May 2023

High Clearance Collision-Free Paths

Barun Thapa

HIGH CLEARANCE COLLISION-FREE PATHS

By

Barun Thapa

Bachelor of Engineering in Computer Engineering
Tribhuvan University
2012

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2023

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

May 2, 2023

This thesis prepared by

Barun Thapa

entitled

High Clearance Collision-Free Paths

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Laxmi Gewali, Ph.D.
Examination Committee Chair

Kazem Taghva, Ph.D.
Examination Committee Member

Mingon Kang, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Graduate College Faculty Representative

Alyssa Crittenden, Ph.D.
Vice Provost for Graduate Education &
Dean of the Graduate College

Abstract

Path Planning is one of the widely investigated research areas in computational geometry and robotics. Given a set of polygonal obstacles inside a rectangular box, and start & goal points, the path planning problem is to construct a collision-free path connecting the start point to the goal point. We review existing well known algorithms for solving the path planning problem. We propose new approaches for constructing a collision-free path with high clearance from obstacles. The main idea of the proposed algorithm is the appropriate generation free-region nodes which can be processed to construct high clearance paths. Neighbors of free-region nodes are carefully joined to obtain a connected graph outside the obstacles. Standard graph searching algorithms are applied in the constructed connected graph to obtain high-clearance paths. We also present methods to remove a few free-region nodes to further increase clearance from obstacles. Finally, we present experimental results on the construction of high-clearance collision-free paths.

Acknowledgements

First and foremost, I would like to thank my advisor Dr. Laxmi Gewali for his guidance, continuous support, motivation and patience throughout the work in this thesis. I am deeply indebted to his vast experience and knowledge in the field of Computational Geometry which has been a great source of help in the completion of the thesis.

Next, I would like to extend my sincere thanks to my committee members for agreeing to be in the committee for my thesis and constantly providing their constructive feedback and suggestions to improve the thesis.

I am also thankful for my wife Nikita Acharya for her moral support and motivation throughout my graduate program culminating in this thesis. I am grateful for her review, proof-reading and feedback on my thesis.

Finally, I am grateful to my friends and roommates Mr. Sweastik Pokhrel and Mr. Bibek Bhattarai for keeping my spirits and motivations high. I would also like to thank my family for their continuous support and well wishes for my progress.

Barun Thapa

University of Nevada, Las Vegas

May 2023

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Connecting two points in a network is a widely investigated problem in computational geometry with application in many areas of science and engineering including geographic information system, transportation, robotics, and game development. The path connecting two points in a network is evaluated using various metrics. Minimising the total length of the path, reducing the implied turn-angle, reducing the number of hops are the commonly used factors for evaluating extracted path for the network.

In applications in robotics, the input is not available as a network. The input is usually in the form of collection of obstacles and it is required to find a good quality path that avoids obstacles. In such applications, the first step is to convert the collection of obstacles into a network. This step is usually called discretization. After a network G is available, the next step is to execute path extracting algorithms on G. In such network all the edges are outside the obstacles. Commonly constructed networks are Visibility Graph, Voronoi Diagram and Triangulated mesh.

In some applications, a pair of non-overlapping collision-free paths are needed. Such pair of paths need to be disjoint from each other except the first and the last node. Having a pair of collision-free paths is desirable in situation where some links could disappear.

In this thesis, we investigate the problem of constructing collision-free paths having high clearance from the obstacles. The objective is to construct paths lying away from obstacles and make it shorter where possible. In Chapter Two, we present a review of existing algorithms (reported in published literature) for constructing high clearance paths. In Chapter Three, we introduce new algorithms for computing collision-free paths based on generating nodes outside obstacles and connecting them by suitable network. In Chapter Four, we present implementation results on the generation of high clearance collision-free paths. The implementation is done in Java programming

language. The front end of the program prototype provides user friendly interfaces. Users can draw polygonal obstacles, generate free-region nodes and the constructed network and a high clearance collision-free paths are displayed. The program allows the users to save generated input for future use via a drop-down menu.

Finally, in Chapter Five, we present some comments on the presented algorithms and discuss possible extensions of the proposed algorithms.

Chapter 2

Review of Collision-Free Paths

In this chapter we describe a brief review of algorithms for constructing collision-free path with high clearance.

2.1 Voronoi Diagram and High Clearance Paths

A structure that has great potential for planning high clearance paths in two dimensions is the Voronoi Diagram mostly studied in computational geometry literature [O'R98]. The Voronoi diagram induced by a set of point sites p_1, p_2, p_3, ... p_n in the plane is the partitioning of the plane into convex cells called Voronoi Cells or Regions R_1, R_2, ... R_n. The region R_i corresponds to site p_i which is such that any point inside R_i is closer to p_i than any other site. The Voronoi Diagram induced by ten point sites is shown in Figure 2.1. Voronoi Diagram satisfies many interesting properties useful for planning collision-free trajectories for robotic vehicles that include the following:

a) A point on the Voronoi Diagram is equidistant from at least two point sites. From any Voronoi vertex, exactly three point sites are equidistant.

b) All regions of Voronoi diagram are convex: some are bounded and others are unbounded. In Figure 2.1, regions corresponding sites p_1, p_6 and p_9 are bounded and other seven regions are unbounded.

c) The dual of the Voronoi Diagram is the triangulation of point sites. These triangulations are called Delaunay Triangulation.

d) The size of Voronoi Diagram is linear in the number of point sites.

e) Paths connecting two points via the Voronoi diagram is such that the path has the best clearance from the point sites. For example, the path connecting two points in Figure 2.1 (shown as red path) has the best clearance from the two point obstacles.

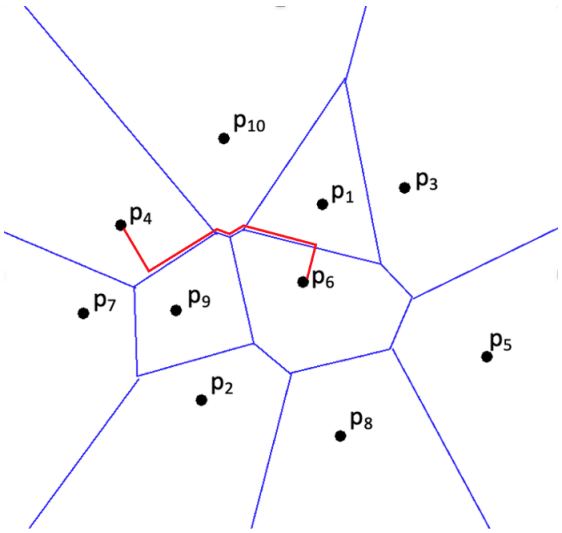


Figure 2.1: Voronoi Diagram of Point Sites and the Best Clearance Path

Our interest in this thesis is to construct a path guided by Voronoi edges when obstacles are polygons.

The Voronoi Diagram of polygonal obstacles are more complex than the Voronoi diagram of point obstacles. An example of Voronoi Diagram of polygonal obstacles is shown in Figure 2.2. The Voronoi Diagram edges are drawn red. In the Voronoi Diagram of point sites, Voronoi edges are lines segments. On the other hand, the edges of Voronoi diagram of polygonal obstacles could be mix of line segments and parabolic/hyperbolic curves.

As observed in [EDPB15], generalized Voronoi Diagram of polygonal objects are very complicated to compute.

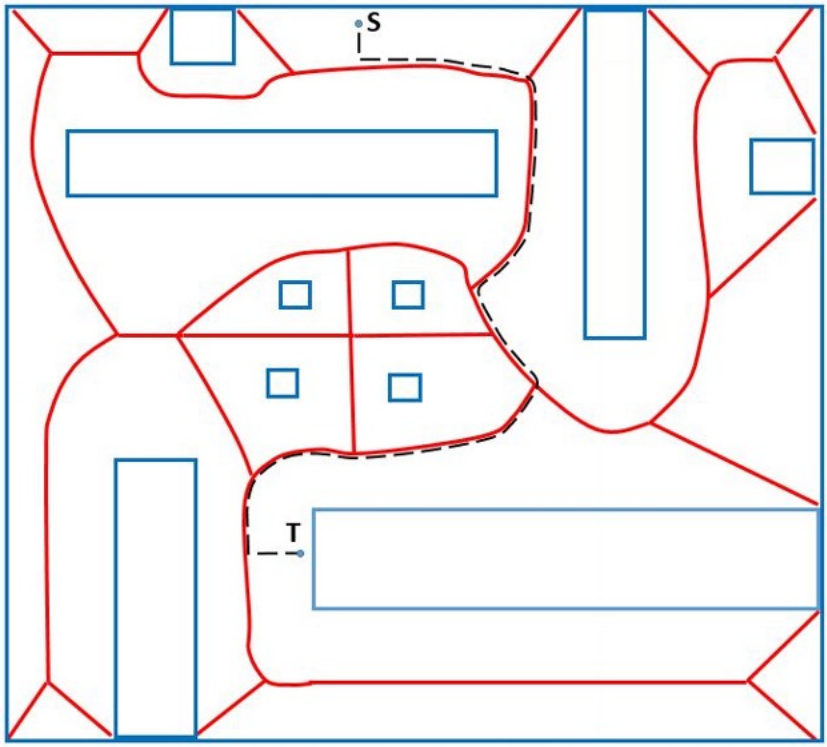


Figure 2.2: Depicting Generalized Voronoi Diagram

2.2 Triangulation and High Clearance Paths

One approach for constructing collision free paths having high clearance has been considered in [JLLC21]. The approach essentially constructs a triangulation T of the free-region. In the triangulation T, free-region nodes are positioned by choosing the centroids of each triangle that tile the free-region. For increasing the quality of resulting high clearance path extra nodes on the edges of obstacles and the bounding box are introduced. Such extra nodes along the edges are referred to as **Steiner** vertices in computational geometry literature [dBvKOS00].

An example of triangulation of the free-region with the addition of Steiner's points on the boundary is illustrated in the Figure 2.3.

After the free-region is triangulated, centroids of the triangles are connected. For connecting centroids only triangles that share edges are considered. The centroids, which are shown as red dots, are connected to form a network as illustrated in Figure 2.4. The start and goal points, S and G, are shown as unfilled dots. These points can be connected to the closest centroid points and a high clearance path from S to G can be extracted which is shown by the network of green edges. It is noted that the network formed by connecting centroids (called **free-region network FRN**)

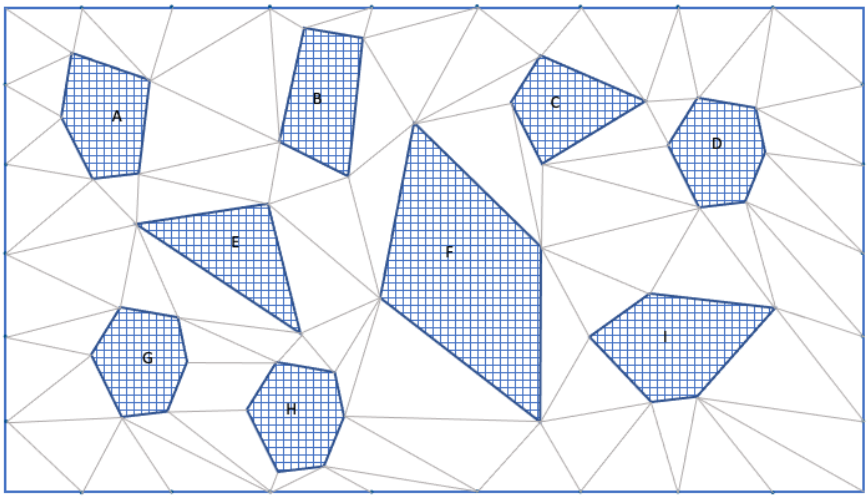


Figure 2.3: Triangulation of Free-Regions

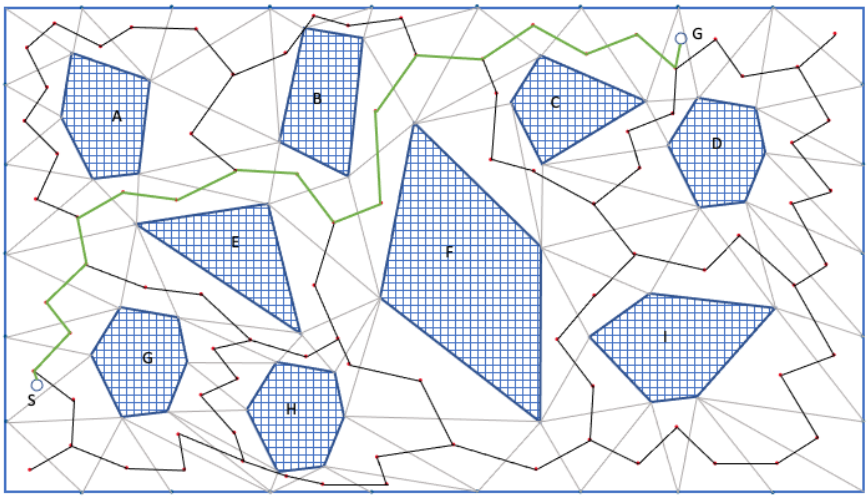


Figure 2.4: Illustrating Centroid Guided Clearance Path

is such that each node is of maximum degree three. Since FRN is the dual of a triangulation it is also a planar network.

The triangulation of the free-region can be done in multiple ways. The triangulation may not always return a high clearance path, as it will depend on the structure of the triangulation. Selecting the best triangulation that can generate a high clearance path is a different research topic on its own. An example of how that path may look like for the example above with different triangulation is illustrated in the Figure 2.5. In the figure, it can be noted that few adjacent centroids may not be connected because they can be obstructed by the polygonal obstacles. These are illustrated at red dashed-lines in the Figure 2.5. Furthermore, if there are many broken connection in the network due to centroids not being visible to each other, there may not be a path from S to G. This demonstrates that the way in which free-region is triangulated determines the quality of the high clearance path.

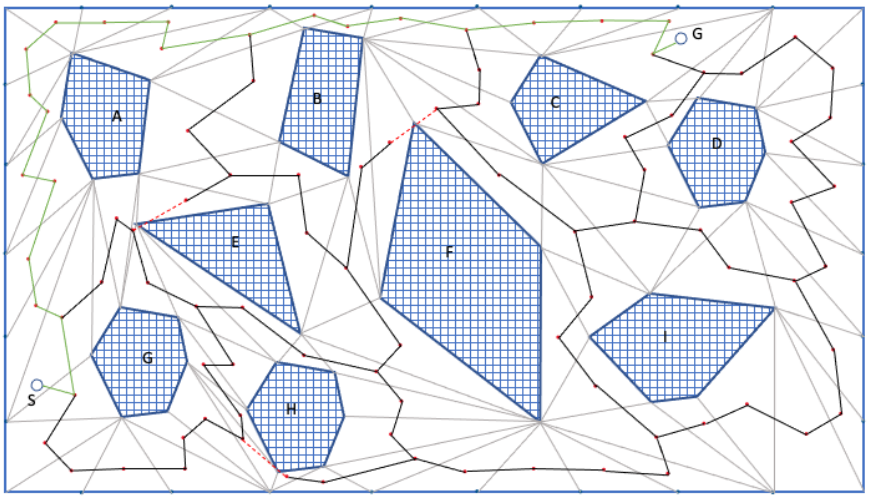


Figure 2.5: Collision-free Path Guided by another Triangulation

Another approach in constructing a high clearance path by using triangulation is presented in [JLLC21]. Here, the the triangulation is achieved by using Constrained Delaunay Triangulations. After the triangulations, the centroids of the triangles are used as vertices and a shortest path is formed by using Dijkstra's algorithm. Next, the mid-points of the edges of the triangles that are intersected by the path are connected to form a new high clearance path.

Chapter 3

Generating Quality Free-Region Nodes

3.1 Problem Introduction

Consider a collection of polygonal obstacles $A, B, C, D, \ldots$, (disjoint or non-overlapping) enclosed inside a rectangular box μ. The m vertices of obstacle A are denoted as $a_0, a_1, a_2, \ldots a_m$. The vertices of other obstacles are denoted in similar manner. The vertices of each obstacle are listed in the order in which they occur along the boundary, in the counterclockwise order. An instance of a collection of nine polygonal obstacles, enclosed in a rectangular box is shown in Figure 3.1.

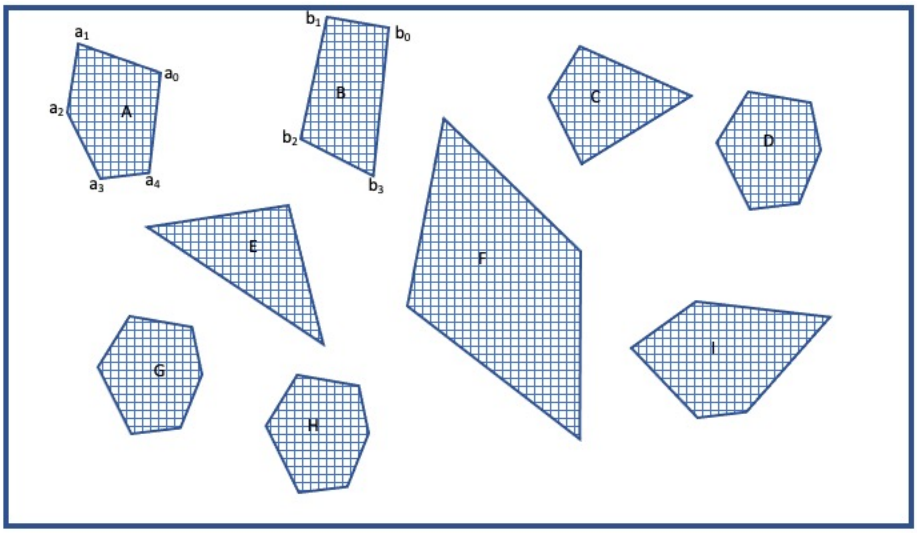


Figure 3.1: A Collection of Polygonal Obstacles in a Rectangular Box

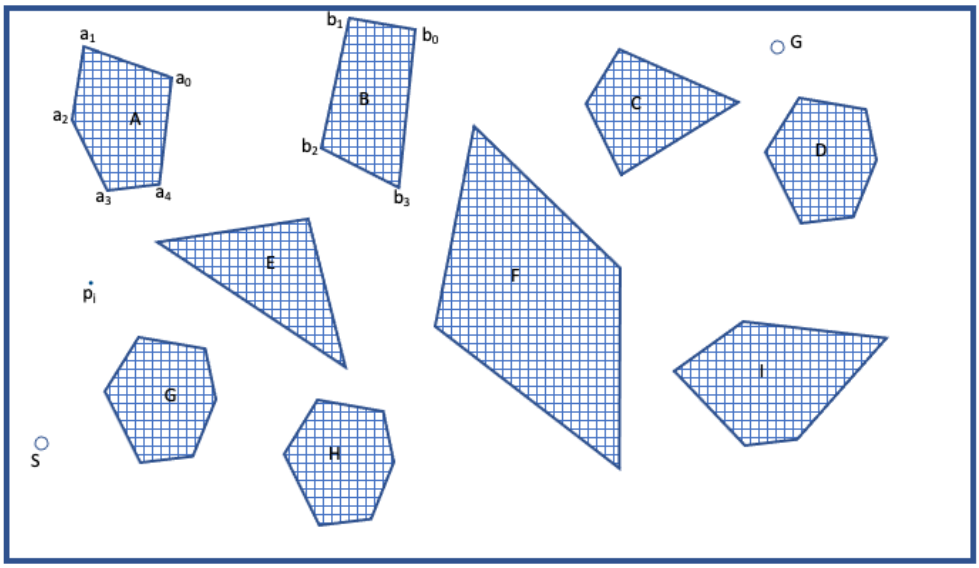


Figure 3.2: Free-Region Point p_i

A point p_i outside of the obstacles is called a **free-region** point. In the Figure 3.2, free-region points S and G are shown (drawn as unfilled dots). Two collision-free paths connecting S to G are shown in the Figure 3.3: one is drawn in red color and the other is drawn as blue. It can be easily observed that the red path has less clearance from obstacles compared blue path. We can imagine many such collision-free paths connecting S to G. Some paths have high clearance from obstacles and other have low clearance. The specific problem we propose to examine is the development of collision-free paths having high clearance from obstacles. The problem can be formally defined as follows.

High Clearance Path (HCP) Problem

Given: (i) A collection of polygonal obstacles *A, B, C, D, ...* (ii) two free-region points S (**start** point) and G (**goal** point).

Question: Find a collision free path P connecting S to G such that P has high clearance from obstacles.

3.1.1 Visibility Graph guided Free-Region Nodes

To investigate the properties of collision-free paths, it is necessary to understand the visibility relationships between the vertices of obstacle polygons. Two points p_i and p_j are said to be

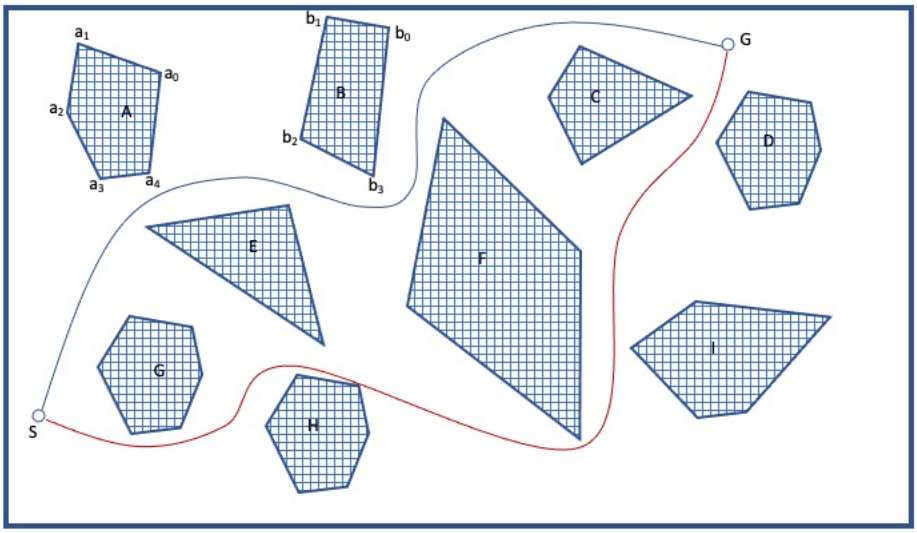


Figure 3.3: Illustrating Low and High Collision-free Paths

mutually visible if the line segment connecting p_i to p_j lies completely in the free-region. This is shown in Figure 3.4, where a_0 is not visible to b_0 as the line segment connecting a_0 to b_0 intersect obstacle B. On the other hand, a_4 is visible to b_3, due to the fact that the line segment connecting a_4 to b_3 lies completely in the free-region. Such visibility relations are explained on the textbooks of computational geometry [O'R98] [dBvKOS00]

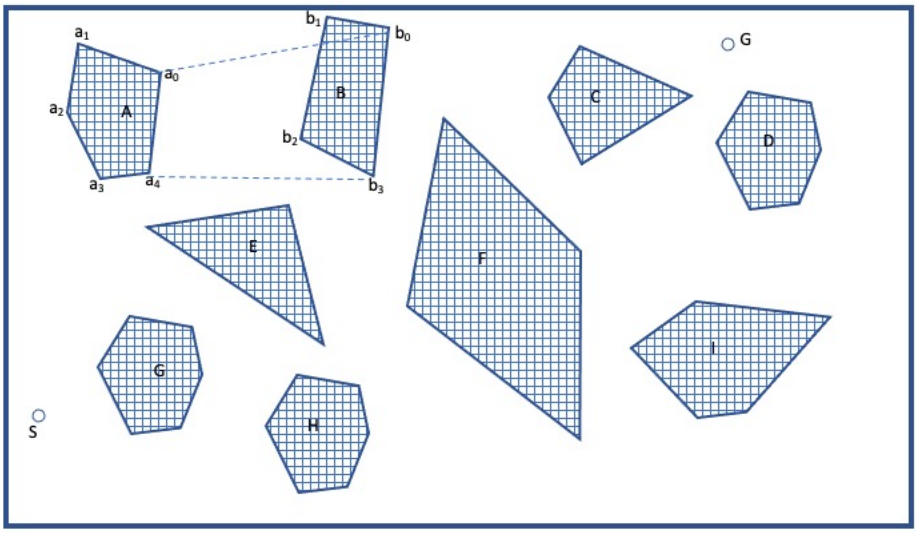


Figure 3.4: Notion of Visibility

Visibility Graph formed by a set of polygonal obstacles is a well investigated structure [GM87].

Specifically, visibility graph is obtained by including all edges corresponding to mutually visible vertices in the collection of obstacles. Figure 3.5 shows a visibility graph of obstacle polygons. In the figure, visibility edges are drawn as thin line segments. It is known that the shortest collision-free path can be computed by applying Dijkstra's shortest path algorithm on the visibility graph [O'R98]. However, the shortest collision-free path extracted from the visibility graph has least clearance from the obstacles. The shortest collision-free path is drawn red-colored which touches several obstacles. Such a path has least clearance from obstacles. This shows that the shortest path extracted from the visibility graph cannot be used for constructing high-clearance path.

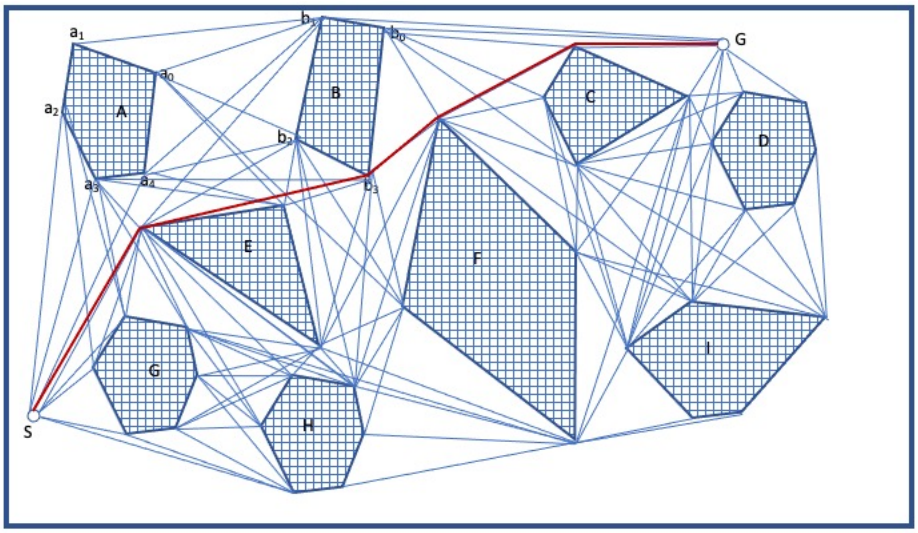


Figure 3.5: Shortest Path Captured by Visibility Graph

A different approach can be taken to obtain a high clearance path from the visibility graph. Instead of just following the visibility edges in the graph to reach the goal node, a new network can be formed from the mid-points of the visibility edges. In this approach, first, the mid points of the visibility edges are generated. Since, the edges are visibility edges, the mid-points of such edges will naturally occur in the free-region as shown in the Figure 3.6. This process is repeated for all the visibility edges to obtain free-region points as illustrated in the Figure 3.7.

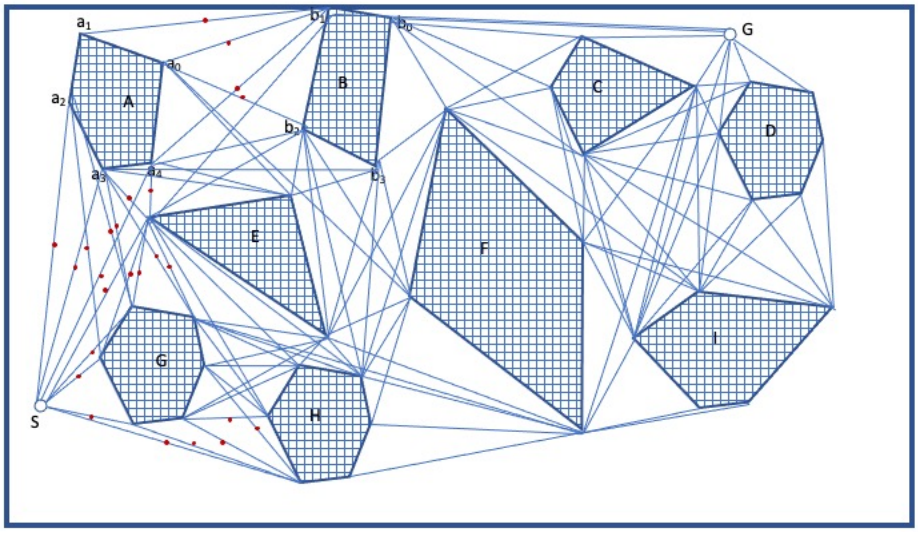


Figure 3.6: Mid-points of Selected Visibility Edges

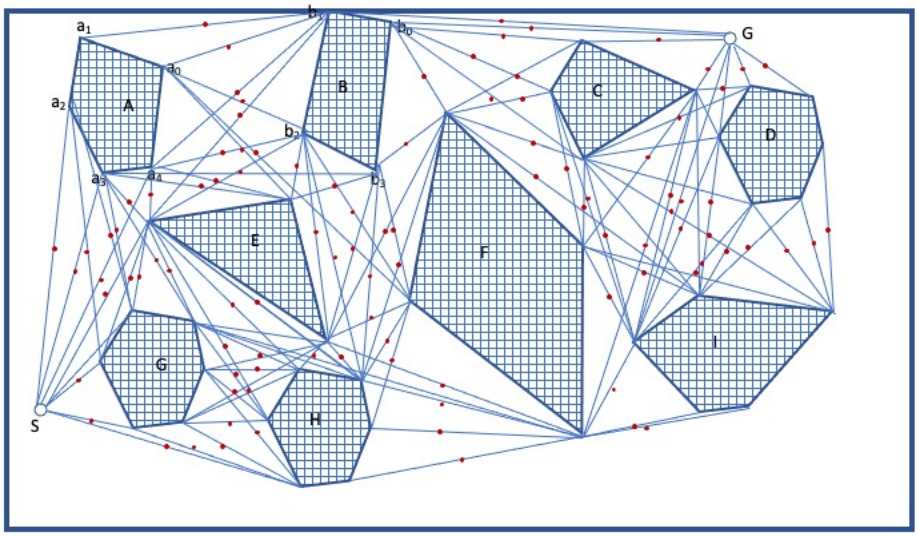


Figure 3.7: Mid-points of All Visibility Edges

If we carefully connect free-region points then a high clearance collision-free path can be constructed as shown in Figure 3.8. How to generate appropriate number of free-region nodes is very critical issue.

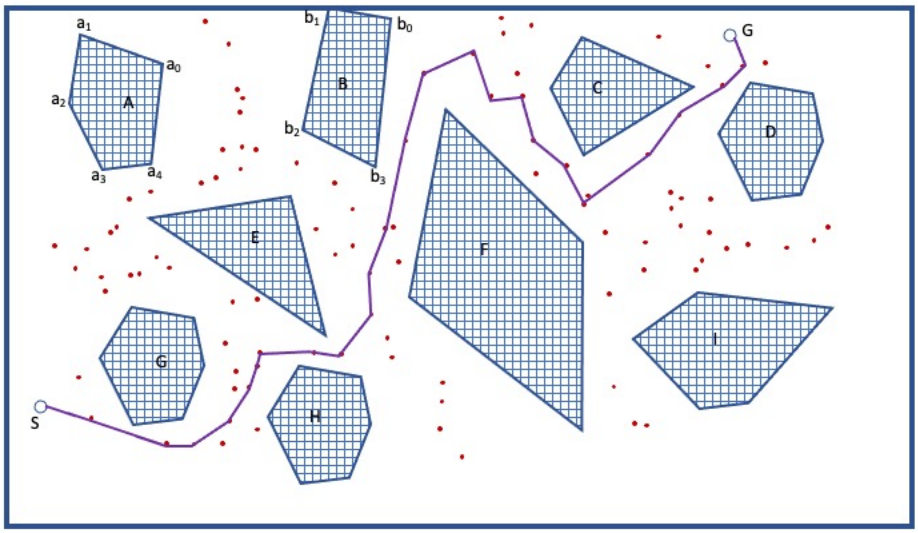


Figure 3.8: A Possible High Clearance Path by Carefully Connecting Free-Region Points

As mentioned earlier, mid-points of visibility edges can be used to construct free-region nodes. Once free-region nodes are generated they need to be connected to obtain a free-region network (FRN in short). FRN can be searched to get a collision-free path. We could use the standard graph searching algorithms to extract a path from FRN. A high level formal sketch(stepwise) of such an approach is listed as 1 below.

Algorithm 1: A High-Level Sketch of the Proposed Collision-free Path Algorithm

Step 1: Compute Visibility Graph formed by the collection of obstacles.

Step 2: Take mid point of each Visibility Edge as free-region node.

Step 3: Let R = q_1, q_2, q_3, ... q_m be the set of free points extracted from Visibility Edges.

Step 4: Construct a Free-Region Network (FRN) by connecting nodes on R.

Step 5: Apply breadth first algorithm in FRN to obtain collision-free path.

Connecting Free-Region Points

Given a set of points q_1, q_2, q_3...q_m in the free-region, as illustrated in the Figure 3.9, it is required to connect the points to form a network, which we refer to as free-region network (FRN). This is required to generate the collision-free high clearance path.

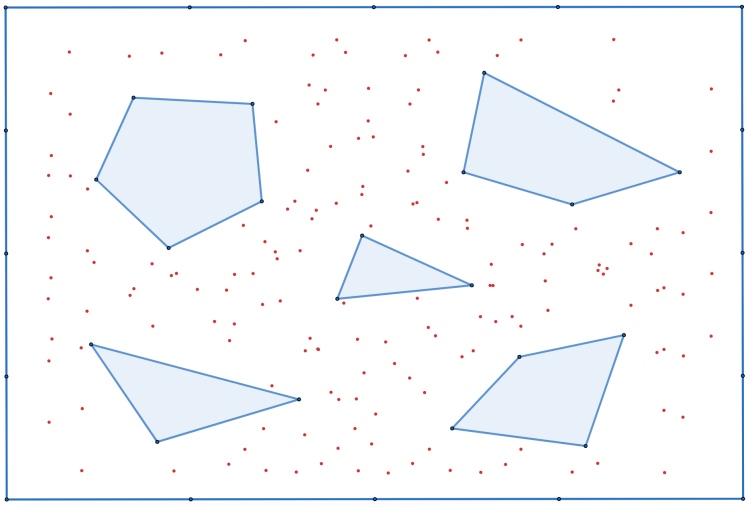


Figure 3.9: Free-Region Points

There can be multiple approaches to connect the free-region points to form a network. Our approach is to connect the points using grid and a guide circle.

Two structures are used to connect free-nodes in this approach. The first structure is the **guiding circle** of radius **r**. The second structure is an orthogonal **square grid** whose cell size is q x q. The starting value of r and q is determined in terms of the separation length δ' between the closest free-region node pair. The grid lines intersect with each other at multiple points and these intersection points can be labeled as $g_1, g_2, g_3....g_n$. From each of these points, a guiding circle of radius r is drawn. The value of r could be between $2\delta'$ and $3\delta'$. The value of q could be between $3\delta'$ and $6\delta'$. The values of r and q can also be selected interactively by the user. Figure 3.10 illustrates embedded grid (black dashed-lines) and guide-circle(orange).

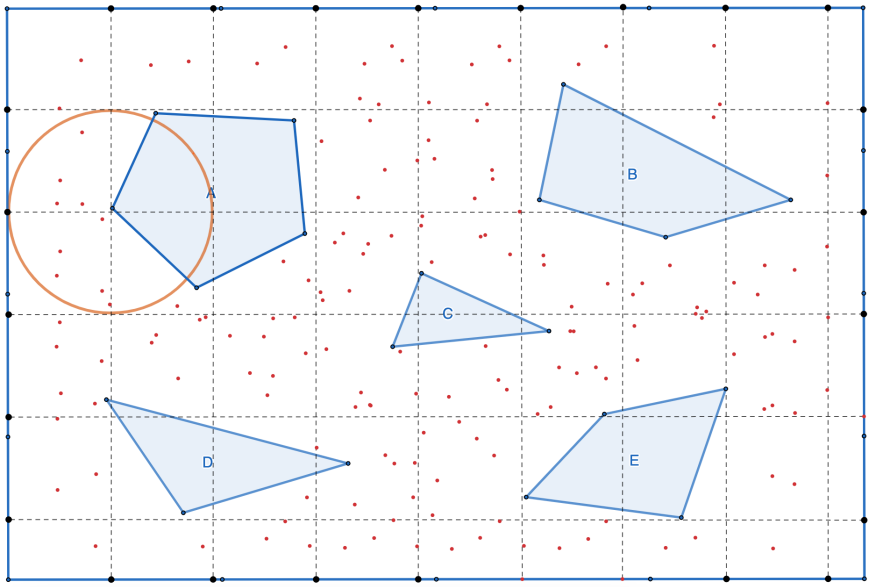


Figure 3.10: Illustrating Embedded Grid and Guide Circle

The nodes inside a guide-circle are connected by an edge as demonstrated in the Figure 3.11.

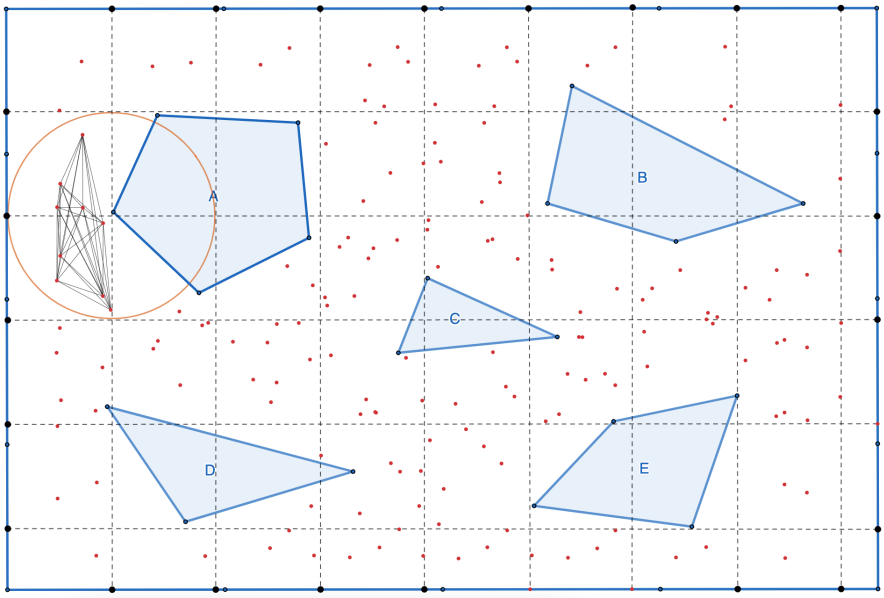


Figure 3.11: Illustrating Basic Block

The nodes, however, can only be connected if the edge does not intersect with any obstacle. Figure 3.12 shows an instance in which some connecting edges intersect with obstacles. This means the free-region nodes that lie inside the guide circle should be visible to each other and not blocked by the polygonal obstacles. In the Figure 3.12, the nodes fp_1 and fp_2 can be connected as they

are visible to each other and are connected with a green line. The nodes fp_1, fp_4 and fp_2, fp_3 cannot be connected as they are blocked by an obstacle, or they are not visible to each other. They are represented by red dashed lines. This essentially means that the high clearance path will not traverse through the nodes fp_1 and fp_4 directly.

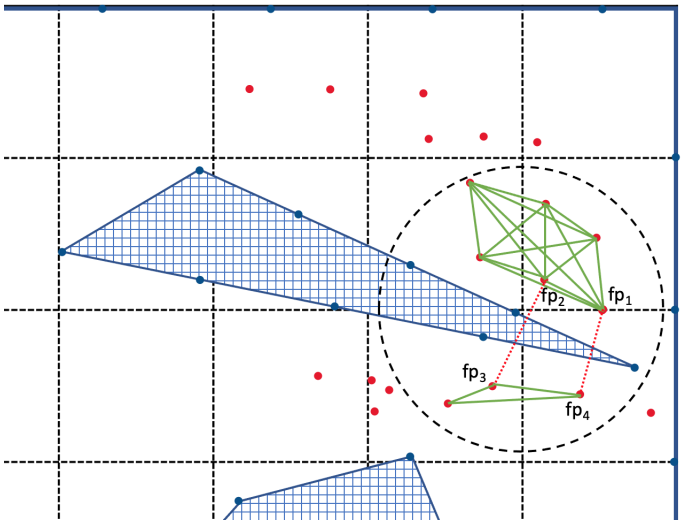


Figure 3.12: Illustrating Connecting Edges that Intersect with Obstacles

When all valid node pair inside a guide circle are connected we get **basic-block** as shown in Figure 3.11. The process of constructing basic-block is repeated by placing guide circle at each grid point. The aggregation of basic blocks give the **navigating network** to construct collision-free path. An example of navigating network is shown in Figure 3.13.

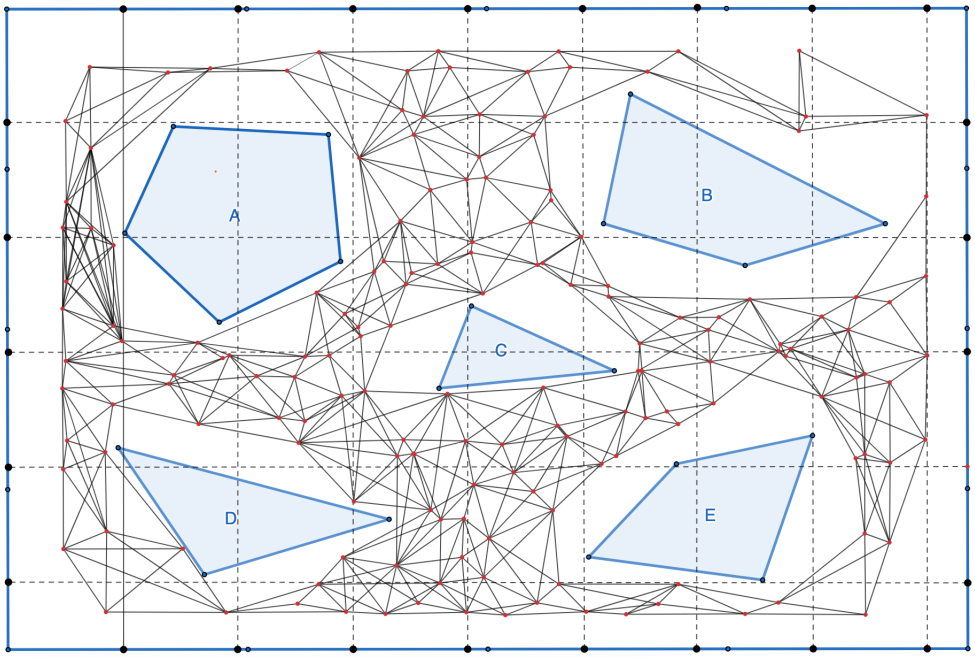


Figure 3.13: Formation of Navigating Network

High Sketch Algorithm to Connect Free-Region Nodes

Algorithm 2: Construct the Navigation Network Algorithm

Input: (i) A collection of polygonal obstacles O_1, O_2,... O_k
(ii) A collection of free-region nodes $q_1, q_2, q_3.....q_n$

Result: Navigation Network (NN)

Step 1:
Compute the radius 'r' of guiding circle by finding the distance between closest pair of free-region nodes

Step 2:
for *each grid point* g_i **do**
| Compute the basic-block by using guide circle centered at g_i
end

Step 3:
Combine basic-block to obtain Navigation Network (NN)

Step 4:
if *NN is connected* **then**
| Stop
else
| Increase the value of r and start from Step 2
end

3.1.2 Avoiding Low Clearance Nodes

Nodes that are closer to the obstacles (than other nodes) can be considered as low clearance nodes. Here, the measure of the low-clearance needs to be quantified clearly. For the purpose of our application we pick a threshold value of smallest distance between obstacles as the basis. If d_{min} is the smallest distance between obstacles then $0.5d_{min}$ is taken as the threshold value. This is illustrated in the Figure 3.14. In the Figure 3.14, the closest distance between polygonal obstacles pairs are drawn with dashed lines. Not all distances are drawn to make figure clear. The smallest distance is observed to be d_4, which is the distance between obstacles B and C, and is marked as d_{min}. The length $0.5d_{min}$ is used as the threshold to identify the low clearance nodes.

The threshold for the low clearance node is set to be $0.5d_{min}$ to ensure that the object traversing the path can move in-between the obstacles without collision. The smallest distance is taken as the diameter for the disc that can pass through. The radius value, $d_{min}/2$, is taken to filter out the low clearance nodes. Any object can be enclosed in a circular boundary in a 2D plane, hence a disc is considered to be a standard object. This is illustrated in the Figure 3.14.

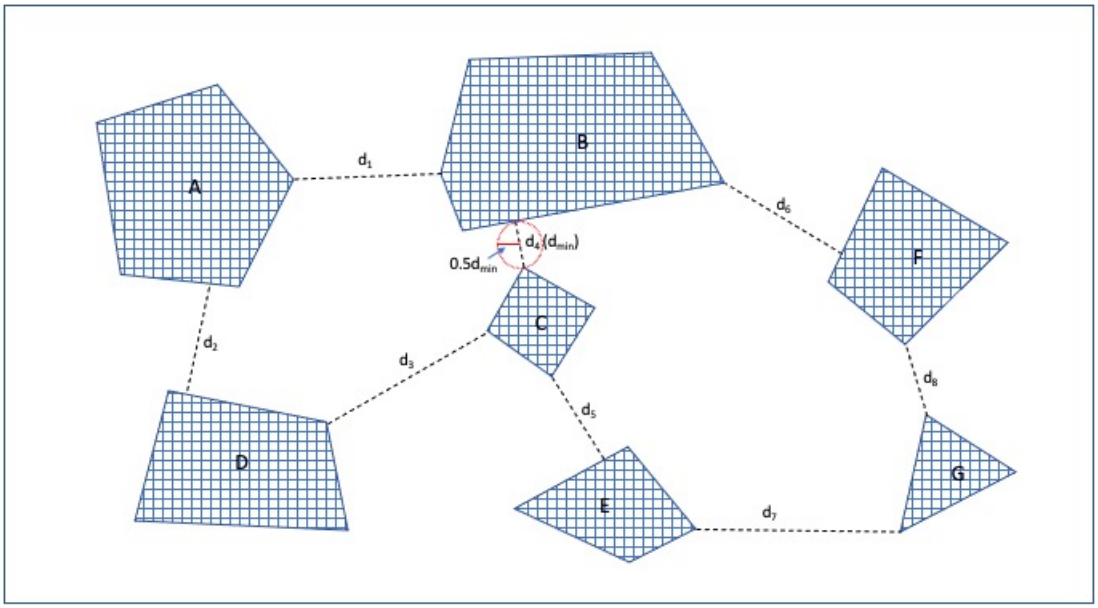


Figure 3.14: Finding Smallest Distance between Polygonal Obstacles

The smallest distance, between two polygonal obstacles, needs to be defined properly to take the measure. The distance that is measured is the distance between nearest points in the polygon boundary. These points can be either vertices or edges. This means the smallest distance, between

two obstacles, could potentially be the distance between two vertices, between two edges or between an edge and a vertex. These cases are illustrated in the Figure 3.14. The distances d_1, d_3, d_7, d_8 are taken between vertex of obstacle pairs. The distances d_4, d_5, d_6 are taken between a vertex and an edge. Finally, the distance between obstacles A and D is taken between the edges.

Algorithm 3: Compute Threshold for Low Clearance Nodes Algorithm

Input: List of polygonal obstacles

1 Select a pair of polygonal obstacles

2 Derive the sets of edges and vertices of the obstacles

3 Compute distance between vertices and edges pairs of two different polygons

4 Find the minimum distance and mark it as threshold

3.1.3 Randomly generated Free-Region Nodes

Free-region nodes can be generated randomly by using the random integer generation library for Java system. Suppose we want to generate a random node in a canvas of size 800x700. We can randomly generate a number between $0 - 800$ inclusive and take it as the x-coordinate. The y-coordinate is similarly generated by picking a random integer between $0 - 700$ inclusive. The Figure 3.15 shows the Java code snippet used to generate a random nodes location.

```
int x1 = (int) (800 * Math.random());
int y1 = (int) (700 * Math.random());
my_point p1 = new my_point(x1, y1);
```

Figure 3.15: Code Snippet for Generating Random Points

The random point so generated could be inside or outside the polygon obstacles. The nodes that are inside the obstacle can be rejected. To check whether the generated node is outside the obstacles or not can be done by appealing to point inclusion in polygon algorithm available in standard computational geometry text book [O'R98].

The randomly generated free-region nodes need to be further processed to filter the nodes that are very close to obstacles. For this purpose we can check the distance of the generated node to the nearest obstacle. If the distance is smaller than a predetermined threshold distance δ. We have

taken the value of δ as the smallest separation between obstacles. The elimination of such nodes can be done by following the same process as described in section 3.1.2.

3.1.4 Free-Region Nodes guided by Obstacle Expansion

Expanded Obstacles

Each polygonal obstacle can be expanded by δ extent to create a δ-envelop. For each edge $e = (a, b)$ a δ-displaced edge $e' = (a', b')$ is constructed, where e' is parallel to e and at a distance δ from e. Here the distance δ is measured perpendicular to e.

Figure 3.16 shows δ-displaced edges (drawn dashed) for all edges of the polygonal obstacles.

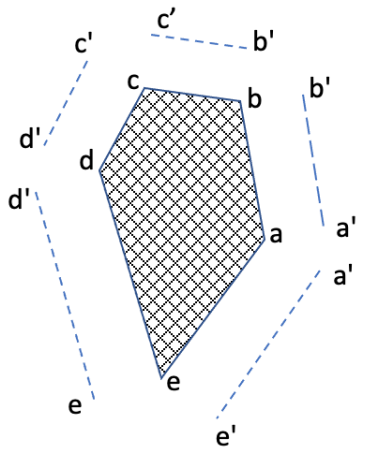


Figure 3.16: Expanded Edges of the Polygonal Obstacles

For each vertex a incident on edges (a, b) and (a, e) an arc is constructed whose center is at a and of radius δ. The endpoints of the arc are δ-displaced points of edges (a, b) and (a, e). Such arcs, labelled as (a', b') and (a', e'), are constructed for all vertices as shown in Figure 3.17.

When δ-displaced edges and arcs are put together we get the δ-envelop of the polygon as shown in Figure 3.18. Observe that δ-envelop is boundary of a convex polygon whose edges are both line segments and arcs. When such envelops are constructed to an instance of high clearance path problem (consisting of three convex obstacles) we get expanded obstacles as shown in Figure 3.19.

Suppose we perform the random generation of free-region nodes then some nodes will fall inside the expanded obstacles and others outside. Those nodes that are inside the obstacles can be discarded. The motivation for expanding the obstacle is to avoid the presence of free-region nodes that are very close to the obstacles. In Figure 3.20, an instance of randomly generated nodes is

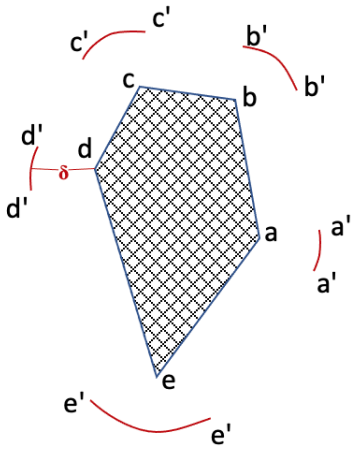


Figure 3.17: Arcs Drawn from Vertices of a Polygonal Obstacle

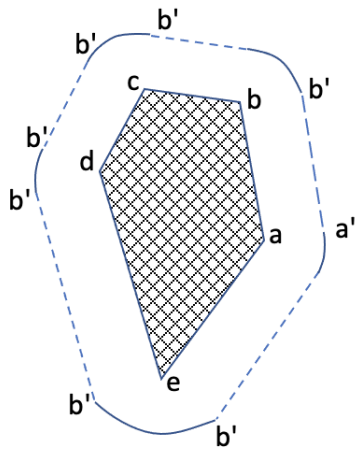


Figure 3.18: Expanded Arcs and Edges of a Polygonal Obstacle

shown where there are seven of them are within annulus region between original obstacle boundary and the boundary of the expanded obstacle.

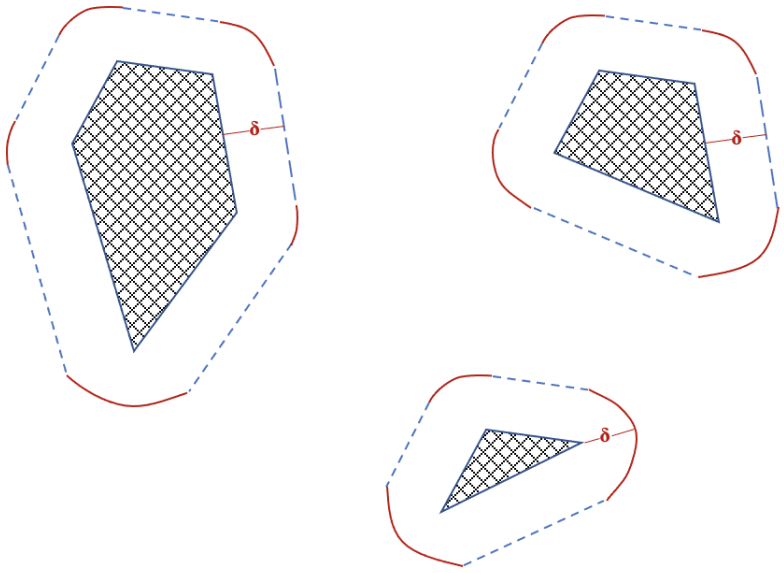


Figure 3.19: Convex Polygonal Obstacles with Expanded Arcs and Edges

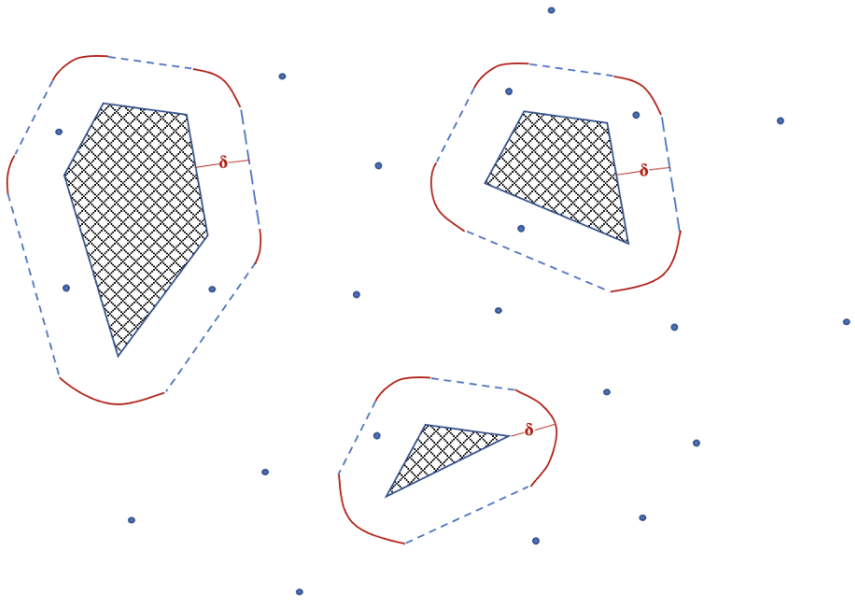


Figure 3.20: Expanded Convex Polygonal Obstacles with Nodes

A formal algorithm for generating free-region nodes not close to the obstacles based on obstacle expansion can be written as follow

Algorithm 4: Generate Free-Region Nodes Algorithm

Input: (i) A collection of polygonal obstacles O_1, O_2,... O_k
(ii) m, number of nodes to generate
(iii) Empty set R

Result: List of free-region nodes not close to obstacles

Step 1:

```
for i = 1 to k do
    for each edge e of O_i do
        construct δ-displaced edge
    end
    for each vertex v_i of O_i do
        construct δ-arc
    end
end
```

Step 2:

Combine δ-displaced edges and δ-arcs in Step 1 to construct expanded obstacles

Step 3:

```
for i=1 to m do
    x = random int in the range low-high
    y = random int in the range low-high
    construct a random node with coordinates x and y and add it to set R
end
```

Step 4:

```
for each randomly generated node n_x do
    if n_x is inside any expanded obstacles then
        remove n_x from R
    end
end
```

Step 5:

Output R

The time complexity of Algorithm 4 can be analysed as follows:

- In Step 1, (nested for loops), each edge of obstacles are processed only once. Similarly, each vertex is processed only once. Hence the total time for Step 1 is $O(n)$ where n is the number of vertices in the obstacles.
- Combining δ-displaced edges and δ-arcs in Step 2 takes $O(n)$ time
- A random integer in a range can be generated in constant time. Hence Step 3 takes $O(m)$ time
- Checking a point inside obstacles can be done in $O(n)$ time by using point in polygon inclusion algorithm [O'R98]. Hence Step 4 takes $O(mn)$

Thus the total time complexity of Algorithm 4 is

$$O(n) + O(n) + O(m) + O(mn) = O(mn)$$

Chapter 4

Implementation and Experiments

In this chapter, the application design, implementation details and the experimental results are discussed. The Java programming language is used to implement the algorithms and JAVA Swing API [Ora23] is used to develop the GUI.

4.1 Interface Design

The main window of the application is built with JFrame object from the Java Swing API. The interface is illustrated in the Figure 4.1. The interface is divided into 4 distinct regions:

1. Menu Bar (Top)
2. Drawing Canvas (Center)
3. Control Panel (Right)
4. Action Bar (Bottom)

The menu bar is built by using JMenuBar object. The remaining sections are JPanels and are set in the BorderLayout grid styling using Center, East and South positioning.

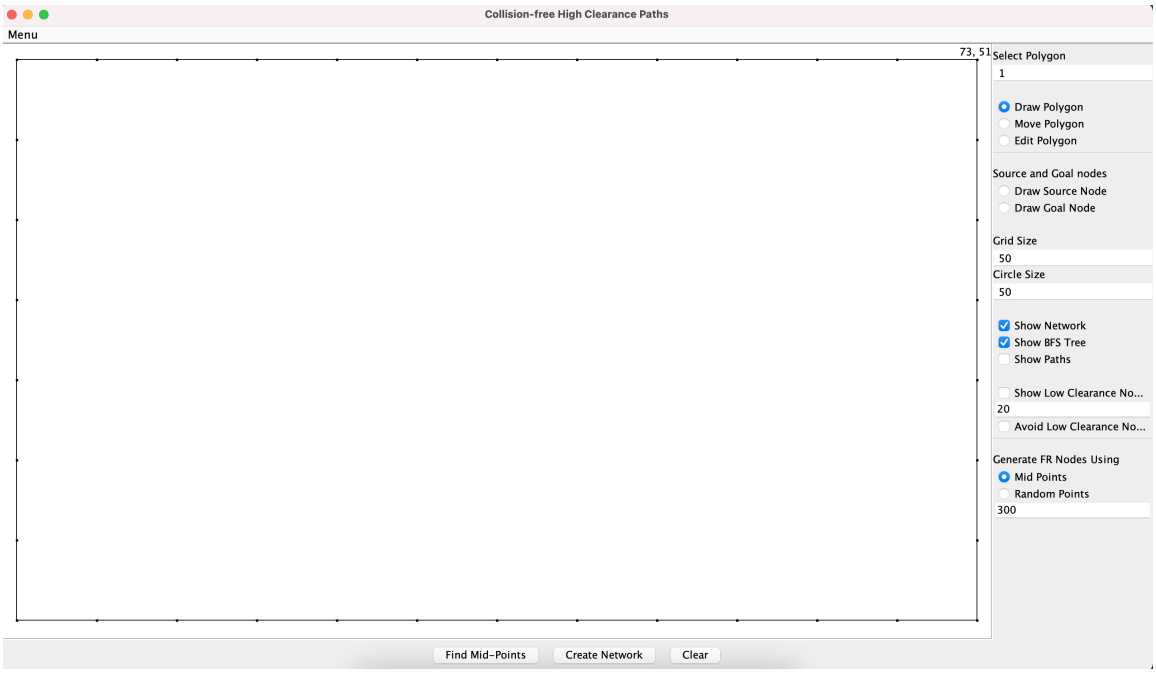


Figure 4.1: A Snapshot of the Application Interface

4.1.1 Menu Bar

The Menu bar sits at the top and has Menu options Read File and Save File. As the name suggests, Read File reads a text file*(.txt)* that contains the co-ordinates of the vertices of a collection of polygons and draws them in the drawing panel, which is detailed later. The Save File options lets users to save the polygons drawn in the drawing panel so that it can be reused. The saved file will have the co-ordinates of the polygons that can be redrawn using the Read File option.

S.No.	Menu Option	Description
1	Read File	Reads a text file and displays the polygonal obstacle layout.
2	Save File	Save the co-ordinates of the polygons in the drawing canvas in a .txt file.

Table 4.1: Menu Bar Description

4.1.2 Drawing Canvas

The Drawing canvas is the center panel where a user can draw polygonal obstacles by using mouse clicks. Similarly, user can draw a Source node and 3 Goal nodes for the path planning. The canvas also displays the Mid-points, BFS tree and the high clearance path. The mid-points, shown as black

dots, are displayed when Find Mid-Points button is clicked. The networks are shown as colored lines that connect the mid points. A general network that connects nearest midpoints are drawn as blue lines, a BFS tree starting from the Source node is drawn as green lines and the High clearance path computed form the BF tree is drawn as red lines. These are explained in details in following section. It also contains a preset rectangular boundary with Steiner's Points along the edges. At the top-right position of this panel, the current position(co-ordinates) of the mouse pointer in the drawing panel is displayed in x, y style. The drawings in the canvas are done using Mouse clicks and Mouse drags.

4.1.3 Control Panel

The Control Panel is a JPanel object positioned to the right side or East side of the application which contains all the functional control for the application. It contains various checkboxes, radio buttons and text boxes to select what and how to perform any action in the canvas. The functions are described in the following table in top to bottom order as displayed in the GUI.

The bottom or South panel consists of three action buttons. Find Mid-Points button computes the mid points between the vertices of the polygons along with the Steiner's point in the rectangular boundary and displays them in the drawing panel. The Create Network button generates a network connecting the mid-points, which is displayed in blue color. The network connections are determined by the values set in the Grid Size and Circle Size text input in the control panel. Furthermore, it also displays the BFS tree (in green color) and the High Clearance Path (in red color). These networks can be shown or hidden based on the checkboxes in the control panel as explained above. The final button is the Clear button, which clears all the polygons and mid-points drawn in the drawing panel along with the networks. This also resets any variables/values that were set while computing the networks.

S.No.	Control	Type	Description
1	Select Polygon	Text Input	Set the Polygon number that is to be operated on. Indicates the selected polygon.
2	Draw Polygon	Radio Button	Draw Polygon, identified by number set in above textbox, on canvas.
3	Move Polygon	Radio Button	Move the polygon as selected in the text input above.
4	Edit Polygon	Radio Button	Edit the vertices of the selected polygon.
5	Draw Source Node	Radio Button	Draws a Source node in the canvas.
6	Draw Goal Node	Radio Button	Draws Goal nodes in the canvas. The nodes are drawn in round-robin system, each click will draw node in the sequence 1, 2 & 3 and then starts again from 1.
7	Grid Size	Text Input	The size, length and width, of the grids that is used to position the circle for empty circle test to determine close nodes and connect them.
8	Circle Size	Text Input	The radius of the circle that is used to perform the circle test to connect the free-region nodes
9	Show Network	Checkbox	Displays the full network of connected free-region nodes when selected and Draw Network button is clicked
10	Show BFS Tree	Checkbox	Displays the BFS tree starting from the source node when selected
11	Show Paths	Checkbox	Displays the path computed from the BFS tree
12	Show Low Clearance Nodes	Checkbox	Marks the low clearance nodes with with encircling circle of radius defined in the Low Clearance threshold box
13	Low Clearance Threshold	Text Input	Defines the threshold for the low clearance nodes
14	Avoid Low Clearance Nodes	Checkbox	When selected, the nodes that are marked as low clearance are avoided while creating the free-region network, BFS tree and high clearance paths
15	Generate Free-Region Nodes Using	Label	
16	Mid Points	Radio Button	When selected, the free-region nodes are generated using the visibility graph mid point algorithm
17	Random Points	Radio Button	When selected, the free-region nodes are generated using the random point generation algorithm
18	Random Points	Text Input	The number of random points to be generated

Table 4.2: Control Panel Description

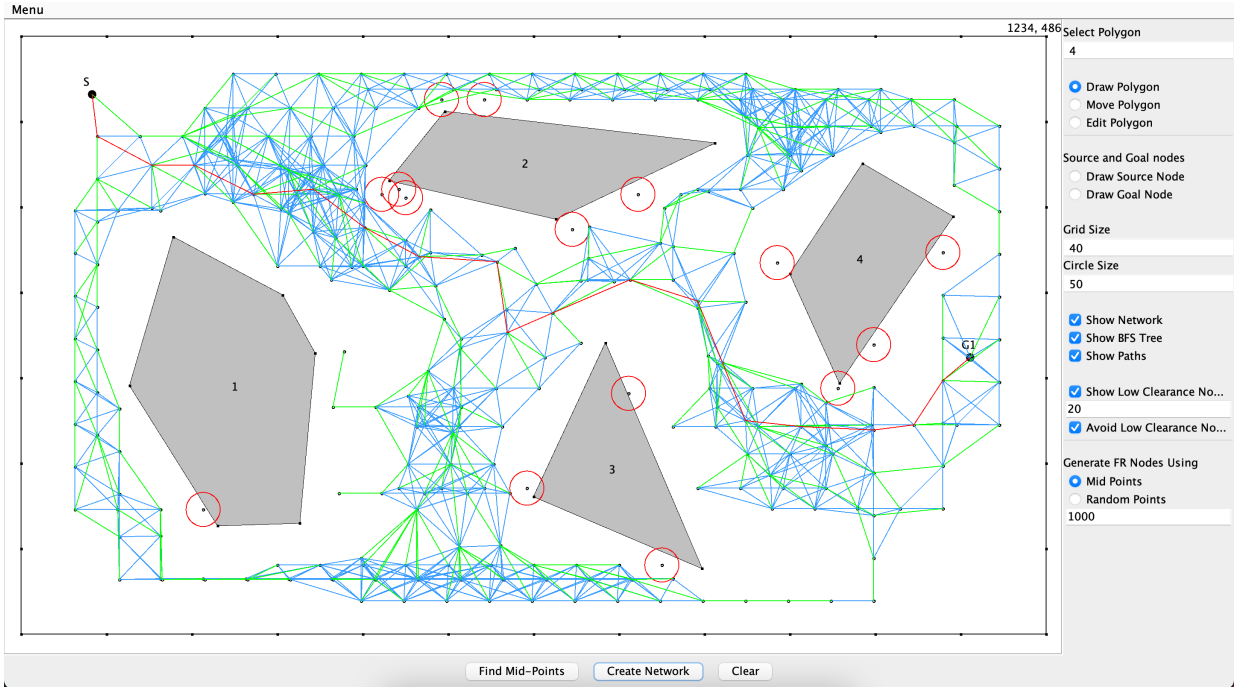


Figure 4.2: Polygonal Obstacles, Network and High Clearance Paths

4.2 Experimental Results

4.2.1 Mid-Point based Free-Region Nodes

In the application, a 7 polygonal obstacle set was designed. A source node was added at position $(114, 104)$. Then three goal nodes were added in different positions.

Sets	Data Label	Goal 1	Goal 2	Goal 3
Set 1	Location (x, y)	580, 91	492, 646	1121, 615
	Normal Least Clearance	51.546	5	19.026
	Enhanced Least Clearance	51.546	55.154	43.278
Set 2	Location (x, y)	125, 680	750, 434	434, 89
	Normal Least Clearance	35.114	27.784	51.546
	Enhanced Least Clearance	45.607	43.278	51.546
Set 3	Location (x, y)	954, 192	733, 549	512, 272
	Normal Least Clearance	51.546	15.62	37.161
	Enhanced Least Clearance	51.546	43.278	51.546

Table 4.3: Clearance Values for Normal and High Clearance Path

Similarly, for a set of 11 obstacles, we are experiments to compute the Euclidean distance of the path together with the total number of nodes in the generated path. We set the low clearance threshold - δ to 25 (px) and observed the values for normal clearance and enhanced(high) clearance

in the path.

Source (x, y)	Goal (x, y)	Clearance	Min Clearance	Nodes in Path	Euclidean Distance
1025, 650	151, 96	Normal	15.81	18	1165.36
		Enhanced	25.55	23	1311.15
	451, 106	Normal	17.49	13	882.55
		Enhanced	25.55	16	1010.74
	117, 322	Normal	16.00	18	1160.46
		Enhanced	25.46	23	1438.91
180, 124	1126, 129	Normal	20.40	19	973.95
		Enhanced	26.48	19	974.28
	497, 659	Normal	1.00	12	741.30
		Enhanced	27.00	12	744.09
	967, 397	Normal	15.81	16	989.83
		Enhanced	25.06	17	974.54

Table 4.4: Normal and Enhanced Clearance Path Comparison

4.2.2 Randomly generated Free-Region Nodes

In the application canvas, 7 polygonal obstacles of different shapes and sizes were added. A source node was placed at $(127, 113)$ position, and a goal node was placed at $(1109, 604)$. Next, random points were generated in the canvas (as mentioned in the section 3.1.3). The points were filtered first based on whether they were inside or outside the polygonal obstacles. The nodes inside the obstacles were removed. Similarly, the nodes that had low clearance, nodes that were closer to an obstacles than 40px, were removed. Now with the eligible free-region nodes, a graph network was created and using Breadth First Search with start point as the source node, and end point as the goal node. Form that path, we computed the number of nodes traversed, the euclidean distance of the path and the minimum clearance of the path from the obstacles. We also recorded the number of free-region nodes and high clearance nodes. This process was repeated for different number of random points and the data is presented in the following table.

Figure 4.3 illustrates the comparison of high clearance nodes, low clearance nodes and nodes that lie inside the polygonal obstacles for different number of randomly generated nodes. We can observe that the randomness is fairly preserved on the number of each of such type of nodes.

Total Nodes	Free-Region Nodes	High Clearance Nodes	No. of Nodes in Path	Euclidean Distance	Minimum Clearance
300	239	208	20	1386.89	40.25
350	274	245	18	1406.95	41.01
400	312	291	16	1354.18	40.31
450	359	337	16	1380.13	41.01
500	409	363	16	1262.49	47.51
550	431	379	13	1231.19	40.46
600	469	439	16	1427.40	45.01
650	499	470	15	1271.52	41.23
700	552	496	14	1291.13	46.09
750	593	544	15	1275.41	48.27
800	628	586	14	1228.52	52.63
850	669	616	15	1210.83	41.86
900	697	633	14	1195.35	41.86
950	756	657	14	1248.47	40.80
1000	789	717	14	1209.17	45.45

Table 4.5: Random Point Generation and Path Statistics

The number of nodes in the generated high clearance path is illustrated in Figure 4.4. It can be observed that the number of nodes in the path decreases as the high clearance nodes increases until total number of generated nodes reach 600. After that the number of nodes in path remain fairly consistent to about 14.

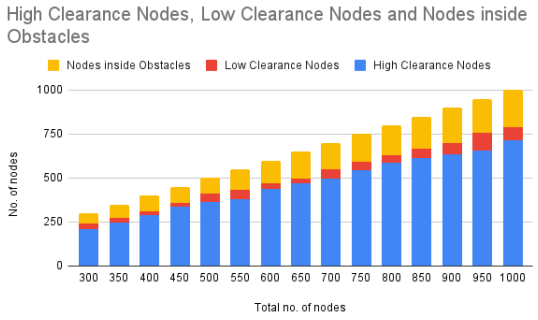


Figure 4.3: Comparison of Randomly Generated Nodes

Furthermore, we experimented with varying free-region space and how the randomly generated nodes would perform. We constructed polygonal obstacles of different shape and sizes to occupy different area in the rectangular boundary. The free space thus obtained were 67%, 75% and 80%. In each of such settings, we randomly generated 600, 800 and 1000 nodes. We then computed the free-region nodes, high clearance nodes, low clearance nodes and nodes that were generated inside the obstacles. Figure 4.5 illustrates the comparison of such nodes across different free-region space.

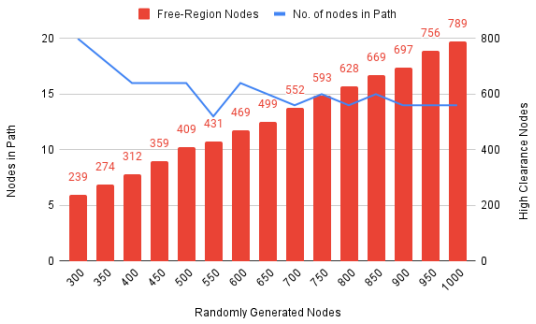
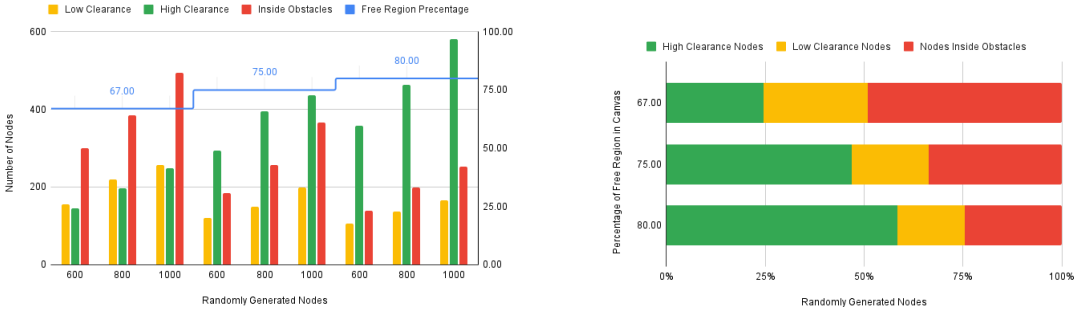


Figure 4.4: Comparison of Number of Nodes in Path



(a) Different Types of Generated Nodes

(b) Percentage of Different Nodes

Figure 4.5: Comparison of Randomly Generated Nodes Against Free-Region Area

We can observe that as the free space increases, so does the high clearance nodes and the low clearance and nodes inside obstacles decreases. Even with an 8% increase, from 67% to 75%, in free space, we can see that the high clearance nodes significantly increases. This behavior is seen to be consistent across the different number of total randomly generated nodes.

Chapter 5

Discussion

We presented a brief review of algorithms for constructing collision-free paths connecting two nodes in a two dimensional Euclidean space containing convex obstacles. We introduced new algorithms for constructing collision-free paths based on generating nodes outside obstacles which could be possible candidates for interior nodes of the path. The generated nodes are processed to make the constructed collision-free path to have more clearance from obstacles. The free-space nodes are generated in two ways: One guided by the visibility edges induced by obstacle set and other randomly generated nodes that are checked to lie outside obstacle set. We presented some experimental results on the quality of generated paths. The experimental results were obtained by implementing presented algorithms in Java programming language.

There could be several extensions of the algorithms presented in this thesis as described next. We developed the presented algorithms in the presence of convex polygonal obstacles. It would be nice to extend our algorithm to the case of non-convex obstacles. One way in this direction would be to replace each non-convex obstacle O_i with O_i' which is the convex hull of O_i. This approach would work correctly only if the convex hull of obstacles do not overlap with each other and the source or the goal node does not lie inside the convex hull of the obstacles. Figure 5.1 illustrates overlapping of non-convex polygonal obstacles. In the figure, the convex hull of the polygons D & E and polygons F & G overlap with each other.

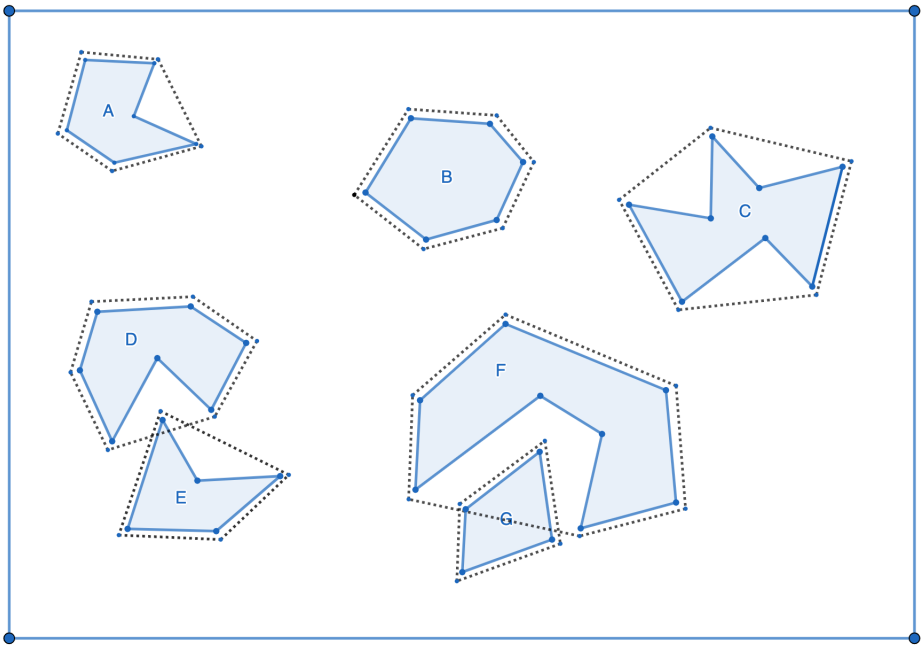


Figure 5.1: Convex Hulls of Polygonal Obstacles

The collision-free path constructed by the presented algorithm could have very sharp turn angles. It would be a good exercise to modify the proposed algorithm to avoid sharp turns in the generated path.

Bibliography

[dBvKOS00] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.

[EDPB15] John Edwards, Eric Daniel, Valerio Pascucci, and Chandrajit Bajaj. Approximating the generalized voronoi diagram of closely spaced objects. *Computer Graphics Forum*, 34(2):299–309, 2015.

[GM87] Subir Kumar Ghosh and David M. Mount. An output sensitive algorithm for computing visibility graphs. pages 11–19, 1987.

[JLLC21] Gene Jan, Ming Lee, Chaomin Luo, and Wei Chiang. Obstacle-avoidance path planning based on delaunay triangulation. 08 2021.

[O'R98] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 1998.

[Ora23] Oracle. Java development kit version 20 api specification. 2023.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Barun Thapa
barunthapa.bvdt@gmail.com

Degrees:

Bachelor of Engineering in Computer Engineering, 2012
Tribhuvan University

Thesis Title: High Clearance Collision-free Paths

Thesis Examination Committee:

Chairperson, Dr. Laxmi Gewali, Ph.D.
Committee Member, Dr. Kazem Taghva, Ph.D.
Committee Member, Dr. Mingon Kang, Ph.D.
Graduate Faculty Representative, Dr. Henry Selvaraj, Ph.D.