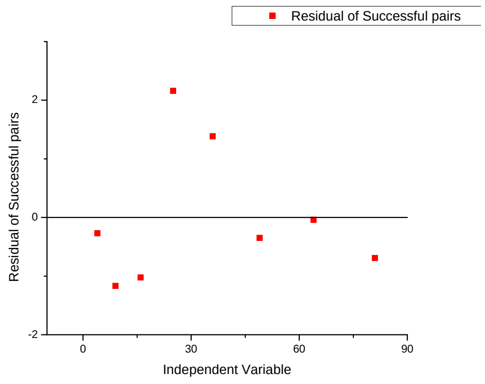


(b) Residual graph of the given graph is 3.21a

Figure 3.21: Pattern 2, Formation 3 for Obstacle $\square$

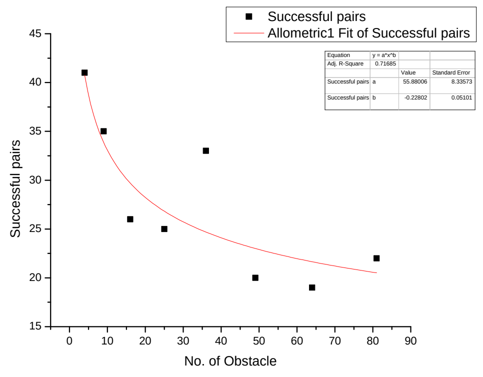


(a) Square obstacle map with source-destination pattern 2 formation 4. Function of the curve is $y = A + B * x$ and $a = 8.3519$, $b = -0.02048$



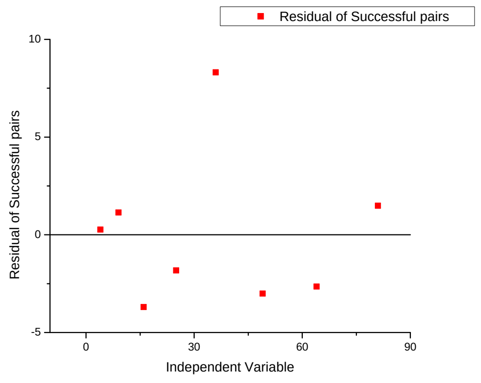
(b) Residual graph of the given graph is 3.22a

Figure 3.22: Pattern 2, Formation 4 for Obstacle $\square$



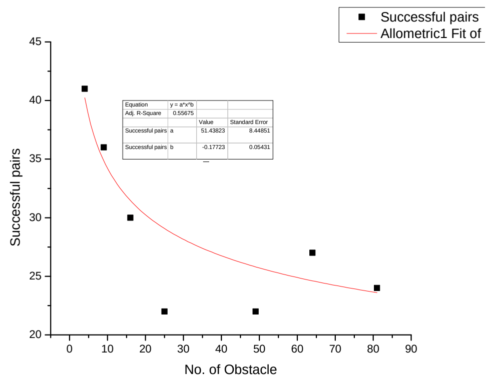
Equation	y = a*x^b		
Adj. R-Square	0.71685		
		Value	Standard Error
Successful pairs	a	55.88006	8.33573
Successful pairs	b	-0.22802	0.05101

(a) Hexagonal obstacle map with source-destination pattern 2 formation 1. Function of the curve is $y = a * x^b$ and $a = 55.88006$, $b = -0.22802$



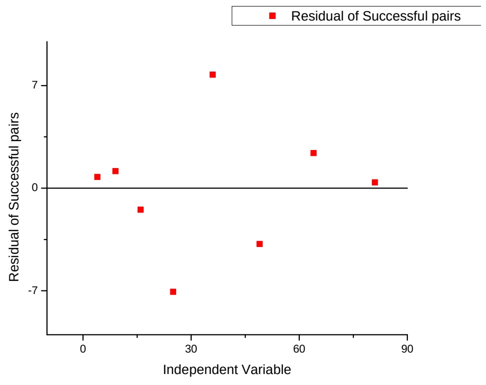
(b) Residual graph of the given graph is 3.23a

Figure 3.23: Pattern 2, Formation 1 for Obstacle ⬡



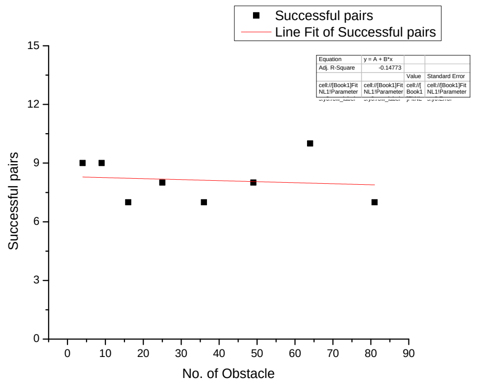
Equation	y = a*x^b		
Adj. R-Square	0.55675		
		Value	Standard Error
Successful pairs	a	51.43823	8.44851
Successful pairs	b	-0.17723	0.05431

(a) Hexagonal obstacle map with source-destination pattern 2 formation 2. Function of the curve is $y = a * x^b$ and $a = 51.43823$, $b = -0.17723$

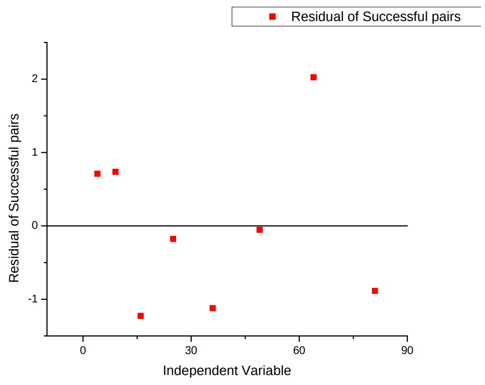


(b) Residual graph of the given graph is 3.24a

Figure 3.24: Pattern 2, Formation 2 for Obstacle ⬡

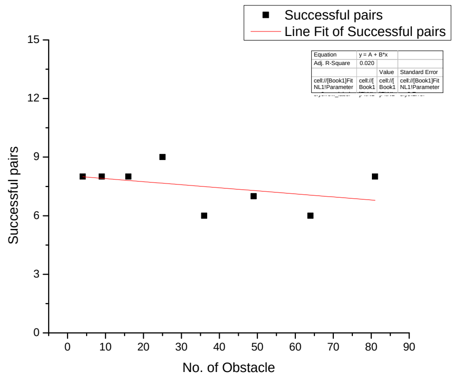


(a) Hexagonal obstacle map with source-destination pattern 2 formation 3. Function of the curve is $y = A + B * x$ and $a = 8.31095$, $b = -0.00524$

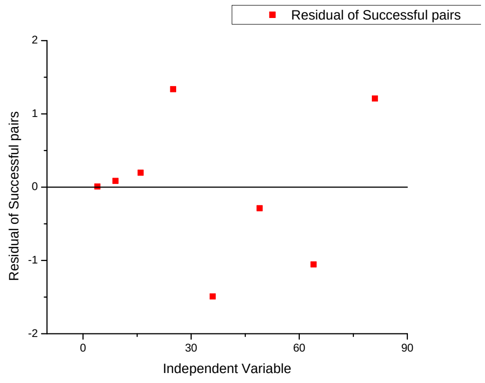


(b) Residual graph of the given graph is 3.25a

Figure 3.25: Pattern 2, Formation 3 for Obstacle ⬡



(a) Hexagonal obstacle map with source-destination pattern 2 formation 4. Function of the curve is $y = A + B * x$ and $a = 8.05448$, $b = -0.01562$



(b) Residual graph of the given graph is 3.26a

Figure 3.26: Pattern 2, Formation 4 for Obstacle ⬡

Chapter 4

Conclusion

In this thesis, we present an algorithm for generating non-intersecting continuous shortest path for multiple robots. This algorithm is based on a visibility graph and the shortest path algorithm is found by Dijkstra's Algorithm. These two algorithms can be replaced by other similar algorithms with minor modification to the proposed algorithm. For example, visibility graph can be replaced with tangent graph[12] to deal with non-polygonal obstacles like circular obstacles or cylindrical obstacles.

From the analysis of results given in Chapter 3, we can say that the paths can intersect amongst each other under of two conditions.

1. In the first place, it is to be noted that intersections originate to avoid obstacles. In this case if there are no obstacles, then the paths will never intersect with each other. That is the straight lines joining the sources to their respective destinations will not have any intersection point. Thus, the number of obstacles and their positions has an effect on the rate of successful non-intersecting paths. From the graphs of simulation results in Chapter3 the number of obstacles(x) and number of successful(y) related by the function $y = a * x^b$.The values of b's in this function $y = a * x^b$ always less than 0. Therefore we can substitute the term x^b by $1/x$.

2. Secondly, intersection originate due to the positions of the robots source and destination points. That is, the straight lines joining the sources to their respective destinations of the given robots intersect with each other. In this case, the number of obstacles may not affect the rate of successful discovery of non-intersecting paths compared with the first condition. As it can observed from the graphs that, for the formation 3 and 4 the relation between number of obstacle(x) and number of success(y) is linear define by the function $y = A + B * x$ and approximately constant that is value of B approximately 0.

If all source and destination pairs maintain the first condition, then *increasing the number of obstacles in the map always decreases the number of* *non-intersecting paths.*

On other hand, if all the source-destination pairs maintain the second condition, then *the number of non-intersecting path remains approximately same for any number of obstacles in the map.* This condition is the **worst case scenario** and the number of *non-intersecting paths is very low* in this case.

4.1 Challenges

This work can be also extended for the concave polygonal obstacles and non-polygonal obstacles. In the present algorithm since the visibility graphs will not work for non-polynomial obstacles and concave polygonals, the algorithm will not produce a desired result.

One of the other, which is remaining is to find the time complexity of the algorithm, which should match approximately with the polynomial fitted into the graphs.

4.2 Future Work

In the proposed algorithm, the function (SuccessClass[Algorithm 3]) is used to classify a source-destination pair into three classes. These classes represent the possibility of finding non-intersecting path for that pair. Based on these classes the algorithm takes decisions to discard source-destination pairs to search for finding non-intersecting paths after certain attempts. These three types of classes are not sufficient for represent the possibility of finding non-intersecting paths for a given source-destination pair. In future, a classification model based on probability can be proposed, such that the function will able to assign a probability of finding non-intersecting path to each given pair.

In the proposed algorithm, we only consider the intersection between the two paths but not the distance between the two paths. An extension of this work can be proposed where the minimum distance between two path is maintained. This problem arises when we work with polygonal robots instead of point robots, as because if the minimum distance is not maintained between the paths, the robots might collide amongst themselves.

Appendix A

An Introduction to setup of Maps using Inkscape

The simulation of the proposed algorithm build in javafx using the IDE IntelliJ IDEA. The maps of the obstacles are given to the simulation program in SVG format. The SVG that is Scalable Vector Graphics is a XML-based vector image format for two-dimensional graphics. An SVG file can be read by the Java DOM(Document Object Model) Parser. The program read the SVG file which is basically an image of obstacle map and create the graph of obstacles in form of vertex set and edge set.

The SVG files are created by the software "Inkscape". The following setting should be maintain to create a image map in SVG format which will compatible with the simulation program -

- Every length should be in *px* unit in the SVG Document.
- The scaling factor of x and y should be 1 that is no scaling to the image is acceptable.
- The polygons drawing should be done by the option "Create stars and polygons".
- The SVG file should be save with extension plain svg no other versions of SVG (like compressed svg, inkscape svg etc) are supported by the simulation.

Bibliography

[1] M. Bennewitz, W. Burgard, and S. Thrun. Optimizing schedules for prioritized path planning of multi-robot systems. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 1, pages 271–276 vol.1, May 2001.

[2] MiÅąel Brezak and Ivan PetroviÄĞ. Path smoothing using clothoids for differential drive mobile robots. *IFAC Proceedings Volumes*, 44(1):1133 – 1138, 2011. 18th IFAC World Congress.

[3] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. *Piecewise Bezier Curves Path Planning with Continuous Curvature Constraint for Autonomous Driving*, volume 68, pages 31–45. 10 2010.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer Berlin Heidelberg, 2013.

[6] Mohamed Elbanhawi, Milan Simic, and Reza N. Jazar. Continuous path smoothing for car-like robots using b-spline curves. *Journal of Intelligent & Robotic Systems*, 80(1):23–56, Dec 2015.

[7] James J. Kuffner Jr. and Steven M. Lavalle. Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE IntâĂŹl Conf. on Robotics and Automation*, pages 995–1001, 2000.

[8] M. Kamath and J. Yang. An intelligent algorithm to generate non-intersecting paths for mobile robots in a multi-robot environment. In *[Proceedings 1989] IEEE International Workshop on Tools for Artificial Intelligence*, pages 641–648, Oct 1989.

[9] J. Kitzinger. *The Visibility Graph Among Polygonal Obstacles: A Comparison of Algorithms*. University of New Mexico, 2003.

[10] Nor Badariyah Abdul Latip and Rosli Omar. Feasible path generation using bezier curves for car-like vehicle. *IOP Conference Series: Materials Science and Engineering*, 226:012133, aug 2017.

[11] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Transactions on Robotics and Automation*, 14(6):912–925, Dec 1998.

[12] Yun-Hui Liu and Suguru Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles: Communication. *The International Journal of Robotics Research*, 11(4):376–382, 1992.

[13] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403–430, Nov 1987.

[14] Ellips Masehian and M. R. Amin-Naseri. A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Robotic Systems*, 21(6):275–300.

[15] N. Omar and Universiti Tun Hussein Onn Malaysia. Fakulti Kejuruteraan Elektrik dan Elektronik. *Path Planning Algorithm for a Car-like Robot Based on Cell Decomposition Method*. Universiti Tun Hussein Onn Malaysia, 2013.

[16] D.F. Rogers and J.A. Adams. *Mathematical elements for computer graphics*. McGraw-Hill, 1976.

[17] Jacob T. Schwartz and Micha Sharir. On the piano movers' problem: Iii. coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers. *The International Journal of Robotics Research*, 2(3):46–75, 1983.

[18] R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. A Bradford book. Bradford Book, 2004.

[19] B. TTTI. *AUTOMOBILE ENGINEERING*.

[20] C. W. Warren. Multiple robot path coordination using artificial potential fields. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 500–505 vol.1, May 1990.

[21] Emo Welzl. Constructing the visibility graph for n-line segments in o(n2) time. *Information Processing Letters*, 20(4):167 – 171, 1985.

[22] Kwangjin Yang, Daehan Jung, and Salah Sukkarieh. Continuous curvature path-smoothing algorithm using cubic b zier spiral curves for non-holonomic robots. *Advanced Robotics*, 27(4):247–258, 2013.

[23] Petr Åǎvestka and Mark H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous Systems*, 23(3):125 – 152, 1998.