PATH PLANNING FOR UNMANNED AERIAL VEHICLES USING VISIBILITY LINE-BASED METHODS

Thesis submitted for the degree of

Doctor of Philosophy

at the University of Leicester

by

Rosli bin Omar

Control and Instrumentation Research Group

Department of Engineering

University of Leicester

March 2011

Abstract

PATH PLANNING FOR UNMANNED AERIAL VEHICLE USING VISIBILITY LINE-BASED METHOD

Rosli bin Omar

This thesis concerns the development of path planning algorithms for unmanned aerial vehicles (UAVs) to avoid obstacles in two- (2D) and three-dimensional (3D) urban environments based on the visibility graph (VG) method. As VG uses all nodes (vertices) in the environments, it is computationally expensive. The proposed 2D path planning algorithms, on the contrary, select a relatively smaller number of vertices using the so-called base line (BL), thus they are computationally efficient. The computational efficiency of the proposed algorithms is further improved by limiting the BL's length, which results in an even smaller number of vertices. Simulation results have proven that the proposed 2D path planning algorithms are much faster in comparison with the VG and hence are suitable for real time path planning applications. While vertices can be explicitly defined in 2D environments using VG, it is difficult to determine them in 3D as they are infinite in number at each obstacle's border edge. This issue is tackled by using the so-called plane rotation approach in the proposed 3D path planning algorithms where the vertices are the intersection points between a plane rotated by certain angles and obstacles edges. In order to ensure that the 3D path planning algorithms are computationally efficient, the proposed 2D path planning algorithms are applied into them. In addition, a software package using Matlab for 2D and 3D path planning has also been developed. The package is designed to be easy to use as well as user-friendly with step-by-step instructions.

Acknowledgements

First of all, thanks to Allah for the blessing and opportunity for me to successfully complete my studies. I would like to express my deepest gratitude to my thesis advisor Professor Da-Wei Gu for his continued guidance and support. His drive and enthusiasm for research is captivating, and he was able to continuously motivate me throughout this research. His insight and knowledge in the subject matter were invaluable to my success. I must also express my gratitude to the sponsors of this research effort namely The Ministry of Higher Education (MOHE) of Malaysia and Universiti Tun Hussein Onn Malaysia (UTHM). Their support is gratefully acknowledged. I would also like to thank my peers in the Control Systems Research laboratory including Dr. Halim Alwi, Naeem Khan, Mangal Kothari, Bharani Chandra, Dr Sajjad Fekri, Dr. Andrade Emmanuel and Dr. Georgios Kladis. All of them could be counted on to provide necessary insight into the research at hand. Special acknowledgement goes to my wife, Zaharah Mohamed Nor. She has given me unconditional support throughout my studies. Last but not least I would like to express my special thanks to my parents and family for their prayers and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Unmanned Aerial Vehicles (UAVs) are a vital means of performing hazardous missions in adversarial environments without endangering human life. They have been used for peaceful purposes in civilian applications such as weather forecasting, environmental research, search and rescue missions, observation during wildfire incidents and traffic control [3]. Fig. 1.1 illustrates a Pathfinder UAV used for environmental research. On the other hand, UAVs have also been used for warfare such as carrying out aerial reconnaissance and surveillance over the opponent's area or attacking strategic facilities in enemy territory. Fig. 1.2 shows an RQ-1 predator which is armed with missiles for combat purposes [1].

Figure 1.1: Pathfinder UAV used for environmental research.

Figure 1.2: A UAV, RQ-1 predator is equipped with missiles

Since UAV requires no human pilot, there is no loss to human life if it crashes or gets attacked during a mission. Besides, UAV also reduce operating costs because it does not require a highly trained pilot onboard as a manned aircraft does. The latter is cost-ineffective often caused by expensive investment needed as part of the pilot's training to cover advanced facilities such as buildings, flight simulators and support equipment including instrumentation, the cockpit and ejection systems. Therefore UAVs are by far the best way forward. In addition, with no human pilot, a UAV can be designed to achieve higher gravitational forces i.e. 50g [2], which results in relatively higher manoeuvrability (a human can sustain up to only 9g). A UAV with higher manoeuvrability may have better performance such as faster speed, smaller minimum turning radius and larger maximum roll angle and hold a higher probability of escaping from enemy's missile attack.

However, many current UAVs still involve a human-in-the-loop to oversee and control the UAVs' operation [4, 31]. This in turn requires a communication link through radio signals between the human operator and the UAV to transmit/receive the command/sensory signals over a frequency spectrum, which is often limited. Furthermore, the radio signal is vulnerable and might be jammed by opponents. In the event of a lost or interrupted signal, as the UAV is dependent on human operators' decisions, it would not be able to execute a mission as desired and to some extent, it

may crash. Thus, the dependency on human instructions through a communication link needs to be minimised or eliminated if possible. This requires the UAV to have the capability of making its own decisions based on the current state and circumstances of its surrounding environments. The capability of doing so will greatly enhance the autonomy of UAVs.

1.2 Autonomy in UAV

Current technologies are capable of operating a UAV in a relatively structured and known environment. However, in a dynamic environment where uncertainties exist such as obstacles that might pop-up during a mission, the technologies are insufficient due to the UAV's inability to make decisions by itself [32]. This requires a new concept called autonomy.

Autonomy means the capability of a UAV to make its own decision based on the information presently available captured by sensors, and potentially covers the whole range of the vehicle's operations with minimal human intervention [5]. Autonomy increases system efficiency because all decisions are executed onboard except for critical decisions such as launching a missile that have to be made by humans [30]. A UAV with autonomy would be able to execute a mission in environments with uncertainties. Furthermore, with autonomy, the UAV can perform a long duration mission, which is beyond the capability of human (operators). Autonomy covers the following areas [6]:

i. sensor fusion
ii. communications
iii. path planning
iv. trajectory generation
v. task allocation and scheduling
vi. cooperative tactics

Additionally, as introduced in [33], there are ten UAV autonomy levels known as Autonomous Control Level (ACL). The ACL and trends in UAV autonomy are illustrated in Fig. 1.3. The concept of ACL as a metric to describe the autonomy in

UAVs is widely accepted [31]. Readers are referred to [33] for a detailed description of ACL.

However, autonomy technology is still in its early stage, fairly undeveloped [5] and is the bottleneck for UAV development in the future [6]. The RQ-1 Predator as shown in Fig. 1.1 for example, at present, can perform up to only level 3 of ACL.

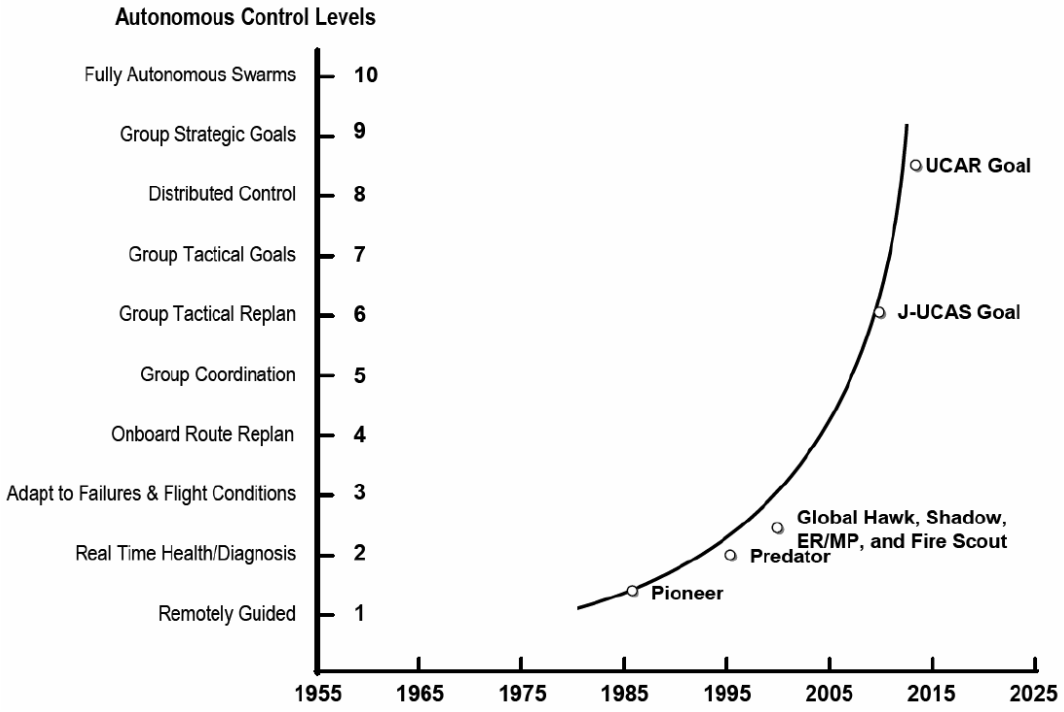


Figure 1.3: UAV autonomy levels and trend (adapted from [33])

The list of autonomy areas included previously, as well as the ACL (Fig. 1.3), have shown that onboard path planning and re-planning, which deals with traversing a vehicle through obstacles is one of the keys components of autonomy.

Research on UAV autonomy including path planning have progressed steadily since the beginning of this century. For example, [31] has designed and conceptually developed a simple UAV path planning mission that is used to reduce the UAV's dependency on human operators, and hence increases the UAV's autonomy level. The so-called Mission Management System (MMS) has been designed, developed and flight-tested in [31]. From sensory data, MMS makes decisions and issues high level commands which are then executed by the Flight Control Systems (FCS).

As path planning plays an important role in enhancing UAV's autonomy level, it has to be considered in the design of a UAV.

1.3 Path Planning Overview and Issues

From a technical perspective, path planning is a problem of determining a path for a vehicle in a properly defined environment from a starting point to a target point such that the vehicle is free from collisions with surrounding obstacles and its planned motion satisfies the vehicle's physical/kinematic constraints [25]. In a report by [12], path planning is associated with a number of terms as follows:

- ***Motion planning***

  This term is frequently associated with manipulator robotics. It involves deliberative high level and low level planning of a way to move a robotic manipulator.

- ***Trajectory planning***

  It is about planning the next movement of a robot. Trajectory planning is similar to motion planning.

- ***Navigation***

  It is a very general term which has several meanings. In general it means "getting there from here". It is also part of path planning, motion planning, obstacle avoidance and localisation.

- ***Global path planning***

  The planning is done prior to vehicle movement. It uses the information from the surrounding world to reach a target point from a starting point. As the information contains global data, the process is slow, but the planned path may be optimal.

- ***Local navigation***

  It is a process of avoiding obstacles by using only acquired data of the current surrounding environment. It is also a process of ensuring the vehicle's stability and safety and runs in real time using a reactive path planning approach.

1.3.1 Criteria of Path Planning

Path planning related problems have been extensively investigated and solved by many researchers [7-10], mostly focusing on ground robotics and manipulators. Important criteria for path planning that are commonly taken into account are the computational time, path length and completeness. A path planning algorithm with less computational time is vital in real time application, which is desirable in dynamic environments. The generated optimal path in terms of path length by a path planning technique will minimise UAV flight time and hence prolongs the UAV's endurance and life cycle, minimises fuel/energy consumption and reduces exposure to possible risks. On the other hand, a path planning approach satisfies the completeness criterion if it is able to find a path if one exists.

However, sometimes, there are trade-offs between such criteria. For example, a path planning method has to disregard the path's optimality in order to increase the computational efficiency. It means that finding a slightly longer path with less computational time may be preferable. On the other hand, higher computational complexity is necessary if an optimal path is required for some reasons. These criteria have to be considered before any path planning technique/algorithm design process takes place.

1.3.2 Path Planning Steps

Typically, path planning of a vehicle $\mathcal{A}$ consists of two phases. The first phase is called the pre-processing phase in which nodes and edges (lines) are built in the environment/workspace $\mathcal{W}$ with $\mathcal{A}$ and obstacles O. In this phase, it is common to apply the concept of a configuration space (C-space) to represent $\mathcal{A}$ and O in $\mathcal{W}$ [9, 12]. In C-space, the vehicle's size is reduced to a point, and accordingly the obstacles' sizes are enlarged according to the size of $\mathcal{A}$. Next, representation techniques are used

to generate maps of graphs. Each technique differs in the way it defines the nodes and edges.

The second phase of path planning is termed the query phase in which a search for a path from a starting point to a target point is performed using (graph) search algorithms.

However there are path planning methods that can find solutions without graph search algorithm such as Mixed Integer Linear Programming (MILP) [4, 105, 116-117] and Evolutionary Algorithm (EA) [118-120].

1.3.3 *C*-space Representation

In path planning for an object, there are a number of methods that are commonly used to represent the environment including potential field (PF) [21-24], cell decomposition (CD) [13-16] and roadmap (RM) [17-20], to name a few. A PF represents the environment by modelling the object as a particle, moving under the influence of potential fields throughout the C-space. The field's magnitude at a particular point in C-space is determined by the fields generated by starting point p_{start}, target point p_{target} and the obstacles O in the C-space. The p_{start} and O are repulsive surfaces (which generate repulsive forces), while the p_{target} is the attractive pole which generates attractive forces [21]. The path is then calculated based on the resulting potential fields from a point with the highest magnitude of the resultant potential field, i.e. p_{start}, to a point with the minimum potential, i.e. p_{target}. The PF has several advantages such as the planning process is done as the vehicle moves and thus is suitable for real time application and the generated path is also smooth. However, conventional PF methods suffer from local minima causing the vehicle to become stuck before it reaches p_{target}, hence it might not satisfy the completeness criterion.

CD-based are among the most popular methods to represent the environment especially for outdoor scenarios [12] as it is the most straightforward technique [29]. This is due to the fact that the cells can represent anything such as free space or obstacles. The first step in CD is to divide the C-space into simple, connected regions termed cells [35]. The cells are regions that might be square, rectangular or polygonal in shape. They are discrete, non-overlapping but adjacent to each other. If the cell

contains obstacle (or part of obstacle), it is marked as occupied, otherwise it is marked as obstacle free. A connectivity graph is then constructed and a graph search algorithm is used to find a path throughout the cells from the starting point to the target point. In order to increase the quality of the path, the size of the cells has to be made smaller, which in turn increases the grid's resolution, and hence computational time. In the literature, there are several variants of CD. These include Approximate Cell Decomposition, Adaptive Cell Decomposition and Exact Cell Decomposition.

Path planning using RM-based methods on the other hand represent the environment by constructing graphs or maps from sets of nodes and edges. Path planning methods which are specific cases of RM are Voronoi diagrams (VD) and Visibility Graphs (VG). The nodes and edges to build a roadmap are defined differently for each method. VD defines nodes that are equidistant from all the points' surrounding obstacles. The paths generated from a graph by VD are relatively highly safe due to the fact that the edges of the paths are positioned as far as possible from the obstacles. However, the paths are inefficient [12] and not optimal in terms of path length. On the other hand, VG uses the vertices of the obstacles including the starting and target points in the C-space as the nodes. A VG (or visibility lines, VL) network is then formed by connecting pairs of mutually-visible nodes by a set of lines E. A pair of mutually-visible nodes means that those nodes can be linked by a line/edge $e \in E$ that does not intersect with any edge of obstacles in the C-space. Additionally, there is a cost associated with each E, possibly in terms of Euclidean distance. One advantage of VL is the capability of finding a path with the shortest length if one exists. A standard VL's computational complexity is $O(N^3)$ to find a path in a C-space with N nodes therefore VL is computationally intractable in the C-space with many obstacles.

1.3.4 Graph Search Algorithms

It has previously been stated that the second step of path planning is to calculate a path using (graph) search algorithms. Two basic search algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS). BFS searches paths in a systematic way which guarantees that the first solution found will utilise the smallest number of iterations [34]. Like BFS, DFS is also systematic but it focuses on one direction and completely misses large portions of the C-space as the number of iterations become very large.

Both BFS and DFS need to consider every node in the graph in calculating the best possible path [12], hence generating a path might take a relatively long time for a large environment with a large number of nodes. In order to address this issue, there are a variety of search algorithms such as Dijkstra's and A* (pronounced A-star) algorithms [12] which consider only a subset of the nodes. Dijkstra's algorithm generates the shortest path by considering the costs from the current node to the starting point. A* on the other hand calculates a path based on the costs from current node to both starting and target points.

1.3.5 Real Time and Off-line Path Planning

A path planner is called real time if it incrementally finds and modifies a path in the course of the UAV's flight. A sensor is used to detect any obstacles with locations that are on the collision course of the UAV path. If the sensor detects obstacles, the information is fed to the path planner, and subsequently, a collision-free path is planned. On the other hand, a path planner is termed offline if it plans the path before the flight starts. The path, which is normally optimal, is constructed based on the data of the environment acquired either by satellite, surveillance or other means.

1.4 Assumptions and Problem Statement

A path planning problem for a UAV in a two-dimensional (2D or $\mathbb{R}^2$) or three-dimensional (3D or $\mathbb{R}^3$) environment through stationary polygonal obstacles, $O = \{O_1, \dots, O_n\} \subset \mathbb{R}^2$ (or $O = \{O_1, \dots, O_n\} \subset \mathbb{R}^3$), from a designated starting point p_{start} to the target point p_{target} have been considered in this thesis. It is assumed that the environment is a well-built urban area and O are hard, rectangle-shaped obstacles (buildings). It is also assumed that the knowledge of the entire or part of the environment such as the geometries, dimensions and locations of O are known *a-priori* either from surveillance, satellite data or other means. The resultant path has to be collision-free and consists of waypoints $W = \{w_0, \dots w_n\}$, which is defined by positions $w_j = \{x_j, y_j\}$ (or $w_j = \{x_j, y_j, z_j\}$ in $\mathbb{R}^3$) where $j = 0 \dots n$. Note that w_0 and w_n are the p_{start} and p_{target}, respectively. Two consecutive waypoints are connected by piece-wise linear segments from p_{start} to p_{target}.

Once the UAV starts its mission by traversing along the planned 2D path, the environment may change where pop-up and/or the previously unknown obstacles might appear on the path. The UAV is assumed to be equipped with sensors of limited range to collect information about the environment such as pop-up and/or the previously unknown obstacles. Using the collected information from the sensors, a new path has to be re-planned in real-time to avoid any collision with the surrounding obstacles.

1.5 Thesis Contributions

In order to address the problems that have been stated in the previous section, several solutions that are the contributions of the thesis are proposed.

The first contribution is the development of a set of algorithms for 2D path planning. The outcome of the proposed algorithms is an optimal, collision-free path with a fixed altitude. The proposed algorithms are based on the Visibility Line (VL) method and Dijkstra's algorithm. Contrary to the VL approach[1], the proposed algorithms find paths by reducing the number of obstacles (as well as nodes (vertices) and edges), which lowers the computation time and is therefore suitable for a real time path planning application. It is emphasised that the VL approach and Dijkstra's algorithm are chosen because they are guaranteed to produce optimal path, if one exists [12]. An optimal path, within the context of this thesis, means the path that has the least distance from p_{start} to p_{target}. It is also worth emphasising that the proposed 2D algorithms possess the aforementioned criteria of path planning and may capable of finding a globally optimal path if the knowledge of the environment is fully and accurately known. The algorithms are also computationally efficient as the number of obstacles that are used for path calculation is relatively small. On the other hand, the proposed algorithms hold the completeness criterion as it will generate a path, if one exists.

The second contribution of the thesis is the development of a set of path planning algorithms in 3D environments which are based on the proposed 2D ones. Unlike 2D, the proposed 3D path planning algorithms consider the heights of obstacles in the environments as well as the altitudes of p_{start} and p_{target}. They apply the concept of planes rotation, in which the nodes, which are used to find 3D paths, can be identified

[1] VL (with Dijkstra's algorithm) uses the entire obstacles in the environment to find an optimal path.

efficiently from the intersections between the rotated planes and 3D obstacles edges. Hence the proposed 3D algorithms solve the problem of conventional VL methods, in which determining the nodes of 3D obstacles are difficult. The proposed algorithms hold the completeness criterion.

Additionally, a couple of Graphical User Interfaces (GUIs) to realise the 2D and 3D algorithms have also been developed. The GUIs are designed to be user-friendly, equipped with step-by-step instructions to guide the user. Using the GUIs for path planning either in a 2D or 3D environment, a random or particular scenario can be generated. The p_{start} and p_{target} can also be located at any points in a provided axis. A collision-free path is then found at the click of a button. The necessary information of the planned paths is also displayed in the GUIs.

1.6 Thesis Structure

This thesis is structured in the following manner:

Chapter 2 presents an extensive literature survey of visibility graph (or visibility line) and graph search algorithms. It begins with defining path planning, and then discussing the importance of path planning and its criteria. An introduction to VL and related research are presented. Also, several established graph search algorithms are briefly explained. The chapter also briefly discusses real-time path planning and path planning in 3D environments.

Chapter 3 discusses the proposed path planning algorithms in 2D environments based on the VL method and Dijkstra's algorithm. The chapter also demonstrates the application of the algorithms to real-time path planning. The safety margin is also introduced for a collision-free path. Also, the improvement of the proposed 2D path planning algorithms is highlighted.

Chapter 4 discusses the proposed 3D path planning algorithms. The concept of rotational planes that are utilised by the algorithms is explained. This chapter also shows the simulations to evaluate the effect of the number of obstacles and rotation angles to computation time and path length.

Chapter 5 presents the Graphical User Interfaces (GUIs) for path planning in 2D and 3D environments. Guidelines on how to use the GUIs are also provided.

Chapter 6 provides conclusions based on the work in this thesis. This chapter also presents possible areas of future research to extend the work developed.

Chapter 2

Path Planning

2.1 Introduction

One of the open issues in the development of autonomous vehicles such as Unmanned Aerial Vehicles (UAVs) is path planning. In its most general form, the path planning problem for an autonomous vehicle $\mathcal{A}$ in an Euclidean space $\mathcal{W}$ can be stated in the following way [45]: Given an initial starting point p_{start}, a target point p_{target} and a set of obstacles O whose geometry is known to $\mathcal{A}$, determine if there exists a continuous obstacle-avoiding motion for $\mathcal{A}$ from p_{start} to p_{target}. If one exists, construct the path for such a motion. Note that $\mathcal{W}$ is called the workspace, represented as $\mathbb{R}^N$, with N=2 or 3 for 2D and 3D, respectively.

Path planning is necessary for autonomous vehicle to find a safe route to be traversed from p_{start} to p_{target}. Research on path planning in environments with polygonal obstacles have been around since the beginning of mobile robots. As such, many path planning techniques, which are categorised under geometric-based, grid-based or potential field, to name just a few, have been documented in ground robotics and manipulators systems [23, 38, 51-57]. Nevertheless path planning for Unmanned Aerial Vehicles (UAVs) have also applied such techniques.

Most existing path planning methods involve a two-step process to generate collision-free paths. The first step is to represent $\mathcal{W}$ either in a two- (2D) or three-dimensional (3D) space with a graph or map. This step is called the pre-processing phase. The next step is the query phase, in which the p_{start} and p_{target} are incorporated into the graph or map. Then a path is calculated through the represented environment using a (graph)

search algorithm. However there are several path planning methods that don't require the graph search algorithms to find paths such as Mixed Integer Linear Programming (MILP) [4, 105, 116-117] and Evolutionary Algorithm (EA) [118-120].

It is important to have a path planning method/algorithm that calculates a safe path in the shortest time possible so that it can be applied in real-time in order to deal with changes in an environment. In a changing environment, a previously unknown or pop-up obstacle might be encountered by a UAV through its onboard sensors during a mission. Quick path re-planning by the UAV's path planner to find an alternative safe path in real-time is important in order to successfully accomplish a given mission.

A good path planning method/algorithm must not only provide a safe path, it also has to be able to find the shortest path. The shortest path is crucial in order to minimise travel time, saves energy/fuel, lower the possible traverse risks exposure and prolong the vehicle life cycle.

However, practically, UAVs fly in a 3D environment. Thus representing the environment in 2D for path planning leads to a path that has constant altitude, which might not optimal. A UAV that flies with constant altitude undoubtedly has the advantage of saving the vehicle's fuel. Instead of ascending, the vehicle would only need to change its heading either to the left or right to avoid obstacles. Ascending consumes extra energy/fuel in order to increase the UAV's thrust level as the UAV has to defy the gravitational force. However, as the real environment is in 3D, it is crucial for a path planning algorithm to be able to generate 3D paths in such an environment because unlike 2D path, a path in 3D has a variable altitude. Such a path may be shorter in distance, which may consume less fuel, less risk and has a longer life cycle than that of a 2D path.

In this chapter, path planning aspects in general are discussed starting with the introduction of the configuration space (C-space) followed by a discussion on the path planning technique using visibility graph (or visibility line (VL)) in $\mathcal{W}$. Then several existing graph search algorithms are discussed. Prior to the conclusion of the chapter, a review of the planning techniques in real time as well as in a 3D environment will be discussed.

2.2 Workspace Representation

The representation of the environment is generally the first phase of the path planning process which involves recognising objects/obstacles in the environment and identifying free space to manoeuvre. In this phase, a map or graph is created considering the configuration of the vehicle and the obstacles. Note that a configuration of an object is defined as a position specification of all points of this object relative to a fixed reference frame [14]. Path planning through polygonal obstacles has led to the development of the configuration space (*C*-space) concept, which allows the specification of the obstacles and the vehicle positions.

In a *C*-space, there are a number of techniques that can be used to represent the environment (including the vehicle and obstacles). This section focuses on the description of *C*-space and reviews the workspace representation using the so-called visibility line (VL).

2.2.1 Configuration space

Configuration space (*C*-space) is the common concept behind most path planning methods to represent the workspace $\mathcal{W}$. *C*-space ($\mathcal{Q}$) is the space of all possible specifications of a vehicle $\mathcal{A}$ and obstacles region O in $\mathcal{W}$ ($\mathcal{W} = \mathbb{R}^2$ in 2D and $\mathcal{W} = \mathbb{R}^3$ in 3D). In path planning, *C*-space is used to ensure that $\mathcal{A}$ doesn't intersect O in $\mathcal{W}$. The *C*-space concept is widely used in path planning problems as it is a key construction and formalism for path planning and it also provides a uniform framework that allows the comparison and evaluations of different algorithms [12, 29]. One way to represent the configuration of $\mathcal{A}$ is to define its centre point $q = (x,y)$ relative to some fixed coordinate frame [41]. If the radius r (or a distance from the furthest point to the centre) of $\mathcal{A}$ is known, it is possible to determine the set of points occupied by $\mathcal{A}$ from the configuration, q. If the notation $V(q)$ represent the set of points, then

$$V(x,y) = \left\{(x',y') \middle| (x-x')^2 + (y-y')^2 \leq r^2 \right\}$$

The above notation shows that, it is sufficient for x and y to completely specify the configuration of $\mathcal{A}$. In C-space with an obstacle region $O = \{o_1, o_2,\ldots,o_p\}$, the set of configuration of obstacle region at which $\mathcal{A}$ will intersect O_i is defined as

$$\mathcal{Q}o_i = \{q \in \mathcal{Q} \mid V(q) \cap o_i \neq 0\}$$

Conversely, the free configuration space in which the vehicle will traverse is

$$\mathcal{Q}_{free} = \mathcal{Q} \setminus (\cup_i \mathcal{Q}o_i)$$

In order to illustrate how the configuration space is created, consider the circular vehicle and an obstacle o_i in a $\mathcal{W}$ as shown in Fig. 2.1(a). By sliding the vehicle around the obstacle as well as the boundary of $\mathcal{W}$, the obstacle configuration $\mathcal{Q}o_i$ is constructed and shown in Fig. 2.1(b). Meanwhile the vehicle transformed into a point in the C-space where the shaded area represents $\mathcal{Q}o_i$ while white region represents $\mathcal{Q}_{free}$ is shown in Fig. 2.1(c) wherein C-space reduces the problem of finding a collision-free path of $\mathcal{A}$ in $\mathcal{W}$ to that of a point in $\mathcal{Q}_{free}$.

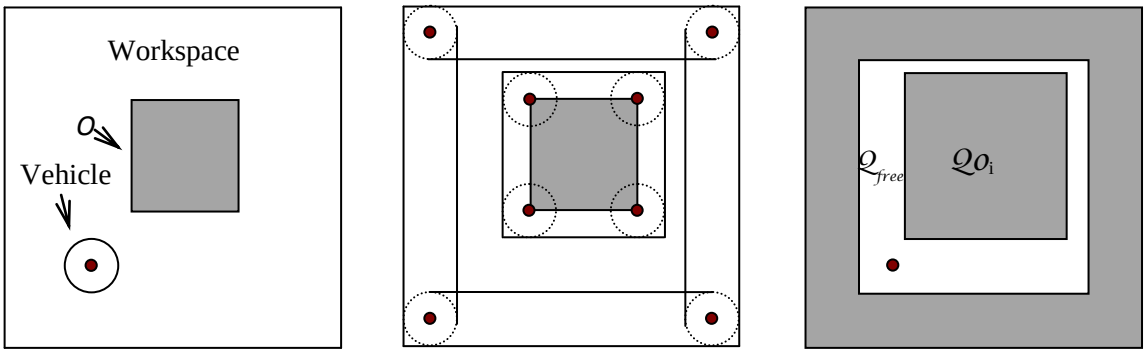
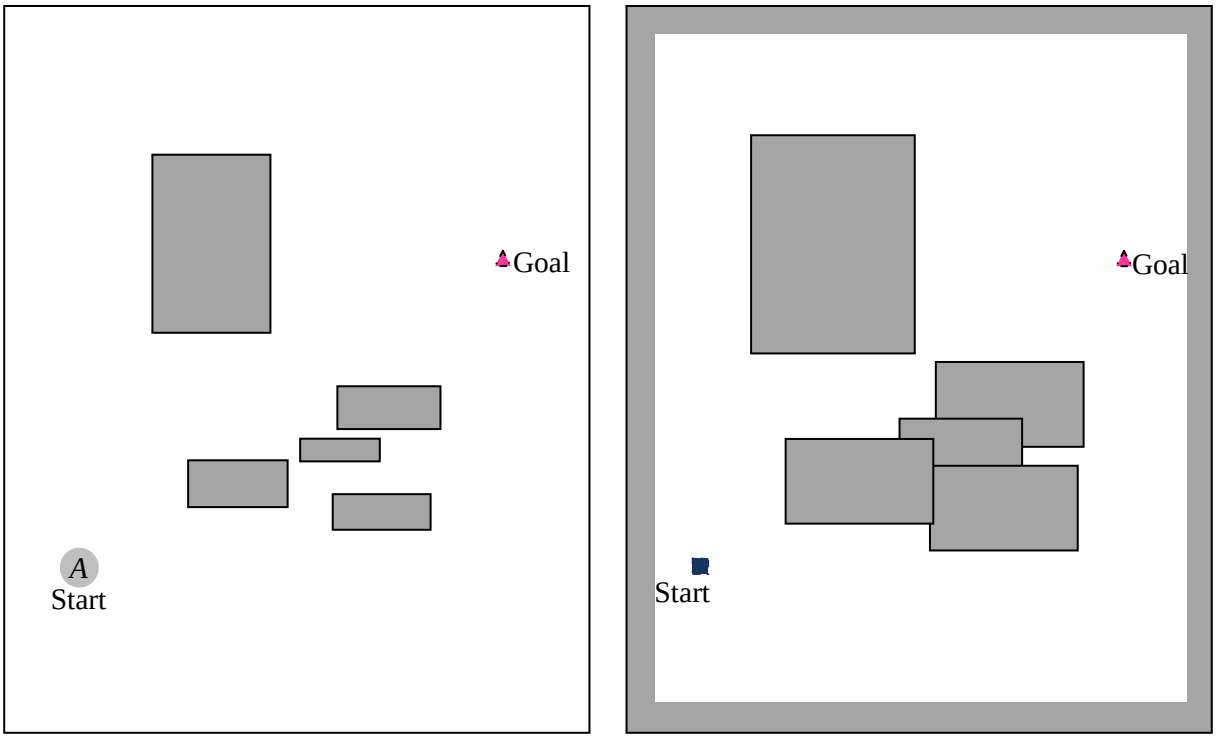


Figure 2. 1: A circular vehicle is transformed into a point in C-space

In Fig. 2.1 (c), the corners of the obstacle are supposed to be curvy; however they are made sharp as most of path planning representations techniques use nodes to find paths.

To demonstrate how to create a C-space from $\mathcal{W}$ for a scenario, which contains a number of obstacles, consider Fig. 2.2 (a) whose obstacles are in their actual sizes. The C-space of the scenario is then created based on the size of $\mathcal{A}$ and illustrated in Fig. 2.2(b). Having the C-space defined, now the problem of finding a path from the *Start* to the *Goal* points as illustrated in Fig. 2.2 (a) is reduced to that of a point in the $\mathcal{Q}_{\text{free}}$ as shown in Fig. 2.2 (b).



(a) (b)

Figure 2.2: A scenario represented in (a) original form (b) configuration space. The darker rectangles in (a) are those with actual dimensions while in (b) are those enlarged according to the size of vehicle *A*. The white areas are the free space.

2.2.2 *C*-Space Representation Techniques

After applying the *C*-space concept to the environment, the next step is to represent the *C*-space. There are three categories of representation techniques including roadmaps, cell-composition and potential fields. Most path planning methods fall under one of those categories. Fig. 2.3 below shows the categorised *C*-space representation techniques.

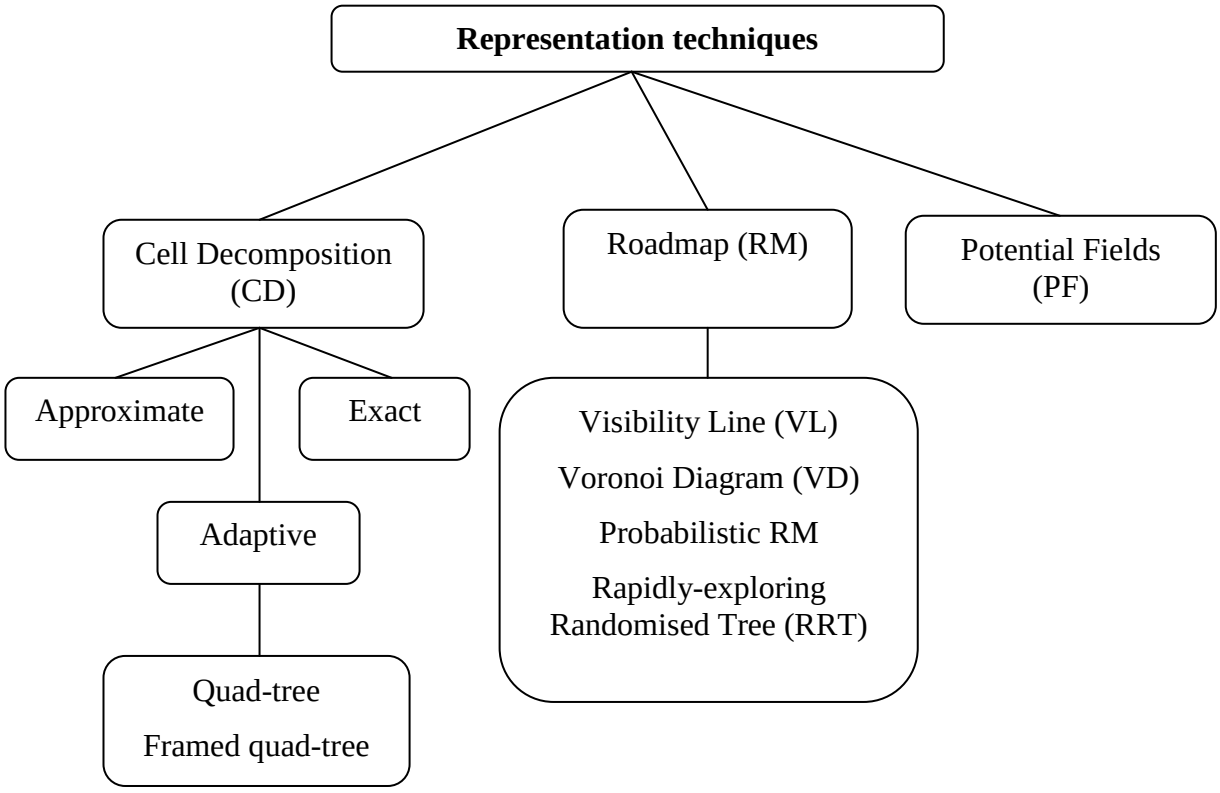


Figure 2.3: Path planning representation techniques categories

2.2.3 Visibility Line

A visibility graph or visibility line (VL) is one of the methods in representing a C-space. It was first proposed by Lozano-Perez and Wesley [9] for path planning in the environments with polyhedral obstacles. Ever since then, researchers [7, 74, 94, 112] have used this method, with some variations, for path planning. The VL of a 2D configuration space which consists of a set of polygonal obstacles O is defined as a network $G(V, E)$, constructed from sets of vertices/nodes V and edges E. The VL network is an undirected graph in which an edge $e \in E$ is a linear segment connecting a pair of mutually visible nodes, v_i, $v_j \in V$ where $i \neq j$. In addition, the edges of the obstacles are also edges of the VL network. Two nodes are mutually visible if the edge connecting both nodes not intersects any edge of E. V consists of all the corners of the obstacles including the starting point and the target point. A path resulted from the VL is the combination of several edges connecting the starting point p_{start} and target point p_{target}. An example of a VL application for path planning is shown in Fig. 2.4. The path is represented by the solid bold lines.

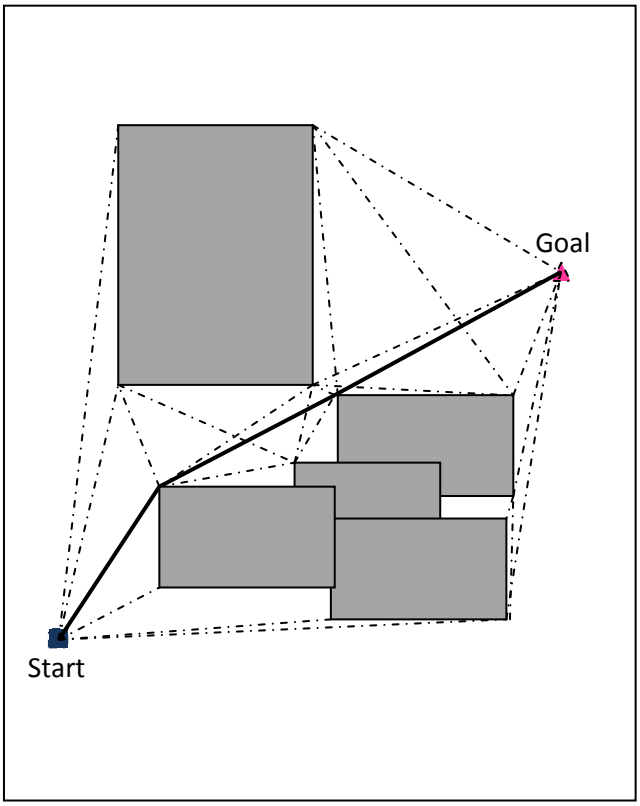


Figure 2.4: A path planned by VL method

2.2.3.1 Related Works Using Visibility Line

The VL method was pioneered by Nilsson [7] for the Shakey Robot project where the graph is created based on planar map called the grid model. The method has been developed since then through studies on the problem of path planning either for ground robots or UAVs through polygonal obstacles [8-10, 68, 56, 70, 71, 72, 43, 47, 72, 74, 94, 112]. Thompson [8] used VL to create a roadmap, and then applied a search algorithm to find an optimal path for a point robot. In 1979, Lozano-Perez and Wesley [9] proposed an algorithm based on VL to solve the problem of finding shortest path for a polyhedral object moving from start to goal points through polygonal obstacles considering its (the object's) dimension. Another work based on VL was undertaken by Tokuta [10] who presented a VGRAPH method that incorporates a starting point and a target point of a robot into the roadmap of a two-dimensional workspace. An algorithm called A VGRAPH Point Incorporation Algorithm (VPIA) was used to incorporate a point in free-space into a roadmap and divided the free space around an obstacle node into an ordered set of areas. A search algorithm was used to determine the containment that implied visibility of the point from the vertex.

Oomen *et al*. [68] used VL to find a solution of autonomous mobile robot path planning in an unexplored obstacles environment. In the proposed solution, the VL was constructed incrementally. A learning element was incorporated in order to construct the VL. Additionally, a sensor with limited range was used to learn information about the obstacles in the environment. However, the generated path was sub-optimal due to the unavailability of complete information about the environment. Like [68], Rao [56] proposed a general framework of robot navigation that could be applied to any situation involving mobile robots or manipulators where a suitable navigation course could be found using the so-called Restricted Visibility Graph (RVG) in an unknown environment. Two algorithms concerning local (*Lnav*) and global (*Gnav*) navigation were proposed. The framework of the proposed algorithms laid a foundation on which navigation systems for mobile robots can be built. Louchene *et.al* [70] presented a strategy for global path planning in a known environment for an Automated Guided Vehicle (AGV) using a VL representation. The proposed strategy consisted of two parts. The first partitions the free working space

according to obstacles models. The second calculates a set of points within the free working space based on the dimensions of the mobile robot.

Many researchers have concentrated on reducing the computational effort required to create the VL network as it is computationally expensive in obstacle-rich environments. Reduced computation time is useful for real-time application. Wooden & Egerstedt [72] derived a significantly reduced roadmap for unstructured polygonal environments suitable for real-time path planning application of outdoor robots. The method called Oriented Visibility Graph (OVG), attached an onboard stereo-based sensor to the robot to detect the obstacles and created the polygonised maps to support the use of the planner. In order to improve the performance over runs, the graphs were saved between runs and dynamic update rules were carried out. Also, the algorithm that was proposed Tokuta [10], as explained above, is suitable to be applied in real time path planning as the VPIA runs in parallel which reduces the computation time. Another real time path planning research project based on VL was done by Huang and Chung [74]. They proposed a method called Dynamic Visibility Graph (DVG), which was claimed to be fast for constructing a reduced roadmap through polygonal obstacles within an active region. DVG enormously decreases the computation time for reconstructing the map and hence is suitable for real time path-planning for single and multiple autonomous vehicles. However, it is difficult to define the area of active region. Other methods for reducing the complexity of VL were proposed by [75, 111]. Both methods were claimed to have low computation loads. Omar and Gu [112] proposed a path planning method, which is based on VL, called Base Line Oriented Visibility Line (BLOVL) to find paths in short time by reducing the numbers of obstacles during the paths calculation. BLOVL was proven through simulations to have paths that are identical to those of conventional VL most of the times.

In addition, since VL results in the shortest path, VL's application in not limited to path planning only, but also extends to Field Programmable Gate Arrays (FPGAs) design [50] and geographic routing [57].

This thesis is the extension of the work of [112] as the proposed method has been proven to be fast in producing optimal path in obstacle-rich environments.

As a conclusion, the aforementioned methods that are based on VL are summarised in Fig 2.5.

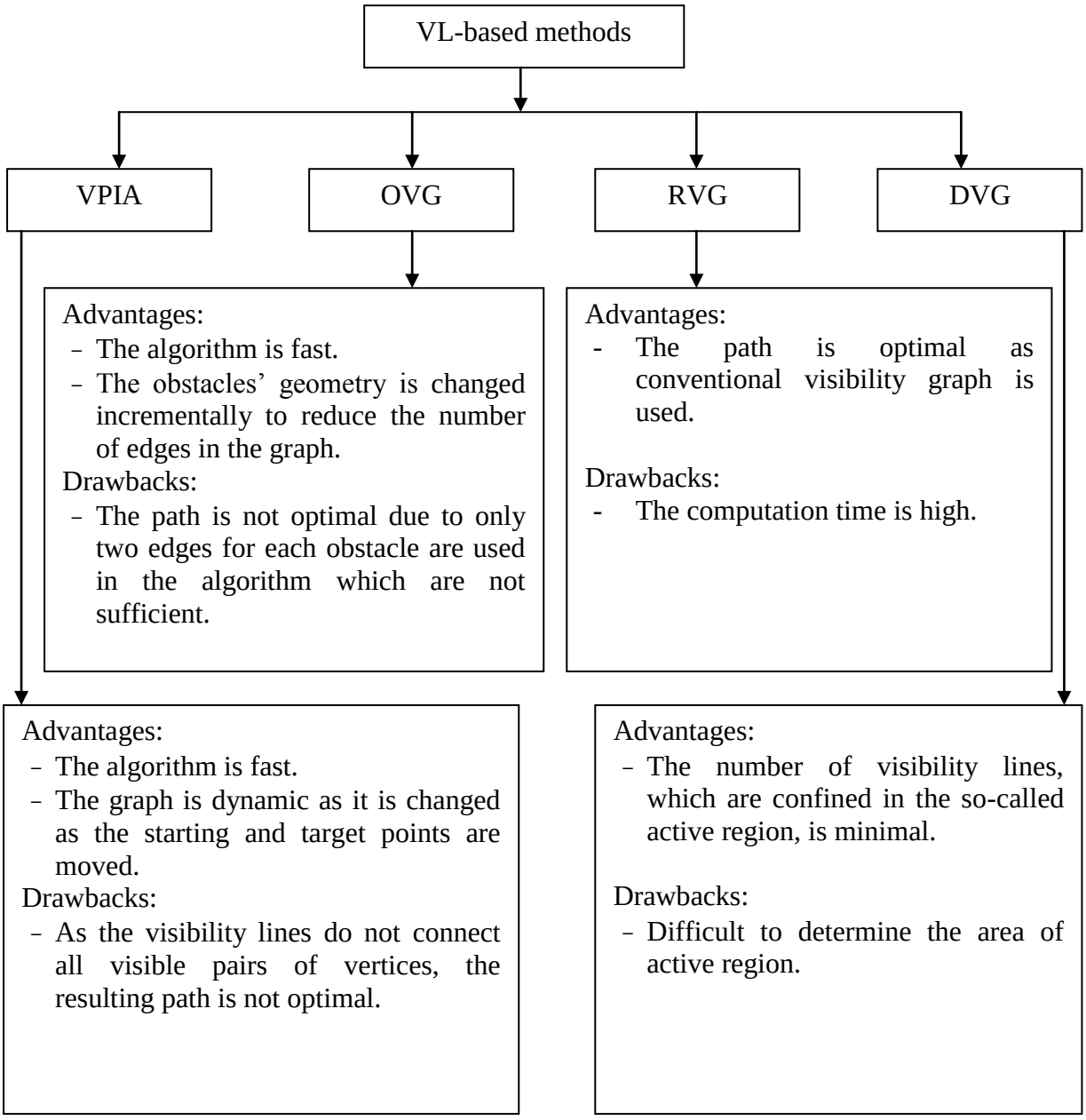


Figure 2.5: A summary of the path planning methods based on VG

2.3 Graph Search Algorithms

Graph search is the second step for path planning after an environment has been represented by a particular method. Graph search algorithms have received considerable attention in the past and are important in path planning. In general, graph search algorithms determine whether a path exists from p_{start} to p_{target} by evaluating certain nodes/states. If no path exists, they will report failure. Several major search algorithms are shown in Fig. 2.6 [12] and a number of them are briefly presented in this thesis.

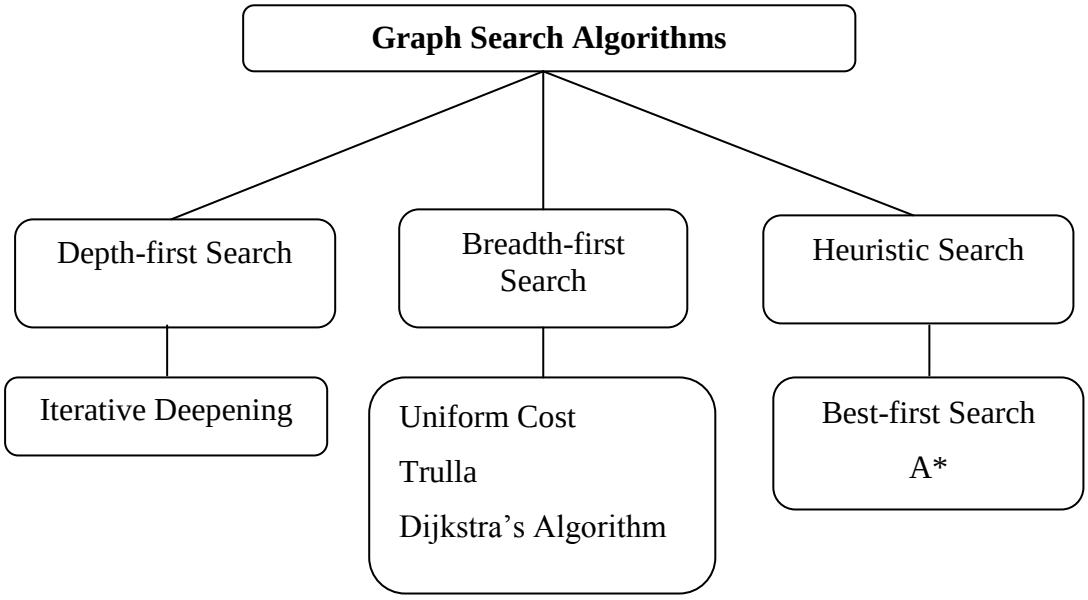


Figure 2.6: Graph search algorithms

2.3.1 Depth-first Search

In Depth-first search (DFS), the deepest node is expanded first as shown in Fig. 2.7. It moves toward the goal as quickly as possible, searching on a path until a dead end is found. As it searches one path through a branch prior to another search at the other path, DFS could miss large portions of the workspace [12,34]. DFS can be applied for finding a path among many possible paths.

However, DFS is an uninformed search, which means it does not use the cost function to decide which direction to go and how far the distance from the current node to the target point is.

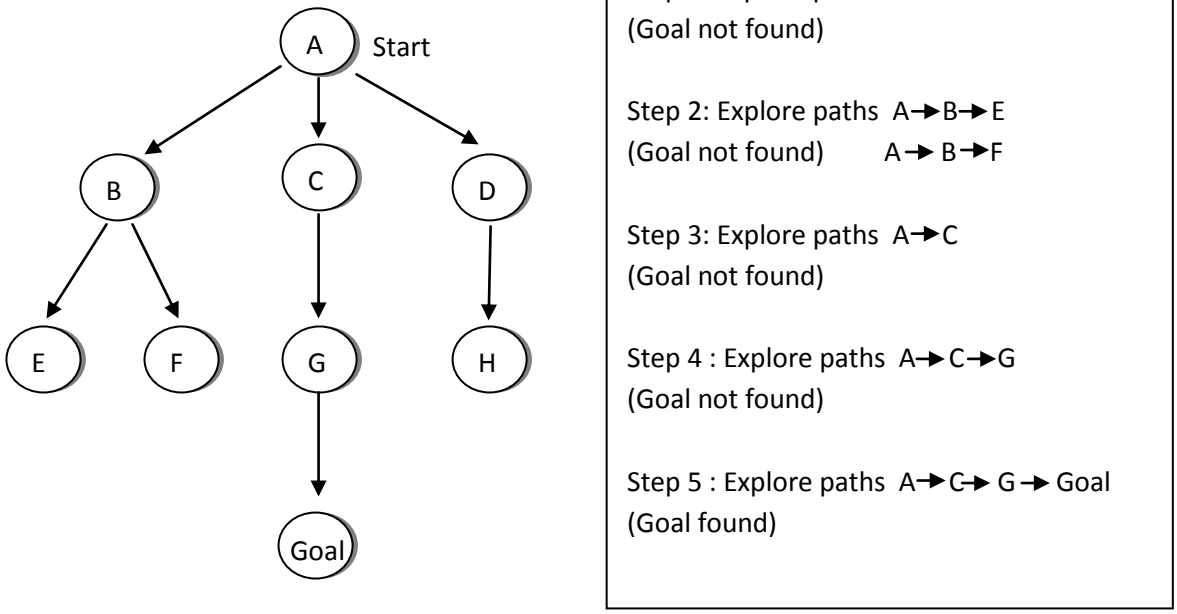


Figure 2.7: Depth-first search (adapted from [12])

2.3.2 Breadth-first Search

The Breadth-first algorithm (BFS) was introduced in 1957 by Moore [73]. In BFS algorithm the shallowest node is expanded first searching all the one-step down nodes of the path prior to the next step taking place as shown in Fig. 2.8 [12]. This makes BFS a systematic search algorithm. However, like DFS, BFS is an uninformed search. BFS finds the shortest path on its first run. It is suitable when there are a small number of solutions which use a relatively short number of steps [12].

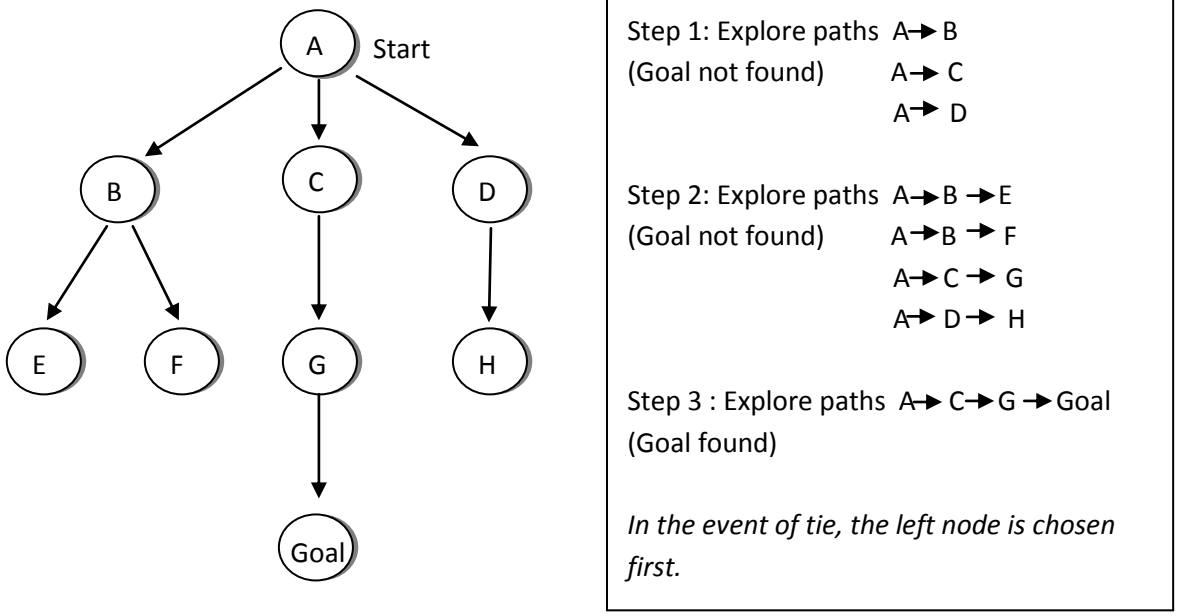


Figure 2.8: Breadth-First search (adapted from [12])

2.3.3 Dijkstra's Algorithm

Dijkstra's algorithm was invented by a Dutch computer scientist, Edsger Dijkstra in 1959 [97]. It is used to find the shortest path based on costs of traversal from p_{start} to all points in a graph. Dijkstra'a algorithm is complete if a solution exists. It measures the distance of node n which is denoted by $g(n)$ with respect to the starting node in the graph. The cost at the node is non-negative and stored in a priority queue. For example a node n that is stored in priority queue has the cost of

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

$f(n)$ is also called the backward cost or cost-to-come. The cost is calculated incrementally during the algorithm execution. As the cost is non-negative, the cost is monotonically increased. For example, if the next node to n is n', and the distance between them is $l(n, n')$, the cost-to come is updated to

$$f(n')=f(n)+ l(n, n')=g(n')$$

Because $l(n, n')$ is non-negative, $f(n')$ is thus greater than f(n).

The algorithm naturally begins at a p_{start} and extends outward within the graph, until all nodes are visited. As a result, Dijkstra's algorithm is a systematic search algorithm.

In order to establish the steps in Dijkstra'a algorithm, let $d(p)$ be the distance from a source node x to a node p; and let $l(p,q)$ be the cost between adjacent/neighbouring nodes p and q. The steps of Dijkstra's algorithm are then as follows:

- *Step1*: Set the priority queue, $PQ=\{x\}$. For each node p not in PQ, set $d(p) = l(x,p)$. For all nodes that are not adjacent to x, set their values to infinity.

- *Step 2*: At each subsequent step, find a node q that is not in PQ where $d(q)$ is minimum. Then add q in PQ and set the parent of q to p. Subsequently update $d(p)$ for all the remaining nodes which are not in PQ by finding its minimum cost using

  $$d(p) = \min\left[d(p), d(q) + l(p,q)\right]$$

  Step 2 is done recursively until node q is the target point.

In order to illustrate how Dijkstra's algorithm works, consider a scenario in Fig. 2.9(a) in which a path has to be found from source node *A* to goal node *E*. As Dijkstra's algorithm starts at node *A* hence the node is put in the priority queue *PQ* as shown in Fig. 2.9(b). The adjacent nodes to node *A* are nodes *B, C* and *D* which are shown in amber. It is found that node *B* has the least cost-to-come i.e. 5. Node *B* is then stored in *PQ*. Node *B*'s neighbours are nodes *C, D* and *E* as shown in Fig. 2.9(c). Of the three nodes, *D* has the least cost from node *A* i.e. 8 thus *D* is kept in *PQ* as shown in Fig. 2.9(d). Node *D* is then expanded to its adjacent nodes i.e. node *E*. At this point there are two remaining nodes that have not been visited yet i.e. *C* and *E*. It is found that of the two remaining nodes, node *C* has the least cost i.e. 9, from source node *A*. hence *C* is put in *PQ* as shown in Fig. 2.9(e). The last node *E* is then examined and placed in *PQ* with the parents of *D* and *A*. As *E* is the goal node, its parents are backtracked. The path via node *D* (also called waypoint) with the lowest cost i.e. 10 is then found and is shown in darker arrows as illustrated in Fig. 2.9(f). Table 2.1 records the priority queue *PQ* and costs of $d(p)$ at each iteration.

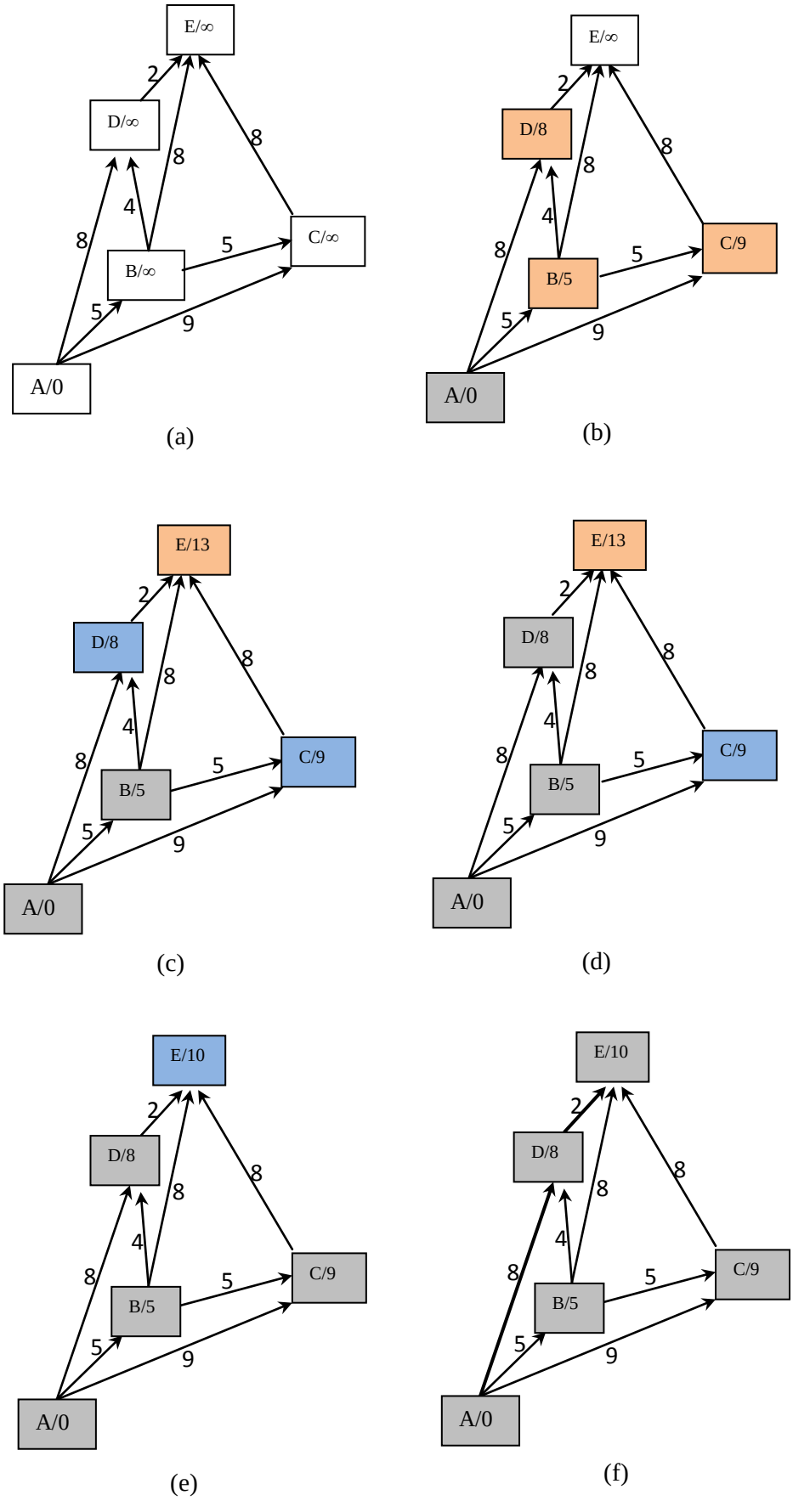


Figure 2.9: Dijkstra's algorithm illustration

Table 2.1: The recorded costs of Dijkstra's algorithm for Fig. 2.8.

Iteration	*PQ*	*d*(B)	*d*(C)	*d*(D)	*d*(E)	*Parents*
1	{A}	5	9	8	∞	-
2	{A,B}	5	9	8	13	{A}
3	{A,B,D}	5	9	8	13	{A}
4	{A,B,D,C}	5	9	8	13	{A}
5	{A,B,D,C,E}	5	9	8	13	{A,D}

2.3.4 Best-first Search

Best-first search falls under the class of heuristic search algorithm, which uses the distance from a current node with respect to the target point in order to find a path in a graph. Heuristic is used for making a guess for such a distance. The heuristic distance $f(n)$ from a node n to target point is defined by

$$f(n) = h(n)$$

The resulting path in a graph using Best-first search is determined by comparing a heuristic cost of the current node with the costs of all the other nodes. The node which has the least cost is then expanded to the neighbouring nodes until the target point is met.

There is no guarantee that Best-first search algorithm will find the shortest path because it by passes some branches in the search tree. Nevertheless, it performs much less searching than Dijkstra's algorithm.

Like Dijkstra's algorithm, Best-first search uses a priority queue that stores the list of nodes. The start node is normally the first node that is stored in the priority queue. As the node is then expanded, all the adjacent nodes that are directly connected to the node are then stored into the priority queue, arranged by their corresponding total heuristic cost. The least cost adjacent node is then expanded next and its neighbours that are not in the queue are added. The process is repeated until the target point is found. The illustration on how Best-first algorithm is used in finding a path in a graph is shown in Fig. 2.10. In the figure the starting point is labelled with *A* and target point is labelled with *E*. Fig. 2.10(b) shows that as Best-first search begins at node *A*, the node is stored in priority queue. The neighbouring nodes that directly linked to node *A* are nodes *B*, *C* and *D* which are shown in amber. It is clear that node *D* has the least

heuristic cost (2), while both nodes *B* and *C* have 8 as their heuristic costs. Node *D* is then expanded and linked to its adjacent node i.e. *E*, as shown in Fig. 2.10(c). As node *E* is the target point, its heuristic cost is 0. The path with the lowest heuristic cost is then found and shown in darker arrows as illustrated in Fig. 2.10(d). It is apparent that, as Best-first search uses heuristics, it visits lesser nodes than that of the Dijkstra's algorithm.

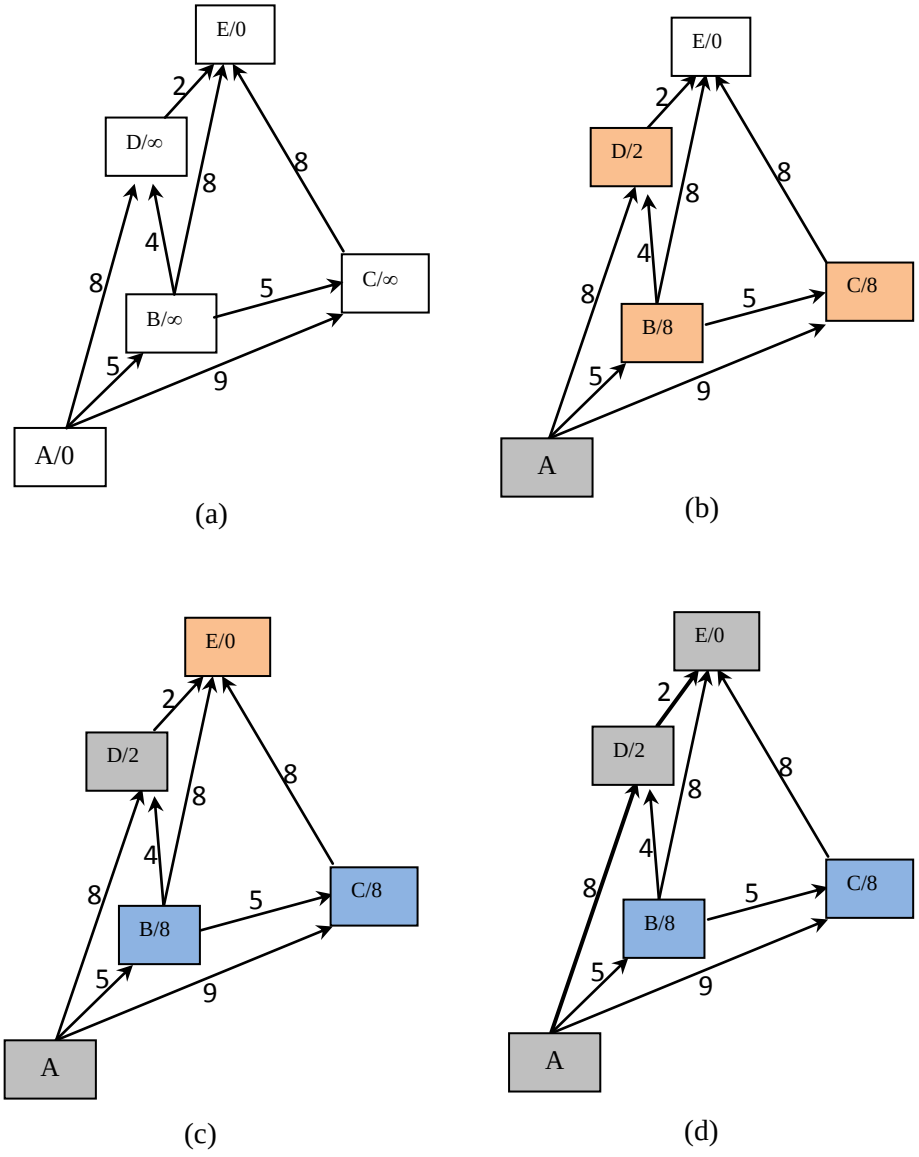


Figure 2.10: Illustration of Best-first search algorithm.

2.3.5 A*Algorithm

Another heuristic search method is A* (pronounced A star) that was pioneered by Hart [96] in 1968. It is a search algorithm that is used to find a solution to a path planning problem. Like Dijkstra's algorithm, the A* search algorithm is systematic and using a backward cost function $g(n)$ from a source node to a current node n. In addition, it uses a forward cost function, called heuristic $h(n)$, which is an estimate of the cost from the node (n) to the goal node. As a result, the total cost function at the current node n can be expressed as follows:

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

Normally the heuristic value is a straight line distance from the current node to the goal ignoring obstacles in between [12]. However, there is no way to estimate the true heuristic value in advance [34]. The heuristic function reduces the total number of states/nodes need to be explored by A*.

As both forward and backward costs are used, it therefore combines the Best-first search and Dijkstra's algorithm. If the backward ($g(n)$) cost is dominant, A* tends to be Dijkstra's algorithm and the result is the shortest path from the source node to the goal, but the search process takes longer. This situation is called admissible heuristic which means the estimated distance $h(n)$ between node n and the source node does not underestimate the true distance from the node to the goal. In the extreme case, A* becomes Dijkstra's Algorithm if the heuristic value $h(n)$ is zero. On the other hand if the forward cost or heuristic weighting is dominant, A* tends to be like the Best-first search, producing a shortest path is not guaranteed, although the path is produced faster. A* becomes -first search if the backward cost is zero.

Like Dijkstra's algorithm, A* has a priority queue that stores the list of nodes. The start node is typically the first node to be stored in the priority queue. The node is then expanded and all the adjacent nodes that are directly connected to the start node are then stored into the priority queue, sorted by their corresponding total cost. The adjacent node with the least cost is then expanded next and its neighbours that are not in the queue are added. The process is repeated until the target point is in the queue.

An example of how A* is used in path finding is depicted in a scenario as shown Fig. 2.11 in which the starting point is node *A* while the target point is node *E*. In Fig. 2.11 (b), A* begins at node *A* hence the node is put in priority queue. The adjacent nodes that directly connected to node *A* are nodes *B*, *C* and *D* which are in amber. From the figure, node *D* has backward cost *g*(*D*) of 8 and 2 forward cost *h*(*D*) which makes the total cost of *f*(*D*) is 10, while nodes *B* and *C* have *f*(*B*) and *f*(*C*) of 13 and 17, respectively. Node *D* is then expanded and its only neighbour is node *E* as shown in Fig. 2.11(c). The path with the lowest cost i.e. 10 is then found and shown in darker arrows as illustrated in Fig. 2.11(d).

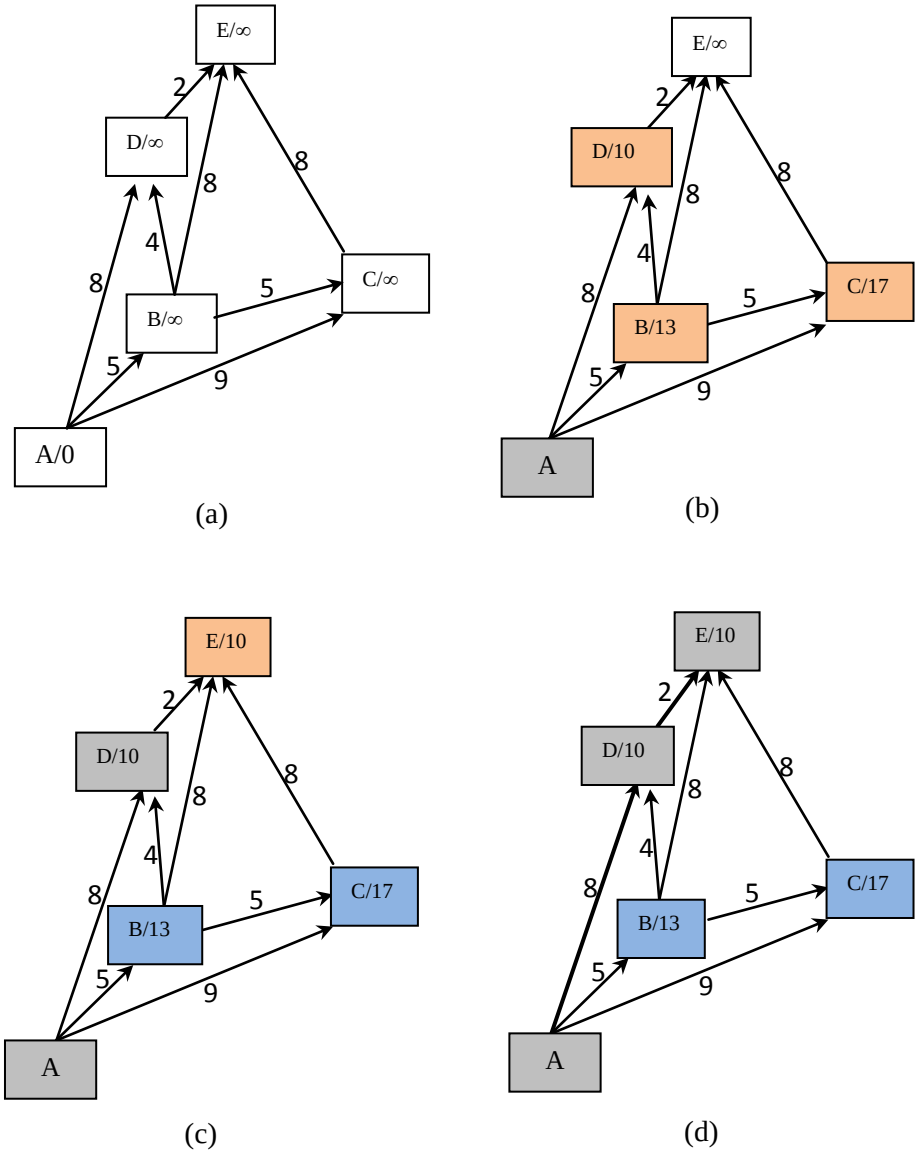


Figure 2.11: Illustration of A* algorithm

A*is complete if a solution exists, provided that the time and memory are unlimited. It will find the target point if it can be possibly found in the map or graph. In an optimal sense, A* is guaranteed to produce a path with the least cost from a starting point to a target point if the heuristic value is admissible, which means smaller than the actual value.

2.4 Path Planning Using Direct Optimisation Methods

As previously mentioned, there are methods that do not require the graph search algorithm to find paths such as Mixed Integer Linear Programming (MILP) and Evolutionary Algorithm (EA).

MILP is capable of producing optimal solutions by expressing linear constraints upon a mixture of continuous and integer variables. This involves the discrete decisions in the optimisation process, which gives some flexibility in the mission problems that are to be solved. MILP is also proven to be efficient in finding solution multiple vehicles path planning problem [105].

EA, on the other hand, finds a path by creating the initial generation of a population at the beginning. This is done by encoding a set of randomly selected feasible solutions. The fitness of each individual of the population is evaluated according to several factors and constraints. Then a set of selected individuals are appointed as parents for the next generation. The final step is to generate the offspring individuals by duplicating a parent with a mutation or combining two parents by crossover [120]. The above process is repeated until the last generation's individual with best fitness is decoded as the optimal solution.

2.5 Real-Time Path Planning

Practically, a UAV operates in an environment that may change over time. For example, an obstacle might pop-up in the environment while the UAV is traversing a planned path. These situation requires a path planning algorithm to quickly re-plan a safe path in real-time. Thus the ability to plan a path in real-time is an important factor that has to be considered before designing a path planning algorithm in dynamic environments.

In order to do so, such an algorithm must be able to adapt to any change in the environment. Thus, for path planning in real time, onboard sensors are very important as the sensors will gather and update all the obstacles' data, which needs to be supplied to the algorithm which is embedded into the UAV's onboard processors. The algorithm will re-plan a path if the sensors detect pop-up obstacles that lie on the planned path.

The issue of real-time path planning has been addressed by many researchers such as [21, 72, 74, 75, 52, 107, 112]. Some of this work have been discussed in sub-section 2.2.3.1. Omar and Gu [112] proposed two algorithms called *Core* and Base Line Oriented Visibility Line (BLOVL), for UAV path planning in real-time. The algorithms were claimed to generate a relatively fast collision-free path as small number of obstacles were considered during the path calculation. As such the proposed algorithms are suitable for real-time path planning. In [74] the so-called fast Dynamic Visibility Graph (DVG) method was proposed for constructing a reduced roadmap among convex polygonal obstacles. DVG was claimed to decrease the computation time of reconstructing the roadmap, and as such it is suitable for real time path-planning for single or multiple autonomous vehicles.

Another example of work in real-time path planning was undertaken by Jason *et.al* [75]. They use the Essential Visibility Graph (EVG) global motion planner so that a realistic, static environment could be modelled in two dimensions. EVG offers a significant reduction in data storage requirements and complexity thus it is suitable for real time path planning.

2.6 Path Planning in 3D Environment

Most of the path planning methods are mainly developed for finding paths for ground robots and hence the paths are in 2D where the altitude is assumed to be constant. However for UAVs, which flies in a 3D environment, it is crucial to have a 3D path planning algorithm that is capable of generating a 3D path for the UAV. This is because such a path with the altitude is taken into account, might be shorter than the 2D path that is planned in 3D environment.

Path planning in a 3D environment, to produce 3D paths, has been studied for many years and includes those that are based on approaches such as VL [71, 94, 95, 112], Potential Fields [22, 93] and Voronoi diagrams [20]. Kitamura *et al.* [22] proposed a method for finding a collision-free path and orientation for a vehicle in a dynamic 3D environment using an octree as a means to represent every object in the environment. In this method, the path of a vehicle from its starting position to the given goal with arbitrary motion in a 3D environment is searched using the potential field method.

Mojtaba and Ghodsi [94] tried to find a new way to define a geometric structure in 3D space based on VL. This method easily defines a new structure called 3D VG to extend the 2D graph to 3D scenes. It runs in $O(n^3 \log(n))$ and this method could be computed in an acceptable time. Chung and Saridis [95] proposed the extended Vgraph algorithm (EVA) to reduce the computation time by using the recursive compensation algorithm (RCA). RCA is used to find the collision-free shortest path in a 3D environment without increasing the Vgraph complexity. It was proven that the EVA can save memory space and the path planning time as well. Omar and Gu [112] introduced a set of 3D path planning algorithms based on VL method. The concept of a rotational plane, on which the VL is created, has also been proposed. The 3D path is obtained after the plane has been rotated at specified angles.

K. Jiang *et al.* [71] proposed a method for the shortest 3D path planning in the presence of polygonal obstacles based on the visibility graph approach. In order to identify the edge sequence which the shortest path may pass across in the three dimensional VL, a collineation was introduced. Then the sub-optimal path was calculated using the so-called principle of minimum potential energy. The process of finding the path was done recursively. However, the drawback of the proposed method was that the processing time is polynomially related to the number of vertices or nodes.

Broz [93] proposed a hybrid and real-time path planning technique that can be used in 3D applications for a set of agents. It works in known, partially known or unknown environments based on Potential Field method. Two separate maps of the same size are used to represent the environment. The obstacles map represents danger weights while threats map represents potential fields of all located and observed threats in the space. The algorithm keeps a mesh that was "widespread over each map" and all

available paths were defined. The changes in both maps and the behaviour of all agents were continuously adapted. The whole algorithm was based on real-time development of the adaptive mesh. Sud *et.al* [20] presented a novel approach for efficient path planning and navigation of multiple virtual agents in complex dynamic using Multi-agent Navigation Graph (MaNG). The MaNG was used to perform route planning and proximity computations for each agent in real time. The algorithm is used for real-time multi-agent planning in pursuit evasion, terrain exploration, and crowd simulation scenarios consisting of hundreds of moving agents, each with an individual goal.

2.7 Conclusion

In this chapter, an overview of path planning was made, including a discussion regarding the workspace $\mathcal{W}$ representation and graph search algorithms. In addition, a brief discussion on real-time and path planning in three-dimensional (3D) has been put forward.

Most path planning methods have two steps in order to find collision-free paths. The first step involves the representation of the $\mathcal{W}$ based on the configuration space (C-Space) of an environment. The C-space is the space of all possible specifications of a vehicle $\mathcal{A}$ and an obstacle region O in $\mathcal{W}$. The second step deals with the calculation of a collision-free path using a graph search algorithm.

Nevertheless there are several path planning methods that don't require the aforementioned steps in order to find solutions such as Mixed Integer Linear Programming and Evolutionary Algorithm.

Popular C-space representation techniques are cell decomposition, roadmaps and potential fields. Under roadmaps, there are several methods, which are widely used for path planning namely, Visibility Graphs (or Visibility Lines), Voronoi Diagrams and Rapidly-exploring Random Tree (RRT). On the other hand, there are a number of graph search algorithms available in the literature such as Breadth-first search, Depth-first search, Best-first search, Dijkstra's algorithm and A* (pronounced A-star). A* will produce an optimal solution if the heuristic value is admissible i.e. less than the

actual value. Dijkstra's algorithm on the other hand guarantees that the resulting path is the shortest if one exists.

One important factor of any path planning algorithm is the ability to find a path in real time. Planning in real time is useful if there are changes in the environment in which the path planning will take place. The ability to re-plan in a short time will guarantee that the UAV will successfully avoid any pop-up obstacles and accomplish a particular mission.

As UAVs fly in 3D environments, traversing a 2D path with constant altitude is inappropriate as this may cause the UAVs to fly along a longer distance path. Hence planning 3D paths is necessary as the path may have a shorter distance hence saving the UAV's fuel/energy, increasing its endurance, prolonging its life cycle and minimising its exposure to risk.

Chapter 3

2D Path Planning Using Visibility Line-Based Methods

3.1 Introduction

One of the criteria of path planning is to produce an optimal (shortest) path that links a starting point p_{start} and a target point p_{target}. An Unmanned Aerial Vehicle (UAV) that is being deployed to accomplish a mission traversing along such path will gain the following advantages:

i. minimal traversal duration
ii. low fuel/energy consumption.
iii. longer life cycle.
iv. minimal exposure to risk.

Planning in real-time in response to any changes in the environment is also a criterion of path planning in addition to producing a path if one exists i.e. complete. These three criteria and the trade–offs between them have to be considered before a path planning algorithm is designed.

One of the path planning approaches that is capable of producing shortest path is Visibility Line (VL) method if it is coupled with Dijkstra's algorithm. Moreover VL is also complete in the sense that a path will be found if the path is available. For these reasons, VL method and Dijkstra's algorithm have been chosen for path planning of

UAV in this thesis. A* search algorithm is not considered because, if it is combined with VL method, the path it produces may not be optimal. The reason being, it is difficult to calculate the heuristic of A* as there is no way to measure the cost of a straight lines that connect nodes to the target point p_{target} in an environment where the lines pass through obstacles. In addition, if the heuristic cost is not admissible i.e. higher than the real cost, the resultant path might not be optimal in terms of the path length.

Although VL satisfies two of the path planning criteria i.e. it produces the shortest path and is complete, its computation time increases significantly with the increase of numbers of obstacles in the environment as will be demonstrated later. This implies that the more obstacles which appear in the C-space, the much longer time it needs to find a path. As a result, VL in its original form is not suitable to be applied in real time path planning applications in obstacle-rich or dynamic environments.

In order to address the above-mentioned issue, two algorithms based on VL have been proposed in this thesis. The proposed algorithms are capable of finding paths in real-time while retaining the advantages of the VL, although sometimes the paths are not the shortest. The first algorithm is called *Core* while the second one is termed Base Line Oriented Visibility Line (BLOVL). The algorithm is called *Core* because it is the fundamental algorithm and repeatedly called to find a path connecting a (current) starting and target points.

Core is used to find an initial path that is optimal using the minimal number of obstacles but the path might not collision-free. BLOVL which contains *Core* on the other hand will find a complete, safe path from p_{start} to p_{target} based on local obstacles in iterative manner. As *Core* is embedded in BLOVL, the proposed algorithm is then called BLOVL. Since BLOVL uses sets of local obstacles, they are relatively faster than the original VL as will be shown later. However the final resultant path might be near-optimal in rare cases.

This chapter explains the idea of the proposed algorithms, *Core* and BLOVL, and an overview of VL is presented in Section 3.2. Subsequently, an example of path planning using VL is demonstrated followed by an in-depth discussion on the *Core* and BLOVL algorithms. Examples of path planning using both proposed algorithms

are demonstrated next. Performance comparisons with respect to the computation time and paths lengths between VL and BLOVL from several randomly generated scenarios is given prior to the conclusion section.

3.2 Visibility Line (VL)

3.2.1 Definitions and Algorithm

The VL of a two-dimensional C-space is defined as a network/roadmap $G(V,E)$, which is constructed from sets of vertices/nodes (V) and edges (E). Each edge $e \in E$ is a linear segment connecting a pair of "mutually-visible" vertices, $v_i, v_j \in V$ where $i \neq j$. V consists of all the corners of the obstacles including p_{start} and p_{target}. Fig. 3.1 shows the algorithm to construct the VL set:

Input: p_{start}, p_{target}, polygonal obstacles.
Output: Visibility Lines, VL set.

```
1:  for every pair of nodes, v_i, v_j where i≠ j
2:      for every obstacle e
3:          if segment (v_i, v_j) intersect e
4:              go to (1)
5:          end if
6:      end for
7:      Insert edge(v_i, v_j) into VL set
8:  end for
```

Figure 3.1: The algorithm to construct VL set

3.2.2 Path Planning Using VL

In order to demonstrate how VL works, a scenario consisting of 15 rectangular obstacles where their sizes and positions are randomly generated together with p_{start} and p_{target} is considered as per depicted in Fig. 3.2. Note that the blue triangle represents the starting point and the magenta square denotes the target point. First a

VL network is created according to the definition and algorithm as mentioned in the previous sub-section. The resultant network is illustrated in Fig. 3.3. By utilising Dijkstra's algorithm then, a shortest path is found and it is shown in Fig. 3.4. It takes 0.50 seconds to generate the path on a computer with 2.4GHz processor and 2GB RAM.

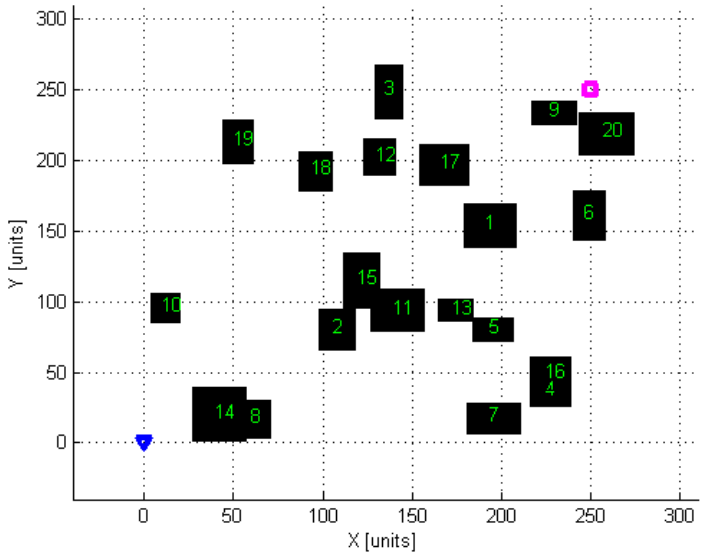


Figure 3.2: A randomly generated scenario for path planning

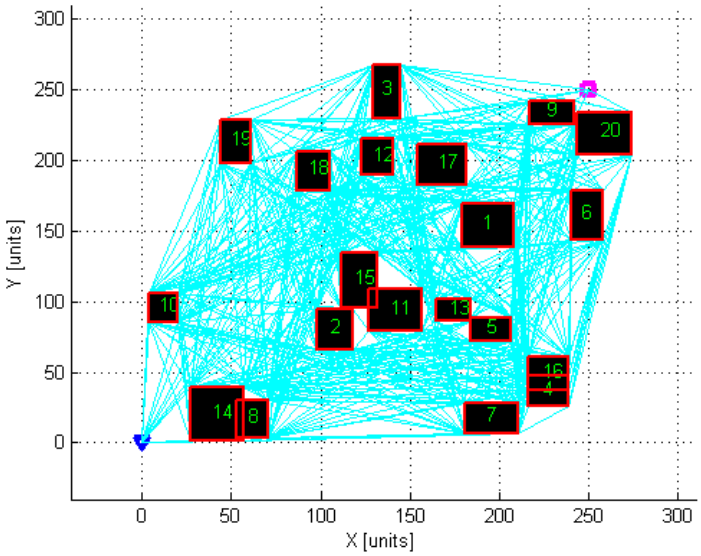


Figure 3.3: The network of visibility line

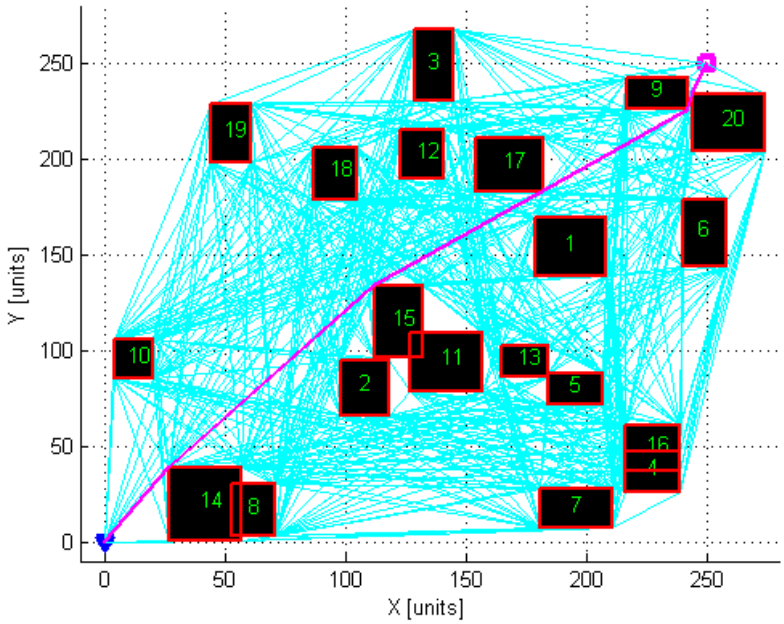


Figure 3.4: The path planned using Visibility line and Dijkstra's algorithm

3.2.3 Advantages and Drawbacks of VL

One of the benefits of VL is the computed path, which consists of a set of waypoints, has the shortest length, if it is coupled with Dijkstra's algorithm. To have a clearer idea on what waypoints are, consider its definition below:

Definition 1. *Waypoints, W is defined as a sequence of points* w_i *{i=0,…,n} of least number, starting from* p_{start} *i.e.* w_0 *to* p_{target} *i.e.* w_n*. A piece-wise linear path is formed if linear segments connect* w_i *to* w_{i+1}*.*

Note that a path is the shortest if it waypoints is a set of nodes of obstacles found using a graph search algorithm. As VL waypoints (not including the starting and target points) are always at certain nodes of obstacles, it is capable of producing a shortest path in terms of Euclidean distance. Consider the following lemma:

Lemma 1. *A necessary condition for a path to have minimum Euclidean distance from* p_{start} *to* p_{target} *in a* $\mathcal{C}$*-space is that all of its waypoints W are the nodes of obstacles O.*

Proof. Suppose that a set of waypoints that consists of a series of points which are not the nodes of the obstacles in C-space. Let B be the first such point in the series. A and C are the points immediately before and after B, respectively. B will not be on the straight line AC, because otherwise B should not be a waypoint. Without loss of generality, consider a path $AB + BC$ formed by points (A, B, C) as shown in Fig. 3.5. If there is no obstacle between A and C, then C should be the next waypoint after A. Also consider points F and G in the series. The following arguments can be observed from Fig. 3.5:

$$AB+BC = AB+BF+FG+GC > (AD+DF)+FG+GC = AD+(DF+FG)+GC > AD+(DE+EG)+GC = AD+DE+(EG+GC) > AD+DE+EC$$

The above arguments show that the path $AD+DE+EC$ formed by points (A, D, E, C), where A, D and E are the nodes of the obstacles, is shorter than the paths which contain points that are in the series. ■

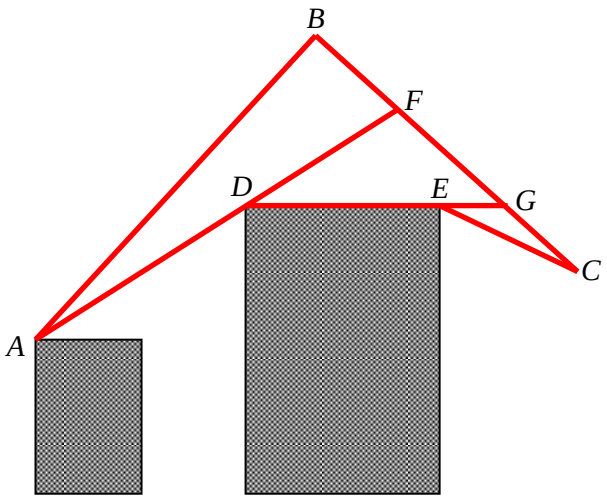


Figure 3.5: A scenario with 2 rectangular obstacles

Besides producing shortest paths, VL is also complete, which means that it always yield a path if one exists. This property is important as it will ensure that the UAV will accomplish a mission in a scenario where generating a path is possible.

One the other hand, VL has a major disadvantage; it is computationally expensive in an obstacle-rich C-space. For VL, the segments s that connect pairs of mutually-visible nodes has the maximum number of

$$s = \sum_{i=1}^{4n+1} i$$

where n is the number of obstacles, each with 4 nodes. For instance, if there are two disjoint rectangular obstacles in the *C*-space, the maximum number of segments is 45. Consequently, with 200 obstacles in the *C*-space, 321201 segments are needed to build the VL. In order to have a better insight into the relationship between the numbers of obstacles and segments, see Fig. 3.6 in which the obstacles' numbers are increased from 1 to 200. The figure shows that the number of segments of VL network increases significantly with respect to the number of rectangular obstacles. As the number of segments is directly related to the amount of computation time, thus the computation time will also be high in obstacle-rich environments. This prevents VL to be used for real time path planning in such environments.

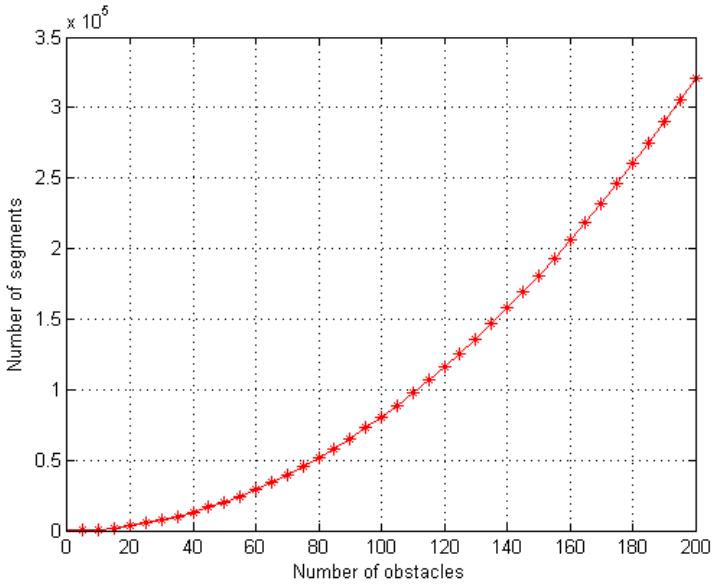


Figure 3.6: The numbers of obstacles in *C*-space increase the number of segments in VL

3.3 *Core* Algorithm

One major disadvantage of VL method is its computation time substantially increases as the number of obstacles grows. In order to address this problem, an algorithm has been designed named *Core*. *Core* can generate a path relatively quickly and is suitable for real time path planning applications in dynamic and obstacle rich environments.

This is due to the fact that a narrow region towards the p_{target} is considered in the algorithm, hence a smaller set of obstacles and vertices are used during the path calculation.

3.3.1 The Idea of *Core*

Core in a nutshell is used to construct a partial VL network from a specific region of the *C*-space. It then finds a path from p_{start} to p_{target} using Dijkstra's algorithm. *Core*'s algorithm is shown in **Algorithm 1** and its process is illustrated in Fig. 3.7. The partial VL network is constructed based on the obstacles that are determined by the so-called base line (BL), which is simply a line that connects p_{start} and p_{target}. BL allows *Core* to use a smaller set of obstacles, O_{Core} rather than the entire obstacles (O) as used by VL. This makes *Core* insensitive to the number of obstacles in the environment.

Algorithm 1: *Core*

1: Create a base line (BL) from starting point, u_{start} to target point, u_{target}.
2: Construct a set of nodes, N_S from each node of each obstacle that lies on the BL and their extensions including u_{start} and u_{target}.
3: Create a cost matrix, C_M from N_S.
4: Find local path, $U(u_0,\ldots,u_m)$ from C_M using Dijkstra's algorithm where $u_0 = u_{start}$ and $u_m = u_{target}$.

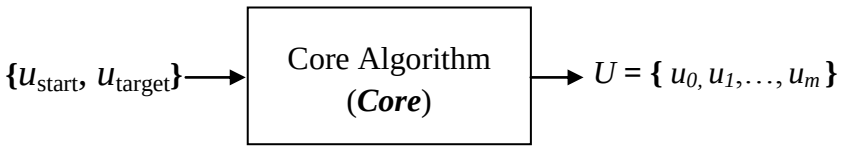


Figure 3.7: The process of *Core*

Referring to **Algorithm 1**, step 1 of *Core* is to create a BL from a starting point u_{start} to a target point u_{target}. The purpose of BL is to determine the obstacles that will be utilised in the path calculation. In order to determine the O_{Core}, BL first identifies the obstacles that lie along it. Then the obstacles, O_{Ext} that overlaps with each obstacle in O_{BL} are recognised. As a result, the obstacles that will be used by *Core* in the path calculation is $O_{Core} = O_{BL} \cup O_{Ext}$.

In the next step of *Core*, a set of nodes N_S is constructed based on O_{Core}, u_{start} and u_{target}. In this work, the nodes of N_S are arranged as shown in Table 3.1[2].

Table 3.1: Nodes arrangement

Node number	Nodes	Obstacles Number
1	Starting point, p_{start}	Nil
2	Target point, p_{target}	Nil
3	Lower left side	1
4	Upper right side	1
5	Upper left side	1
6	Lower right side	1
7	Lower left side	2
8	Upper right side	2
9	Upper left side	2
10	Lower right side	2

Subsequently, in step 3 of *Core*, a cost matrix C_M is determined from N_S. C_M is a structured database that contains the indexes and Euclidean lengths of inter-visible nodes pairs and the obstacles numbers, which the nodes belong to. The length will be set to infinity if a pair of nodes in N_S is not mutually-visible. Finally *Core* finds the path using Dijkstra's algorithm from the constructed network.

3.3.2 Path Planning Using *Core*

In order to demonstrate how *Core* algorithm works, consider the previous scenario as shown in Fig. 3.2. By step 1 of *Core*, a set of obstacles O_{BL} that lie along the BL is first identified as shown in Fig. 3.8. It is clear from the figure that the obstacles with numbers 14, 15 and 9 are the members of O_{BL}. Then a set of obstacles, O_{Ext} that overlaps with each obstacle in O_{BL} is recognised. This means O_{Ext} consists of obstacles {8,11}. The O_{Core} that have been identified by BL then consists of obstacles {14, 15, 9,

[2] In the table it is assumed that there are two disjoint rectangle obstacles i.e. o_1 and o_2 that lie on BL.

8, 11} as shown in Fig. 3.8. Notice that in the figure, BL is represented by the dashed red line and O_{Core} are outlined in red.

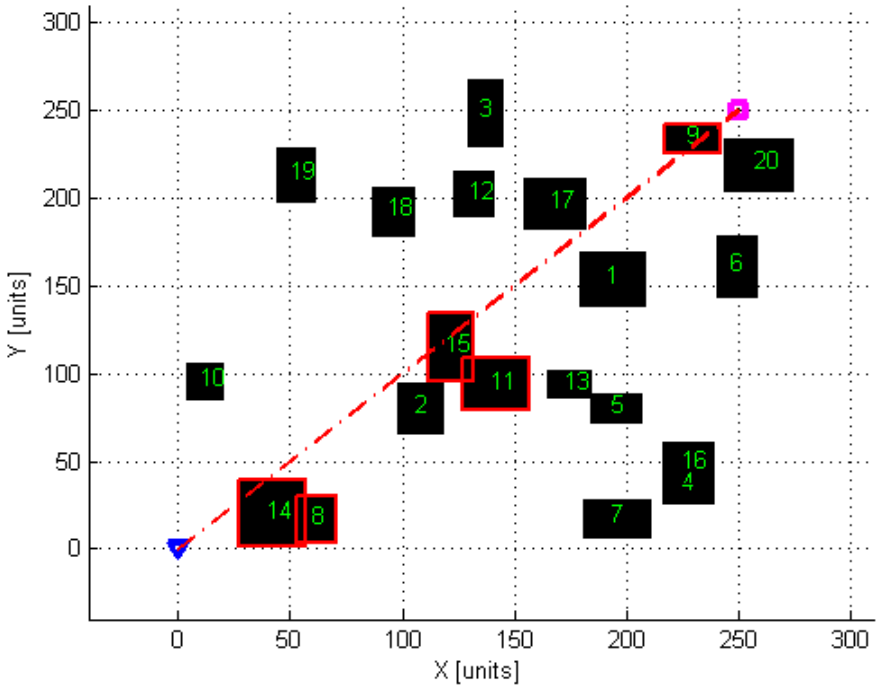


Figure 3.8: The base line (dashed red) that is used to identify the obstacles for path planning

Then N_S is identified and is shown in Table 3.2. The node number in N_S is arranged in the brackets of the first column of the table. Then *Core* generates C_M from N_S. The generated C_M is shown in Table 3.3. By removing the infinity costs and since VL is non-directed graph, C_M as in Table 3.3 can be simplified into the arrangement as shown in Table 3.4. The pairs of nodes that are not infinity in Table 3.4 will then be used to construct a VL network. The resultant VL network using *Core* is illustrated in Fig. 3.9. Finally *Core* finds a path using Dijkstra's algorithm based on C_M and it is depicted in Fig. 3.10. The waypoints of the path are shown in Table 3.5. It is noticeable that the waypoints w_1, w_2 and w_3 correspond to nodes 17, 21 and 10, respectively.

Table 3.2: The list of nodes

N_S	Nodes	Obstacles Number
1	Starting point, p_{start}	Nil
2	Target point, p_{target}	Nil
31(3)	Lower left side	8
32(4)	Upper right side	8
33(5)	Upper left side	8
34(6)	Lower right side	8
35(7)	Lower left side	9
36(8)	Upper right side	9
37(9)	Upper left side	9
38(10)	Lower right side	9
43(11)	Lower left side	11
44(12)	Upper right side	11
45(13)	Upper left side	11
46(14)	Lower right side	11
55(15)	Lower left side	14
56(16)	Upper right side	14
57(17)	Upper left side	14
58(18)	Lower right side	14
59(19)	Lower left side	15
60(20)	Upper right side	15
61(21)	Upper left side	15
62(22)	Lower right side	16

Note that *Core* finds the path in 0.016 seconds on a computer equipped with 2.4GHz processor, 2GB RAM. The computation time is faster compared to that of VL by over 30 times. However, as can be seen from Fig. 3.10, the path that has been planned by *Core* is not collision-free as some segments of the path intersecting the edges of obstacles 17 and 20. These obstacles have not been included during the path calculation because they are not on the BL or overlapped with the obstacles on BL. In

the next section, this problem will be addressed by introducing another algorithm called Base Line Oriented Visibility Line (BLOVL).

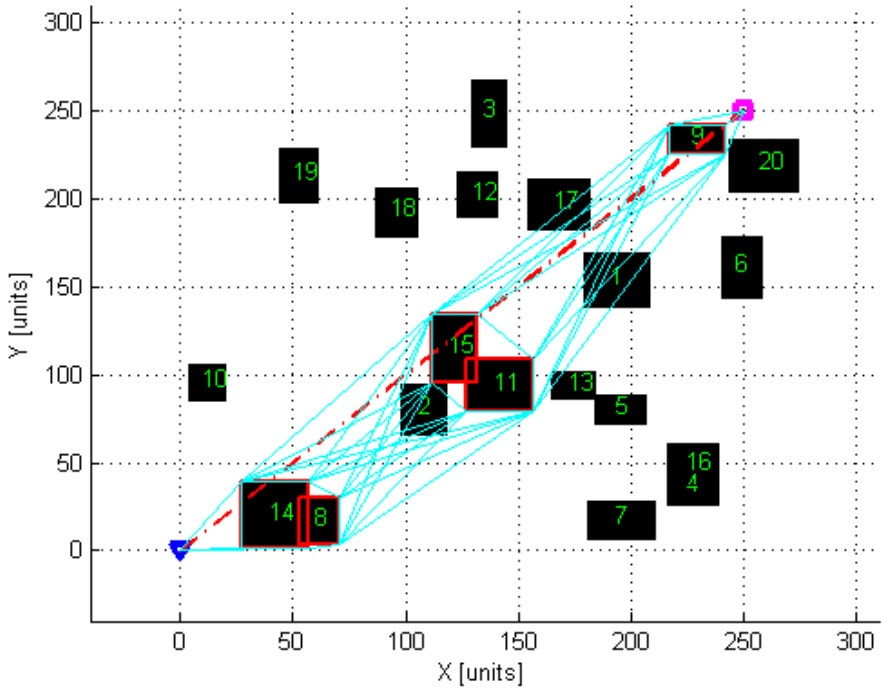


Figure 3.9: The network created by *Core*

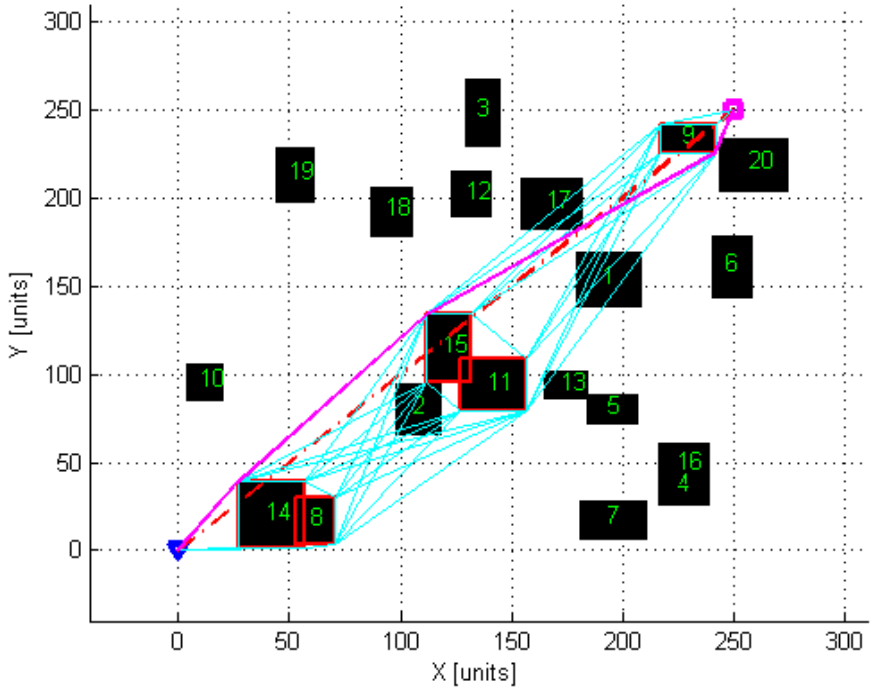


Figure 3.10: The planned path by *Core*

Table 3.3: The matrix of cost, C_M

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	27.0	∞	47.4	57.0	∞	∞	∞	∞
2	∞		∞	∞	∞	∞	∞	11.3	34.0	25.3	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	∞		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞		∞	27.0	∞	∞	∞	∞	74.4	∞	∞	99.0	∞	16.6	∞	∞	77.7	∞	111.8	∞
5	∞	∞	∞	∞		∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
6	∞	∞	∞	27.0	∞		∞	∞	∞	∞	94.4	∞	∞	114.8	∞	∞	∞	14.1	101.6	∞	137.3	∞
7	∞	∞	∞	∞	∞	∞		∞	16.0	25.0	∞	131.5	∞	158.8	∞	∞	∞	∞	∞	125.3	139.6	∞
8	∞	11.3	∞	∞	∞	∞	∞		25.0	16.0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
9	∞	34.0	∞	∞	∞	∞	16.0	25.0		∞	∞	145.9	∞	173.7	∞	∞	∞	∞	∞	137.4	150.6	∞
10	∞	25.3	∞	∞	∞	∞	25.0	16.0	∞		∞	144.6	∞	169.8	∞	∞	∞	∞	∞	143.4	159.3	∞
11	∞	∞	∞	74.4	∞	94.4	∞	∞	∞	∞		∞	∞	30.0	∞	80.6	107.7	∞	22.7	∞	∞	∞
12	∞	∞	∞	∞	∞	∞	131.5	∞	145.9	144.6	∞		∞	30.0	∞	∞	∞	∞	∞	35.4	∞	∞
13	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞		∞	∞	∞	∞	∞	∞	∞	∞	∞
14	∞	∞	∞	99.0	∞	114.8	158.8	∞	173.7	169.8	30.0	30.0	∞		∞	107.7	136.0	∞	∞	∞	∞	∞
15	27.0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞		∞	38.0	30.0	∞	∞	∞	∞
16	∞	∞	∞	16.6	∞	∞	∞	∞	∞	∞	80.6	∞	∞	107.7	∞		30.0	∞	79.2	∞	109.8	∞
17	47.4	∞	∞	∞	∞	∞	∞	∞	∞	∞	107.7	∞	∞	136.0	38.0	30.0		∞	102.3	∞	127.5	∞
18	57.0	∞	∞	∞	∞	14.1	∞	∞	∞	∞	∞	∞	∞	∞	30.0	∞	∞		∞	∞	∞	∞
19	∞	∞	∞	77.7	∞	101.6	∞	∞	∞	∞	22.7	∞	∞	∞	∞	79.2	102.3	∞		∞	38.0	∞
20	∞	∞	∞	∞	∞	∞	125.3	∞	137.4	143.4	∞	35.4	∞	∞	∞	∞	∞	∞	∞		20.0	∞
21	∞	∞	∞	111.8	∞	137.3	139.6	∞	150.6	159.3	∞	∞	∞	∞	∞	109.8	127.5	∞	38.0	20.0		∞
22	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	

Table 3.4: The simplified C_M

Nodes pair		Cost
1	15	27.0185
1	17	47.4342
1	18	57.0088
2	8	11.3137
2	9	33.9559
2	10	25.2982
4	6	27
4	11	74.411
4	14	98.9798
4	16	16.6433
4	19	77.6981
4	21	111.79
6	11	94.4034
6	14	114.7693
6	18	14.1421
6	19	101.6366
6	21	137.2662
7	9	16
7	10	25
7	12	131.4876
7	14	158.7734
7	20	125.2557
7	21	139.603
8	9	25
8	10	16

Nodes pair		Cost
9	12	145.9075
9	14	173.6923
9	20	137.4373
9	21	150.6287
10	12	144.6167
10	14	169.8058
10	20	143.4015
10	21	159.2608
11	14	30
11	16	80.6226
11	17	107.7033
11	19	22.6716
12	14	30
12	20	35.3553
14	16	107.7033
14	17	136.0147
15	17	38
15	18	30
16	17	30
16	19	79.2086
16	21	109.7725
17	19	102.3426
17	21	127.4755
19	21	38
20	21	20

Table 3.5: The waypoints generated by *Core*

Waypoints (*U*)	***x***	***y***
u_0	0	0
u_1	27	39
u_2	112	134
u_3	242	226
u_4	250	250

3.4 BLOVL Algorithm

The previous section has demonstrated that the resultant path by *Core* is not collision-free as several segments of the path intersect with a number of obstacles' edges in the *C*-space. In order to overcome such a problem, BLOVL, which steps are stated in **Algorithm 2** as shown in Fig. 3.11, has been designed. Fig. 3.12 shows the process of BLOVL in flow chart. *Core* is part of BLOVL which will be called repeatedly to find a local path from one waypoint to p_{target} or next waypoint until the global target point is reached. BLOVL is complete as it will keep finding a path sequentially by checking a set of obstacles at each sequence until all obstacles have been checked. However BLOVL will stop finding a path once one is found.

3.4.1 The Idea of BLOVL

The general idea of BLOVL is to plan a local path using *Core* and further repeatedly calculate a collision-free path between two consecutive waypoints, considering the obstacles in-between of those two waypoints. These are done until the target point is reached. The number of repetitions depends on the complexity of the environments. Less number of obstacles requires *Core* to be called in less time and vice-versa.

	Algorithm 2: BLOVL(p_{start}, p_{target})
1	set $j = 0$ and $w_j = p_{start}$
2	**while** $w_j \neq p_{target}$ **do**
3	set $u_{start} = w_j$ and $u_{target} = p_{target}$
4	call $[m,U]$=*Core*(u_{start}, u_{target})
5	**If** $m = 1$ then
6	set $w_{j+1}=u_1$
7	**else**
8	set $u_{start} = u_0$; $u_{target} = u_1$
9	goto line 4
10	**end if**
10	UAV flies from w_j to w_{j+1}
12	set $j = j+1$
13	**end while**

Figure 3.11: BLOVL algorithm.

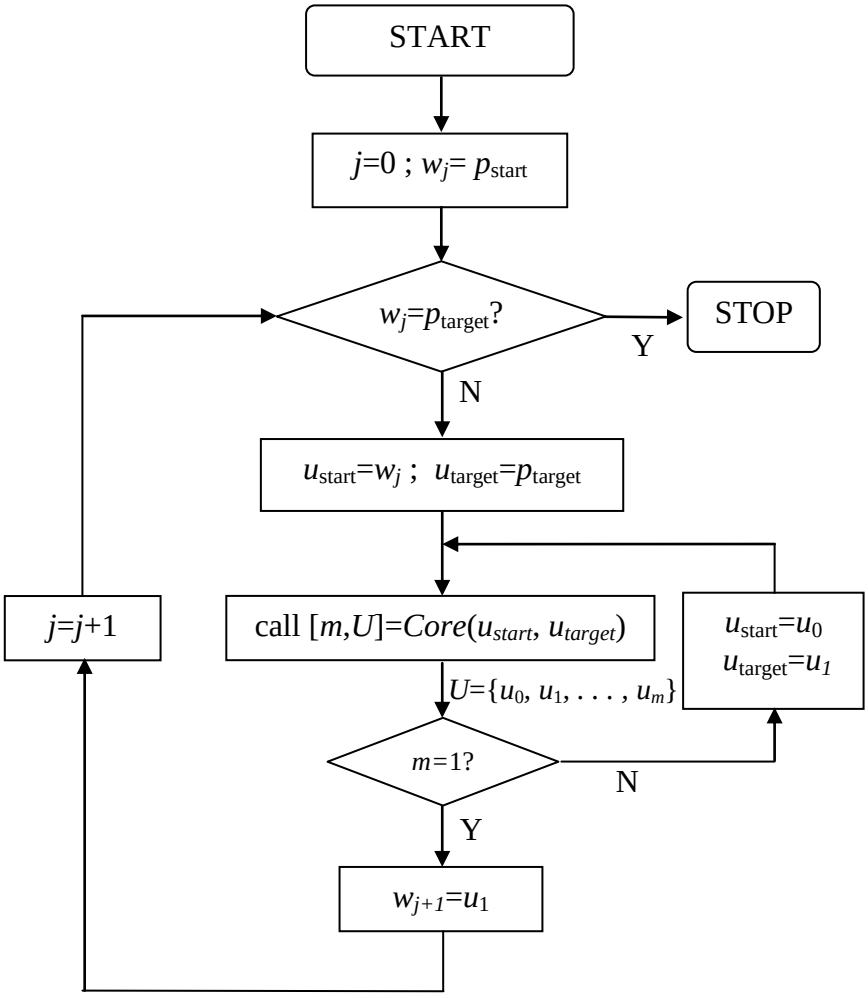


Figure 3.12: BLOVL Algorithm

In particular, the BLOVL algorithm starts with setting the first waypoint index j to 0. Then, the current waypoint w_j is assigned to the global starting point, p_{start}. Subsequently, w_j is compared with the global target point p_{target}. If w_j is the p_{target}, BLOVL will then be stopped as the target point has been reached. Otherwise, the local starting point u_{start} and u_{target} are set to w_j and p_{target}, respectively. Notice that u_{start} and u_{target} are the inputs of *Core* as BL will connect these two points. *Core* is then called to calculate a local path U, which contains a series of waypoints $\{u_0, u_1, \ldots, u_m\}$ and might not be collision-free as previously demonstrated. Note that m represents the number of segments in the path. The smallest possible segment m produced by *Core* is 1, when the resultant local path is $U = \{u_0, u_1\}$. If the segment $m > 1$, it indicates that there are more than 2 waypoints between u_{start} and u_{target}. Thus in the next repetition of BLOVL, u_{target} will be assigned to a new location, u_1, with u_{start} retains its location to u_0. *Core* is called again to find a local path between u_{start} and u_{target}. If there is no obstacle between u_{start} and u_{target} i.e. $m = 1$, the next waypoint w_{j+1} is set to u_1 and the

index j is increased by 1 accordingly. The u_{start} and u_{target} to be fed into *Core* in the next repetition of BLOVL are the u_1 and p_{target}, respectively. The above process will be repeated until w_j is the p_{target}.

3.4.2 Path Planning Using BLOVL

In order to demonstrate the BLOVL, consider the previous scenario as shown in Fig. 3.2. After providing all the necessary parameters, BLOVL will then call the *Core* algorithm to plan an initial path. The input of *Core* is the p_{start} and p_{target}. The resultant path, U containing waypoints $\{u_0, u_1, u_2, u_3, u_4\}$ is shown in Fig. 3.10. According to lines 5 and 6 of BLOVL algorithm, since U has more than two waypoints i.e. the number of line segments m=4, *Core* will be called again. The local starting point is set to u_0 and local target point is u_1. A local path between u_0 and u_1 will then be worked out again by *Core*. This action is shown in Fig. 3.13(a). As m>1, this indicates that there is obstacle between u_0 and u_1. Next *Core* is called again with different starting and target points where the starting point is u_1 of U while target point is the global target point, p_{target}. p_{target} is fed into *Core* because it will ensure that the path is always the shortest one from the current starting point. The above steps are done incrementally until the target point is reached. The path at each repetition is shown in Figs. 3.13(a-f) where green triangles denote the starting points while yellow squares represent the target points. Fig. 3.14 depicts the complete path that has been planned by BLOVL using the scenario as shown in Fig. 3.2. The path is calculated in 0.18 seconds using the same computer as mentioned above. Table 3.6 shows the waypoints of the path.

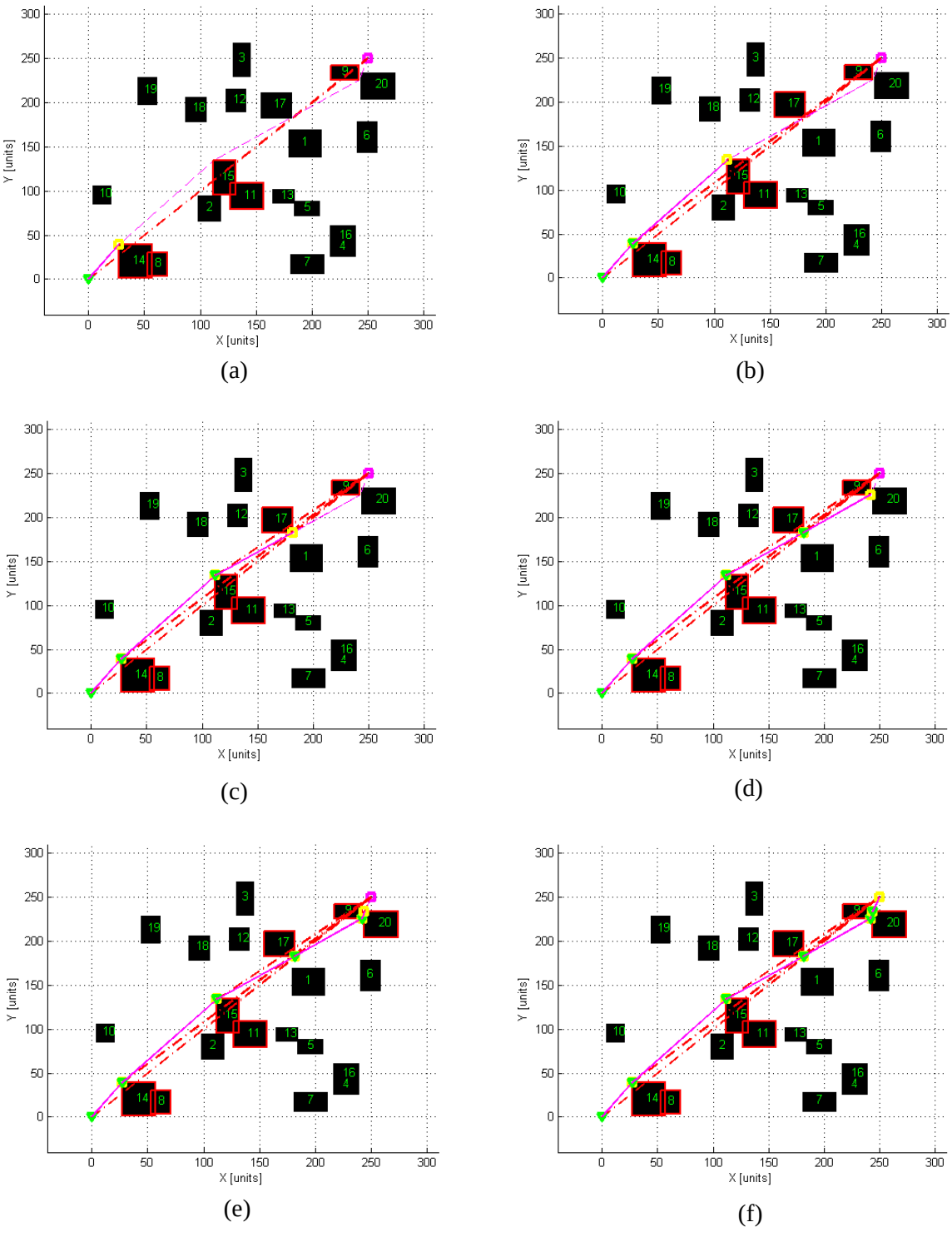


Figure 3.13: The paths generated by *Core* and BLOVL

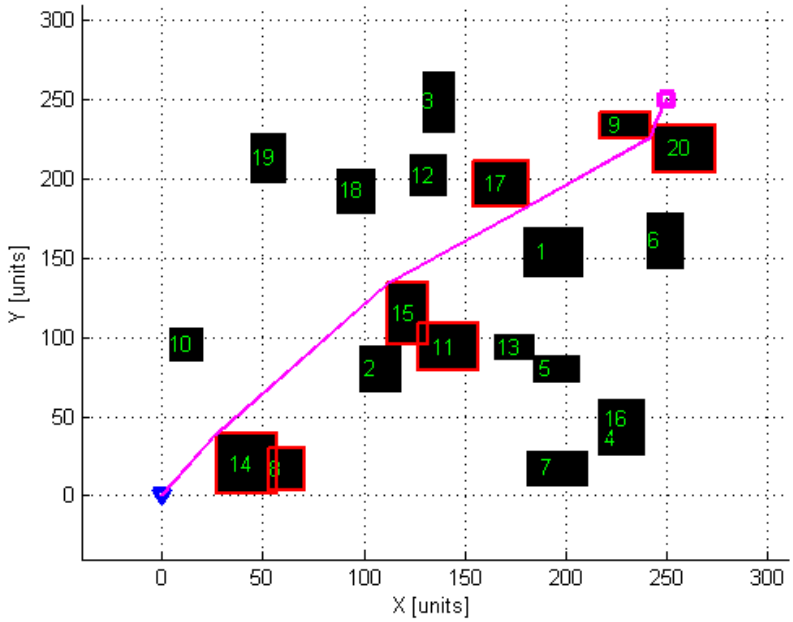


Figure 3.14: The final path calculated by BLOVL.

Table 3.6: The waypoints generated by BLOVL

Waypoints (*W*)	***x***	***y***
w_0	0	0
w_1	27	39
w_2	112	134
w_3	182	183
w_4	242	226
w_5	244	234
w_6	250	250

In terms of completeness, BLOVL is guaranteed to find a path if one exists. Consider a complex and structured scenario as illustrated in Fig. 3.15. BLOVL successfully calculates the path as shown in Fig. 3.16(a) with a length of 317.43 units in 0.047 seconds. On the other hand VL method consumes 0.35 seconds to compute the path, as depicted in Fig. 3.16(b), which is identical to that of BLOVL.

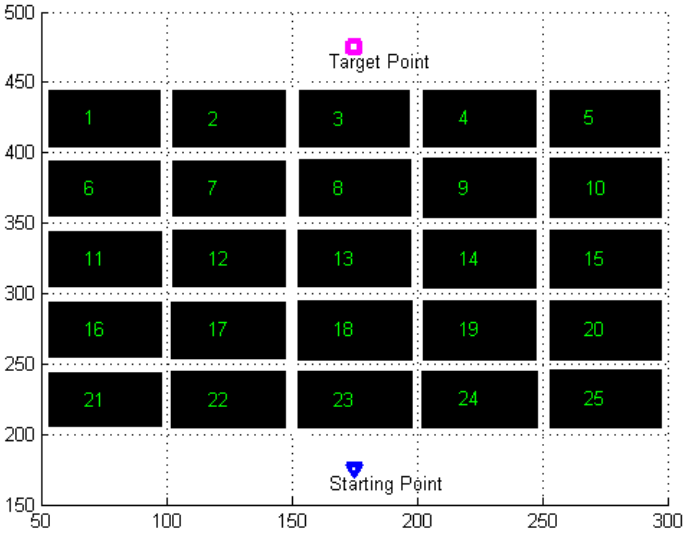
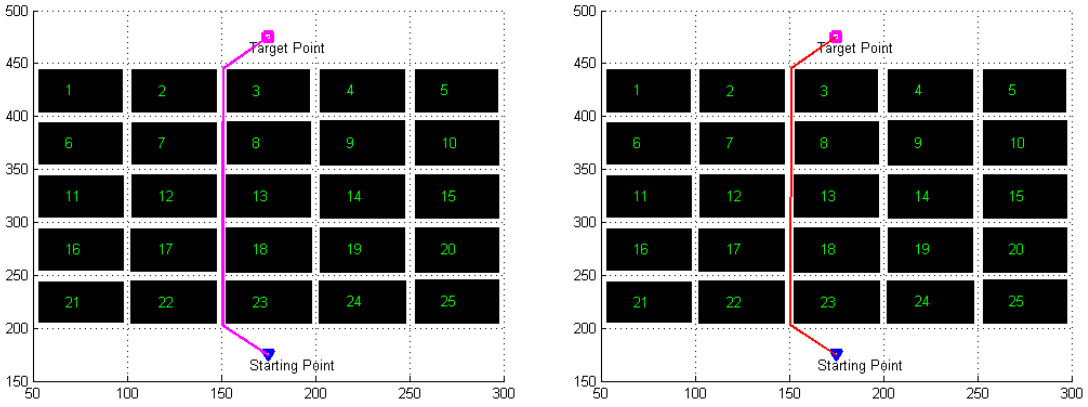


Figure 3.15: Path planning complex and structured scenario



(a) (b)

Figure 3.16: The resultant paths calculated by (a) BLOVL, (b) VL

3.5 Performance Comparisons of VL and BLOVL

This section compares the performance between VL and BLOVL in terms of the computation time and resultant paths lengths. As previously mentioned, one disadvantage of VL method is that the computation time is related to the number of obstacles. A large number of obstacles in *C*-space causes much longer time for the VL to find a path. For example, using the scenario as depicted in Fig. 3.2, the computation time to find the path as shown in Fig. 3.4 is 0.50 seconds on a personal computer with 2.4 GHz processor, 2GB RAM. Using the same scenario, BLOVL takes only 0.18 seconds to generate the path as shown in Fig. 3.14. BLOVL is more than twice faster than VL for path planning with 20 obstacles[3] in that particular scenario. In order to gain a better insight of the performance of the two methods i.e. VL and BLOVL, consider a scenario with an increased number of obstacles as depicted in Fig. 3.17. BLOVL finds the path as shown in Fig. 3.18 in 0.26 seconds, while VL takes 2.30 seconds to generate the path as illustrated in Fig. 3.19. VL is over 8 times slower than BLOVL with both methods produce identical paths.

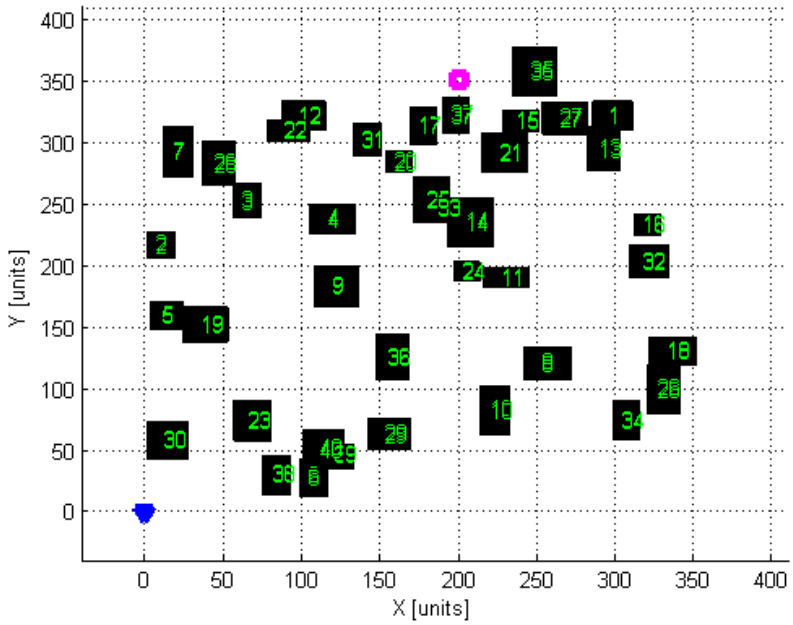


Figure 3.17: A scenario with increased number of obstacles

[3] However, the computation time of BLOVL depends on the obstacles on the base line and their extension as well as their size. Larger obstacles will have more extensions, and at one level, the obstacles that are considered for path calculation are the entire obstacles in the *C*-space. As for VL, the computation time is solely related to the number of the whole obstacles in the *C*-space and it is almost constant for each number of obstacles.

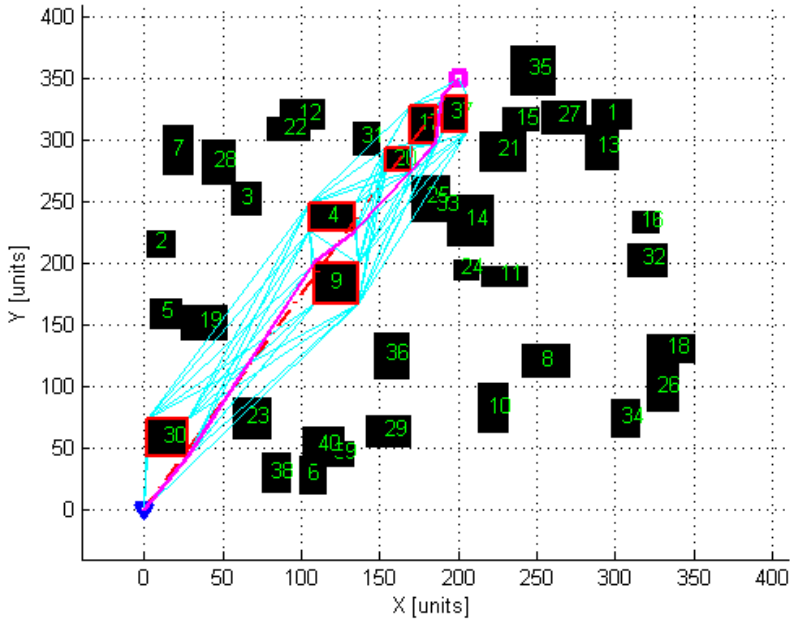


Figure 3.18: A path generated by BLOVL

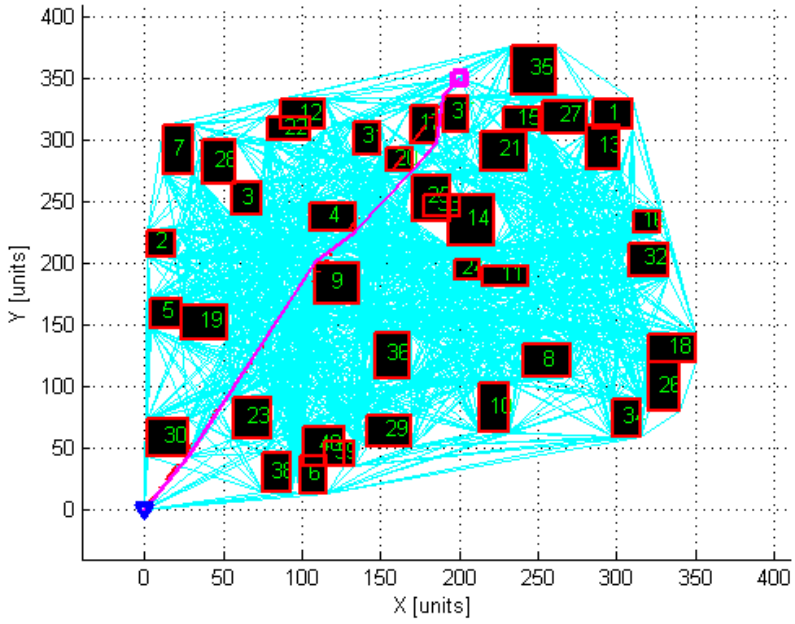


Figure 3.19: A path generated by VL

Increasing the numbers of obstacles will definitely increase the computation time of both VL and BLOVL. In order to see the performances of VL and BLOVL in scenarios with different numbers of obstacles, simulations using 10, 20, 30, 50, 100, 150, 200 and 300 obstacles were performed. Each number of obstacles was randomly simulated 200 times to ensure the reliability of the finding. The simulated computation time of VL are shown in Fig. 3.20 while the computation time of BLOVL are shown in Fig. 3.21. To have a clearer insight of the performance of both methods, a log scale graph as shown in Fig. 3.22 is presented. It is apparent from the figure that, as the number of obstacles increases, VL's computation time also raise significantly.

Table 3.7 compares the average computation time of both VL and BLOVL with respect to the number of obstacles. Column three of the table lists how many times BLOVL is faster in average than VL for each obstacle number. The table proves that BLOVL is significantly faster than VL in finding paths. The main reason why BLOVL is faster than the conventional VL is BLOVL generates less number of visibility lines during the paths calculation. For example, in an environment with 300 obstacles, VL generates 721,801 visibility lines while BLOVL produces only 15,238.

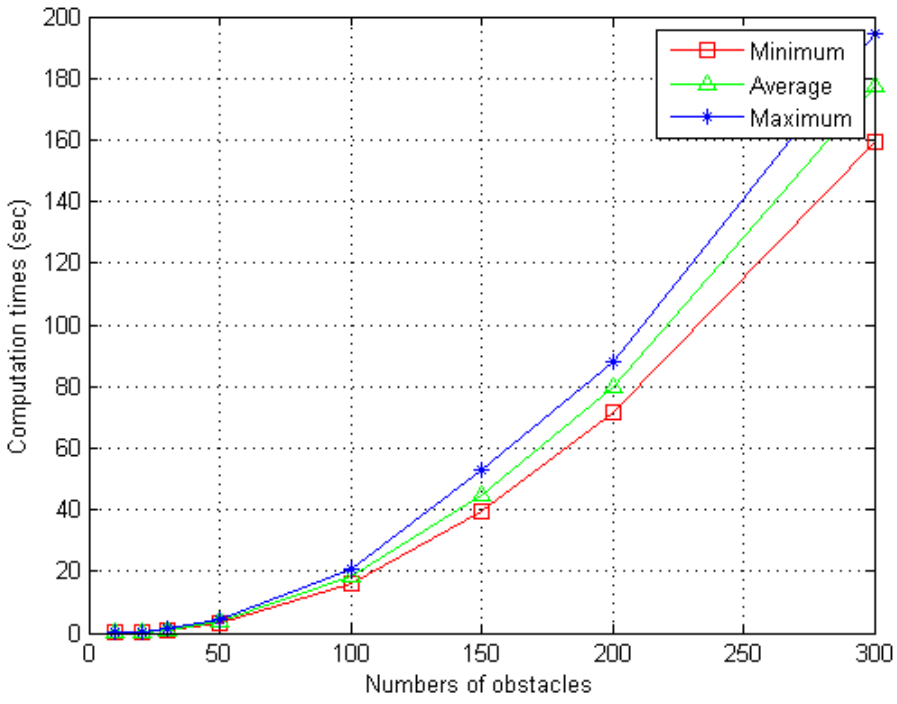


Figure 3.20: Computation time of VL from different number of obstacles

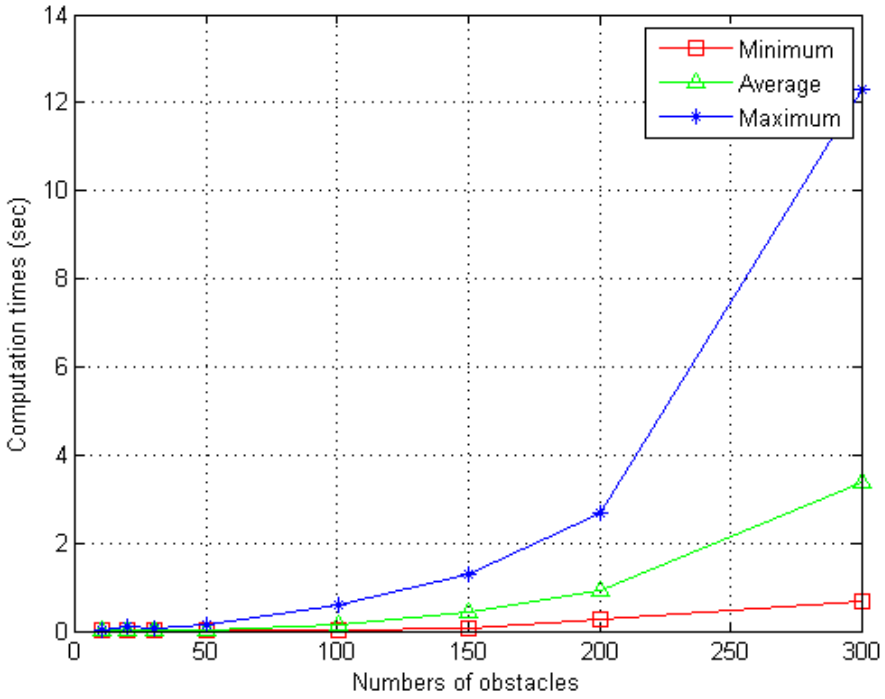


Figure 3.21: Computation time of BLOVL from different number of obstacles

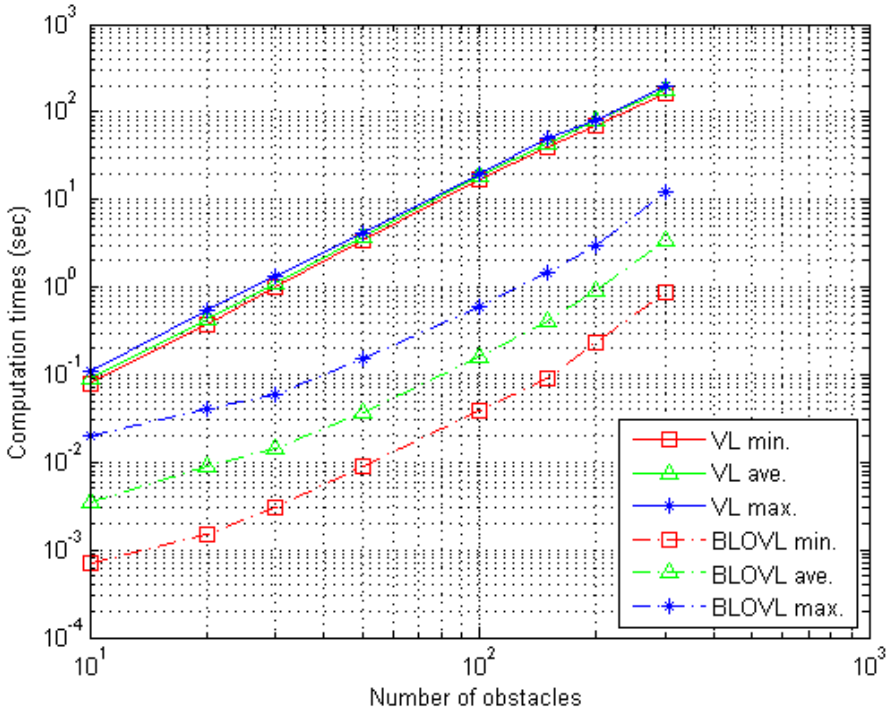


Figure 3.22: Comparison of VL's and BLOVL's computation time in log scale

Table 3.7: Comparison of VL and BLOVL computation time

Obstacle Numbers	Average computation time (s)		Ratio of BLOVL/VL computation time
	VL	BLOVL	
10	0.0904	0.0035	25.7655
20	0.4306	0.0092	47.0597
30	1.1286	0.0147	76.9581
50	3.7887	0.0367	103.3613
100	18.4947	0.1565	118.2110
150	44.5318	0.4119	108.1152
200	80.0284	0.9187	87.1121
300	177.2581	3.3795	52.4508

On the other hand, the paths lengths resulting from the simulations are depicted in Fig. 3.23. It is clear from the figure that a number of paths generated by BLOVL are slightly longer than VL especially in scenarios with higher number of obstacles. This happens because BLOVL considers local path between two consecutive waypoints of the previously planned path to ensure that there is no obstacles between the two waypoints. The next section will illustrate this situation.

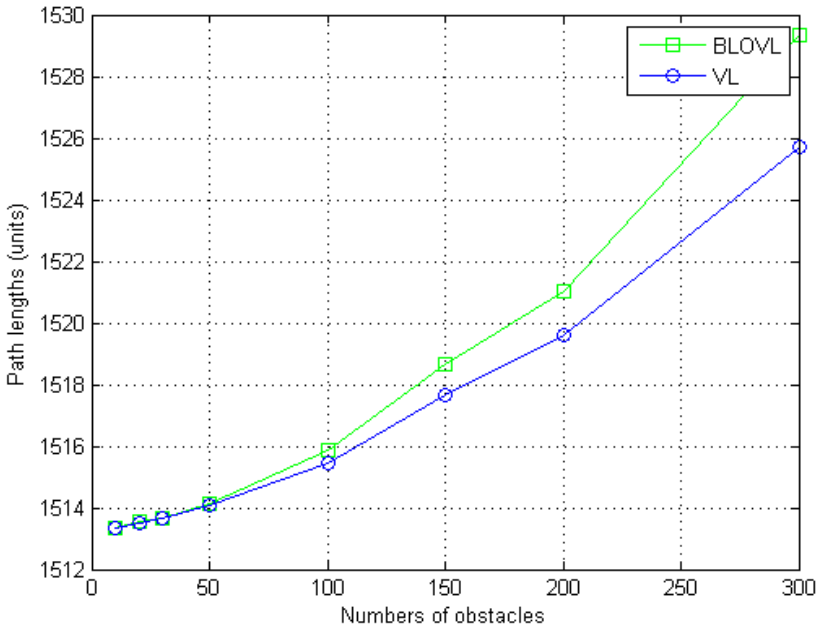


Figure 3.23: Paths lengths of VL and BLOVL in environments with different number of obstacles.

3.6 BLOVL Path

It has been shown in the previous section that BLOVL sometimes produces slightly longer paths compared to the conventional VL, especially in obstacle-rich environments. In order to get an idea why BLOVL produces a longer path, consider the scenario as shown in Fig. 3.24, in which a path is to be planned from p_{start} to p_{target} using both BLOVL algorithm and VL method. Note that the grey rectangles are the obstacles. The resultant paths are shown in Fig. 3.25 where the red and black lines are the paths that have been planned by VL and the first repetition of BLOVL, respectively. In the figure, the dashed black line represents the base line (BL), and the red-outlined rectangles are the obstacles that have been identified by the BL or O_{Core}. At this point, the path by BLOVL is shorter than that of VL as it disregards a large number of obstacles in the scenario. However, the path is not collision-free. BLOVL then checks the collision between two consecutive waypoints i.e. u_0 and u_1. It is found that there is an obstacle that has not been considered during the previous repetition. A new path is then planned by *Core* of BLOVL and is shown in Fig. 3.26.

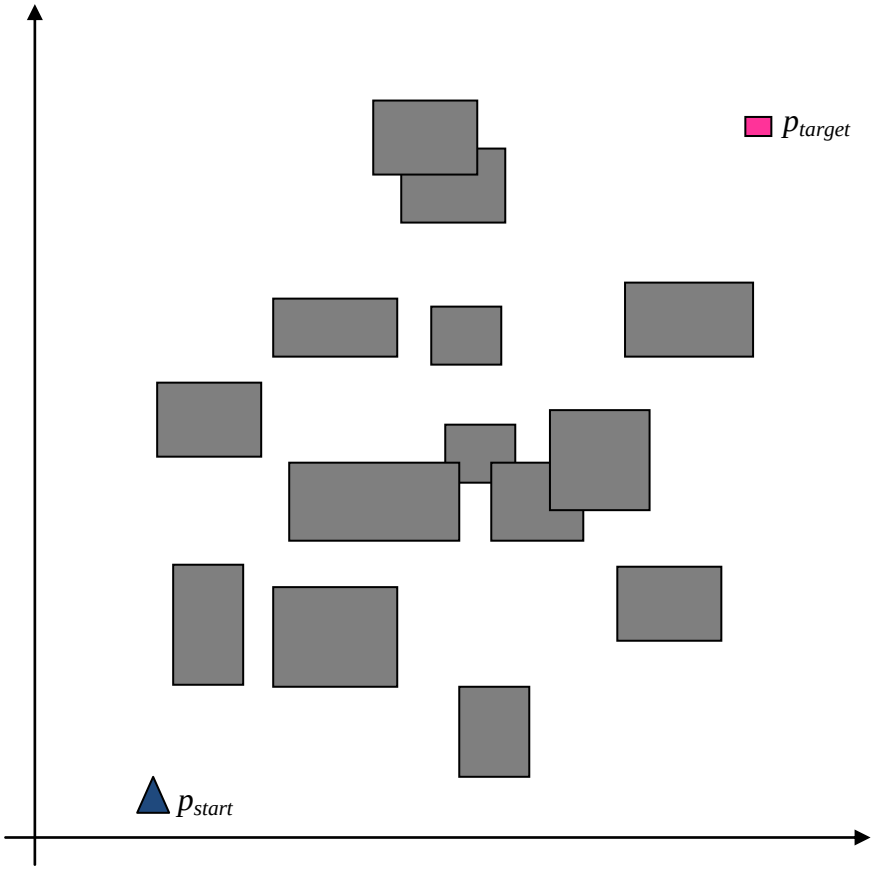


Figure 3.24: A scenario for path planning.

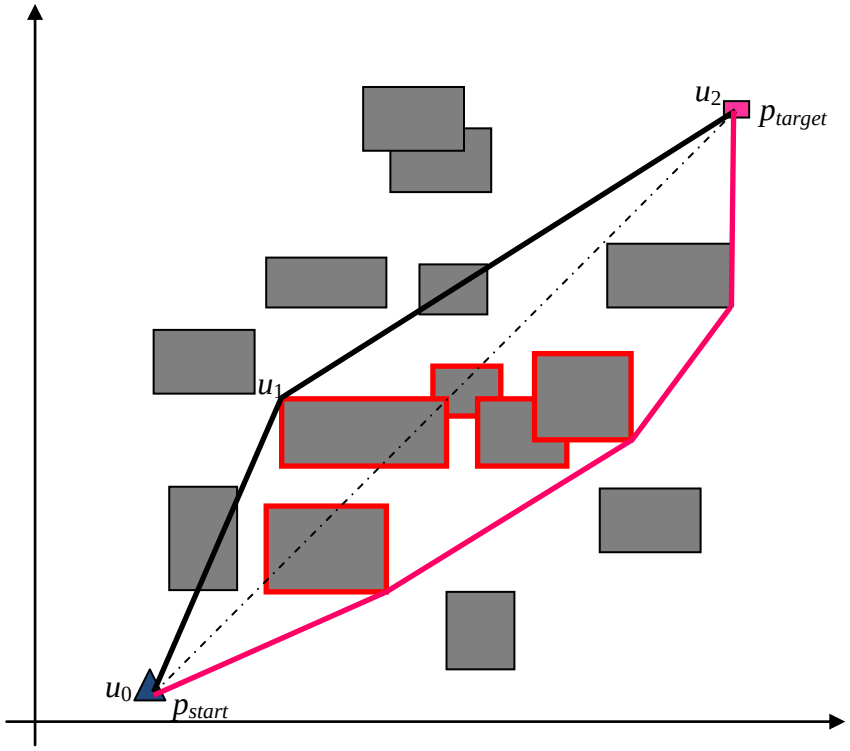


Figure 3.25: The paths planned by BLOVL (black) and VL (magenta)

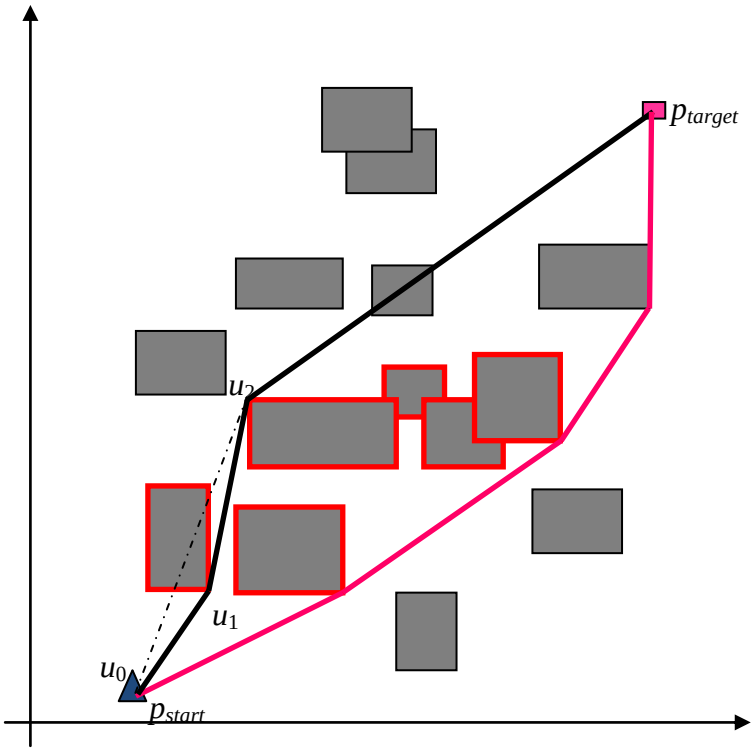


Figure 3.26: BLOVL updates the path as the path planning progresses.

BLOVL keeps checking the planned path to ensure that there is no collision between two consecutive waypoints as shown in Figs. 3.27 and 3.28. As a result, BLOVL plans longer path, compared to the one that is planned by VL.

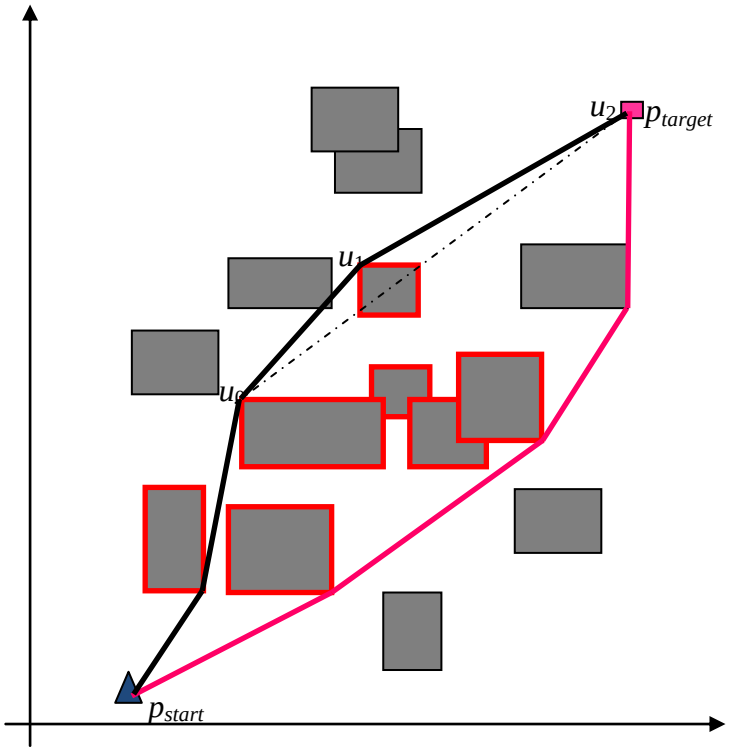


Figure 3.27: BLOVL updates the path as the path planning progresses.

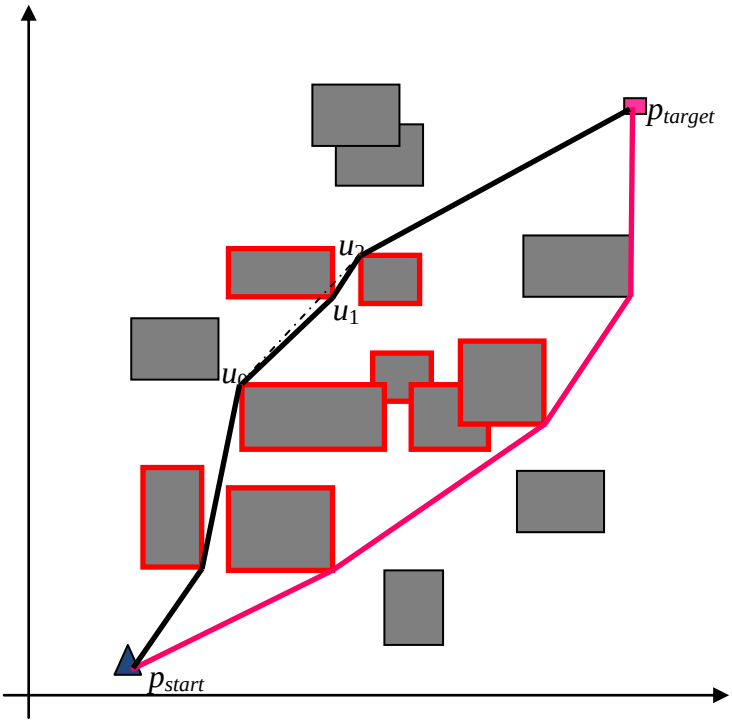


Figure 3.28: The paths planned by BLOVL (black) and VL (magenta)

3.7 Safety Margin

VL and BLOVL produce paths which contain waypoints that pass through obstacles' nodes. This is highly unsafe and will lead to collisions between the vehicle and obstacles. One way to avoid such a collision is to extend the size of the obstacles by a certain margin before the path is planned. The margin is called the clearance or safety margin, M_S. The safety margin is important for the UAV to be able to traverse the planned path safely without hitting any obstacle throughout the entire C-space, until it reaches the target point. In calculating M_S, the kinematic/physical constraints of the UAV such as maximum roll angle (φ_{max}) and minimum turning radius R_{min} have to be considered. The relationship between minimum R_{min} and φ_{max} of a UAV can be defined as follow [122]:

$$R_{min} = \frac{V^2}{g \times tan(\varphi_{max})}$$

where V is the vehicle's velocity. In order to ensure that the path is safe for the UAV so that it can traverse through the obstacles' nodes, M_S is determined based on a worst case scenario. For VL-based methods in C-space with rectangle obstacles (as stated in Section 1.4), the worst case scenario that can happen when two consecutive segments of a path is made of two consecutive edges of the obstacle, which has the angle of 90 degree as shown in Fig. 3.29. Note that in the figure, the red lines are the planned path. *Ms* is then calculated as follows:

$$M_s = \sqrt{2} \times R_{min} - R_{min}$$

The enlarged obstacle by M_S and the resultant path are shown in Fig. 3.30.

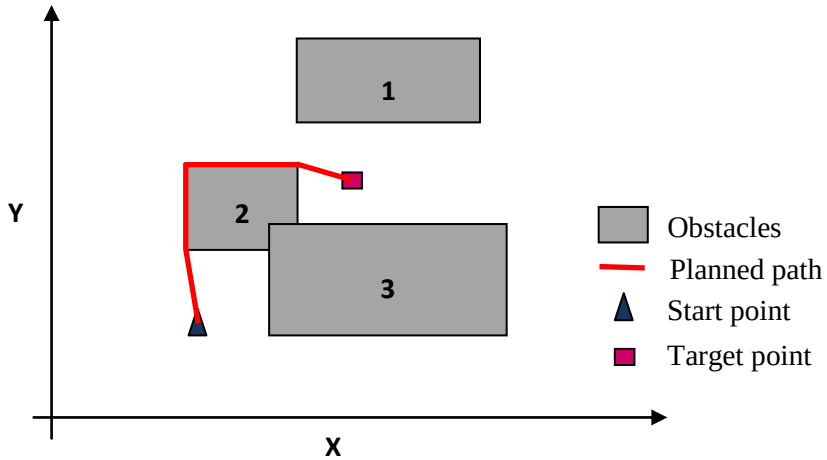


Figure 3.29: A piece-wise linear segments path in the worst case scenario

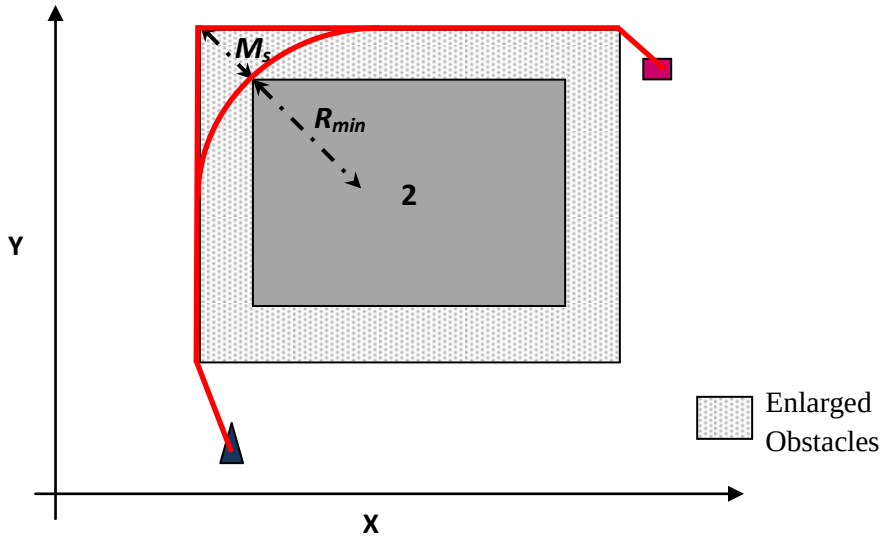


Figure 3.30: The enlarged obstacle with safety margin

In order to demonstrate the implementation of the safety margin in a path planning process, consider a randomly generated scenario with 20 obstacles as shown in Fig. 3.31. The grey rectangles are the enlarged obstacles after a *Ms* is applied. The UAV's speed and maximum roll angle are set to 50km/h (or 16.67m/s) and 50 degrees, respectively. R_{min} and M_S are then 28.32 and 11.73 units, respectively. The planned path using the BLOVL algorithm applying the safety margin is depicted in Fig. 3.32.

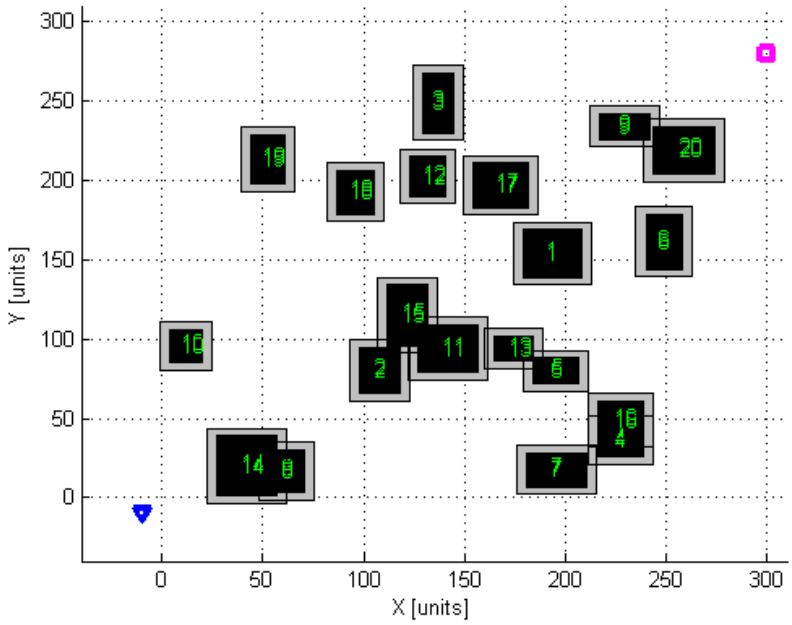


Figure 3.31: The obstacles with safety margin.

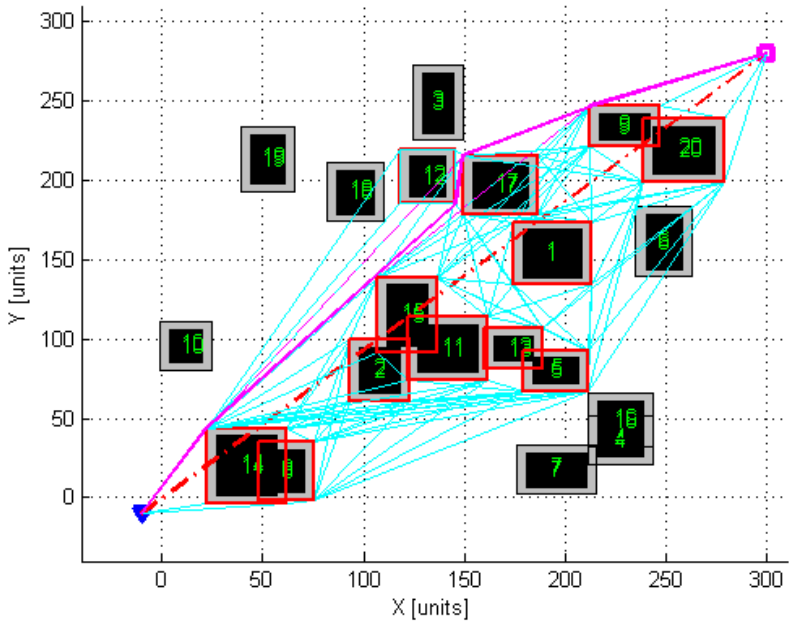


Figure 3.32: A path with safety margin generated by BLOVL

As has been demonstrated, the safety margin clearly changes the path from semi collision-free to fully collision-free, considering the vehicle's physical constraints.

3.8 BLOVL for Real Time Path Planning

In the previously demonstrated path planning process using BLOVL, it was assumed that all the obstacles data were known either from satellite data, surveillance information or other means beforehand. However, in certain situations, not all the data are available. Some obstacles might appear suddenly (pop-up) on the UAV path. Such occurrences require path re-planning in real time in order to avoid a collision between the UAV and the pop-up obstacles.

Assuming that the UAV is equipped with on-board sensors which can sense and measure the pop-up obstacles' locations, geometry and sizes accurately, BLOVL is capable of finding a collision-free path in such an occasion in real time as the planning considers a small set of obstacles and are executed sequentially from one waypoint to the next one until the target point is reached. It means that, if there is a pop-up obstacle between two consecutive waypoints, the algorithm will find a new path, considering local obstacles, which enables the realisation of real-time path planning.

In order to demonstrate a real-time path planning using BLOVL, consider the scenario as shown in Fig. 3.33. The obstacles that are known prior to path planning are shown in black. The grey areas surrounding the obstacles are the safety margins. The sensor range is assumed to be 50 units. There will be a random pop-up obstacle during the flight. It is also assumed that the obstacle pop-ups outside the sensor range to give the UAV enough time to re-plan a new path and a sufficient distance to manoeuvre to avoid the pop-up obstacle.

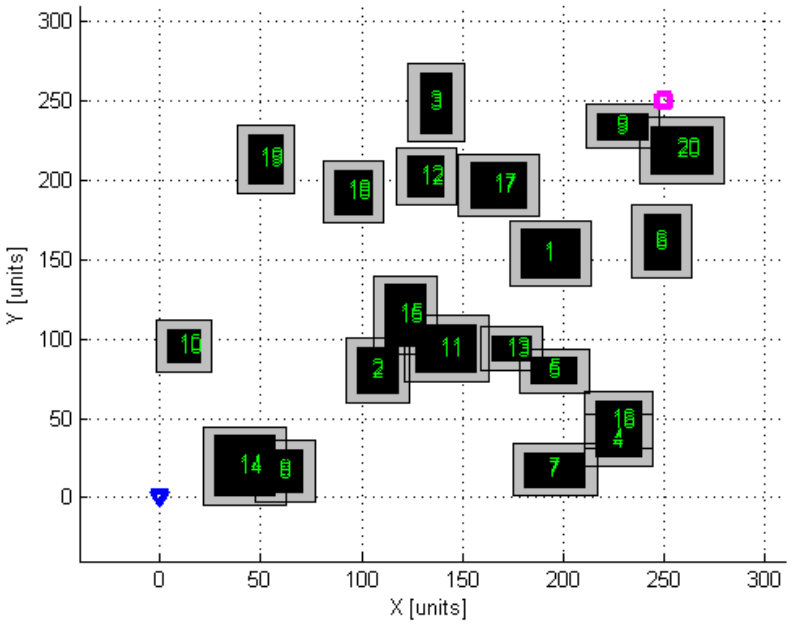


Figure 3.33: The scenario in which a real-time path planning will take place.

The BLOVL real-time path planning starts with calculating an initial path using *Core* as shown (in magenta) in Fig. 3.34(a). Note that in the figure, blue triangle represents the starting point while the magenta square denotes the target point. The path is currently unsafe because one segment of the path intersects with interior of an obstacle. However, the path will be later re-planned as the UAV approaches the collision area.

Fig. 3.34(b) illustrates the UAV has started the mission traversing along the planned path going to the first waypoint. The green-dotted semi-circle in the figure represents the UAV's sensor range. As the UAV reaches the first waypoint as shown in Fig. 3.34(c), BLOVL examines the feasibility of traversal between the current waypoint and the next one. In this example, the segment connecting the current waypoint is

obstructed by an obstacle, requiring the path to be re-planned. The resulting re-planned path is shown in Fig. 3.34(d). The UAV is then traversing the path to the next waypoint. However, after a while, there is a pop-up obstacle (shown in yellow) on the current path. When the UAV's sensor detects the obstacle, BLOVL starts to re-plan a new path as depicted in Fig. 3.34(e). The above steps are repeated and are depicted in Figs. 3.34 (f-h). The UAV follows the path avoiding the pop-up obstacle until it reaches at the target point as illustrated in Fig. 3.35.

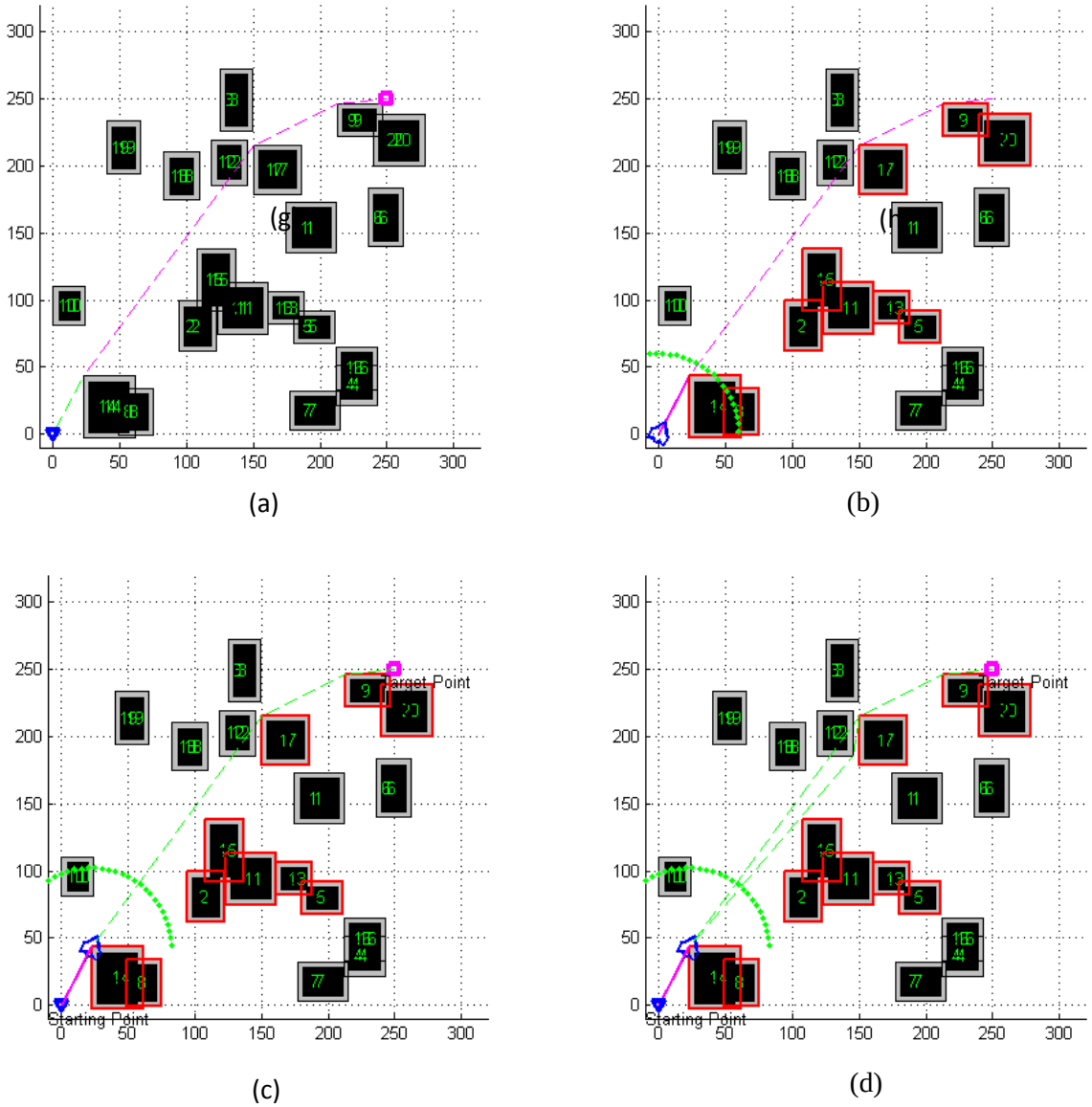
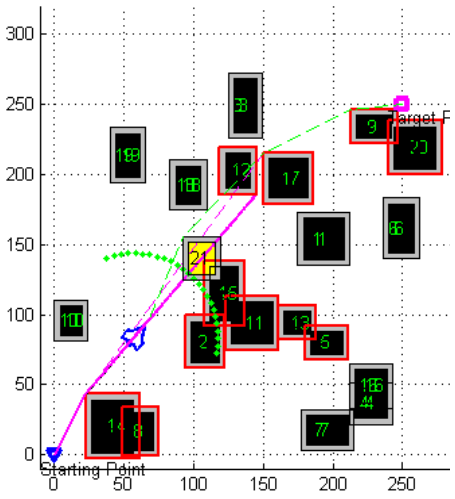
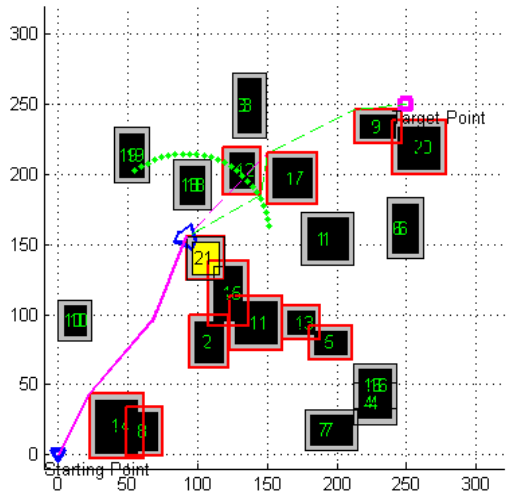


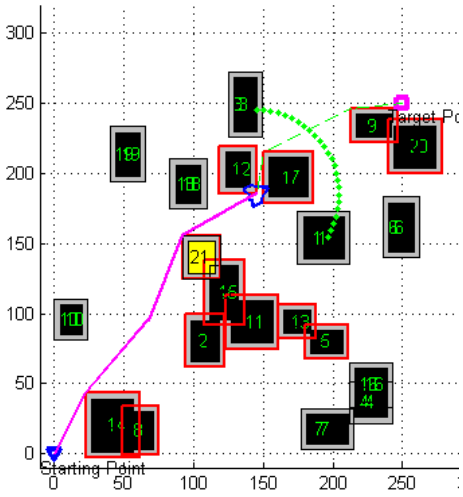
Figure 3.34**:** BLOVL for real time path planning



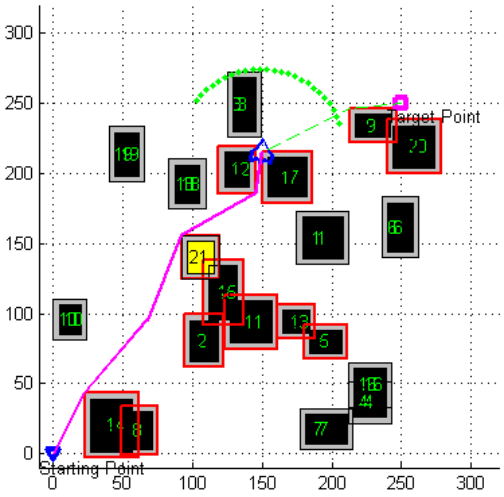
(e)



(f)



(g)



(h)

Figure 3.34: BLOVL for real time path planning (contd).

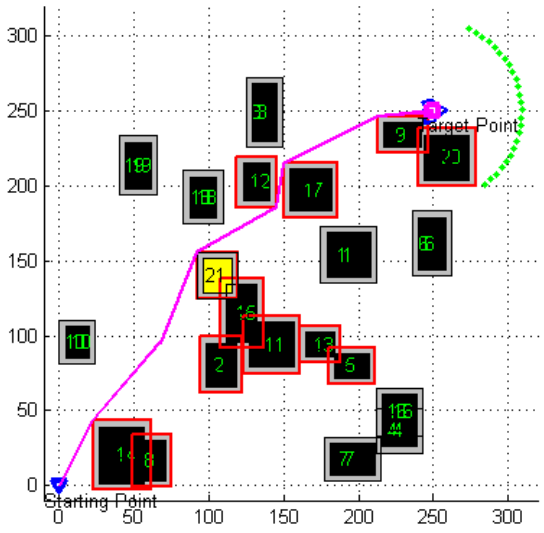


Figure 3.35: The resultant path using BLOVL with a pop-up obstacle.

3.9 Improvement to BLOVL

As has been demonstrated in Section 3.5, BLOVL is relatively faster in comparison with VL in planning a path. This is due to the fact that BLOVL uses a relatively less number of obstacles during the path calculation. In order to further accelerate the BLOVL's computation time, an improvement to BLOVL is proposed in this section.

It has been previously emphasised that BLOVL uses a base line (BL) that connects the p_{start} to p_{target} in order to identify a set obstacles for path calculation. In a large environment with many obstacles, the BL might identify a high number of obstacles, which results in a much longer time to generate a collision-free path. This situation is illustrated in Fig. 3.36. In this particular scenario, BL identifies 17 obstacles that will be in considered in path planning. Although the BL uses a significantly lower number of obstacles in comparison with that of the VL, its computation time might still be high which may hinder BLOVL in carrying out a path planning in real-time.

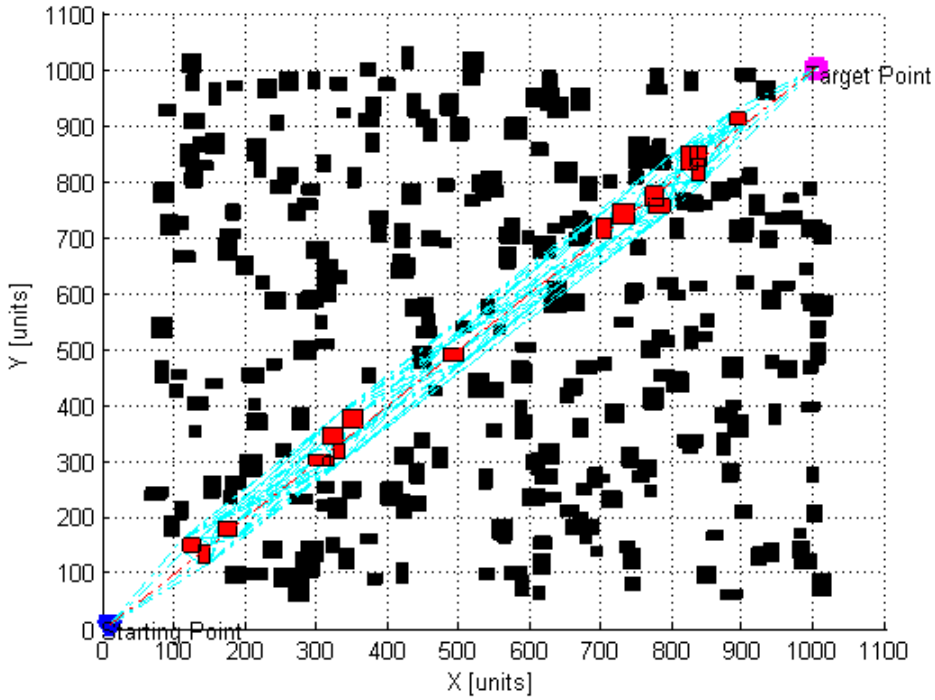


Figure 3.36: BL identifies the obstacles to be used by BLOVL.

In order to address this issue, BLOVL with limited range of BL is proposed. The approach is then called BLOVL with limited range. BLOVL with limited range will further reduce the number of obstacles used for path planning which results in reduced computation time. Using BLOVL with limited, only the obstacles that are within the limit of the BL's range are identified and used for path calculation. For example, using the identical scenario as in Fig. 3.36, a limited BL with a length of 300 units identifies only 3 obstacles, which are shown in red in Fig. 3.37. This number is much less than the number of obstacles identified by normal BLOVL. This in turn reduces the overall computation time.

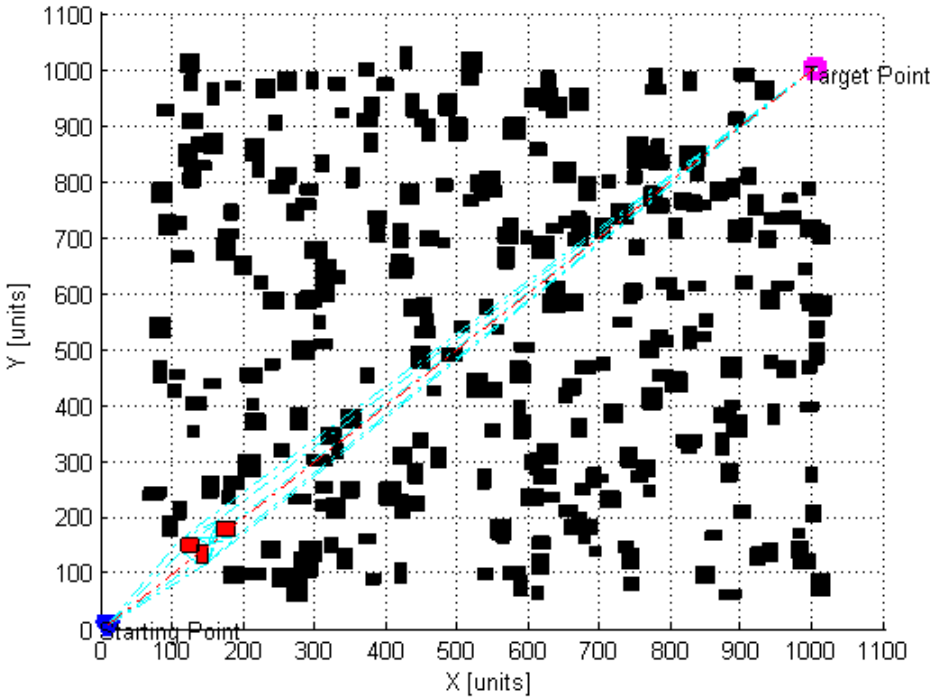


Figure 3.37: BL with limited range identifies the obstacles to be used by BLOVL

In order to demonstrate the efficacy of the BLOVL with limited range, simulations using several numbers of obstacles i.e. 50,100,150,200,250 and 300 were performed. Each number of obstacles was generated 100 times randomly. The size of the environment of each scenario was set to $[1000 \times 1000]$ units. The starting and the target point were fixed to (0,0) and (1000,1000), respectively. Fig. 3.38 shows the scatter plot of the computation time of BLOVL and BLOVL with limited range, respectively.

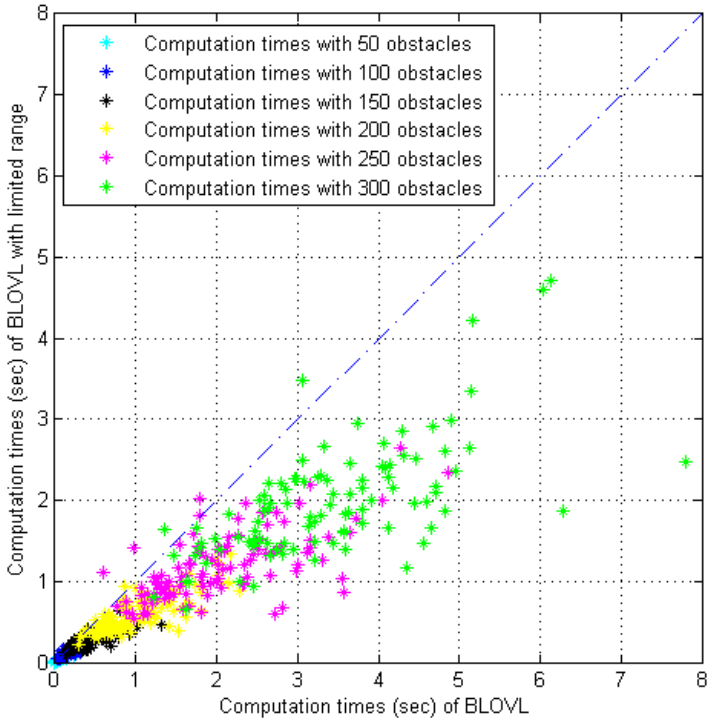


Figure 3.38: Computation time comparison between BLOVL and BLOVL with limited range

It is transparent that, as the number of obstacles increases, the differences in the average (of computation time) between those two approaches also increase. The figure clearly proves that BLOVL with limited range considerably outperforms the BLOVL in terms of computation time.

On the other hand, Fig. 3.39 illustrates the scatter plot of the average paths lengths of BLOVL and BLOVL with limited range, respectively. Although BLOVL with limited range uses relatively lesser number of obstacles, it still produces paths with similar lengths to those of normal BLOVL.

Thus, it is concluded that the proposed BLOVL with limited range has better performance in terms of computation time in an environment with higher number of obstacles compared to that of BLOVL. Although its computation time are significantly reduces, the lengths of the planned paths by BLOVL with limited range are quite identical to the normal VL

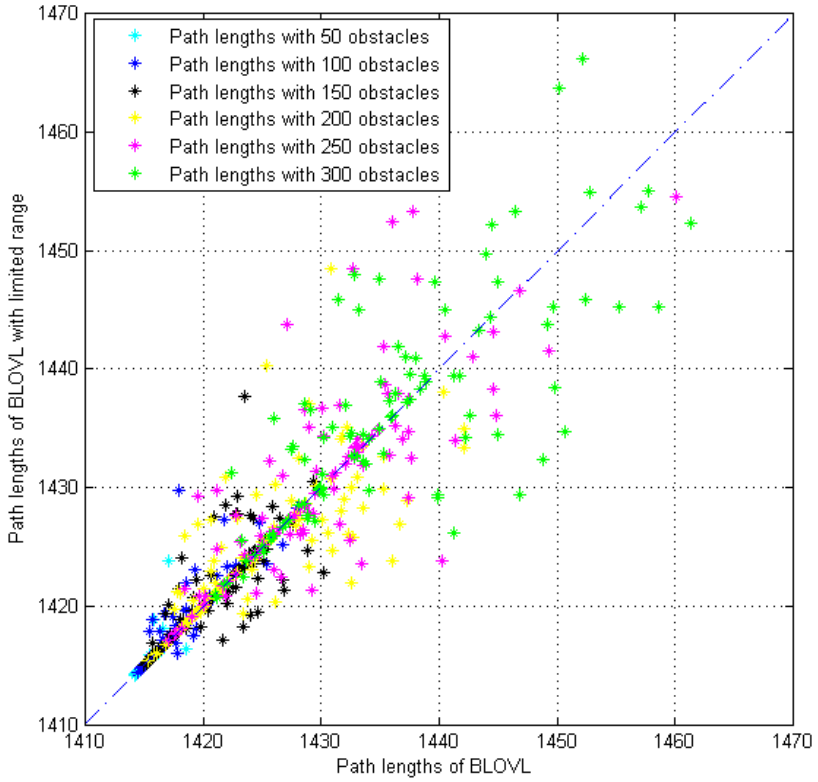


Figure 3.39: Paths lengths comparison between BLOVL and BLOVL with limited range

3.10 Conclusion

Visibility Line (VL) is a path planning method that is capable of producing optimal path if one exists. However, the VL method is computationally expensive when there are many obstacles in the environments. As a result, VL is not suitable to be applied in real time, in obstacle-rich and dynamic environments.

This chapter has proposed algorithms for 2D path planning based on the VL method. The proposed algorithms are capable of addressing the drawback of VL as they plan collision-free paths in computationally tractable manner.

The first proposed algorithm, *Core* is used to find a path from a set of obstacles that are determined by the so-called base line (BL). BL is a segment that connects the starting point and the target point. All obstacles that lie on BL and their extension are the set of obstacles that will be used for path planning. As the set contains a relatively small number of obstacles, the path is quickly calculated.

However the path planned by *Core* might not collision-free, hence the algorithm called the Base Line Oriented Visibility Line (BLOVL) has been designed. BLOVL runs sequentially by checking visibility between two consecutive waypoints of a planned path. If those two waypoints are obstructed by obstacles, a new path will be re-planned avoiding the blocking obstacles. This process is done repetitively until the target point is reached.

Through simulations, BLOVL has been proven to generate a path relatively faster than VL. Although BLOVL uses less number of obstacles during a path planning process, the length of generated paths are similar to VL's. To ensure the completeness of the algorithm, BLOVL will keep finding a path until at one level in which all obstacles are used for the path calculation.

Furthermore BLOVL is capable of re-planning a new path if there is pop-up obstacle, which lies on a UAV's current path. BLOVL is also designed to perform a path planning task in real-time.

To further improve the performance of BLOVL in terms of computation time, BLOVL with limited range has been introduced. The idea behind this approach is to use a limited range of base line so that a smaller number of obstacles are considered for path calculation.

Chapter 4

3D Visibility Line Based Path Planning

4.1 Introduction

In the previous chapter, a visibility line (VL)- based path planning algorithm called Base Line Oriented Visibility Line (BLOVL) have been developed and demonstrated. The algorithm efficiently produces a 2D path in a 2D environment (*X*-*Y* plane) with fast planning time by minimising computational loads.

In this chapter, a set of three-dimensional (3D) path planning algorithms are proposed. The algorithms utilise the so-called rotational plane approach to find 3D paths. Besides, as BLOVL is fast in finding a path, the proposed algorithms are based on it.

This chapter is arranged as follows. Path plannings in a 3D scenario applying the Base Line Oriented Visibility Line (BLOVL) are presented in the following section. Later, a couple of algorithms called *BasePlane*, *FindIntersection* are introduced and demonstrated. Next the BLOVL3D1 algorithm is introduced and elaborated, followed by a demonstration of finding a 3D path on a plane using the *BasePlane*, *FindIntersection* and BLOVL3D1 algorithms. The following section proposes the BLOVL3D2 algorithm that combines the *BasePlane*, *FindIntersection* BLOVL3D1 algorithms, and show an example of BLOVL3D2 calculating a 3D path in a 3D scenario. Simulations of paths planning using the proposed algorithms with different number of rotation angles and obstacles are carried out prior to conclusion.

4.2 Direct Applications of BLOVL 2D Path Planning Algorithm in 3D Environment

As introduced in Chapter 2, the visibility line (VL) method considers all nodes of obstacles in a 2D environment to find a 2D path. For 2D obstacles, the nodes are the corners of the obstacles.

In a 3D environment with a set of 3D obstacles, a VL based method may not be able to find a 3D path due to the infinite number of nodes along each border's edge of the obstacles. A border's edge is defined as a line that separates two adjacent faces of an object. For example, if a 3D obstacle was modelled as a cuboid which has 6 faces, there were 12 borders' edges that separated such faces.

However, it is possible to find a path in a 3D scenario using BLOVL if a starting point p_{start} has an equal altitude with that of a target point p_{target}. In a scenario where the p_{start} altitude is not identical to the altitude of p_{target}, a path can be found if a project point of the starting point, p_{ps} or the project point of the target point, p_{pt} is first defined. A project point p_{ps}/p_{pt} is the (x,y) coordinate of p_{start}/p_{target} that is raised or lowered to the altitude of p_{target}/p_{start}. Using the project point, a path from p_{start} and p_{target} can be calculated using BLOVL in two ways. The first one begins with defining a project point p_{pt}, followed by calculating a horizontal path from p_{start} to p_{pt}. To complete the path planning process, a vertical path connecting p_{pt} and p_{target} is worked out. The second way of finding a path from p_{start} to p_{target} with different altitude starts with defining a project point p_{ps}. Subsequently, a vertical path connecting p_{target} and P_{ps} is identified. Then a path from P_{ps} to p_{target} is calculated to complete the path planning process. On the other hand, the 2D path planning approach can be applied to find a path on a vertical plane as will be demonstrated later.

4.2.1 p_{start} and p_{target} With Identical Altitude

A scenario in which p_{start} has an equal altitude with the p_{target}'s is shown in Fig. 4.1. Note that in Fig. 4.1 and afterwards, the blue triangles denote p_{start} while the magenta squares represent p_{target}.

As p_{start} and p_{target} have an equal altitude, the 2D path planning algorithm can be applied to find a path using BLOVL. Briefly, the first step is to define a set of nodes whose altitudes are set to the height of p_{start} and p_{target}. Then a visibility lines network is created from the nodes set as shown in Fig. 4.2. The nodes are represented by the green dots. Finally, Dijkstra's algorithm is used to find a 2D path as illustrated in Fig. 4.3.

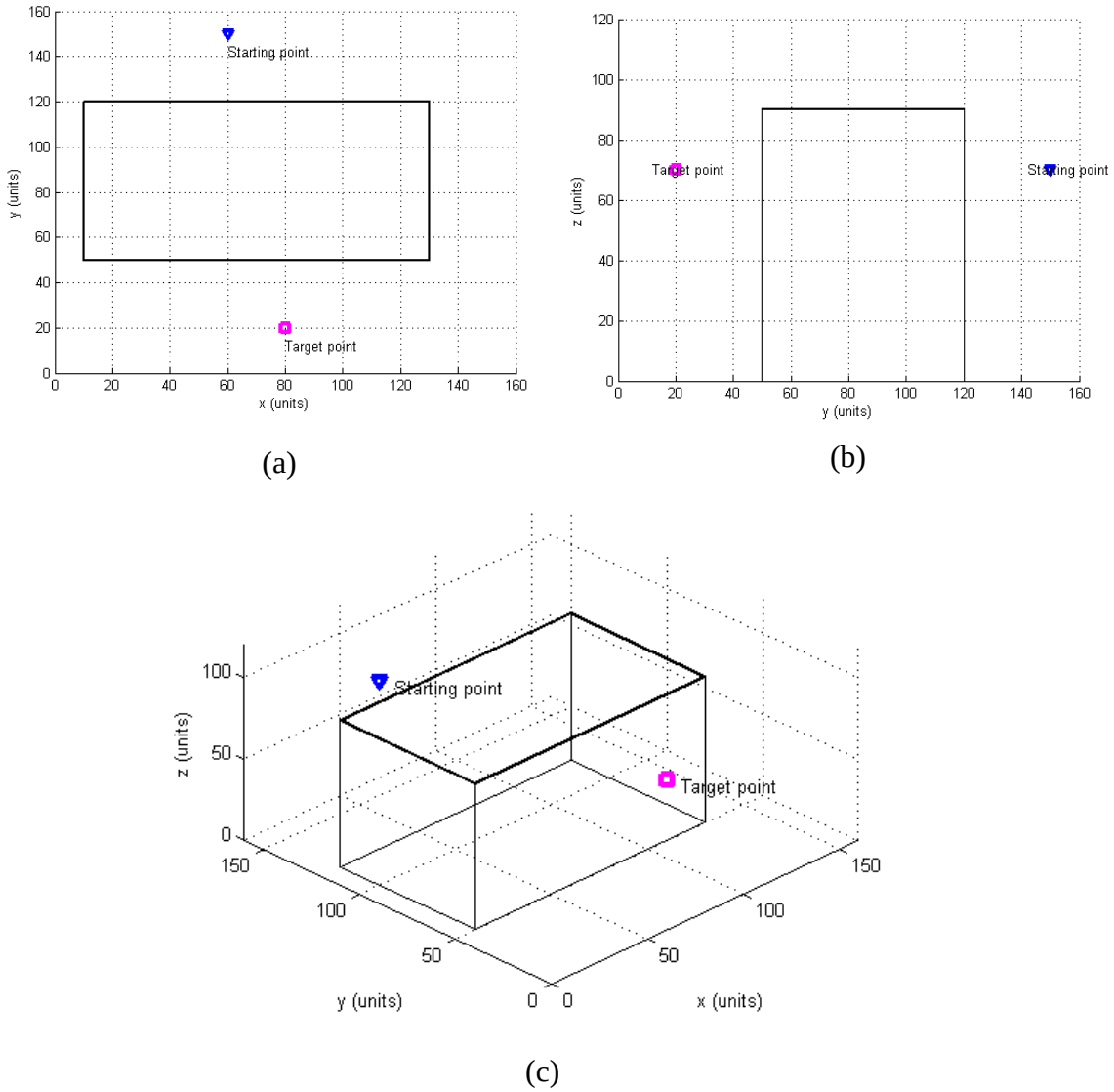
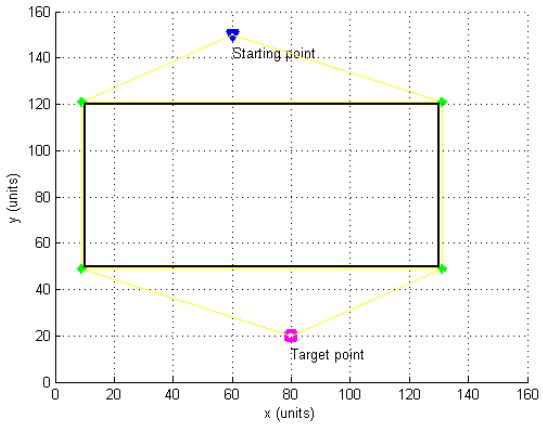
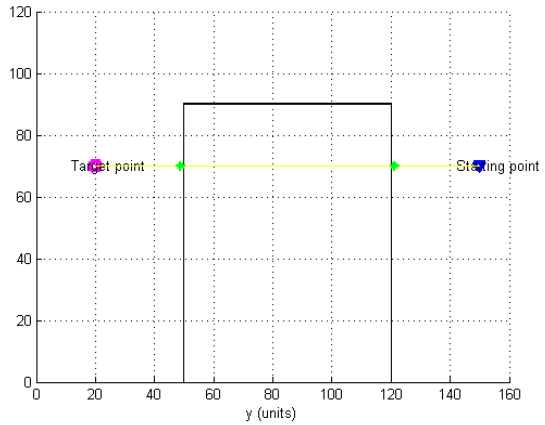


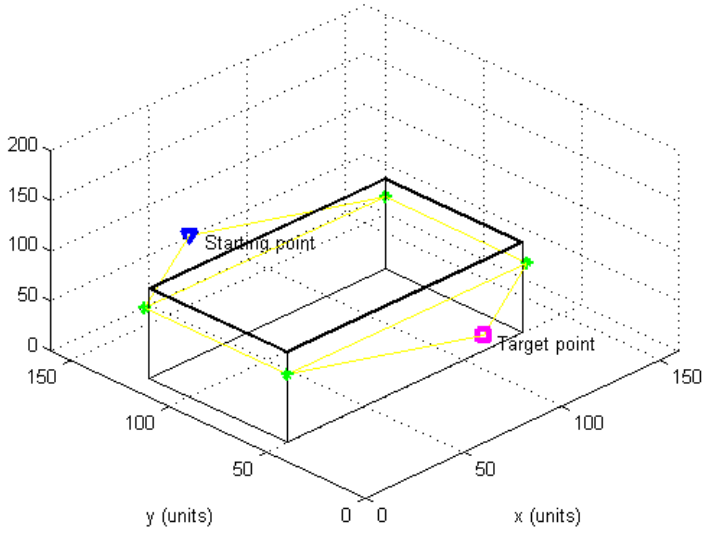
Figure 4.1: A 3D environment with a 3D obstacle, in which the starting and target points have an equal altitude; (a) top view, (b) side view, (c) 3D view.



(a)

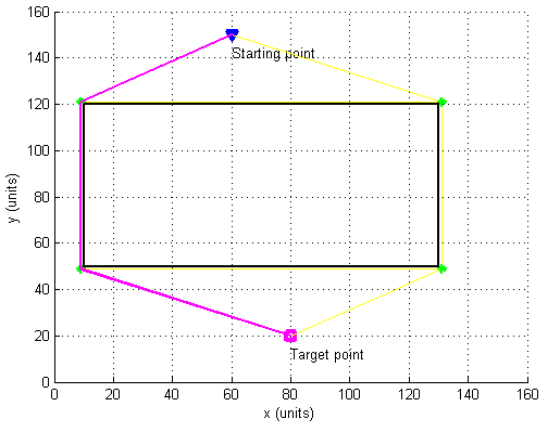


(b)

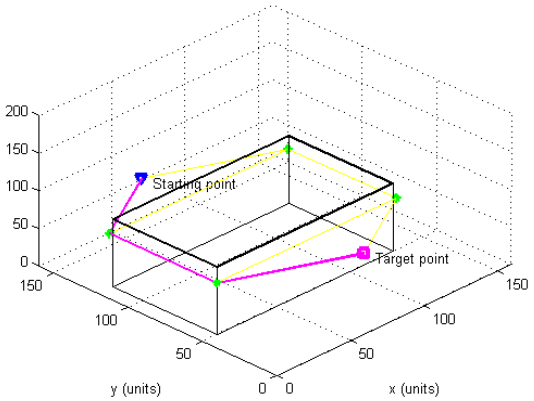


(c)

Figure 4.2: The visibility lines network; (a) top view, (b) side view, (c) 3D view.



(a)



(b)

Figure 4.3: The 2D path (solid magenta lines) in 3D scenario; (a) top view, (b) 3D view.

4.2.2 p_{start} and p_{target} With Different Altitudes

Consider a 3D environment with an obstacle as illustrated in Fig. 4.4, in which the starting point and target point have different altitudes.

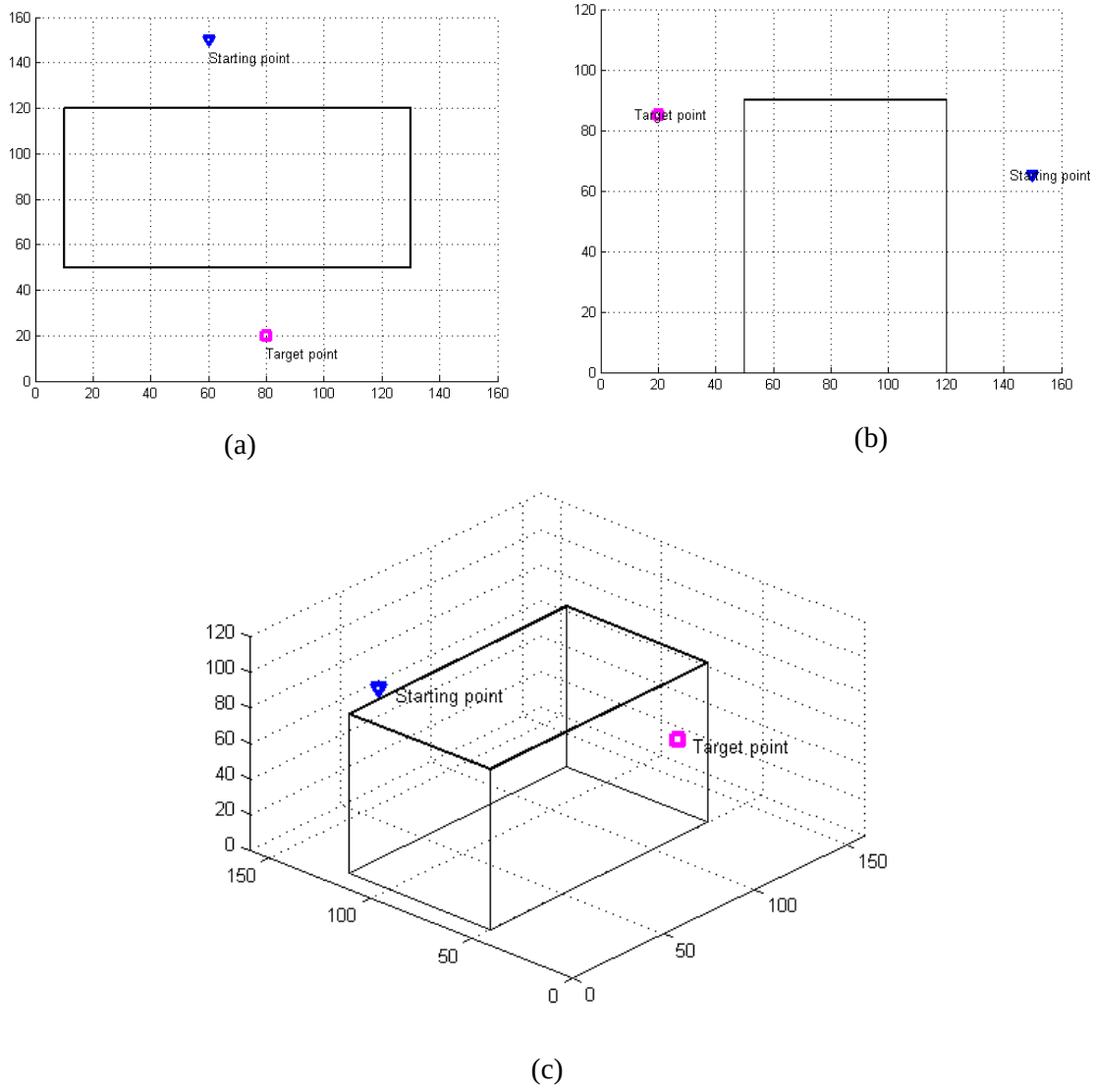
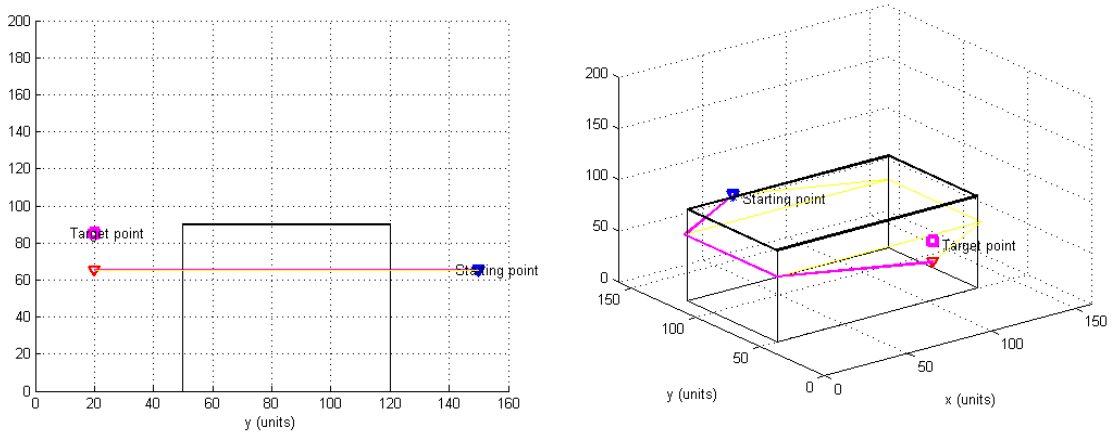


Figure 4.4: A 3D environment with a 3D obstacle. (a) top view, (b) side view, (c) 3D view.

To find a path in such an environment using the project point, consider the following cases:

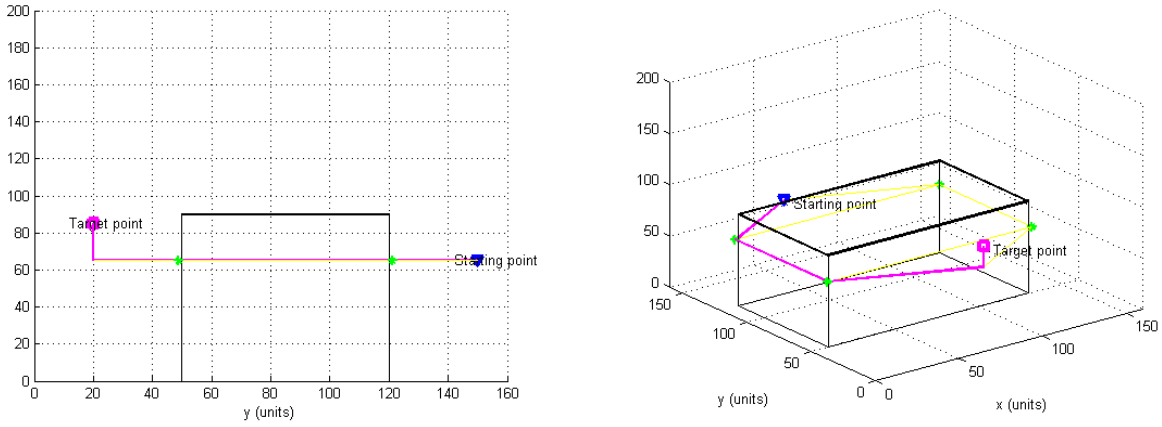
Case 1

In this case, a project point p_{pt} with respect to p_{target} is first identified. Using BLOVL, a path from p_{start} to p_{pt} is then calculated using BLOVL. The path is shown in Fig. 4.5. To complete the path planning process, p_{pt} is then connected to p_{target}. The complete path is shown in Fig. 4.6.



(a) (b)

Figure 4.5: A horizontal path connecting the starting point and the project point p_{pt}, which is represented by the red triangle. (a) side view, (b) 3D view.

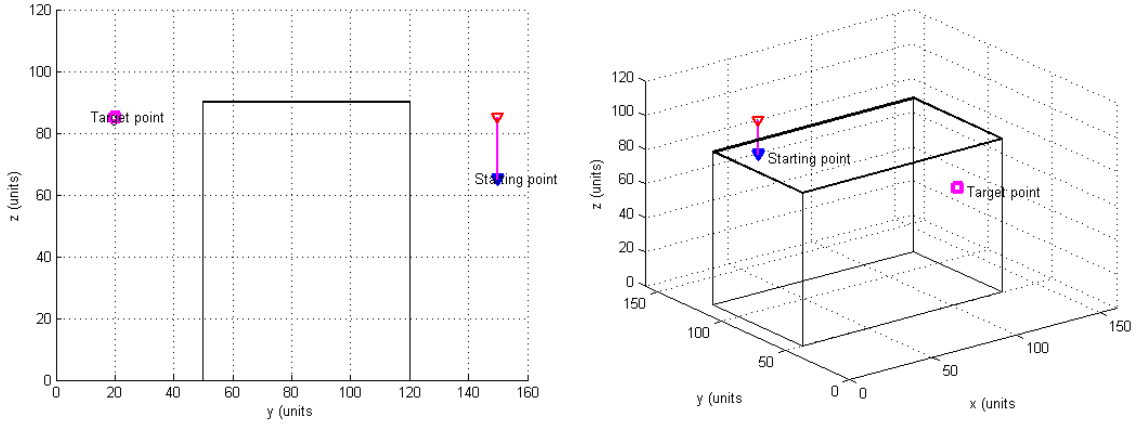


(a) (b)

Figure 4.6: A path that has been planned in a 3D environment. (a) side view, (b) 3D view.

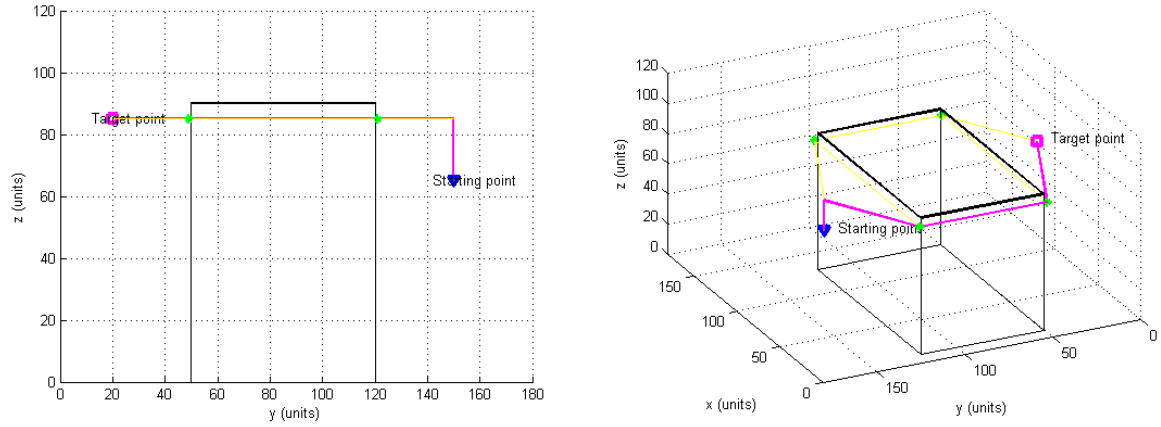
Case 2

This case is the dual of the previous case, in which a project point p_{ps} with respect to p_{start} is first defined. Then a vertical path connecting the starting point to p_{ps} is found as illustrated in Fig. 4.7. The subsequent step is to find a path from p_{ps} to p_{target}. The complete path is shown in Fig 4.8.



(a) (b)

Figure 4.7: A vertical path connecting p_{start} and p_{ps} (red triangle). (a) side view, (b) 3D view.



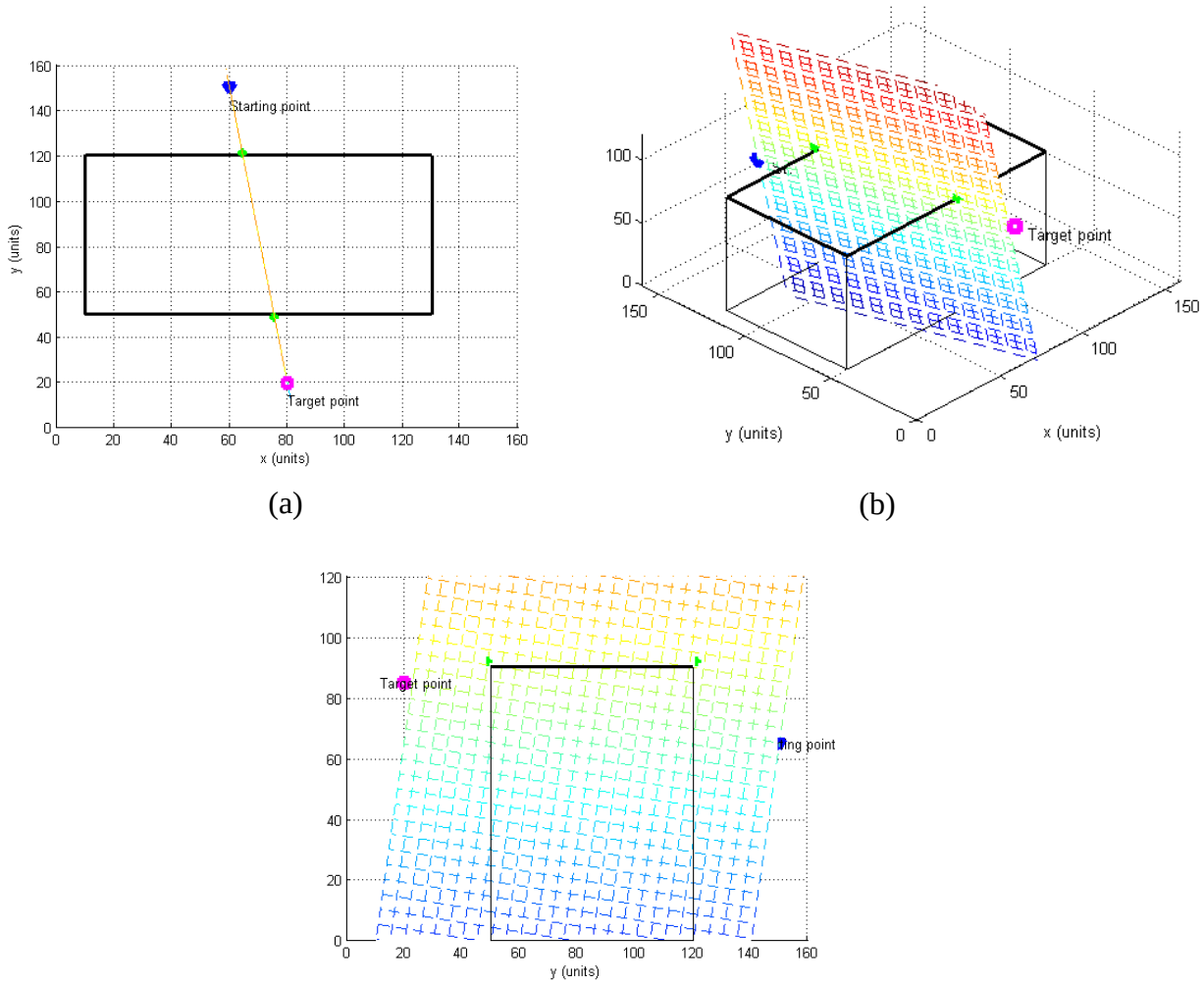
(a) (b)

Figure 4.8: A path (solid magenta lines) that has been planned in a 3D environment. (a) side view, (b) 3D view.

4.2.3 Find a Path on a Vertical Plane

The methods demonstrated in sub-sections 4.2.1-4.2.3 are quite straight forward in finding the 2D paths in the 3D environments. However, the paths are not feasible for a fixed-wing UAV as they consist of vertical segments. This sub-section will demonstrate how a 3D path is calculated on a vertical plane which results in a path with no vertical segments.

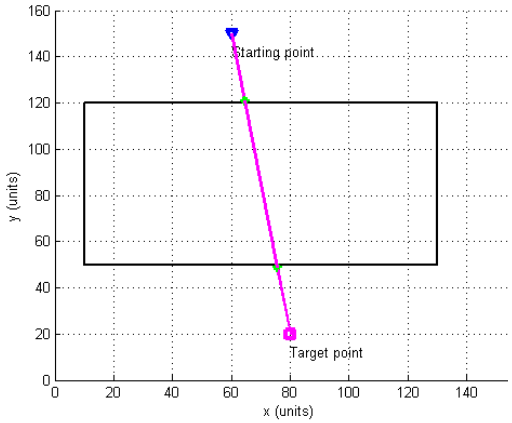
Consider the 3D scenario as shown in Fig. 4.4. A vertical plane is first created where the range of the plane is set to be between the starting point and the target point. The nodes used to create a VL network are subsequently determined from the intersections between the plane and the obstacle borders' edges. The resulting nodes are represented by the green dots in Fig. 4.9. Finally Dijkstra's algorithm is applied to find a path as depicted in Fig. 4.10.



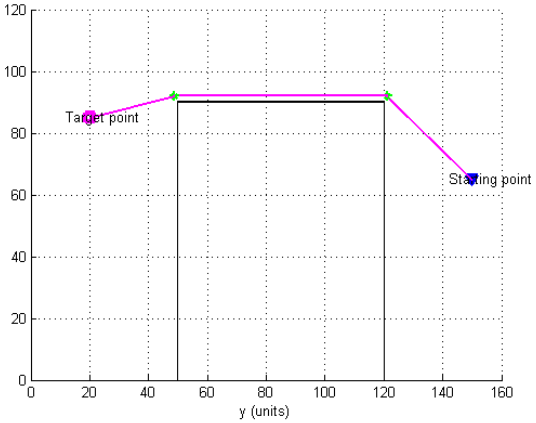
(a) (b)

(c)

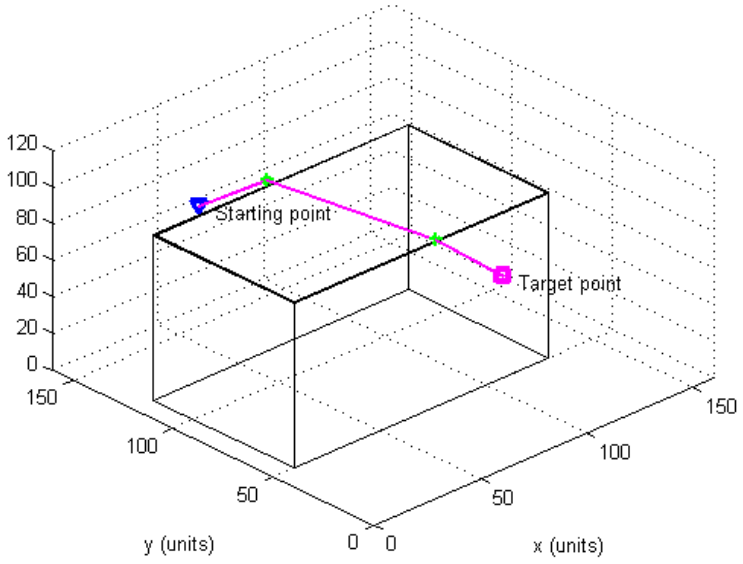
Figure 4.9: The vertical plane in which the green dots are the intersection points. (a) top view, (b) side view, (c) 3D view.



(a)



(b)



(c)

Figure 4.10: A 3D path found the vertical plane; (a) top view, b) side view, (c) 3D view.

4.3 Path on the Base Plane

The vertical plane, as has been briefly explained in the previous section, is efficient in determining a set of nodes, which are the intersections points between the plane and the borders' edges the 3D obstacle. The resulting nodes from the intersection between the plane and the obstacle are then used to create a VL network and, subsequently, a 3D path can be found from the network.

This section will further explain and demonstrate how a base plane is created and rotated at an arbitrary angle, and how the intersection points are determined. It also demonstrates a path planning process on the base plane.

4.3.1 Creating a Base Plane

In order to create a base plane, an algorithm called *BasePlane* is developed. The *BasePlane* algorithm, consists of three steps, is shown in Fig. 4.11.

Algorithm: *BasePlane*

1. Create a base line, BL_{3D} that connects u_{start} and u_{target}. Find the pitch angle β, which is formed by the BL_{3D} and the global horizontal (*X*-*Y*) plane, $Pxyu_{start}$.
2. Define a local plane $Px'y'u_{start}$, formed by the BL_{3D} and $BL\perp_{3D}$. The $BL\perp_{3D}$ is on $Pxyu_{start}$, orthogonal to BL_{3D} and passes u_{start}.
3. Define a local coordinate system on $Px'y'u_{start}$ with e u_{start} as the origin, $BL\perp_{3D}$ as the *X*-axis and BL_{3D} as the *Y*-axis of the $Px'y'u_{start}$.

Figure 4.11: The *BasePlane* algorithm

The *BasePlane* algorithm starts with creating a 3D base line, BL_{3D} ranging from a current starting point, u_{start} with coordinates $\left(u_{start}^{x}, u_{start}^{y}, u_{start}^{z}\right)$ to a target point, u_{target} with coordinates $\left(u_{target}^{x}, u_{target}^{y}, u_{target}^{z}\right)$. Unlike the base line (BL) of the *Core* algorithm, BL_{3D} of *BasePlane* considers the altitudes of u_{start} and u_{target}.

Then a pitch angle, β, which is the angle between BL_{3D} and the global horizontal plane (X-Y axis), $Pxyu_{start}$ can be found from the following equation:

$$\beta = arctan\left(\frac{\left(u^z_{target} - u^z_{start}\right)}{\sqrt{\left(u^y_{target} - u^y_{start}\right)^2 + \left(u^x_{target} - u^x_{start}\right)^2}^2}\right) \tag{4.1}$$

β is illustrated in Fig. 4.12(b). It is zero if the starting point and the target point have an equal altitude. β is important for a coordinate transformation of a local plane that will be shown later.

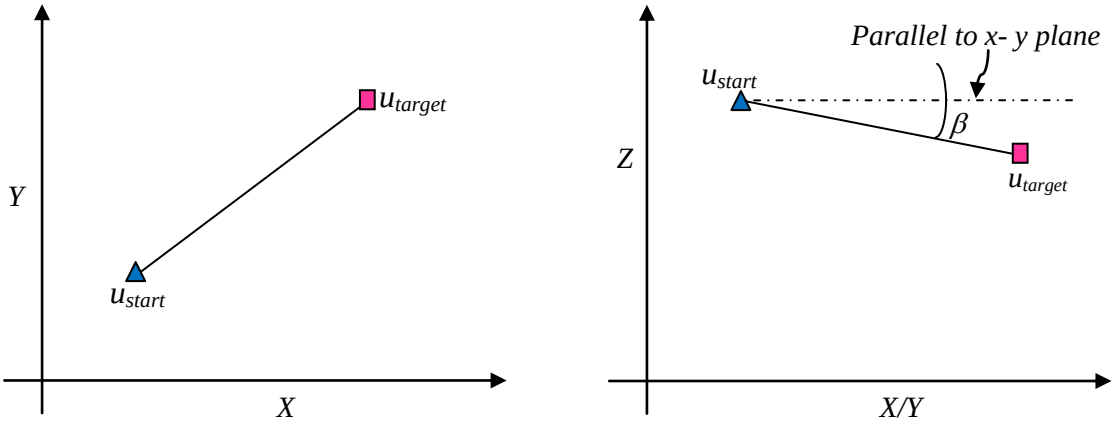


Figure 4.12: The illustration of β. (a) top view, (b) side view.

Next, a line perpendicular to BL_{3D} and intersects u_{start} called the $BL\perp_{3D}$ is defined, followed by the definition of a local plane called $Px'y'u_{start}$ in which the BL_{3D} and $BL\perp_{3D}$ are the local X- and Y-axes, respectively. The $Px'y'u_{start}$ resulted from the *BasePlane* algorithm is illustrated in Fig. 4.13. Note that the u_{start} is the origin of the $Px'y'u_{start}$.

In order to define the local coordinate system of $Px'y'u_{start}$ with respect to the Y-axis of $Pxyu_{start}$, the orientation γ (heading angle) of the BL_{3D} has to be known and can be calculated using

$$\gamma = arctan\left(\frac{u^y_{target} - u^y_{start}}{u^x_{target} - u^x_{start}}\right) - \pi/2 \qquad (4.2)$$

γ is illustrated in Fig. 4.14.

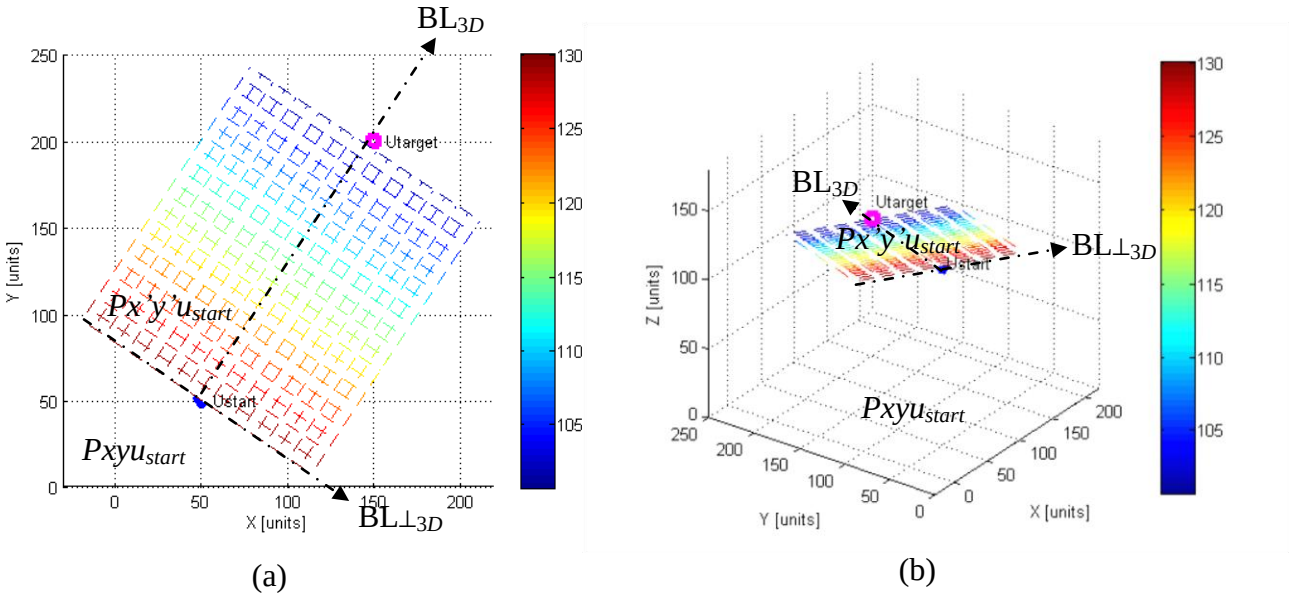


Figure 4.13: The *Px'y'u*$_{start}$ generated by the *BasePlane* algorithm. (a) top view, (b) 3D view

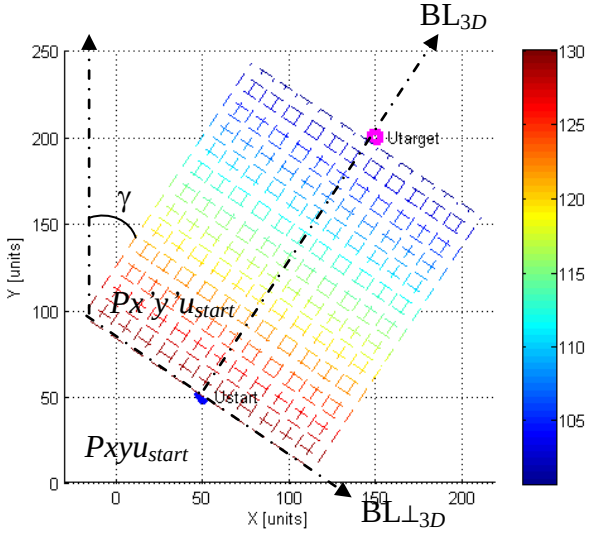


Figure 4.14: The orientation γ is the angle between the the *Y*-axis of *Pxyu*$_{start}$ and the BL$_{3D}$ of *Px'y'u*$_{start}$.

The coordinate transformation for $Px'y'u_{start}$ is then performed using the following steps:

1. Rotate the plane by -γ about the *Z*-axis of the translated $Pxyu_{start}$ using the matrix transformation R_1 below:

$$M_1 = R_1(Pxyu_{start} - u_{start}) \tag{4.3}$$

where

$$R_1 = \begin{bmatrix} \cos(-\gamma) & \sin(-\gamma) & 0 \\ -\sin(-\gamma) & \cos(-\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

2. Rotate the M_1 by -β about the *X*-axis of the global plane using the following the matrix transformation R_2:

$$M_3 = R_2 M_1 \tag{4.5}$$

where

$$R_2 = \begin{bmatrix} 1 & 0 & 0 \\ \cos(-\beta) & \sin(-\beta) & 0 \\ -sin(-\beta) & cos(-\beta) & 1 \end{bmatrix} \tag{4.6}$$

As a result, the relationship between the coordinates of $Px'y'u_{start}$ and the ones of $Pxyu_{start}$ can be established from

$$Px'y'u_{start} = M_3 = R_2R_1(Pxyu_{start} - u_{start}) \tag{4.7}$$

4.3.2 Finding Intersection Points

A VL-based path planning method requires a set of nodes to be defined prior to a path calculation. In a 2D path planning using VL-based method, the nodes are simply the corners of obstacles whose altitudes are ignored, and hence the nodes are finite in number. Unlike the 2D path planning, the number of nodes in a 3D path planning is infinite because 3D obstacles heights are taken into account. Thus the corners of such obstacle cannot be used for determining the nodes. In this thesis, it is proposed that the nodes are the intersection points between a plane and the 3D obstacles.

In this subsection, the *FindIntersection* algorithm whose purpose is to find those intersection points or nodes is introduced. The algorithm is shown in Fig. 4.15.

The *FindIntersection* algorithm first transforms the coordinates of each obstacle with respect to $Px'y'u_{start}$. The transformed coordinates O_T of the obstacles is obtained using

$$O_T = R_3 R_2 R_1 (O - u_{start}) \tag{4.8}$$

where O are the coordinates of the obstacles with respect to the global plane.

The next step of *FindIntersection* is to find pairs of borders edges of O_T within the environment, which has a range of $[X_{min}\ X_{max}\ Y_{min}\ Y_{max}]$. Each pair of border edge is a line segment that starts from p_a and ends at p_b.

Algorithm: *FindIntersection*

1. Transform the obstacles coordinates according to $Px'y'u_{start}$.
2. Identify all borders' edges of obstacles. Each edge consists of a pair of nodes, p_{a_i} and p_{b_i} where $i = 1 \ldots n$. n is the total number of the borders' edges.
3. Divide the local plane ($Px'y'u_{start}$) into two triangles. Each triangle is formed by three nodes, p_1, p_2 and p_3, which are determined based on the range of the plane.
4. Find the intersection point p_{int_i} between each triangle and edge i from

$$p_{int_i} = p_{a_i} + (p_{b_i} - p_{a_i})q \tag{4.9}$$

where

$$\begin{bmatrix} q \\ r \\ s \end{bmatrix} = \begin{bmatrix} p_{a_{i_x}} - p_{b_{i_x}} & p_{2_x} - p_{1_x} & p_{3_x} - p_{1_x} \\ p_{a_{i_y}} - p_{b_{i_y}} & p_{2_y} - p_{1_y} & p_{3_y} - p_{1_y} \\ p_{a_{i_z}} - p_{b_{i_z}} & p_{2_z} - p_z & p_{3_z} - p_{1_y} \end{bmatrix}^{-1} \begin{bmatrix} p_{a_{i_x}} - p_{1_x} \\ p_{a_{i_y}} - p_{1_y} \\ p_{a_{i_z}} - p_{1_z} \end{bmatrix} \tag{4.10}$$

Figure 4.15: The *FindIntersection* algorithm.

$Px'y'u_{start}$ is then divided into two triangles each consists of three nodes p_1, p_2 and p_3. These nodes are determined from the range of the plane, $[X'_{min}\ X'_{max}\ Y'_{min}\ Y'_{max}]$. Note that the coordinate $[X'_{min}\ Y'_{min}]$ is the origin of $Px'y'u_{start}$. The parameters r and s as stated in equation (4.10) indicate that the intersection point is on the line between p_a and p_b if $r + s \leq 1$.

The illustration of a line-plane intersection is shown in Fig. 4.16, in which the red dot is the intersection point between the $Px'y'u_{start}$ and the solid upright line. Note that the plane has been rotated at an arbitrary angle. Also note that the line can be one of the obstacles borders' edges. Readers are referred to [121] to find a further explanation about the line-plane intersection.

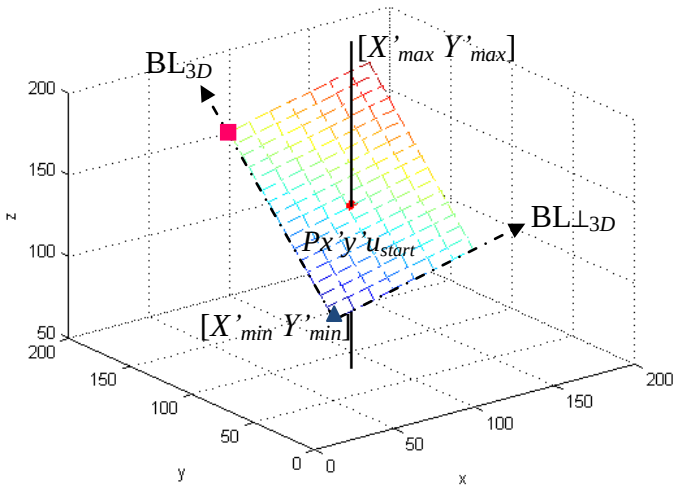


Figure 4.16: A line-plane intersection found using the *FindIntersection* algorithm.

In a real scenario, however, a 3D obstacle consists of several borders' edges (lines). For instance, a cuboid obstacle has eight (not including the edges at the base). Applying the *FindIntersection* algorithm in a scenario with a cuboid obstacle, the intersection points between $Px'y'u_{start}$ and the obstacle borders' edges can be determined and are shown in Fig. 4.17. The intersection points are represented by the red dots.

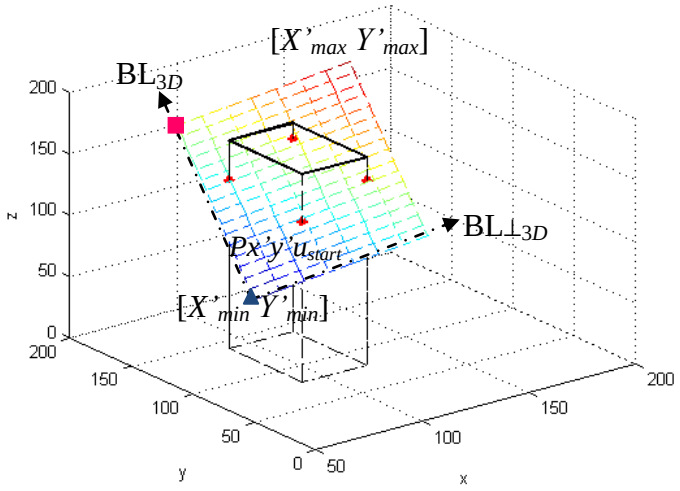


Figure 4.17: The borders' edges of a 3D obstacle intersect with a plane.

4.3.3 Finding a 3D Path on the Base Plane

This sub-section demonstrates a 3D path planning on the base plane that has been created by the *BasePlane* algorithm. Additionally, in order to ensure that the path on the plane, another algorithm called BLOVL3D1 is proposed. BLOVL3D1, which is based on BLOVL, is shown in Fig. 4.18. Note that the *FindIntersection* algorithm is embedded in BLOVL3D1 to ensure that the intersection points are only calculated between the base plane and certain obstacles to accelerate the computation time.

Algorithm: *BLOVL3D1*(u_{start}, u_{target})

1 set j = 0 and $w_j = u_{start}$
2 **while** $w_j \neq p_{target}$ **do**
3 set $v_{start} = w_j$ and $v_{target} = u_{target}$
4 Find the obstacles that lie on BL_{3D} and their extension.
5 Call *FindIntersection* to define a set of nodes, Ns_{3D}.
6 Create a cost matrix, Cm_{3D} from Ns_{3D}.
7 Find a 3D path $V=\{v_0,\ldots,v_m\}$ from Cm_{3D} using Dijkstra's algorithm.
8 **If** m = 1 then
9 set $w_{j+1}=v_1$
10 **else**
11 set $v_{start} = v_0$; $v_{target} = v_1$
12 goto line 4
13 **end if**
14 set $j = j+1$
15 **end while**

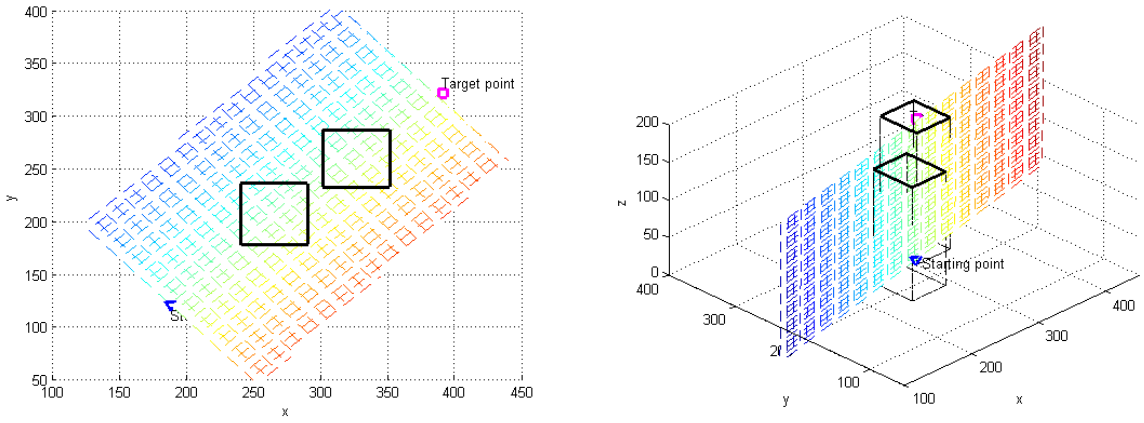
Figure 4.18: The BLOVL3D1 algorithm

Like BLOVL, the main idea of BLOVL3D1 is to perform a path planning process on a plane iteratively on a plane. It means that, while the target point has not been reached, the algorithm will keep planning a path on the plane. To ensure that the algorithms possess the completeness criterion, at each repetition, the starting point is updated to the second waypoint of the previously planned path until the starting point is the target point. By this way, it is guaranteed that the algorithm is able to find a path if one exists.

In order to reduce the computation time, the path finding process of BLOVL3D1 considers only the obstacles that lie on the base line BL_{3D} and their extension. Note that BL_{3D} is a line that connects p_{start} and p_{target}. Unlike BL of BLOVL, BL_{3D} considers the altitudes of p_{start} and p_{target}.

Furthermore, *m*, which is the number of segments between two consecutive waypoints, plays an important role in determining whether a planned path is collision-free or not. A value of *m* that larger than one indicates that the path between two consecutive waypoints is being blocked. In this case, a collision-free path has to be calculated.

To demonstrate the BLOVL3D1 algorithm in finding a path on a plane, consider an environment with a pair of 3D cuboids obstacles as illustrated in Fig. 4.19. The plane $Px'y'u_{start}$ that has been created by *BasePlane* with an arbitrary angle, on which a path will be calculated, is also shown in the figure.



(a) (b)

Figure 4.19: A scenario with two 3D obstacles with a plane; (a) top view, (b) 3D view.

The BLOVL3D1 algorithm first determines the obstacles that lie on BL_{3D} and their extensions. It then calls the *FindIntersection* algorithm to find the intersection points between the plane and obstacle borders' edges.

The *FindIntersection* algorithm first transforms the obstacles coordinate system with respect to the $Px'y'u_{start}$, and subsequently, determines the intersection points, Ns_{3D} between the plane and the obstacles borders' edges. The transformed local plane, the obstacles and the intersections points are depicted in Fig. 4.20. In the figure, the shape of the obstacles has been sheared as it is projected orthogonally towards the plane. Note that a convex-shaped obstacle will remain convex after a projection is performed.

Now the path planning problem has been reduced from planning in a 3D environment into in a 2D one i.e. on the $Px'y'u_{start}$. Following the next step of BLOVL3D1, a cost matrix, Cm_{3D} is then created based on Ns_{3D}. The visibility lines network on the plane can now be created from Cm_{3D} using the VL algorithm. The network is illustrated in Fig. 4.21. Dijkstra's algorithm is then applied to calculate a path on the plane. The resulting path, which is in magenta, is depicted in Fig. 4.22.

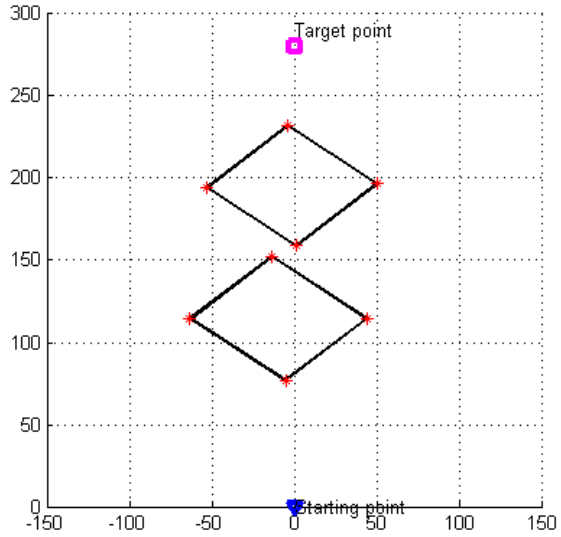


Figure 4.20: The transformed local plane ($Px'y'u_{start}$) and the intersection points shown by the red dots.

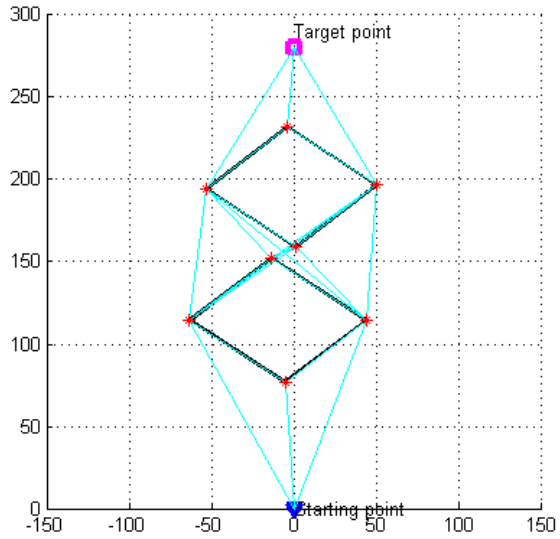


Figure 4.21: The visibility lines network is represented by the cyan lines.

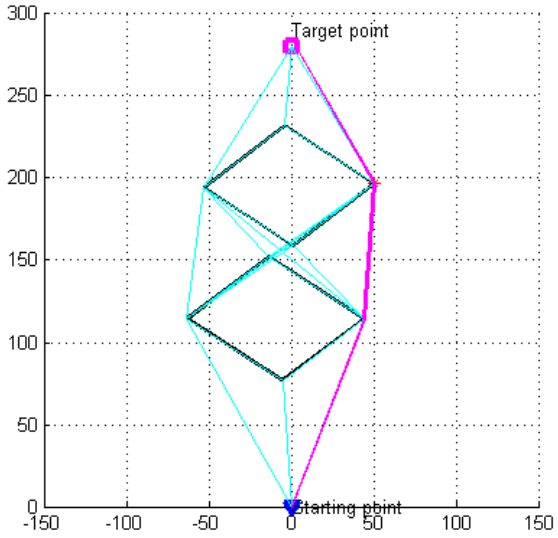


Figure 4.22: The path (magenta lines) on the plane found using Dijkstra's algorithm.

4.4 BLOVL 3D Algorithms

The *BasePlane* and the *FindIntersection* algorithms that have been proposed and demonstrated in the previous section are necessary to create a plane, $Px'y'u_{start}$ and find intersection points between $Px'y'u_{start}$ and obstacles, respectively. To find a path on the plane from the intersection point, the BLOVL3D1 algorithm has been proposed.

In this section, a 3D path planning algorithm that combines and governs the aforementioned algorithms is proposed. The proposed 3D path planning algorithm is called BLOVL3D2 and is shown in Fig. 4.23. The process of BLOVL3D2 is depicted in Fig. 4.24.

The algorithm starts with initialising the necessary parameters i.e. $k = 0$, $Ps_k = p_{start}$ and $i = 0$. Ps is the global path consisting of waypoints, which is updated during the path planning process. The final value of k determines the number of waypoints in Ps. i represents the index of rotation angles and $\boldsymbol{\alpha}$ is the vector that contains b rotation angles.

Algorithm: ***BLOVL3D2***

1: Set k=0, Ps_k=p_{start} and i=0
2: Define the rotational angle vector, $\boldsymbol{\alpha}$=$\{\alpha_1,\ldots,\alpha_b\}$
3: **while** $Ps_k \neq p_{target}$ **do**
4: u_{start}= Ps_k; u_{target}=p_{target}
5: call *BasePlane* to generate a local plane $Px'y'u_{start}$
6: **while** $i \neq b$+1 **do**
8: call *BLOVL3D1*
9: Save waypoints $W^{\alpha i}$ into $\boldsymbol{W}^{\boldsymbol{\alpha}}$
10: Increase i by 1.
11: Rotate $P\ x'y'u_{start}$ by α_i degree using $Rotate_{3D}$.
12: **end while**
13: Compare all paths in $\boldsymbol{W}^{\boldsymbol{\alpha}}$ and find the shortest, Ws. Ws=$\{w_{s0},\ldots w_{sn}\}$
14 Transform Ws with respect to the global coordinate system.
15: Increase k by 1 and update Ps_k =w_{s1}. Set i to 0.
16: **end while**

Figure 4.23: The BLOVL3D2 algorithm

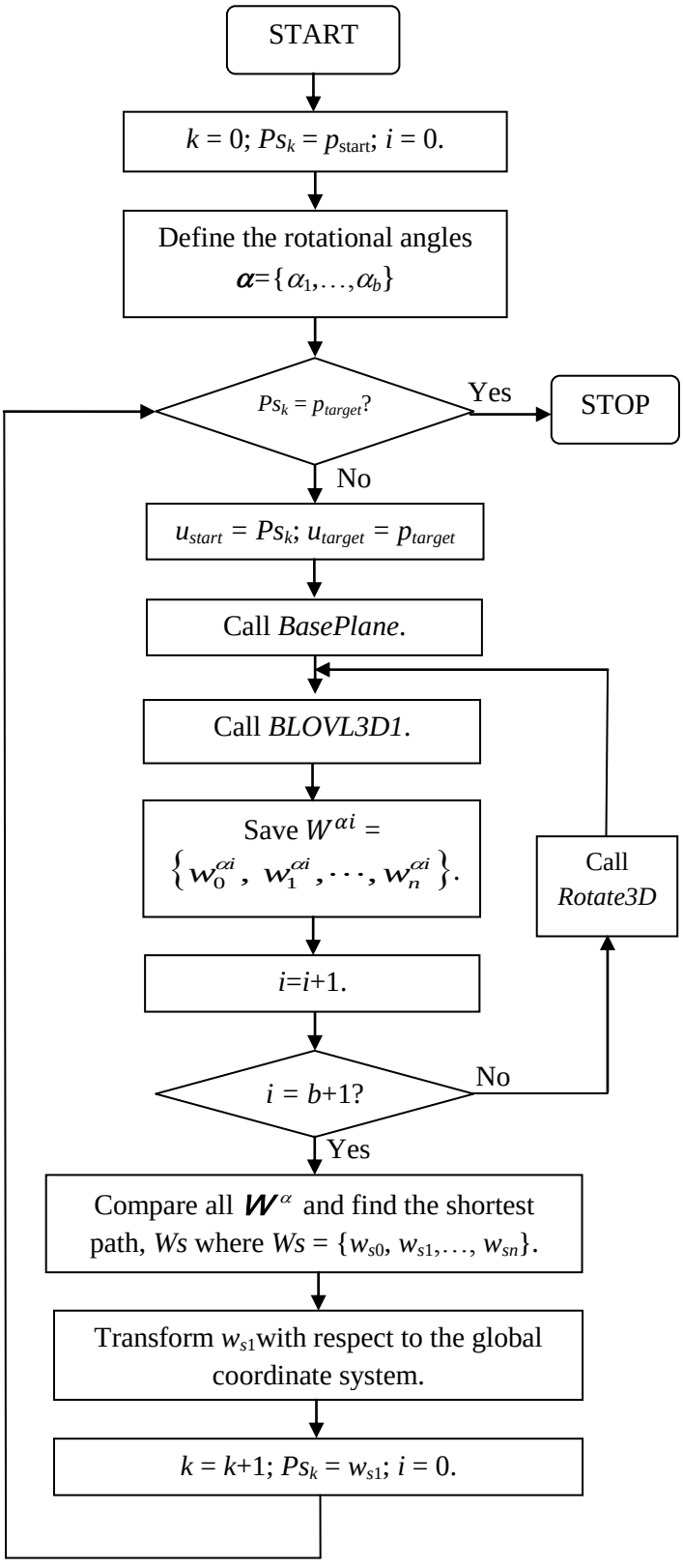


Figure 4.24: The BLOVL3D2 process

In the next step of BLOVL3D2, Ps_k is compared with p_{target}. If Ps_k is equal to p_{target}, the path planning process is stopped. Otherwise, Ps_k is set to u_{start} while u_{target} is set to p_{target}. Note that the u_{start} and u_{target} are the necessary inputs for BLOVL3D1 that is embedded in the algorithm.

After all the necessary parameters have been initialised, BLOVL3D2 calls the *BasePlane* algorithm to generate a local plane, $Px'y'_{ustart}$ ranging from u_{start} to u_{target}. At this point, if the index i does not exceed b, the BLOVL3D1 algorithm will carry on finding a path on $Px'y'_{ustart}$. The resulting waypoints $W^{\alpha i}$ are then saved in $\boldsymbol{W^{\alpha}}$.

Subsequently, i is increased by 1 and $Px'y'_{ustart}$ is rotated by α_i using the $Rotate_{3D}$ algorithm. The *Rotate3D* algorithm will be introduced and demonstrated in the following sub-section. BLOVL3D1 is called again to find a path $W^{\alpha i}$ on the rotated $Px'y'_{ustart}$. At each rotation of $Px'y'_{ustart}$, the resulting $W^{\alpha i}$ calculated by BLOVL3D1 is saved in $\boldsymbol{W^{\alpha}}$. After $Px'y'_{ustart}$ has been rotated by all the angles, the waypoints with the shortest path, *Ws* are then selected from $\boldsymbol{W^{\alpha}}$. *Ws* consists of waypoints w_{s0}, w_{s1}, ..., w_{sn} where n+1 is the number of waypoints in *Ws*.

Next, k and Ps_k are updated to k+1 and the second waypoint of *Ws*, i.e. w_{s1}, respectively. Consequently u_{start} is set to Ps_k and u_{target} is to p_{target}. The next iteration is then started, in which a path is calculated on a new $Px'y'_{ustart}$ ranged from u_{start} to u_{target}. The above steps are kept repeated until Ps_k is p_{target} to meet the completeness criterion.

4.4.1 Rotating a Base Plane

Finding a 3D path on $Px'y'u_{start}$ solves the problem of path planning in 3D environment with 3D obstacles. However, the path on $Px'y'u_{start}$ may not be the shortest, especially in an obstacle-rich environment. In such an environment, a shorter path may results if the path goes over the obstacles. This in turn needs the plane to be rotated. An example of a path planning with a plane rotated at 90 degrees (and hence the plane is vertical) was demonstrated in Section 4.2.3. Thus in this section, the *Rotate*3*D* algorithm is introduced and demonstrated. The purpose of this algorithm is to rotate a base plane at a particular angle. The algorithm is shown in Fig. 4.25.

Algorithm: *Rotate$_{3D}$*

1. Rotate the plane *Px'y'u$_{start}$* by α degree about BL$_{3D}$.
2. Define the coordinate transformation of the *Px'y'u$_{start}$* plane with respect to the global plane.

Figure 4.25: The Rotate$_{3D}$ algorithm

According to *Rotate$_{3D}$*, it first rotates the *Px'y'u$_{start}$* plane by α degree about the BL$_{3D}$ axis and the plane coordinate is then transformed accordingly. Note that α is an angle between 0 and 180 degrees. The transformation is done based on the rotation angle α, the orientation or heading angle γ and the pitch angle β of *Px'y'u$_{start}$*. (γ and β have been defined in Section 4.3.1). The plane rotation and subsequently its coordinate transformation of *Px'y'u$_{start}$* are performed and updated using the aforementioned steps with an additional one,

$$M_4 = R_3 Px'y'u_{start} \tag{4.11}$$

where

$$R_3 = \begin{bmatrix} \cos(\alpha) & 0 & -sin(\alpha) \\ 0 & 1 & 0 \\ sin(\alpha) & cos(\alpha) & 1 \end{bmatrix}$$

The relationship between the rotated plane $Px'y'u_{start}$ and the global one Pxy_{ustart} is therefore

$$Px'y'_{ustart} = M_4 = R_3 R_2 R_1 (Px'y'u_{start} \quad - u_{start}) \tag{4.12}$$

To demonstrate how a rotation of a base plane is performed, consider the base plane that has been created using the *BasePlane* algorithm as illustrated in Fig. 4.26. In the figure, the blue triangle represents the starting point while the magenta square is the target. The plane will be rotated by 30 degrees about the BL$_{3D}$ line. The resulting rotated plane is shown in Fig. 4.27.

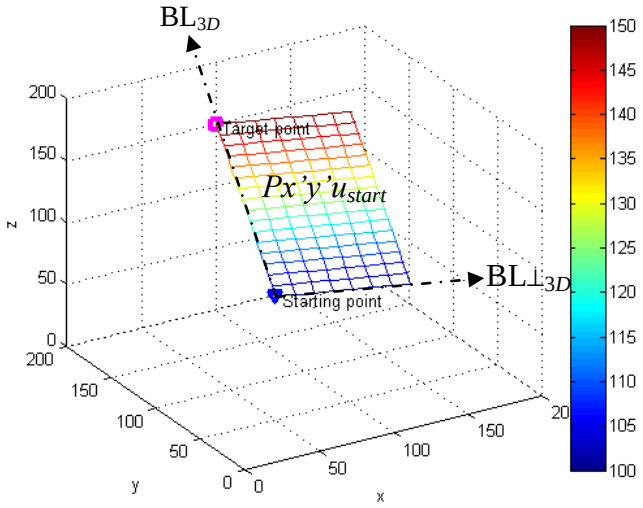


Figure 4.26: A local plane $Px'y'u_{start}$ to be rotated by 30 degrees about BL_{3D}.

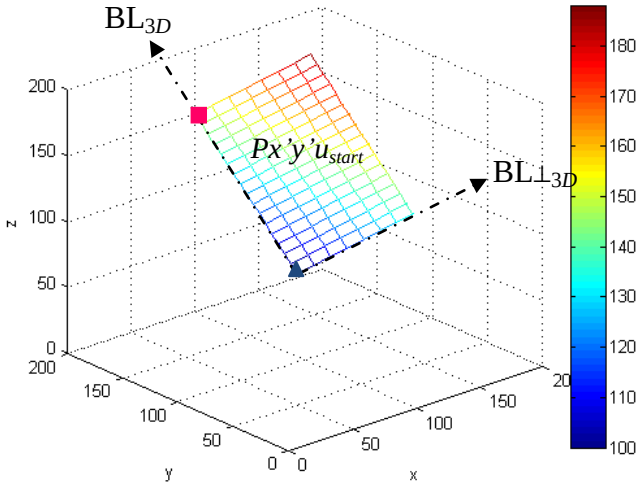


Figure 4.27: The plane in Fig. 4.16 is rotated about the BL_{3D} line.

Note that the BL_{3D} is chosen as the axis of rotation of the plane because this will guide the path towards the target point in order to reduce the path length.

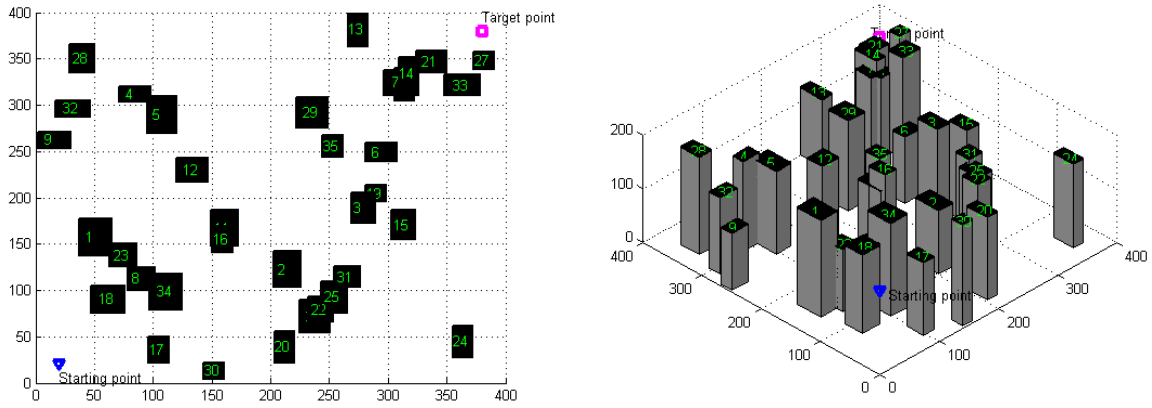
4.4.2 3D Path Planning Using BLOVL3D2

To demonstrate the 3D path planning using BLOVL3D2, consider a scenario with 35 randomly generated cuboids obstacles as shown in Fig. 4.28. The range of the scenario is $[400 \times 400]$ and the coordinate of p_{start} is set to (20,20,130) while the p_{target} is (380,380,160).

The rotational angle vector is assumed to be α = {0,15,30,45,60,75,90,105,120,135,150,165}. Notice that no angle greater than 180 degrees is being used because, a plane that is rotated by α_i degree is identical to a plane rotated by $\alpha_i + 180$.

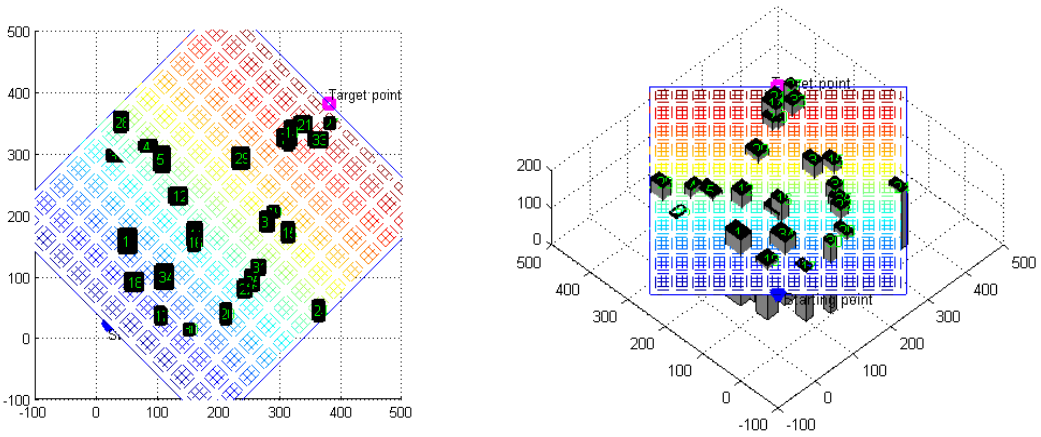
In order to find a 3D path using the BLOVL3D2 algorithm, a number of iterations have to be performed until the current starting point is the target point.

In the first iteration, a plane $Px'y'u_{start}$ is first generated between u_{start} to u_{target} using the *BasePlane* algorithm. The plane is shown in Fig. 4.29 from two different viewpoints.



(a) (b)

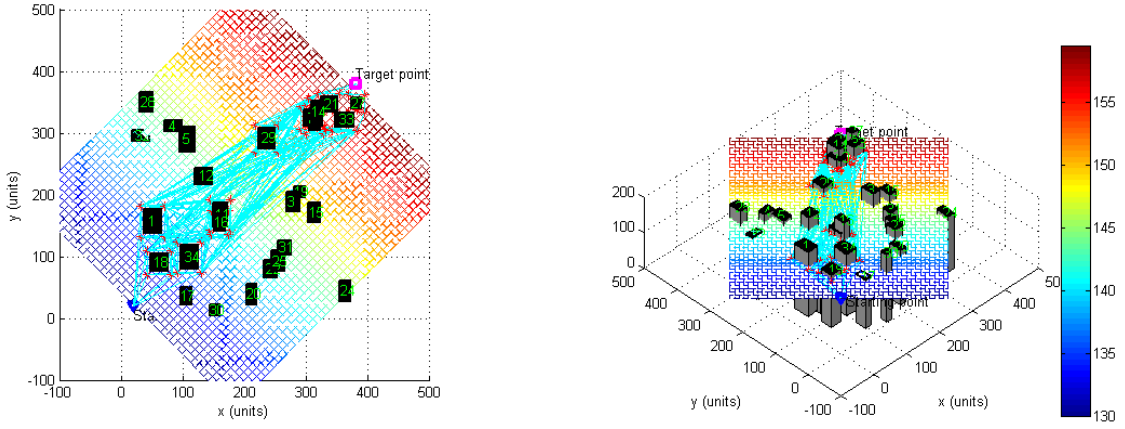
Figure 4.28: A 3D scenario with 35 obstacles. (a) top view, (b) 3D view.



(a) (b)

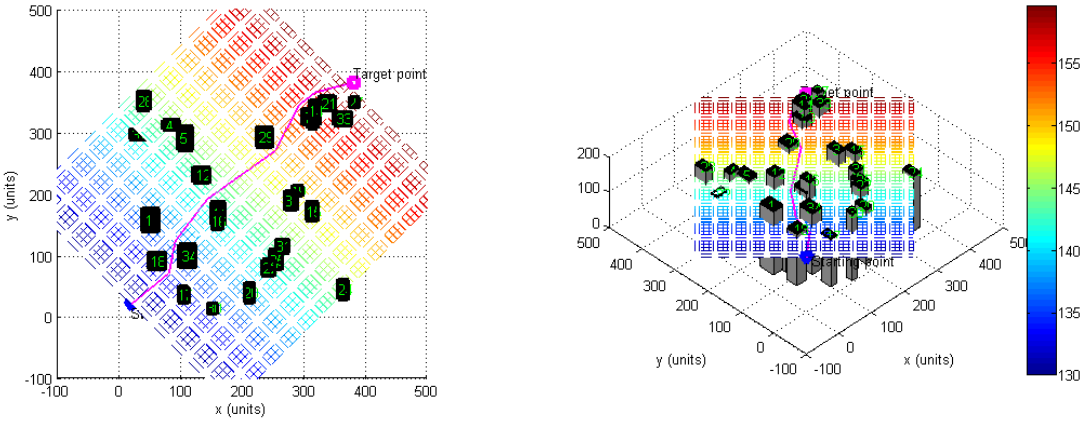
Figure 4.29: The $Px'y'u_{start}$ plane generated by *BasePlane*. (a) top view, (b) 3D view

The BLOVL3D1 algorithm is then called to find a set of nodes, Ns_{3D} which are the intersection points between the plane and the obstacles borders' edges through the *FindIntersection* algorithm. From Ns_{3D}, a visibility lines network is created. The nodes and the visibility lines network are shown in Fig. 4.30. Note that the red dots in Fig. 4.30 represent the Ns_{3D} and the VL network is represented by the cyan lines. Subsequently BLOVL3D1 finds an optimal path on the plane, $W^{\alpha i}$ (where i=0 at this point) containing a set of waypoints. The path, which is represented by the solid magenta lines, is illustrated in Fig. 4.31. $W^{\alpha 0}$, which has been transformed into the global plane coordinate system is listed in Table 4.1.



(a) (b)

Figure 4.30: The nodes (red dots) obtained by the *FindIntersection* algorithm and the visibility lines network (cyan lines) generated by the BLOVL3D1; (a) top view, (b) 3D view.



(a) (b)

Figure 4.31: A path on $Px'y'u_{start}$ represented by the magenta segments found by BLOVL3D1; (a) top view, (b) 3D view.

Table 4.1: Waypoints generated by BLOVL3D1 at 0 degree.

$W^{\alpha 0}$	X	Y	Z
$w_0^{\alpha 0}$	20.00	20.00	130.00
$w_1^{\alpha 0}$	81.23	70.77	134.67
$w_2^{\alpha 0}$	91.77	123.23	137.29
$w_3^{\alpha 0}$	143.77	191.23	142.29
$w_4^{\alpha 0}$	254.23	269.77	150.17
$w_5^{\alpha 0}$	291.77	343.23	154.79
$w_6^{\alpha 0}$	318.77	364.23	156.79
$w_7^{\alpha 0}$	380.00	380.00	160.00

The next step of BLOVL3D2 is to update the index *i* to 1, and subsequently rotate the plane by α_1 = 15 degrees using *Rotate3D* as illustrated in Fig. 4.32. BLOVL3D1 is again applied to determine a set of nodes from the intersection between the obstacles borders' edges and the rotated plane and subsequently generates a visibility lines network as illustrated in Fig. 4.33. BLOVL3D1 then calculates a shortest path on the plane, $W^{\alpha 1}$ as shown in Fig. 4.34. The waypoints of the path, which have been transformed according to the global plane coordinate system, are recorded in Table 4.2

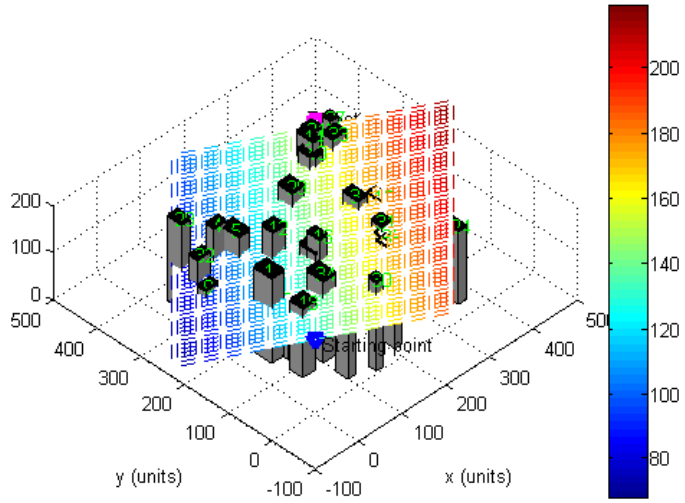


Figure 4.32: The plane is rotated by 15 degree.

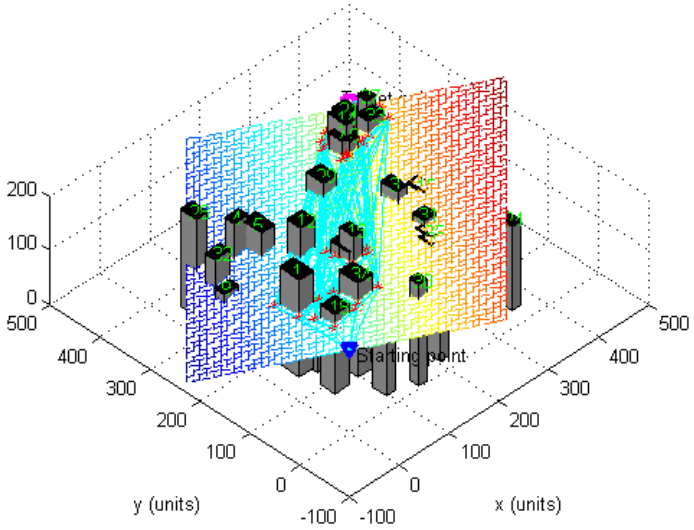


Figure 4.33: The nodes and visibility lines network of the plane rotated by 15 degree.

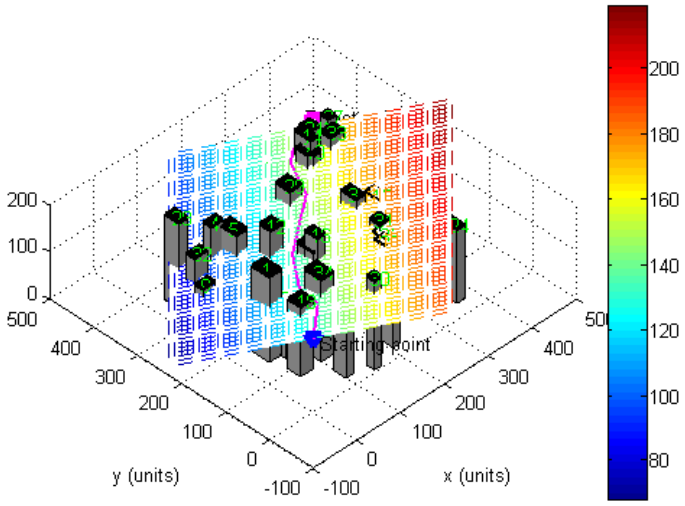


Figure 4.34: The shortest path on plane rotated by 15 degrees.

Table 4.2: The waypoints generated by BLOVL at 15 degrees rotation angle.

$W^{\alpha 1}$	X	Y	Z
$w_0^{\alpha 1}$	20.00	20.00	130.00
$w_1^{\alpha 1}$	80.10	71.90	136.22
$w_2^{\alpha 1}$	92.90	122.10	131.75
$w_3^{\alpha 1}$	144.90	190.10	133.71
$w_4^{\alpha 1}$	253.10	270.90	146.79
$w_5^{\alpha 1}$	292.90	342.10	145.45
$w_6^{\alpha 1}$	304.90	356.10	146.16
$w_7^{\alpha 1}$	319.90	363.10	148.59
$w_8^{\alpha 1}$	380.00	380.00	160.00

In the current iteration, the above-mentioned steps are kept repeated until the plane is rotated by all the pre-defined angles as shown by Fig. 4.35. As a result, the shortest path, $Ws = \{w_{s0},\ldots, w_{s1},\ldots, w_{sn}\}$ is found when the plane is rotated by 150 degrees. The path is illustrated in Fig. 4.36 and its waypoints, which have been transformed into the global coordinate system, are listed in Table 4.3.

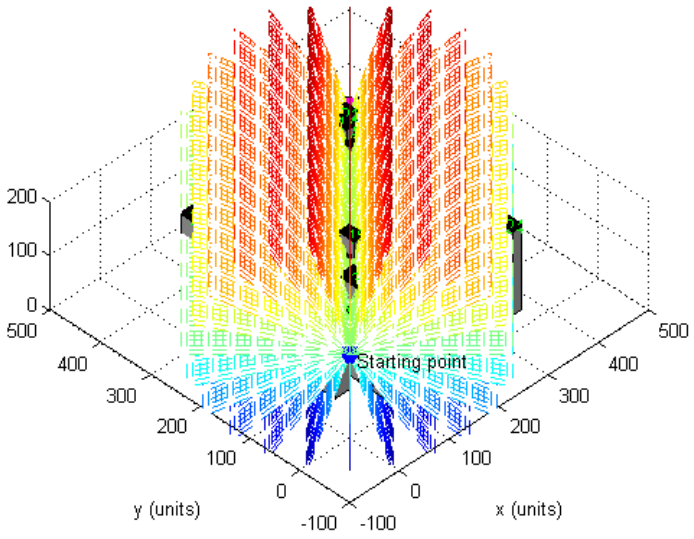
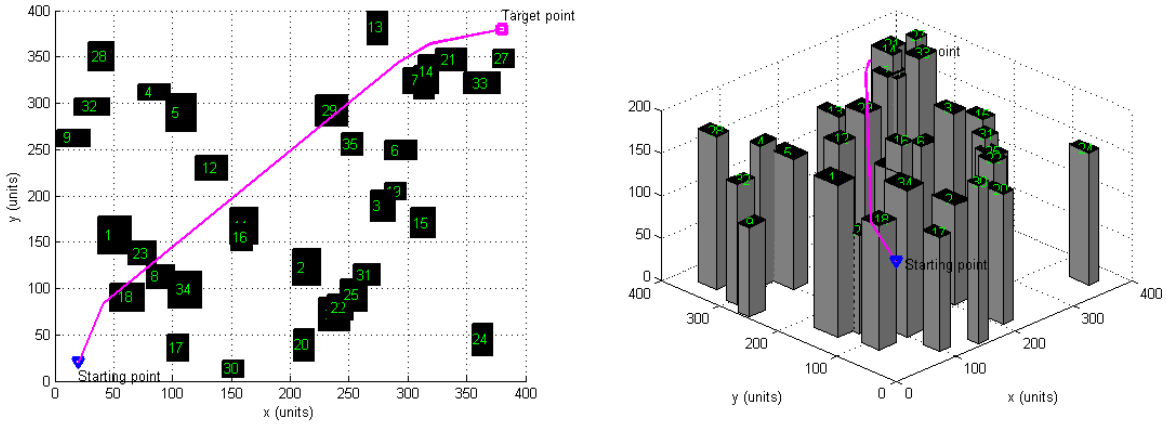


Figure 4.35: The plane rotated at {0:15:165) degrees to find a 3D path from u_{start} to u_{target}.

Table 4.3: The waypoints generated by BLOVL3D2 in the first iteration.

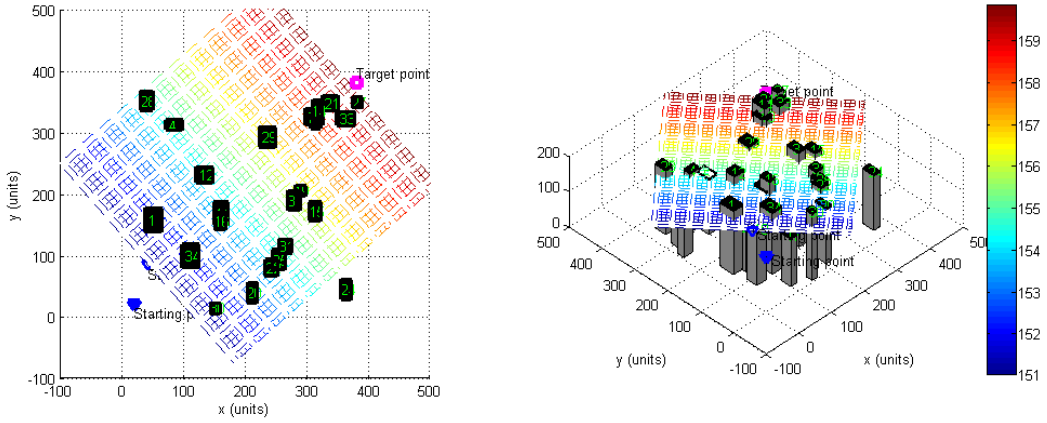
W_s	X	Y	Z
w_{s0}	20.00	20.00	130.00
w_{s1}	42.90	85.26	151.00
w_{s2}	304.90	356.10	175.84
w_{s3}	319.90	363.10	175.38
w_{s4}	380.00	380.00	160.00



(a) (b)

Figure 4.36: The path obtained by BLOVL3D2 in the first iteration. (a) top view, (b) 3D view.

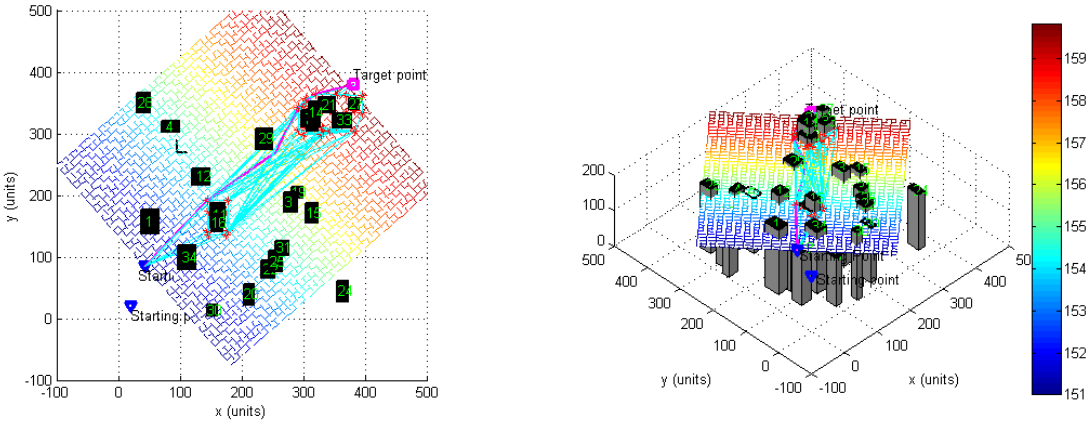
In the next iteration, the necessary parameters have to be updated and re-initialised. The value of k is updated to k+1 (in this example k now becomes 1) and accordingly, Ps_1 is set to the second point of the previous shortest path i.e. w_{s1}. As listed in Table 4.3, w_{s1} = (42.90, 85.26, 151.00). Concurrently, u_{start} is set to Ps_1 and u_{target} is to p_{target}. Also i is re-initialised to 0. A new plane ranging from the new u_{start} to p_{target} is then generated by *BasePlane*. The plane is shown in Fig. 4.37.



(a) (b)

Figure 4.37: The plane generated by *BasePlane* from the second waypoint of the previous shortest path to p_{target} (a) top view, (b) 3D view.

Subsequently, Ns_{3D}, which is a set of nodes, resulted from the intersection points between the plane and the obstacles borders' edges is determined. This is followed by creating a VL network and finding a path on the plane using BLOVL3D1. The Ns_{3D}, the visibility lines network and the path on the plane are depicted in Fig. 4.38 while the path waypoints are recorded in Table 4.4.



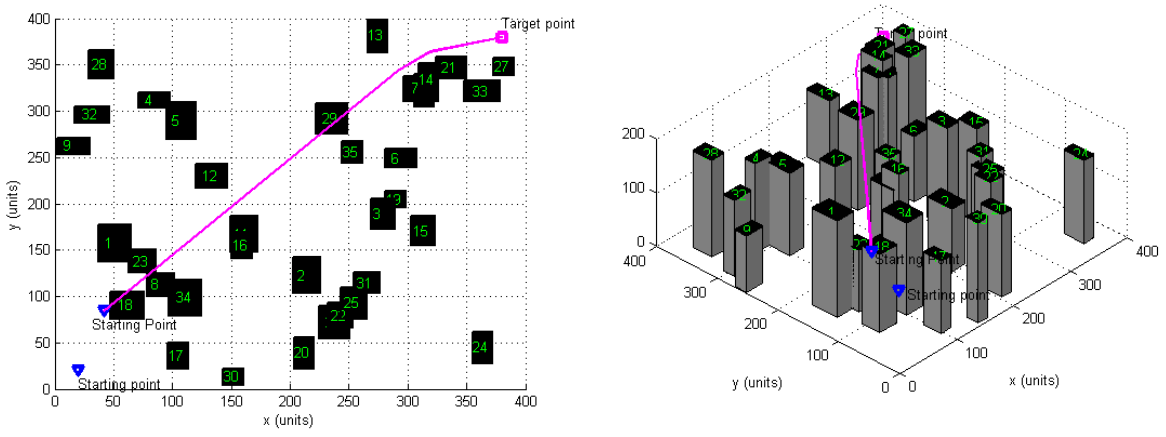
(a) (b)

Figure 4.38: The nodes, the visibility lines network and the path at 0 rotation angle of second repetition (a) top view, (b) 3D view.

Table 4.4: The waypoints generated by BLOVL3D2 in the second iteration at 0 rotation angle.

$W^{\alpha 2}$	X	Y	Z
$w_0^{\alpha 2}$	42.90	85.27	151.00
$w_1^{\alpha 2}$	144.90	191.23	153.95
$w_2^{\alpha 2}$	253.10	269.77	156.65
$w_3^{\alpha 2}$	292.90	343.23	158.18
$w_4^{\alpha 2}$	304.90	356.10	158.55
$w_5^{\alpha 2}$	319.90	363.10	158.87
$w_6^{\alpha 2}$	380.00	380.00	160.00

In the next steps of BLOVL3D2, the plane is rotated by all the angles contained in $\boldsymbol{\alpha}$, where at each rotation a path is calculated and is saved in $\boldsymbol{W^{\alpha}}$, followed by the determination of the shortest path from $\boldsymbol{W^{\alpha}}$. In this iteration, the path on the plane rotated by 150 degrees is the shortest with a length of 458.47 units. The path is illustrated in Fig. 4.39 and its waypoints are listed in Table 4.5.



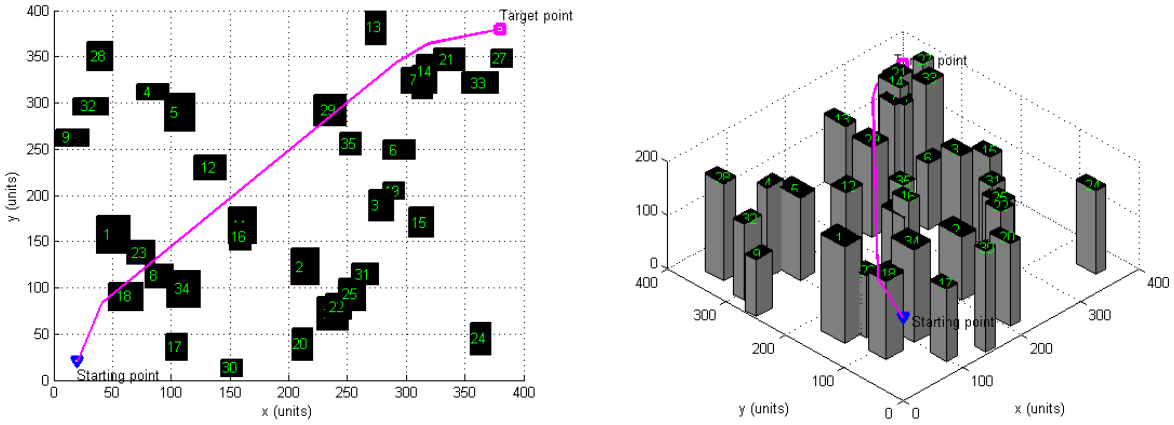
(a) (b)

Figure 4.39: The path obtained by BLOVL3D2 in the second iteration. (a) top view, (b) 3D view.

Table 4.5: The waypoints generated by BLOVL3D2 in the second iteration at 150 rotation angle.

W_s	X	Y	Z
w_{s0}	42.90	85.27	151.00
ws_1	304.90	356.10	176.70
ws_2	319.90	363.10	174.36
ws_3	380.00	380.00	160.00

The above processes are further executed until Ps_k is the p_{target}. The resulting path is shown in Fig. 4.40 and the waypoints of the path, *Ps* are recorded in Table 4.6.



(a) (b)

Figure 4.40: The resulted path (solid magenta line) planned by BLOVL3D2. (a) top view, (b) side view, (c) 3D view.

Table 4.6: The waypoints of the path shown in Fig. 4.40 generated by BLOVL3D2.

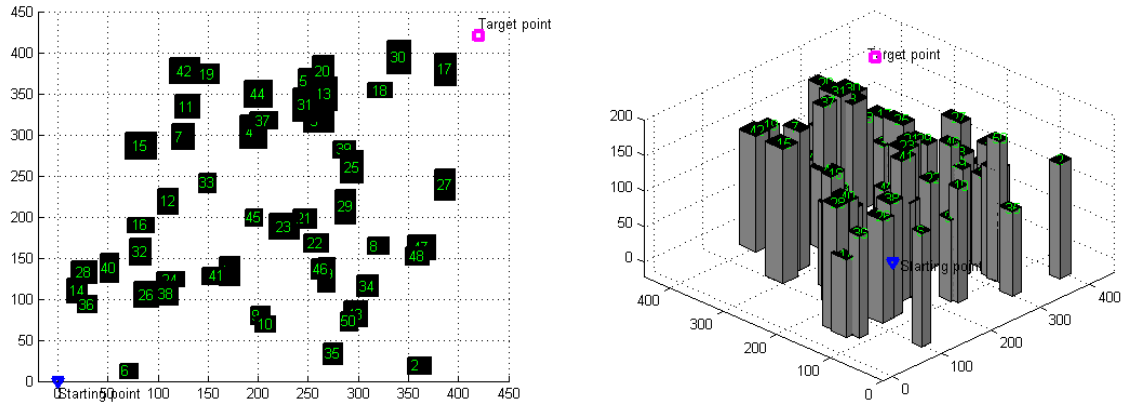
Ps	***X***	***Y***	***Z***
Ps_0	20.00	20.00	130.00
Ps_1	42.90	85.27	151.00
Ps_2	304.90	356.10	176.70
Ps_3	319.90	363.10	173.22
Ps_4	380.00	380.00	160.00

4.4.3 BLOVL3D2 Performances

In this section, the performance of BLOVL3D2 algorithm using different number of rotation angles and obstacles in terms path lengths and computation time are presented and examined.

4.4.3.1 Different Numbers of Rotation Angles

In order to examine the performance of BLOVL3D2 using several numbers of rotation angles, consider a scenario as shown in Fig. 4.41. The range of the scenario is set to $[420 \times 420]$ and consists of 50 cuboids obstacles. The coordinate of p_{start} is set to (0,0,140) whereas p_{target} is (420,420,155). There are six sets of rotation angles that will be used for path calculation as listed in Table 4.7.



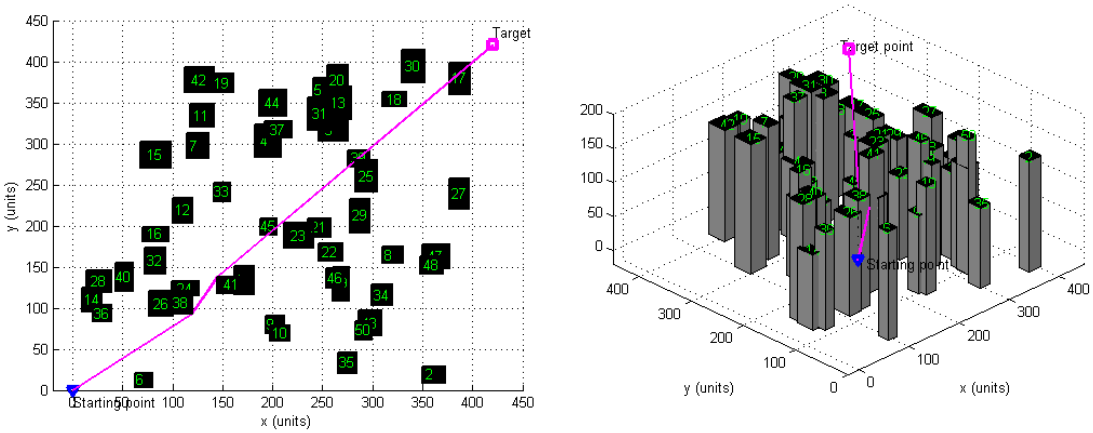
(a) (b)

Figure 4.41: The scenario used to examine the performance of BLOVL3D2 using different sets of rotational angles. (a) top view, (b) 3D view.

Table 4.7: The sets of angles used in the simulation.

Set	Rotation angles
1	{0}
2	{0:90 }
3	{0:60:120}
4	{0:45:135}
5	{0:30:150}
6	{0:15:165}

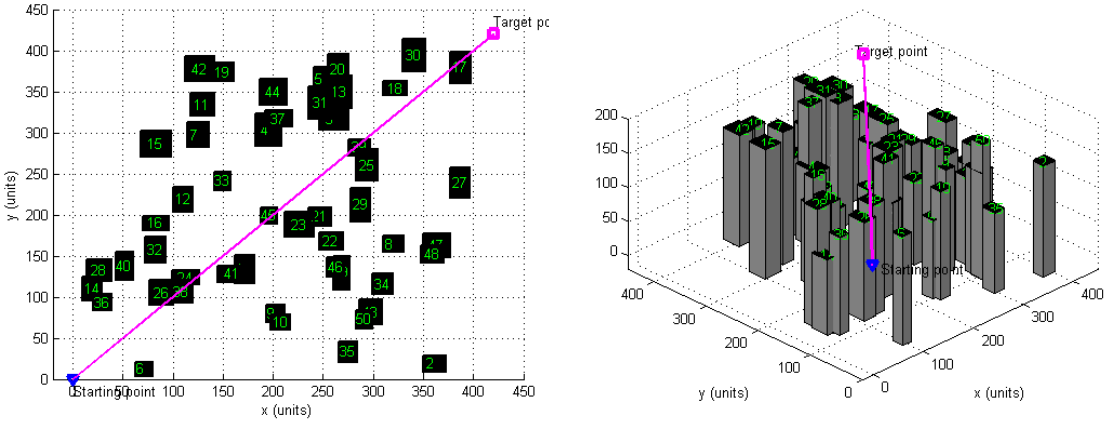
Using the first set of rotation angle, which contains only one angle, i.e. 0 degree, the path found by BLOV3D2 is shown in Fig. 4.42. The path has a length of 597.38 units and is calculated in 0.43 seconds.



(a) (b)

Figure 4.42: The path (solid magenta lines) planned by BLOVL3D2 using {0} degree rotation angles. (a) top view, (b) 3D view.

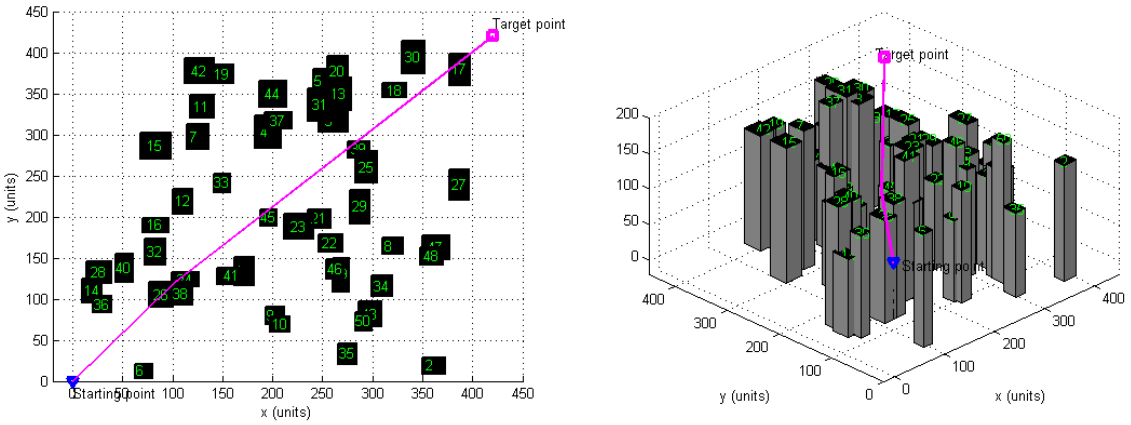
The second set of the rotation angles, which contains 0 and 90 degrees, produces a path with a length of 596.57, which is shorter than the previous one. The path, which is depicted in Fig 4.43, is calculated in 0.65 seconds.



(a) (b)

Figure 4.43: The path (solid magenta lines) planned by BLOVL3D2 using {0,90} degrees rotation angles. (a) top view, (b) 3D view.

Consequently, using the third set of the rotation angles, which consists of 0, 60 and 120 degrees, BLOVL3D2 calculates the path as shown in Fig 4.44 in 1.01 seconds. The path length is 596.94 units.



(a) (b)

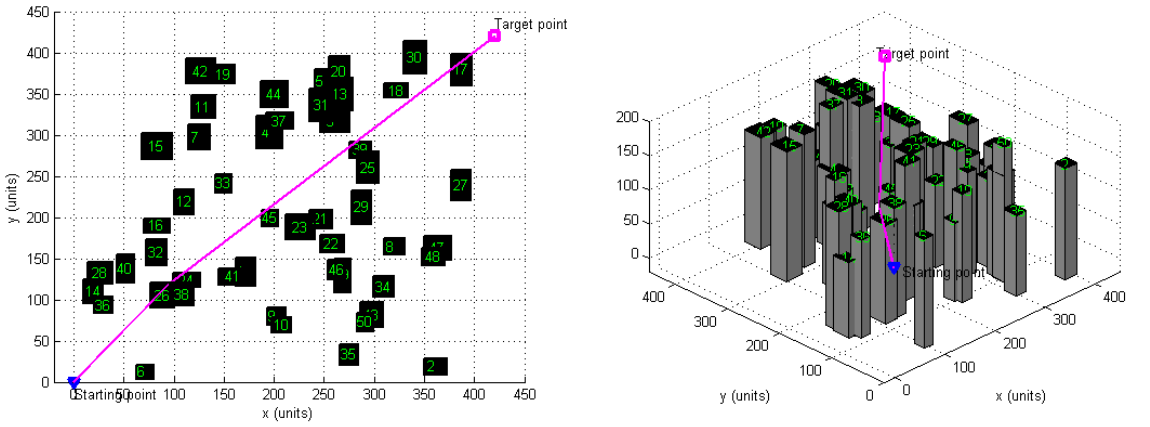
Figure 4.44: The path (solid magenta lines) planned by BLOVL3D2 using {0:60:120} degrees rotation angles. (a) top view, (b) 3D view.

BLOVL3D2 with the rotation angles of {0:45:135}, which are contained in the fourth set, plans a path that is identical with that of the second set as shown in Fig. 4.43. However, using this set of rotation angles, the computation time is slightly increased to 1.09 seconds.

The next set of rotation angles, which consists of {0:30:150} degrees, results in a path as shown in Fig. 4.45. The path length is 595.84 units and it is calculated in 1.65 seconds.

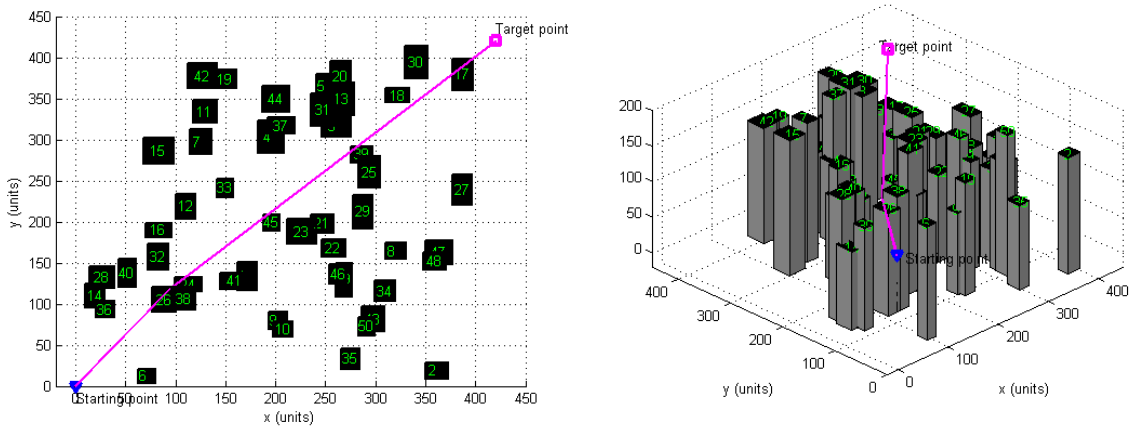
Finally, the sixth set of the rotation angles produces a path with a length of 595.51 units as illustrated in Fig. 4.46. The time taken to calculate the path was 3.05 seconds.

Through the simulations, it can be observed that, as the number of rotation angles is increased, the resulting paths are getting shorter.



(a) (b)

Figure 4.45: The path (solid magenta lines) planned by BLOVL3D2 using {0:30:150} degrees of rotation angles. (a) top view, (b) 3D view.

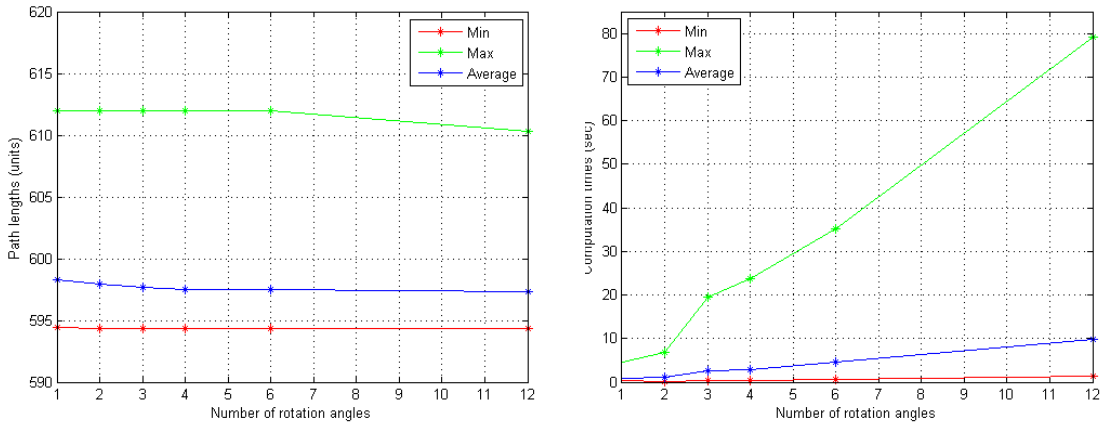


(a) (b)

Figure 4.46: The path (solid magenta lines) planned by BLOVL3D2 using {0:15:165} degrees rotation angles. (a) top view, (b) 3D view.

In order to have a better idea on the effect of the number of rotation angles, simulations using 100 random scenarios have been performed. Each scenario contained 50 cuboids obstacles and its range was fixed to $[420 \times 420]$. The minimum dimension ($[W \times L \times H]$) of each cuboids obstacle was $[15 \times 15 \times 100]$ while the maximum was $[30 \times 40 \times 200]$. The p_{start} and p_{target} were positioned at (0, 0) and (420,420), respectively. The minimum heights of p_{start} and p_{target} were set at 130 and 140 units, while the maximum of those were 160 and 170, respectively. In each

scenario, the sets of rotation angles as listed in Table 4.7 were utilised to find 3D paths. The resulting maximum, minimum and average paths lengths and computation time are plotted in Fig. 4.47(a) and Fig. 4.47(b), respectively. To have their exact values, the average path lengths and computation time for each set of rotation angles are recorded in Table 4.8.



(a) (b)

Figure 4.47: The simulations results of BLOVL3D2 using different number of rotation angles in 100 random scenarios, each with 50 cuboids obstacles; (a) path lengths, (b) computation time.

Table 4.8: The average of path lengths and computation time using sets of rotation angles.

	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6
Ave. path lengths (units)	598.28	597.98	597.63	597.47	597.47	597.35
Ave. computation time (s)	0.98	1.09	2.61	2.94	4.66	9.76

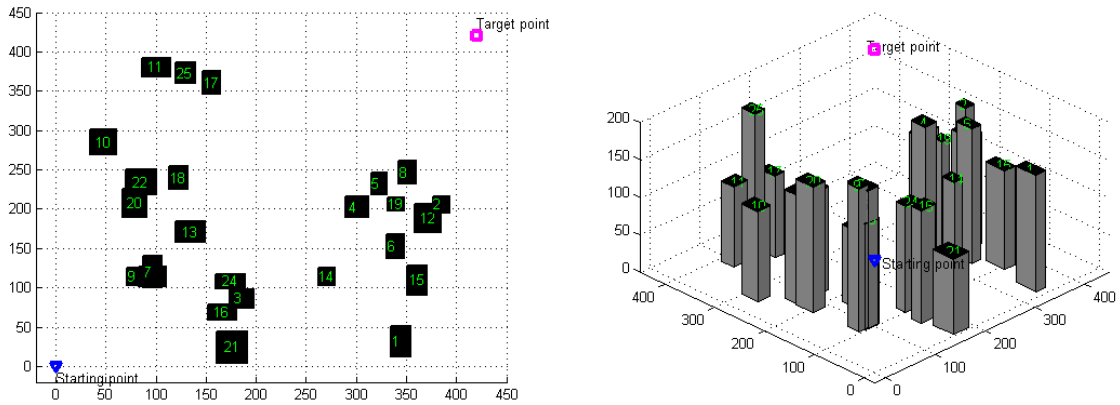
As can be seen from Fig. 4.47(a), a higher number of rotation angles used in the BLOVL3D2 leads to a shorter path length at the expense of computation time. As there is a trade-off between path length and computation time, the selection of the number of rotation angles has to be made based on the type of mission i.e. off-line or real-time. If an off-line path planning is required, the number of rotation angles has to be large to ensure the resulting path, which is planned prior to the mission, is as shortest as possible. On the other hand, the number of rotation angles should be minimal for a mission that requires real-time path planning so that a 3D path can be calculated in a relatively short time.

4.4.3.2 Different Numbers of Obstacles

In order to measure the performance of BLOVL3D2 using different number of obstacles, scenarios containing 25, 50 and 75 cuboids obstacles have been used for simulations. As a matter of fact, the number of obstacles affects the density or ratio of the occupied space by the obstacles to the C-space area. Generally, a higher density scenario results in a longer computation time and a longer path. The selection of different obstacles numbers as stated above ensures that the performance of the proposed algorithm in terms of path length and computation time is informative.

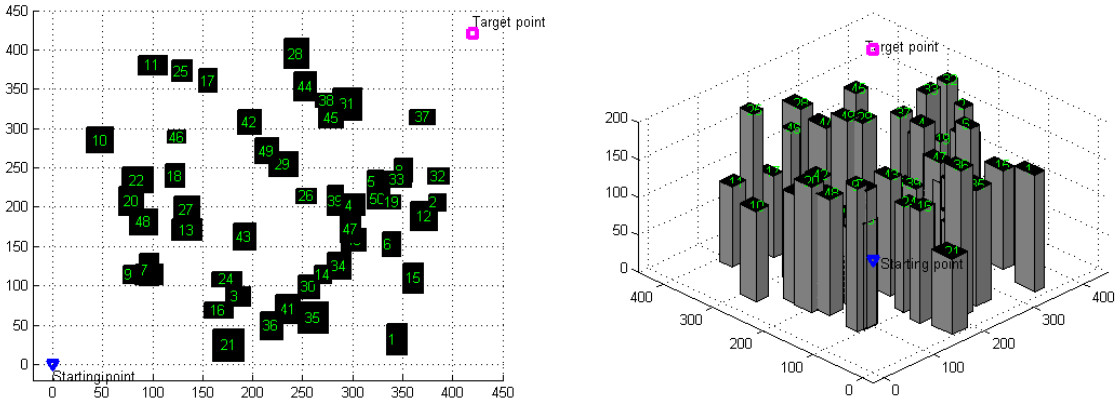
In the simulations, each number of obstacles was generated in 100 random scenarios, thus, as three numbers of obstacles were used, there were 300 scenarios in total. The obstacles minimum dimension $[W \times L \times H]$ was set to [15×15×100] and their maximum was [30×40×200]. Each scenario covers an area of 420 by 420 units in X- and Y- axis, respectively. The starting point, p_{start} coordinate was fixed to (0,0) while the target point p_{target} was (420,420). The heights of the starting point were randomly varied between 130 to 160 units and the target point heights were between 140 to 170 units. The rotation angles were set to {0:45:135}.

Figs. 4.48-4.50 illustrate the first three scenarios of the simulations with 25, 50, 75 cuboids obstacles, respectively. Note that the scenario as shown in Fig. 4.48 had the lowest density while the one shown in Fig. 4.50 possessed the highest.



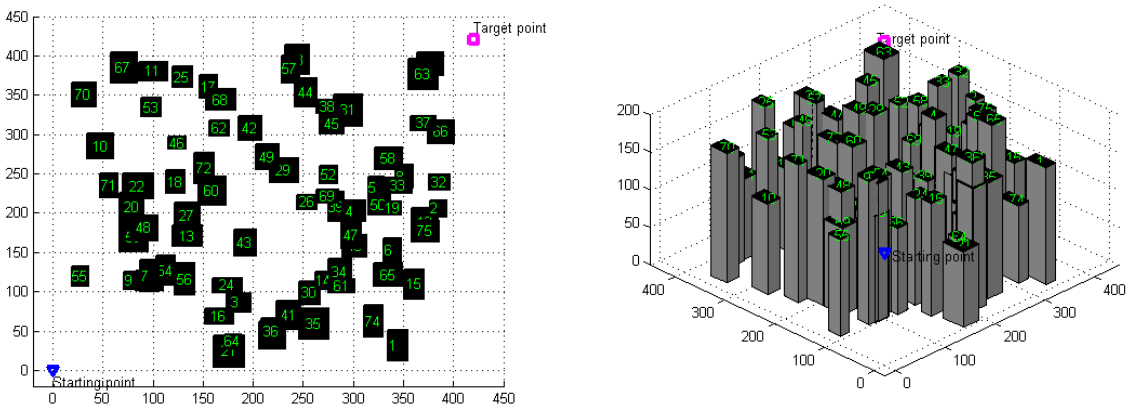
(a) (b)

Figure 4.48: A scenario with 25 cuboids obstacles. (a) top view, (b) 3D view.



(a) (b)

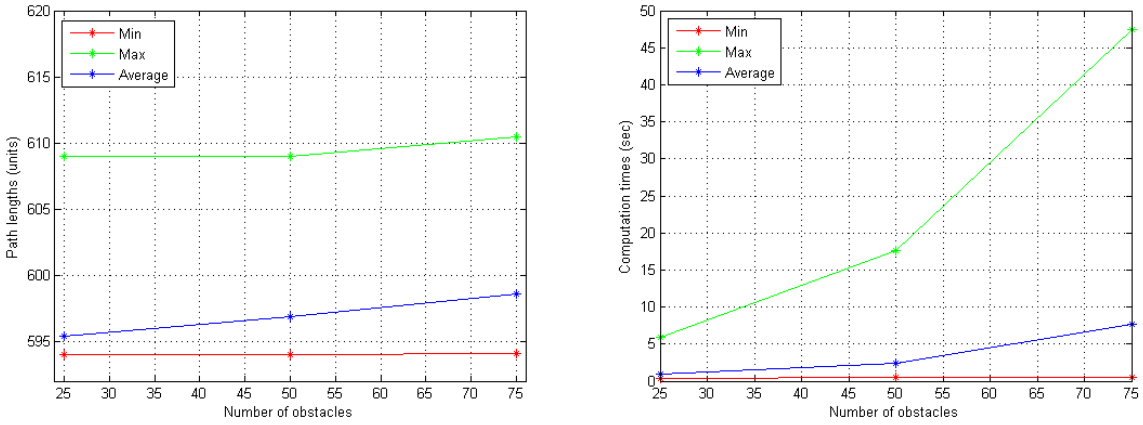
Figure 4.49: A scenario with 50 cuboids obstacles; (a) top view, (b) 3D view.



(a) (b)

Figure 4.50: A scenario with 75 cuboids obstacles; (a) top view, (b) 3D view.

The minimum, average and maximum path lengths and computation time in 100 random scenarios for each number of obstacles are shown in Fig. 4.51(a) and 4.51(b), respectively. From the figures, it is clear that the average path lengths and computation time are proportional to the numbers of obstacles.



(a) (b)

Figure 4.51: The results of the BLOVL3D2 simulations using different number of obstacles; (a) paths lengths, (b) computation time.

4.5 Conclusion

In this chapter, a set path planning algorithm in 3D environments has been introduced and demonstrated. The proposed algorithms are based on the base line oriented visibility lines (BLOVL) algorithm, which has been introduced in Chapter 3. The proposed 3D path planning algorithm, called BLOVL3D2, uses rotational planes on which visibility lines (VL) networks are created in order to find 3D paths.

To see the effect of the numbers of rotation angles in BLOVL3D2, six sets of rotation angles have been used in the simulations. The results of the simulations have shown that despite BLOVL3D2 consumes longer computation time, a higher number of rotational angles leads to a relatively better 3D path in terms of path's length.

In addition, simulations to evaluate the effect of number of obstacles have also been performed. In the simulations, there were 300 random scenarios with 25, 50 and 75 obstacles. Each number of obstacles was generated in 100 scenarios. From the results of the simulations, it is concluded that the BLOVL3D2 algorithm takes a longer time to plan a path in an environment with a higher number of obstacles. The relationship between the number of obstacles and the average computation time is slightly non-linear as illustrated by Fig. 4,51(b). In terms of path lengths, BLOVL3D2 plans longer path in environments with higher number of obstacles, however, the relationship between the number of obstacles and the average path lengths is slightly non-linear.

Chapter 5

Software Packages for Path Planning

5.1 Introduction

At the University of Leicester, there is a path planning software package that has been developed that serves two purposes; first, it validates the effectiveness of the proposed algorithms visually. Second, it implements and presents the algorithms in an intuitive way. The software package, which has been developed using Matlab consists of two Graphical User Interfaces (GUIs), each with its own aim. The first GUI is used to execute the BLOVL algorithm, while the second executes the BLOVL3D1 and BLOVL3D2 algorithms. The former and the latter GUIs are referred to as 2D GUI and 3D GUI, respectively in this chapter.

Finding a collision-free path using a path planning algorithm normally requires several inputs such as the number of obstacles, dimensions and position, a starting point, a target point and the vehicle's speed and kinematic constraints. The inputs have to be keyed-in according to a certain logical order. For example the starting and target points cannot be located before all the obstacles data (locations and dimensions) are made available, otherwise they might be in the obstacle region, causing the path planning to be incomplete. On the other hand, the path planning process cannot be executed if any of the required input is not supplied. The developed software package is designed to address this issue as the path planning process is implemented systematically and in user-friendly manner. Before explaining the software packages further, it is useful that the important objects as shown in Fig. 5.1 that have been used in the GUIs are introduced.

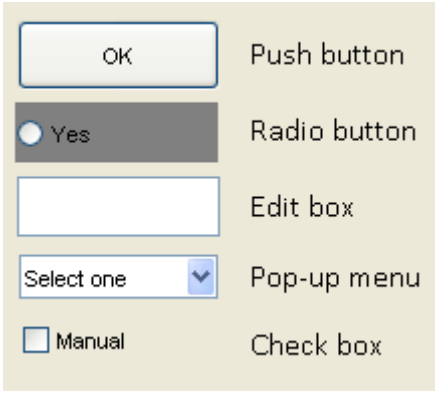


Figure 5.1: The objects used in the GUI

This chapter is arranged as follows. The aims of the GUIs and their functionalities are introduced first in order to demonstrate the way in which the 2D GUI is carried out based on the proposed algorithms to find a 2D path in a random scenario. Then, a real-time 2D path planning is shown using the 2D GUI. Additionally, the 3D path planning processes using the 3D GUI is demonstrated in order to find 3D paths through the applications of both the BLOVL3D1 and BLOVL3D2 algorithms.

5.2 The GUIs Aims

The 2D GUI is used to visually demonstrate the real time path planning based on the BLOVL algorithm. This is useful as it demonstrates the basic idea of the proposed algorithm in dealing with pop-up obstacles. On the other hand, the 3D GUI implements both the BLOVL3D1 and BLOVL3D2 algorithms, demonstrating the concept of rotational planes in finding 3D collision-free paths for different starting and target points altitudes.

5.3 The GUIs Features

The 2D and 3D GUIs are designed in such a way that their user will be able to operate them with minimal guidance and practice. Additionally, both GUIs come with step-by-step instructions located at the top of the GUIs. As the GUIs commence and progress throughout the path planning process, the instructions will guide the user to perform the next action after a particular step has been undertaken.

In a user-friendliness sense, all objects in the GUIs as shown in Fig. 5.1 are arranged to follow a chronological order and with some objects disabled, so that further progress can only be made after particular buttons are pressed or a certain parameters are keyed-in. Moreover, most of the objects are grouped into a number of frames, located at the left hand side of the GUI for convenience purposes and as check points to ensure that necessary inputs/parameters are keyed-in for the path planning process.

Using the 2D GUI, the movement of the UAV traversing the planned 2D path is visually animated. It displays the UAV's traversal along the planned collision-free path towards the target point. If pop-up obstacles are detected on the UAV's path by the UAV's sensor, the proposed algorithms through the 2D GUI will calculate a new collision-free path. The UAV will then traverse the newly planned path and repeat the previous step if pop-up obstacles are detected, until it reaches the target point. This feature allows the *real time* path planning process can be viewed. Furthermore, there is an option that allows the user to view the traces of a path throughout a path planning session. As the proposed algorithms are executed sequentially, there might be several traces of the optimal paths from the starting point/waypoints to the target point. This feature applies to both 2D and 3D GUIs.

In relation to the 3D GUI, the rotational plane displayed in the provided axis is shown sequentially, provided that particular check boxes are properly ticked. These features are useful to check the validity of the proposed algorithms' execution.

5.4 Path Planning using 2D GUI

The developed 2D GUI for path planning in 2D is shown in Fig. 5.2. It finds a collision-free path with fixed altitude, based on the proposed 2D path planning algorithms. In this GUI, the starting point p_{start}, target point p_{target}, obstacles and the resultant path are assumed to be at the same heights. They are all displayed in the provided axis, which is situated at the right hand side of the GUI.

5.4.1 Operating the 2D GUI

The GUI is designed to be user-friendly; the user will be guided throughout a path planning session by a series of instruction located in the *Instructions* box at the top of

the GUI. Once the GUI is launched as shown in Fig. 5.2, the user is asked by the *Instructions* box to do the following:

Welcome to the path planning package by the University of Leicester. To begin, go to STEP 1 and set the range of search space.

STEP 1 consists of several sub-steps as shown in Fig. 5.3. The main purpose of *STEP 1* is to provide the environment, in which the path planning in 2D will take place with the necessary parameters.

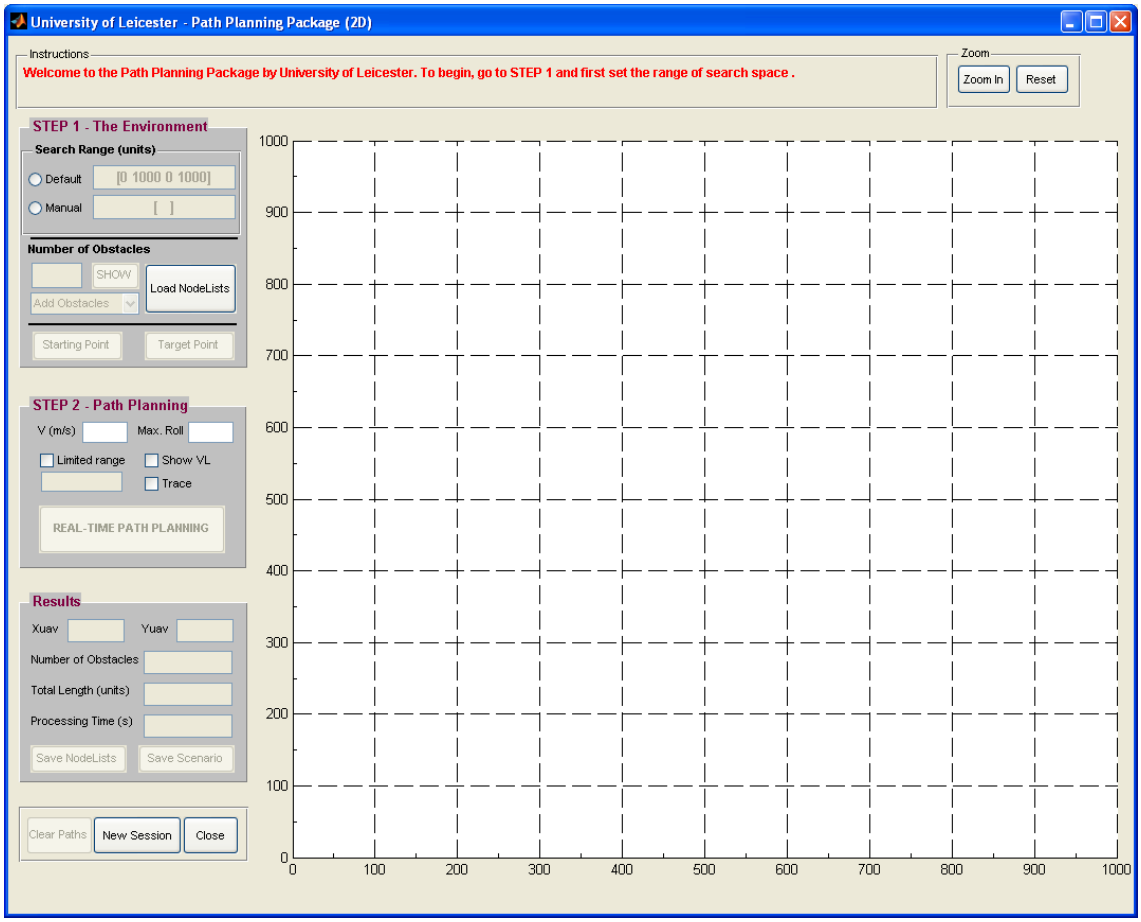


Figure 5.2: The GUI for 2D path planning using the proposed algorithms

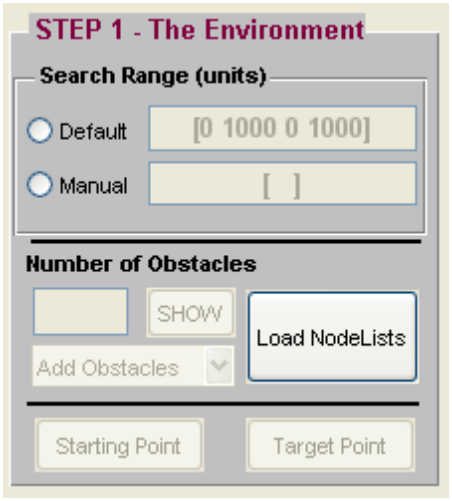


Figure 5.3: The *STEP 1* of the 2D package

As instructed, the range of the search space has to be set by either selecting the *Default* or by manual entry the *Manual* radio-button in the *Search Range* (*units*) option. If the range entered is not as in the required format, a warning window as shown in Fig. 5.4 will pop-up and suggests the correct format. Assuming that the area of *C*-space is 500x500 units, the range is then manually set to [0 500 0 500].

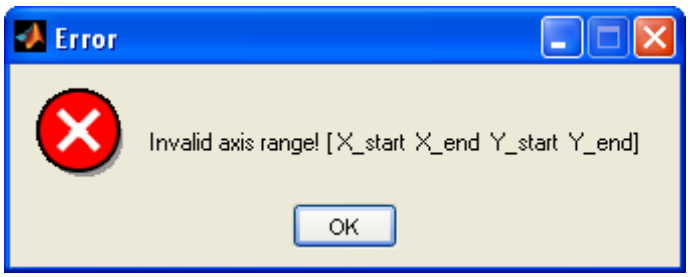


Figure 5.4: A warning window asking user to enter the range correctly.

Then the GUI through the *Instructions* box shows the following:

Now enter the number of obstacles or load the nodelist in STEP1.

The obstacles' number has to be keyed-in in the provided editable box. Otherwise a file that contains a node set needs to be supplied. The node set can be imported from a file that contains previously saved data by pressing the *LoadNodelists* pushbutton.

For demonstration purposes, 50 obstacles are assumed in the search space, hence 50 is keyed-in in the box and the *SHOW* pushbutton is then pressed. Now, a scenario with 50 obstacles in a search space of 500x500 has been generated and displayed in the provided axis as shown in Fig. 5.5. The obstacles in the search space can be added by pressing the *Add Obstacles* pop-up menu. The dimension of the added obstacles can be adjusted using the mouse to match with the real obstacles size in the real world.

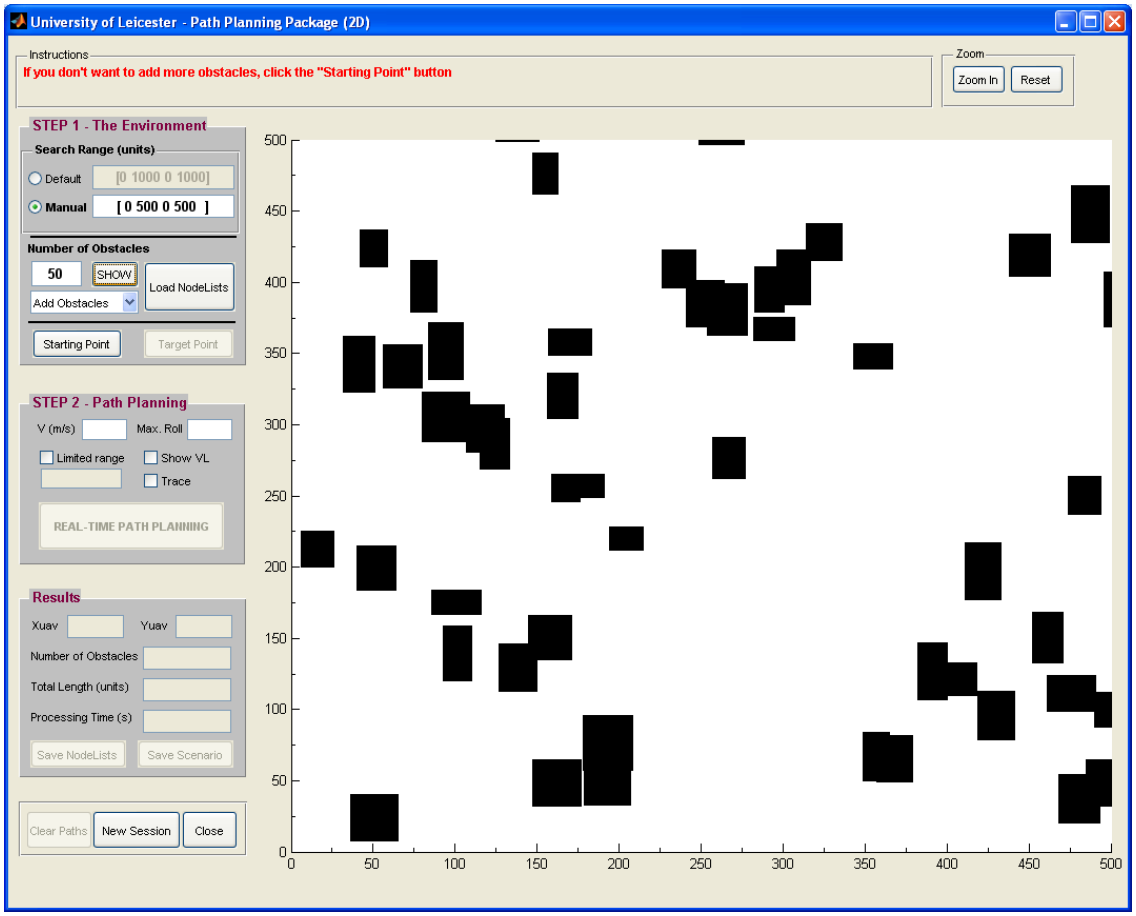


Figure 5.5: A scenario of 500x500 units with 50 obstacles has been set.

Then the starting point p_{start} has to be located by pressing a pushbutton called *Starting Point*. It can be located anywhere within the range of the search space. As soon as the p_{start} is located, the pushbutton for locating the target point, which is named *Target Point* is enabled. As long as the p_{start} is not located in the search space, the *Target Point* button is kept deactivated. The p_{target} location then needs to be specified in the search space using the mouse. Suppose the p_{start} and p_{target} are located at (50, 100), and (250, 450), respectively. Fig. 5.6 shows the GUI with the generated scenario after *STEP 1* has been completed.

The next step is to key-in the necessary parameters for path planning, considering the UAV's kinematic constraints such as the vehicle's speed and maximum bank angle. All these are grouped in the *STEP 2 – Path Planning* frame and shown in Fig. 5.7. After all the required parameters are keyed-in, a path can be generated using the proposed algorithms by pressing the *REAL-TIME PATH PLANNING* pushbutton. The GUI will then display the path for a UAV with 50 km/h speed and 50 degree bank angle in the provided axis as illustrated in Fig. 5.8.

In order to display the visibility lines created by the algorithms, the *Show VL* check box as shown in Fig. 5.7 has to be ticked. The visibility lines of the previous scenario, generated from the p_{start} and the second waypoint to p_{target} are shown in Fig. 5.9 and Fig. 5.10, respectively. Displaying the visibility lines is useful because it reflects the number of obstacles involved in the path calculation.

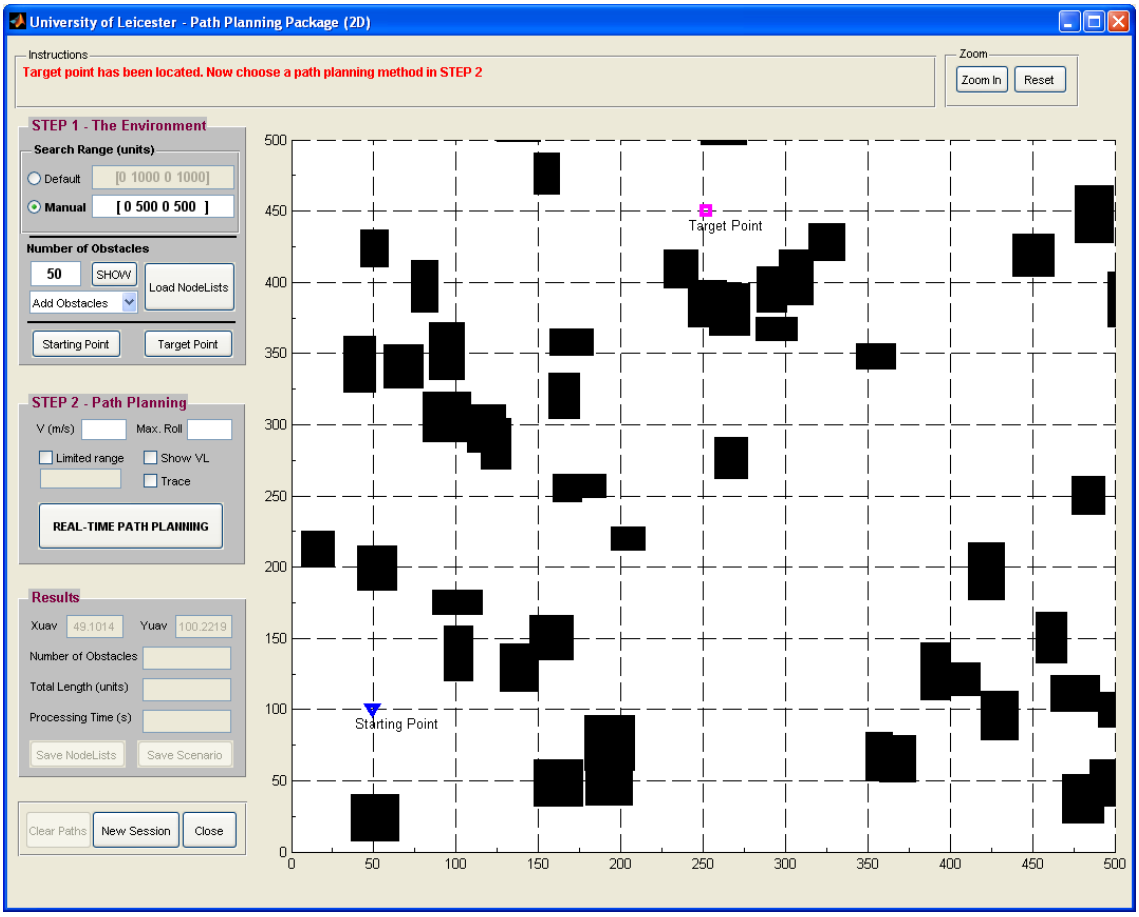


Figure 5.6: A complete scenario with the area of 500x500 units, 50 obstacles, a starting point and target point.

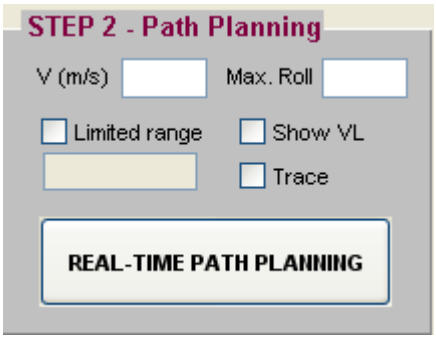


Figure 5.7: Necessary parameters are required to plan a 2D collision-free path.

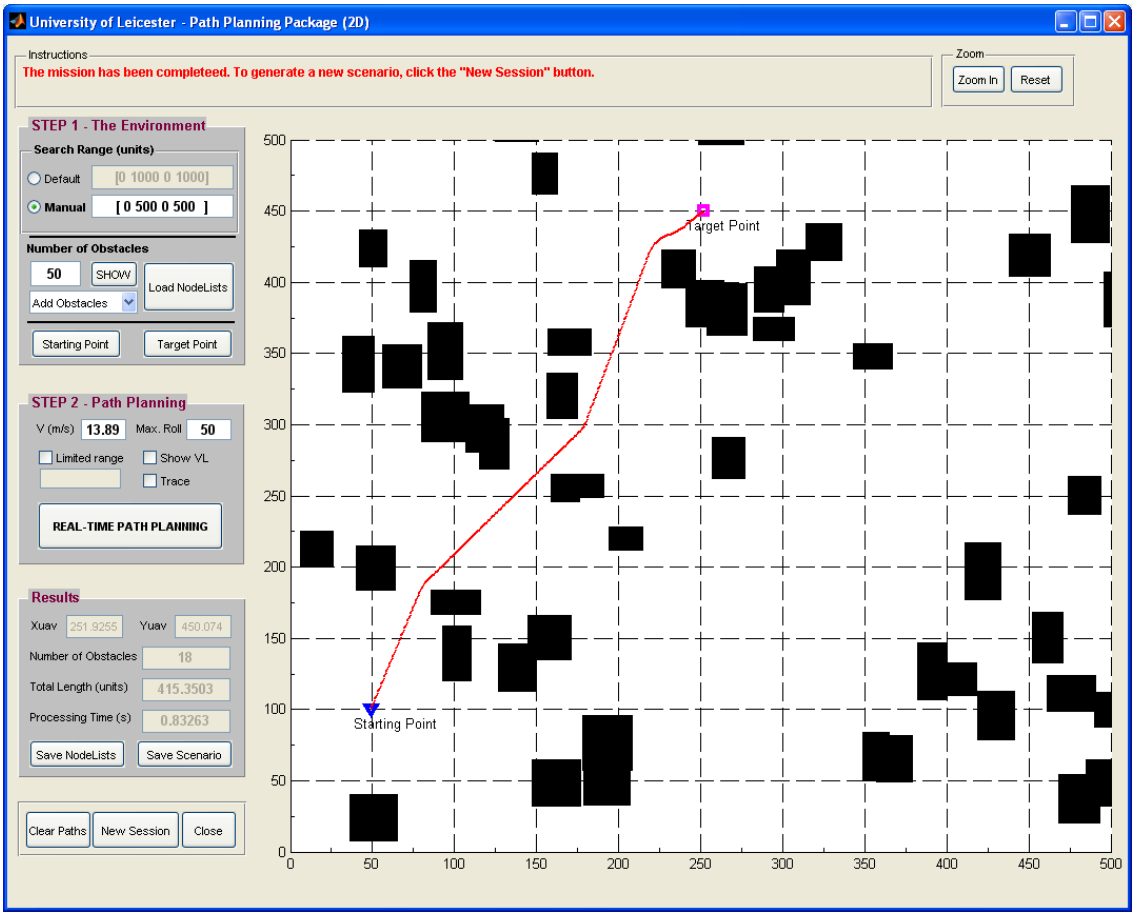


Figure 5.8: The planned path shown in red, satisfying the kinematic constraints

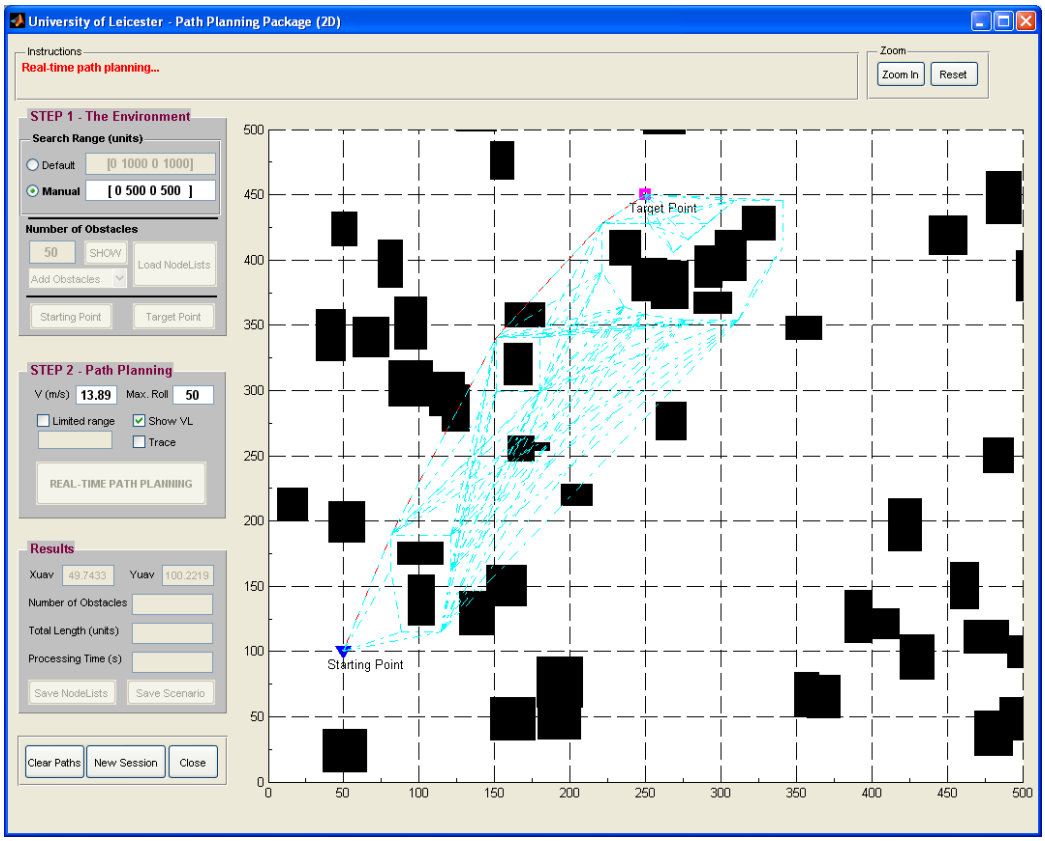


Figure 5.9: The visibility lines are shown from the starting point

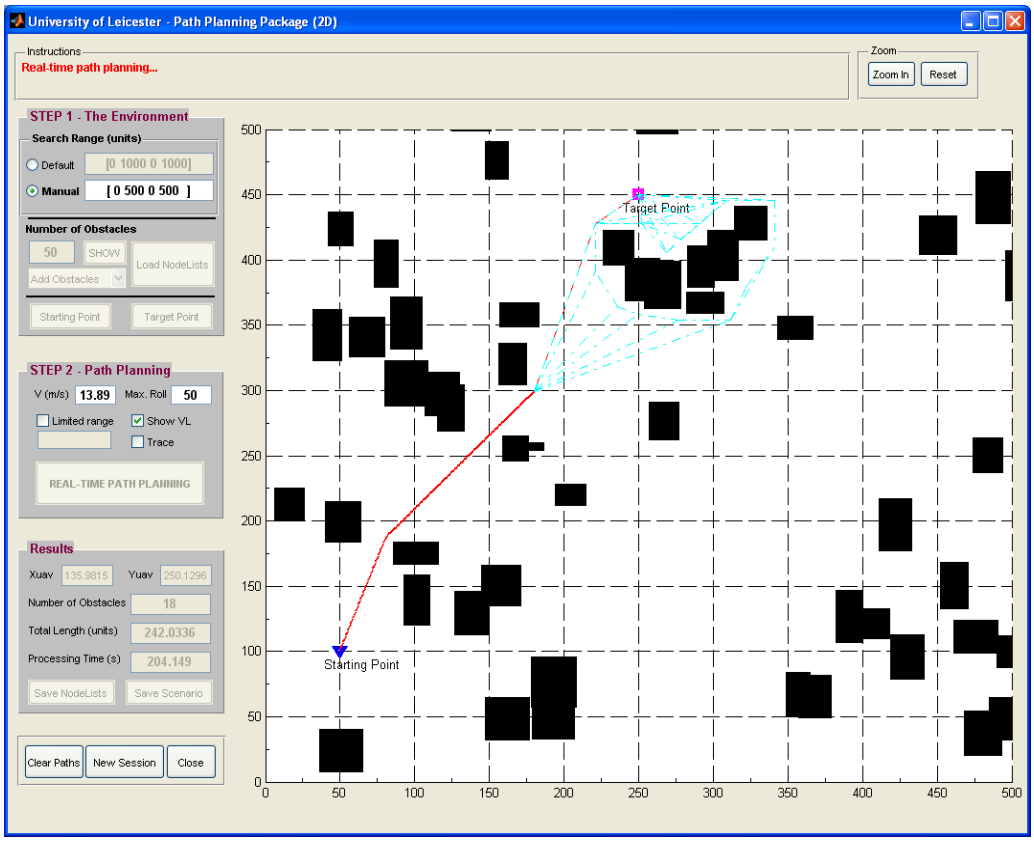


Figure 5.10: The visibility lines are shown from the second waypoint

In addition, by ticking the *Trace* check box in *STEP 2*, the trace of the planned path from each waypoint will be displayed as depicted in Fig. 5.11. The solid red line is the final path, while the dashed red line refers to the path that is planned previously (in this case, it is from the starting point).

Another feature of the GUI is that it displays all the important information about the planned path in the *Results* box located at the lower left part of the GUI as depicted in Fig. 5.12. The displayed information represents the current position of the UAV, the number of the considered obstacles that have been used to calculate the path, the total path length and the computation time. Additionally, the node list of the current scenario can be saved to be used in the future.

The specific area of the search space can be zoomed-in to display a clearer view of the selected area. This is done by pressing the *Zoom In* push button, which is located at the GUI's top right. Finally, a new session can be started by pressing the *New Session* pushbutton which is situated at the left bottom area of the GUI. The same afore-mentioned procedures have to be followed to find a collision-free path in 2D.

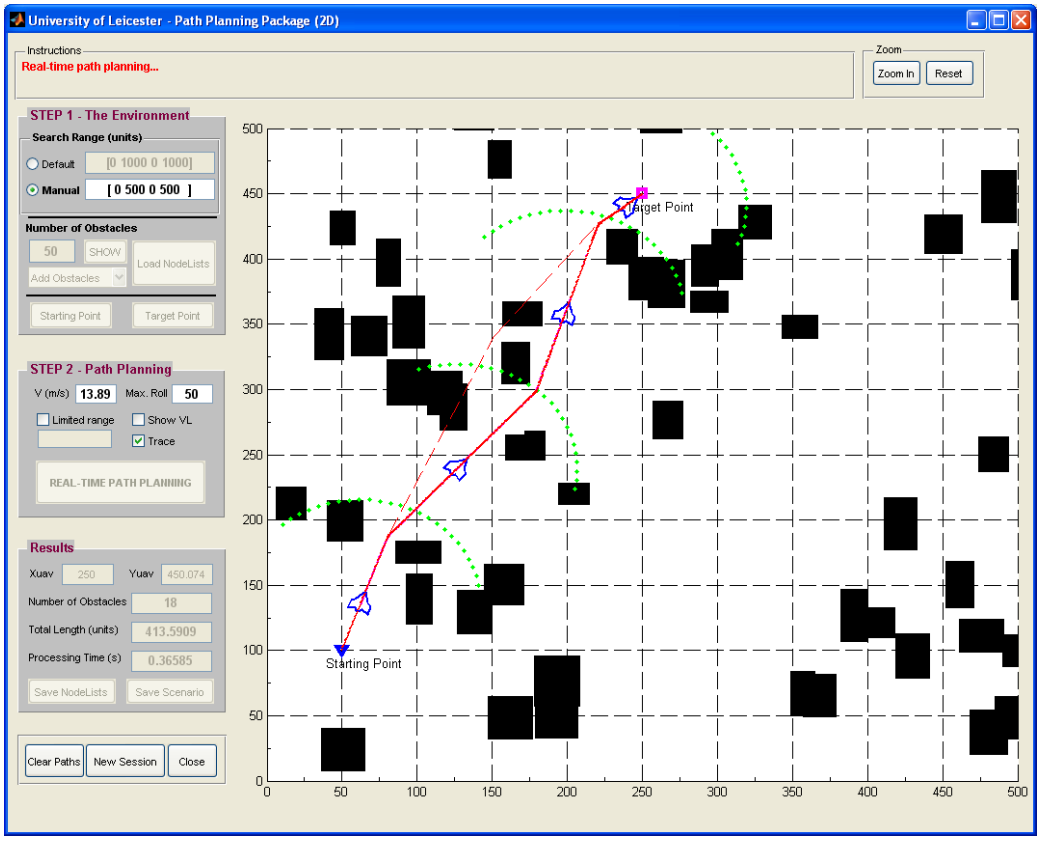


Figure 5.11: The traces of the planned path.

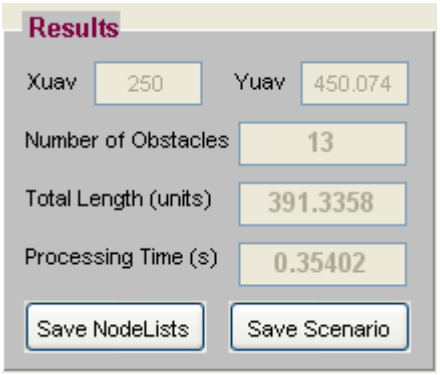


Figure 5.12: The information of the planned path.

5.4.2 Real-Time Path Planning Using the GUI

Demonstrating a path planning process in real-time using the proposed algorithms is one of purposes of the developed 2D GUI. The GUI is capable of displaying the planning and re-planning processes in a scenario with pop-up obstacles. A new path, which is locally optimal is planned when the UAV's sensors detect the pop-up obstacles.

Consider a scenario as shown in Fig. 5.13 in which pop-up obstacles will appear during the UAV's traversal. The scenario consists of 75 *a priori* defined obstacles with p_{start} and p_{target} located at (100,100) and (600,700), respectively. The UAV's speed is assumed to be 50 km/h and its maximum bank angle is 50 degrees. Fig. 5.14 shows a globally optimal path (dotted red line) planned, based on the information provided. The UAV traversing the planned path is animated in the GUI as shown in Fig. 5.15. The dotted green semi-circle is the UAV's sensor coverage with limited range. As the UAV traverses the path, four previously unknown obstacles pop-up and one of them is on the UAV's path. At this moment, the path planner does not take any action as the obstacle is yet to be detected by the onboard sensors. As the obstacles are detected as illustrated by Fig. 5.16, the UAV will quickly re-plan a new path as depicted in Fig. 5.17 based on the proposed 2D path planning algorithms. The path is not collision-free since the algorithm considers only the obstacles on the base line and their extensions in re-planning the path at this moment. The path is again re-planned as soon as the sensors detect more pop-up obstacles on the UAV's path. This situation is illustrated in Fig. 5.18. The UAV carries on the flight until it reaches at the target point, p_{target} as shown in Fig. 5.19.

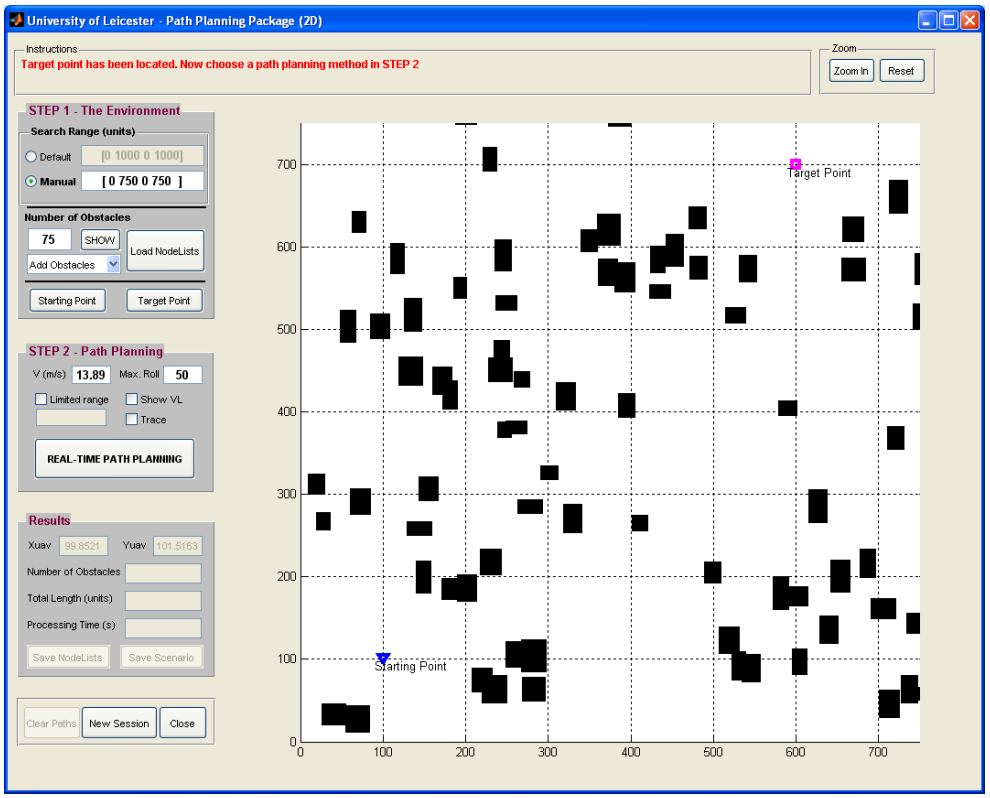


Figure 5.13: The scenario that is used to demonstrate the real-time path planning.

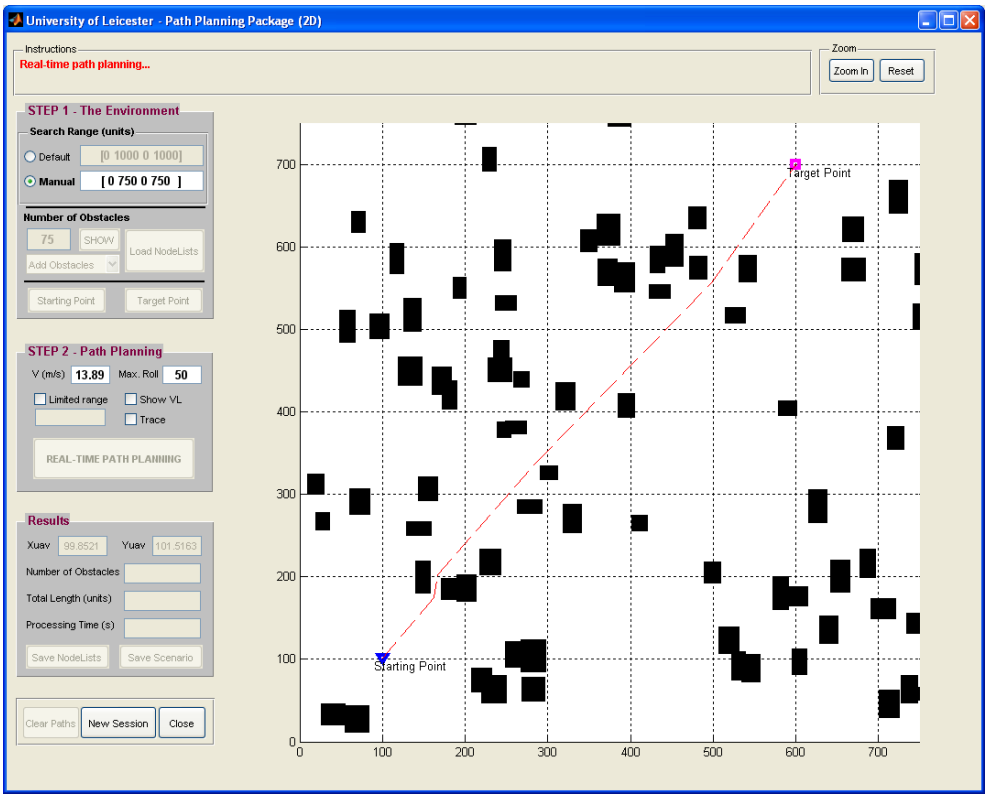


Figure 5.14: The globally optimal path.

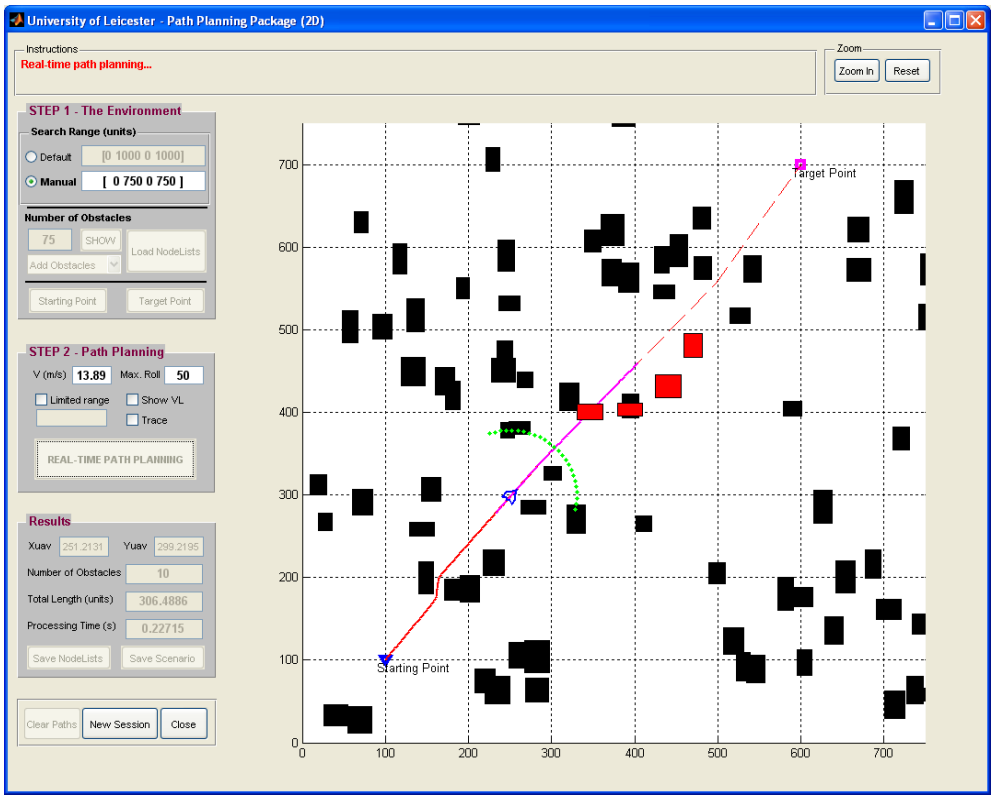


Figure 5.15: The UAV is traversing the planned path.

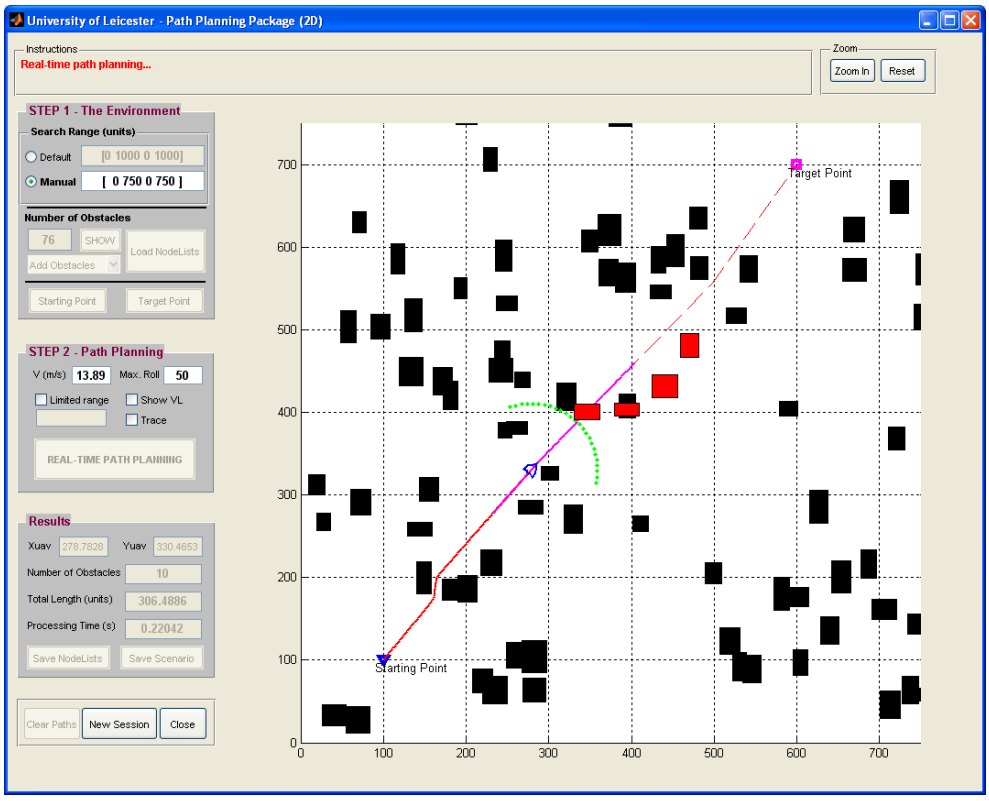


Figure 5.16: The UAV detects the pop-up obstacle.

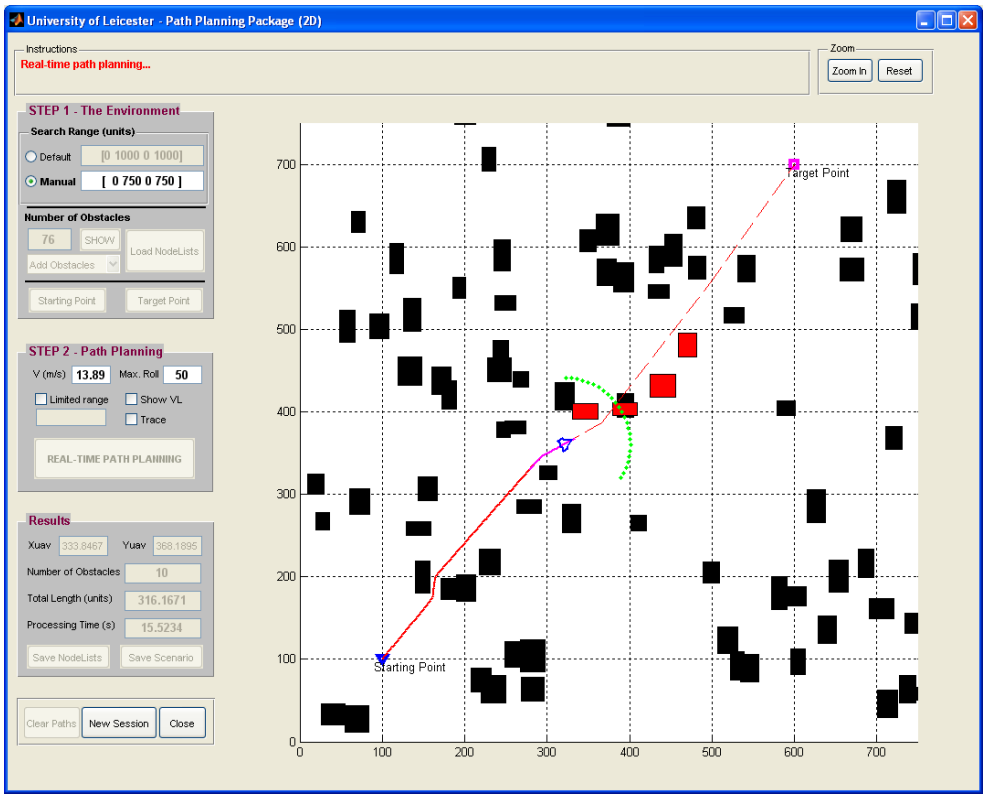


Figure 5.17: The new path is re-planned when the pop-up obstacle is detected.

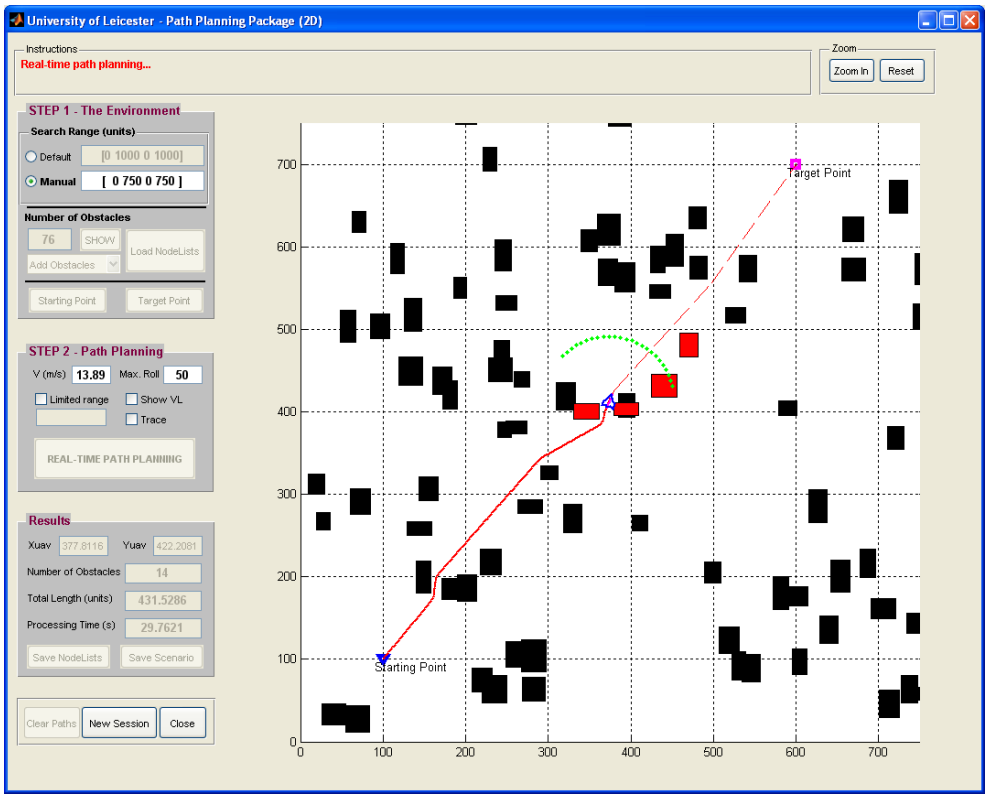


Figure 5.18: The UAV traverses the re-planned collision-free path.

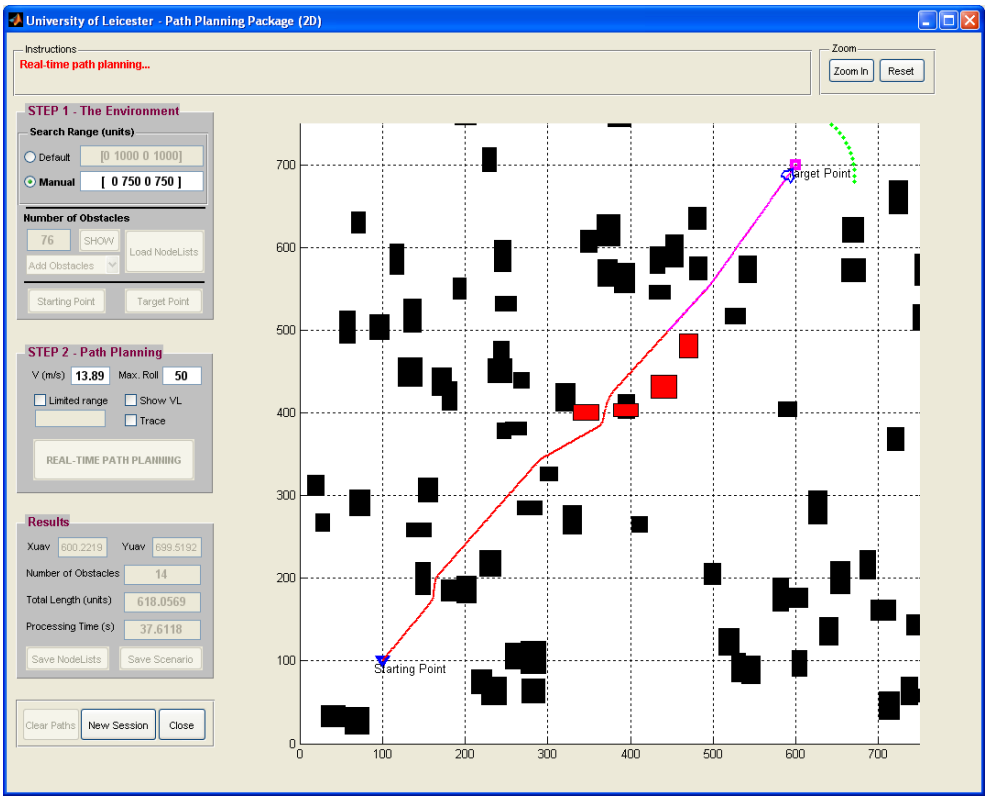


Figure 5.19: The UAV successfully reaches at the target point.

5.5 Path Planning Using 3D GUI

The 3D GUI is shown in Fig. 5.20. The package executes the proposed 3D path planning algorithms to find a 3D path in a given scenario. Unlike 2D GUI, the 3D GUI takes into account the altitudes of the obstacles, the starting point p_{start} and target point p_{target} during the path calculation process.

5.5.1 Operating the 3D GUI

The way to operate the 3D software package is similar to that of the 2D GUI. First the area of the search space has to be defined. In *STEP 1* of 3D GUI, unlike in the 2D package, the search space consists of six values i.e. [x_{start} x_{end} y_{start} y_{end} z_{start} z_{end}], as the range in the *z*-axis is included. *STEP 1* of 3D GUI is shown in Fig. 5.21.

Next, the number of obstacles in the provided box needs to be inputted. Pressing the *SHOW* button will display the obstacles, whose dimensions and positions are randomly generated. There is also an option to load a set of nodes from a file. An example of a scenario covering the area of 500x500x200 with 75 obstacles is shown in

Fig. 5.22. More obstacles which are available in several shapes can be added by pressing the *Add Obstacles* pop-up menu. The size of the added obstacles can be adjusted according to the real obstacles' sizes in real world using the mouse.

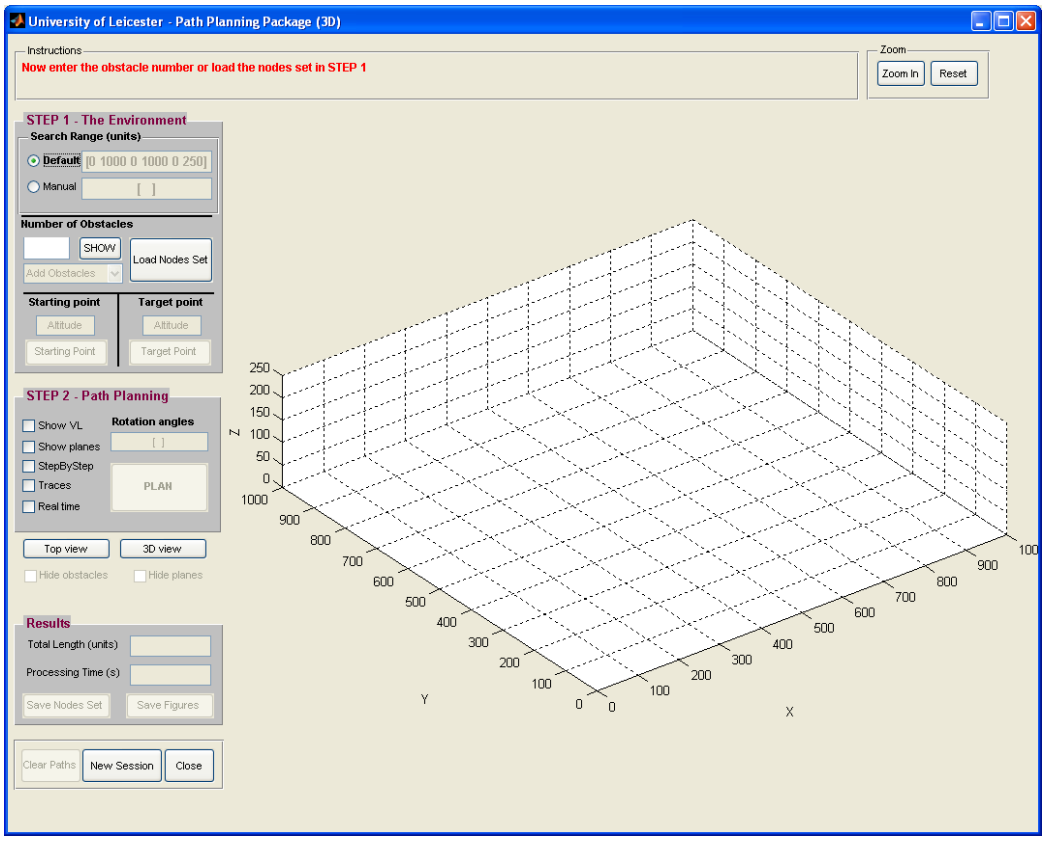


Figure 5.20: A GUI for 3D path planning using the proposed algorithms

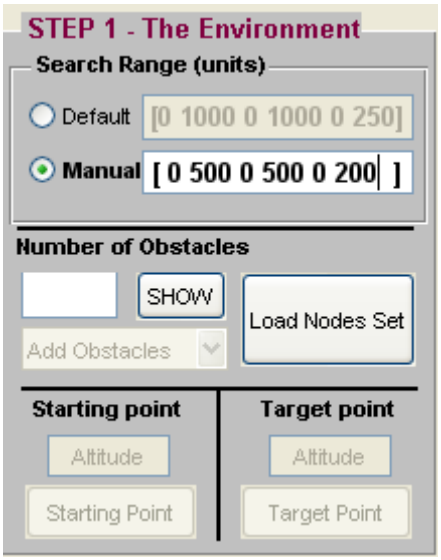


Figure 5.21: The *STEP 1* of the 3D package.

After that, the altitudes and locations of p_{start} and p_{target} must be defined. An example of a complete scenario is shown in Fig. 5.23. In the scenario the locations of the p_{start} and p_{target} are assumed to be at coordinates (20,20,170) and (400,500,150), respectively. The third values in the coordinates refer to the altitudes of the starting and target points.

In the next step of the 3D GUI, called *STEP 2*, more information should be supplied to the GUI, such as the algorithms to be used (either BLOVL3D1 or BLOVL3D2) and the rotation angles. *STEP 2* of the package is shown in Fig. 5.24. For the BLOVL3D1 algorithm, the rotation angles are not required as they are fixed to 0 and 90 degrees. After all the necessary parameters are supplied in *STEP 2*, the *PLAN* button is pressed to find a 3D path. The paths that are planned by the BLOVL3D1 and BLOVL3D2 algorithms through the 3D GUI are shown in Figs. 5.25 and 5.26, respectively. Note that for BLOVL3D2, the plane rotation angles are set to {0,30,60,90,120,150} degrees.

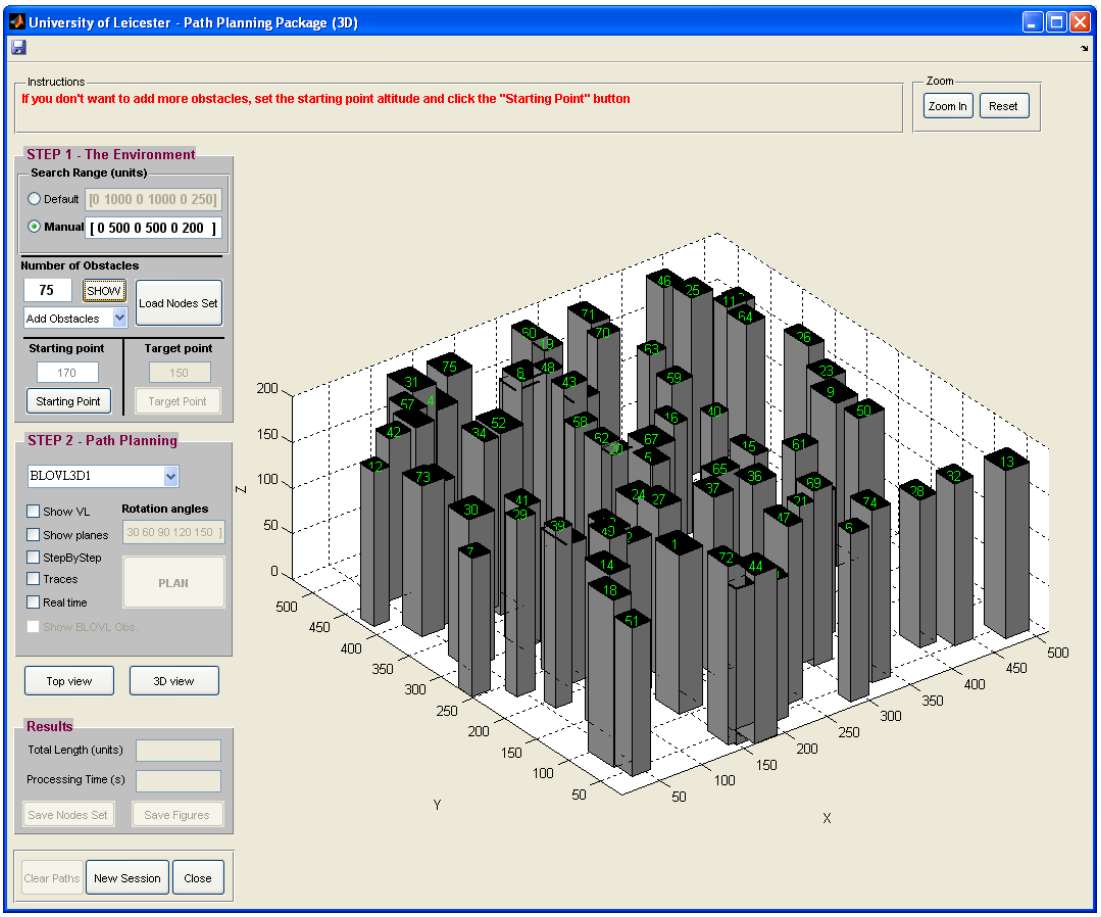


Figure 5.22: The 3D scenario with 75 obstacles

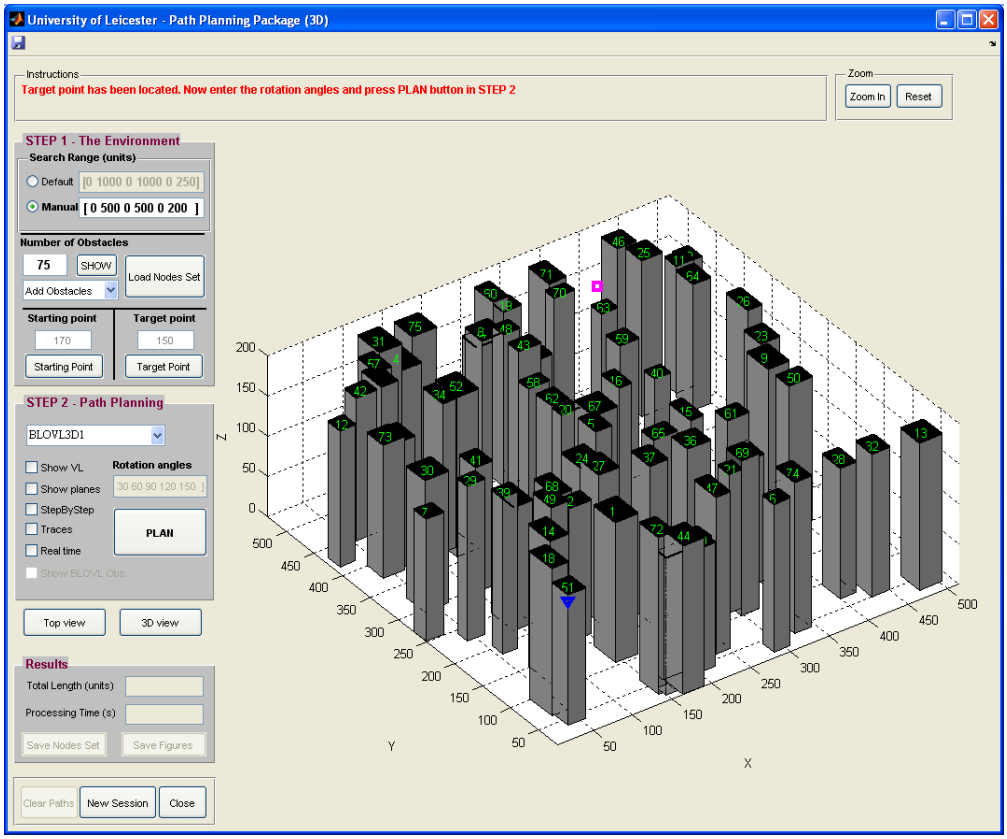


Figure 5.23: The complete 3D scenario with 75 obstacles, starting point (blue triangle) and target point (square magenta)

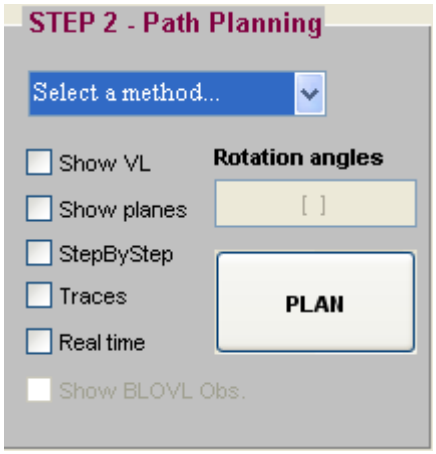


Figure 5.24: Necessary parameters are required to plan a 3D collision-free path.

The plane can be displayed in the axis by ticking the *Show planes* check box as shown in Fig. 5.24. The plane is shown sequentially if the *StepByStep* checkbox is also ticked. Using the same scenario, the planes for BLOVL3D1, which is seen from the top and in 3D views, are depicted in Fig. 5.27 and Fig. 5.28, respectively. On the other hand, the planes generated by BLOVL3D2, viewed from top and in 3D are illustrated in Figs. 5.29 and 5.30, respectively.

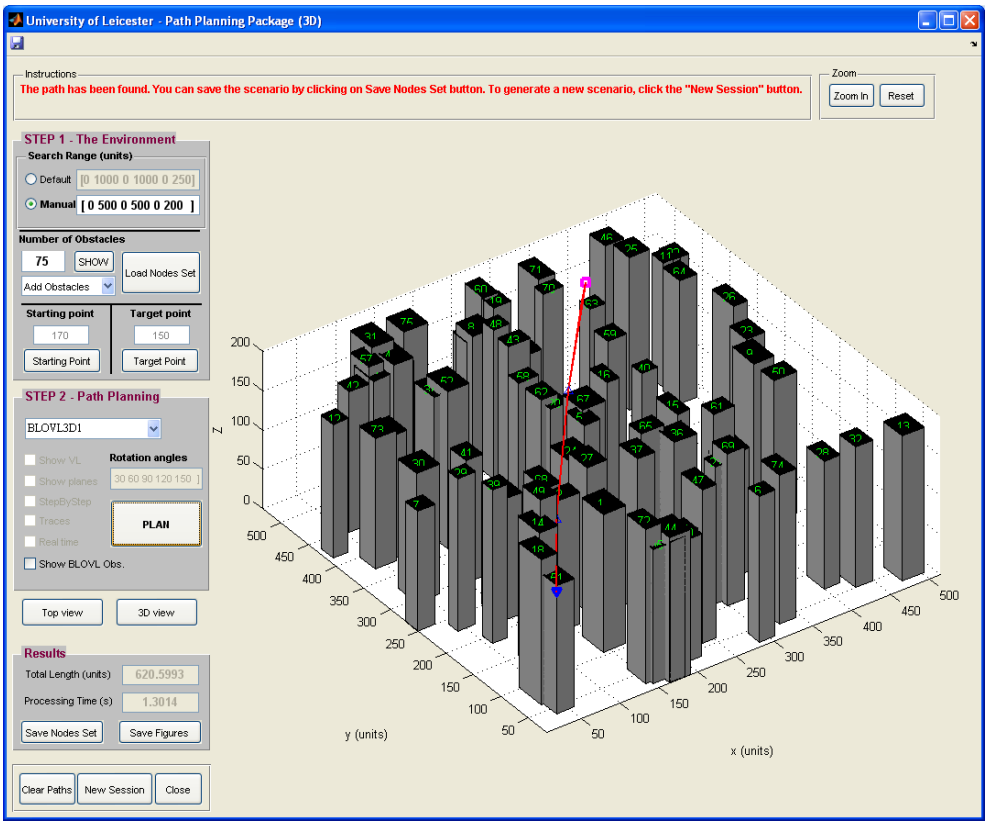


Figure 5.25: The 3D path planned by BLOVL3D1 algorithm.

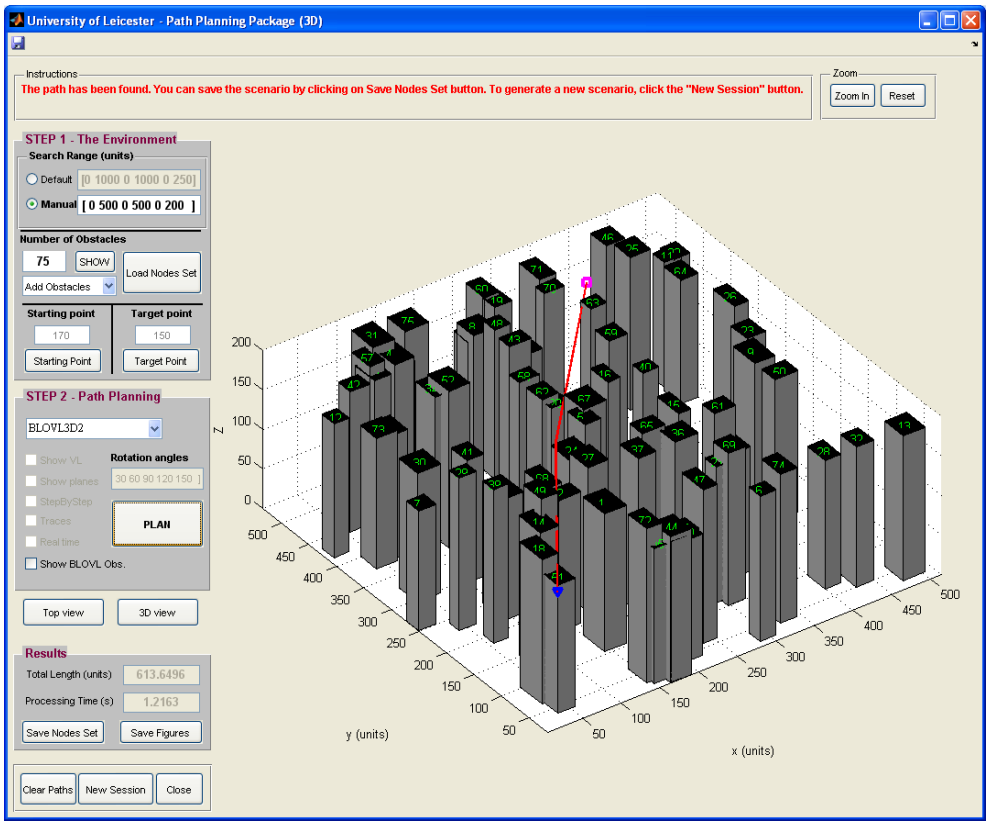


Figure 5.26: The 3D path planned by BLOVL3D2 algorithm.

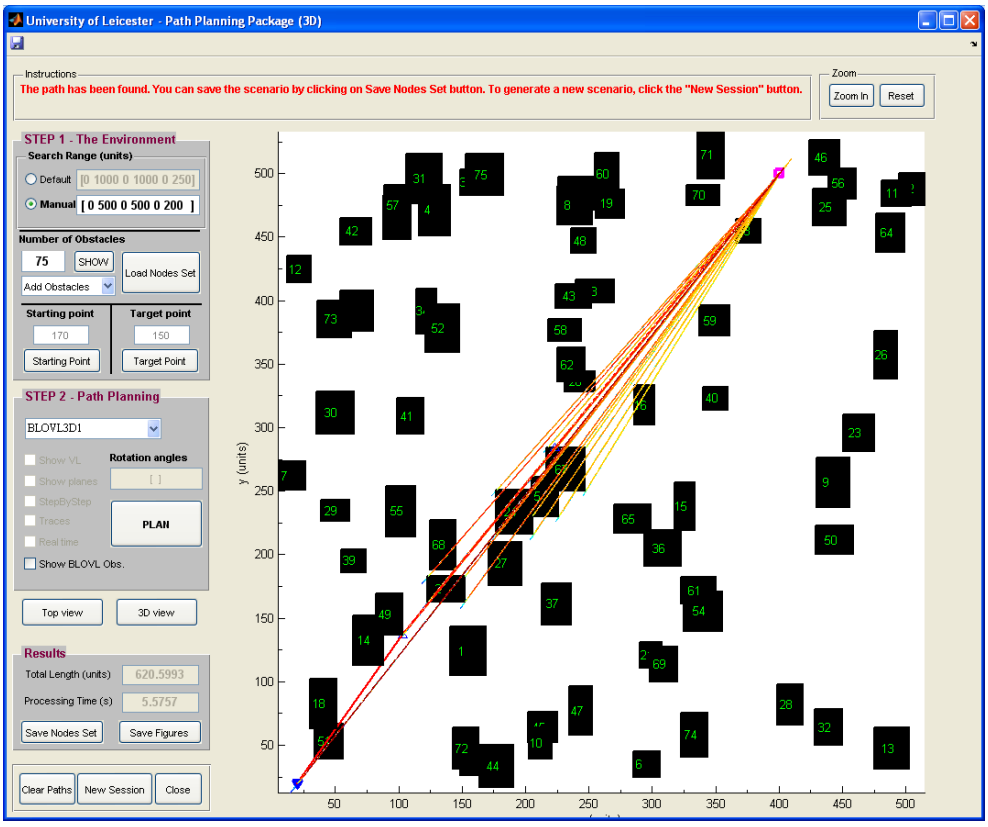


Figure 5.27: The planes generated by BLOVL3D1 are seen from top view

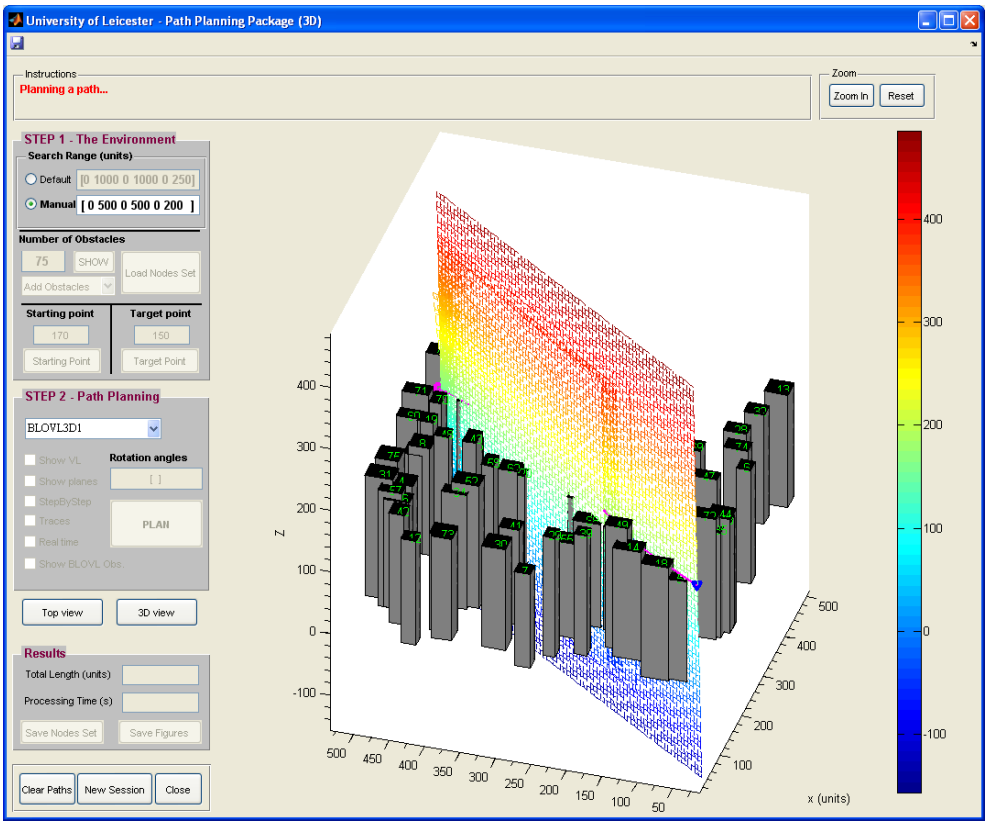


Figure 5.28: The planes generated by BLOVL3D1 are in 3D view.

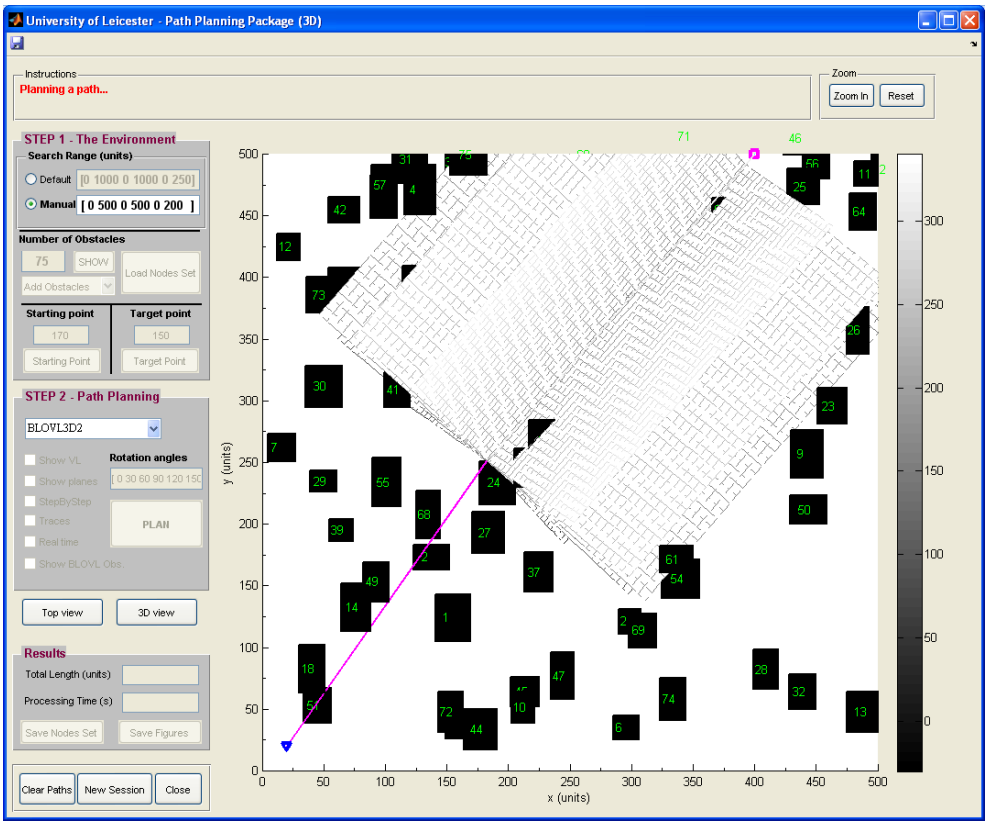


Figure 5.29: The planes generated by BLOVL3D2 are viewed from top

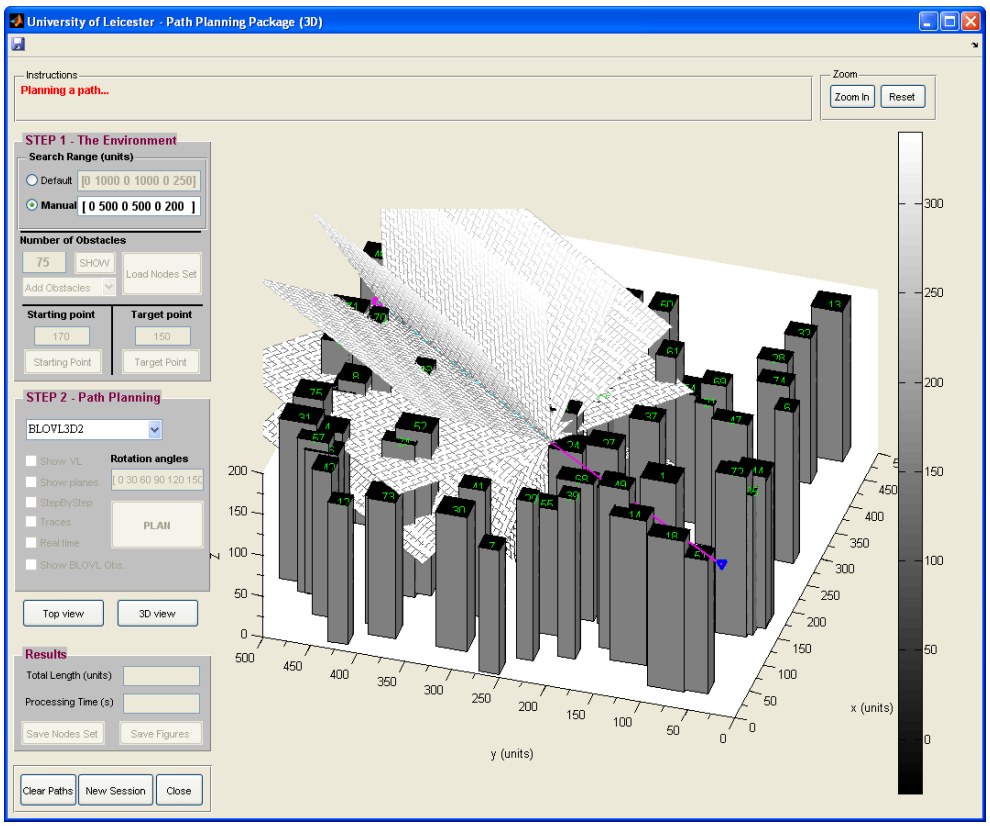


Figure 5.30: The planes generated by BLOVL3D2 are viewed in 3D.

The results of the planned path such as the path length and its computation time can be seen in a frame called *Results*, which is located at the lower left part of the GUI as shown in Fig. 5.31. The scenario can also be saved to be used in the future.

Figure 5.31: The *Results* frame showing the information about the planned path.

In addition, a specific area of the search space can be zoomed-in so that the selected area can be seen more clearly. This is done by pressing the *Zoom In* button, located at the GUI's top right. A new 3D path planning session can be started by pressing the *New Session* button which is situated at the left bottom of the GUI, and the same afore-mentioned procedures have to be performed in order to find a 3D collision-free path.

5.6 Conclusion

The software package that has been developed at the University of Leicester consists of two Graphical User Interfaces (GUIs), which are called 2D GUI and 3D GUI. Both GUIs make the path finding based on the Visibility Graph method in 2D or 3D scenarios, respectively easier as the path planning process is done systematically.

The purposes of the software package are to validate the effectiveness of the proposed algorithms visually and to realise and present the proposed algorithms in an intuitive way as the package was designed to be user-friendly. Furthermore, the GUIs can be operated with minimal guidance and practice, equipped with step-by-step instructions, which are located at the top of the GUIs.

The 2D GUI can be used for real time path planning in 2D using the proposed 2D path planning algorithms i.e. BLOVL. One can also use the 3D GUI that has been specifically designed to plan 3D paths in 3D scenarios. The 3D GUI combines both the 3D algorithms i.e. BLOVL3D1 and BLOVL3D2.

The relevant information about the planned path such as path length and processing time can be seen in the GUIs. In addition, the scenarios, which were used to generate collision-free path can be saved for future use.

Chapter 6

Conclusions and Future Work

Briefly, path planning involves a problem of finding a safe path from a starting point to a target point. There are three criteria for path planning; computational efficiency, path optimality and completeness.

These criteria have to be considered before any path planning method/algorithm is designed. A path planning method which is computationally efficient is capable of planning in real-time in dynamic environments while a path planning method that produces an optimal path can save the vehicle's fuel, reduce the potential risks and prolong the vehicle's life cycle. On the other hand, a path planning method that holds the completeness criterion will find a path if one exists.

This chapter discusses the works that have been undertaken in this thesis on path planning, considering the above-mentioned criteria. The next section discusses and concludes the developed path planning algorithms including the path planning software package. The last section proposes possible extensions of the work that have been developed in this thesis.

6.1 Conclusions

6.1.1 Path Planning in 2D environments

Two path planning algorithms in two-dimensional (2D) environments have been proposed in Chapter 3. The first algorithm, *Core* is used to find a path through a set of obstacles in the environment represented by the configuration space (*C*-space). A base line (BL), which is defined in the *Core* algorithm, is a line segment that connects the

starting point p_{start} and target point p_{target}. Its purpose is to determine a set of obstacles O_{Core} that will be used for path calculation. As O_{Core} contains a relatively small number of obstacles, the visibility lines (VL) network can be built in a relatively short time.

Core then plans a path based on the VL network using Dijkstra's algorithm. *Core*, however, produces a path that might not be collision-free. Thus another algorithm called Base Line Oriented Visibility Line (BLOVL) has been proposed. As a matter of fact, *Core* is a part of BLOVL, which runs iteratively. BLOVL checks the visibility between two consecutive waypoints i.e. the local starting point (u_{start}) and the next waypoint (u_{target}), of the previously planned path, at each iteration. *Core* will be called if those two waypoints are blocked by obstacles. This procedure guarantees that the resulting path is collision-free. In order to further accelerate BLOVL's computation time, base line (BL) with limited range has been introduced.

Simulation results showed that BLOVL is computationally efficient in generating a collision-free path, suitable for a real-time path planning. Additionally the planned paths were similar with those of the VL method. On the other hand the proposed algorithms also hold the completeness criterion as the path planning runs iteratively until the target point is reached.

6.1.2 Path Planning in 3D environments

Path planning in a 3D environment is necessary to ensure that the path has a shorter distance in comparison with a 2D path. This in turn saves the UAV's fuel/energy, increases its endurance, prolongs its life cycle and minimises the exposure to possible risks.

A 3D path planning algorithm i.e. BLOVL3D2, which are based on BLOVL algorithm, have been proposed in Chapter 4 using rotational plane approach. Several sub-algorithms including *BasePlane*, *Rotate3D*, *FindIntersection* and BLOVL3D1 have also been proposed. *BasePlane* is used to create a local base plane on which a visibility lines network will be built and consequently a path will be planned. On the other hand, *Rotate3D* rotates the local plane at predefined angles. *FindIntersection* calculates the intersection points between a rotated plane and obstacles while the BLOVL3D1 algorithm finds a path on the rotated local plane.

BLOVL3D2, which combines *BasePlane*, *Rotate3D*, *FindIntersection* and BLOVL3D1, has been simulated using different numbers of rotation angles and obstacles in random scenarios. The results of the simulations have shown that a higher number of rotation angles leads to a relatively better 3D path in terms of path's length. Also, the simulations show that the BLOVL3D2 algorithm takes longer time to plan a path in environment with higher number of obstacles. The relationship between the number of obstacles and the average computation time is slightly non-linear. In terms of path lengths, BLOVL3D2 plans longer path in environments with higher number of obstacles.

6.1.3 Path Planning Package for 2D and 3D environments

A path planning software package that has been developed at the University of Leicester using Matlab has been introduced and demonstrated in Chapter 5. The purpose of the package is to check the validity of the proposed algorithms visually and to implement and present the proposed algorithms in an intuitive way.

There are two Graphical User Interfaces (GUIs) in the package, in which the first GUI is used to execute the BLOVL algorithm, while the second executes the BLOVL3D2. In developing the package, several features have been incorporated into the GUIs as follows:

- step-by-step instruction, which is located at the top of the GUIs, is provided.
- all objects in the GUIs such as editable boxes, push buttons etc. are arranged in a chronological order.
- most of the objects are grouped into a number of frames, which are located at the left hand side of the GUI. Putting them in one area makes them easy to be seen and handled.
- the movement of the UAV is visually animated in 2D GUI.
- there is an option that allows the user to view the traces of a path throughout a path planning session.
- Regarding the 3D GUI, the rotational plane can be displayed in the provided axis.
- related information about the planned path such as the path length and the processing time are displayed.

- the scenarios can be saved for future use.

6.2 Future work

This section briefly addresses the proposed future work to extend the current study. The BLOVL and BLOVL3D2 algorithms that have been explained in Chapters 3 and 4, respectively, assume that the obstacles' sizes and positions in the *C*-space are known accurately. Future work should consider uncertainties in the obstacles' sizes and positions. As VL-based methods produce waypoints that pass through the obstacles nodes, considering uncertainties would guarantee the resultant paths are in the free region of *C*-space.

Also, the proposed algorithms assume that all obstacles are static. However, in the real scenario, obstacles might move from one place to another. Hence it is worth considering moving obstacles in designing a path planning algorithm in the future.

Additionally, the proposed 2D and 3D path planning algorithms only consider a single UAV. Missions executed by multiple UAVs achieve better and faster results; therefore this would be an ideal area of development in future.

Also there were no path smoothing and path tracking included in the proposed algorithms. As the paths are formed by piece-wise linear segments which have abrupt heading changes near waypoints, the UAV might not be able to traverse such paths. In order to overcome that drawback, path smoothing or path tracking would be another topic to explore for future work.

Finally there was no consideration in the ascending/descending angle of the path in the BLOVL3D2 algorithm. Thus, it is hoped that future work would take this aspect into account by incorporating angle restrictions based on the kinematics constraints of a particular UAV.

References

[1] http://www.richard-seaman.com
/Aircraft/AirShows/Nellis2006/Highlights/Predator2006.jpg

[2] W. A. Kamal. Safe trajectory planning techniques for autonomous air vehicles. PhD Thesis, 2005.

[3] http://www.theregister.co.uk/2006/08/30/uav_project

[4] J. S. Bellingham, M. Tillerson, A. G. Richards and J. P. How. Multi-Task Allocation and Path Planning for Cooperating UAVs. In *Proceedings of Conference on Cooperative Control and Optimization*, 2001.

[5] R. Frampton, UAV autonomy, In *MOD Codex Journal*, Issue 1 Summer, 2008, [Online]
www.science.mod.uk/codex/Issue1/Journals/documents/Issue1_2Journals_UAV_autonomy.pdf .

[6] http://www.theuav.com/

[7] Nilsson, N.J. A mobile automaton: An application of artificial intelligence techniques. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 509-520, 1969.

[8] Thompson,A.M., The navigation of the JPL robot. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 749- 57, 1977.

[9] T. Lozano- Perez and M. A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Contmum*. *ACM*, 22, pages 560-570, 1979.

[10] A. Tokuta. Extending the VGRAPH algorithm for robot path planning. In *International Conference in Central Europe on Computer Graphics and Visualization*. 1998.

[11] A. Louchene, N. E. Bouguechal, A. Dahmani, S. Yahiaoui, and S. Merrouchi. Automated guided vehicle path planning without obstacles expansion. In *Proceedings*

of the IEEE International Conference on Control Applications, pages 1333-1337, 1998.

[12] J. Giesbrecht and Defence R&D Canada. Path planning for unmanned ground vehicles. *Technical Memorandum DRDC Suffield TM 2004-272,* 2004.

[13] R. Brooks and T. L.-Perez. A subdivision algorithm in configuration space for findpath with rotation. In *Proceedings of the 8th International Conference on AI,* pages 799-806, 1983.

[14] D. Zhu and J. Latombe. Constraint reformulation in hierarchical path planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 1918-1923, 1995.

[15]. H. Samet. An overview of quadtrees, octrees and related hierarchical data structures. *NATO ASI Series*, F40, pages 187-260. 1988.

[16]. H. N. T. Naniwa and S. Arimoto. A quadtree-based path-planning algorithm for a mobile robot. In *Robotic Systems*, pages 668-681, 1990.

[17] D. M. Coleman and J. T. Wunderlich. O3: An optimal and opportunistic path planner (with obstacle avoidance) using voronoi polygons. In *IEEE International Workshop on Advanced Motion Control*, pages 371-376. 2008

[18] Y. Qu, Q. Pan and J. Yan. Flight path planning of UAV based on heuristically search and genetic algorithms. In P*roceedings of the 31st Annual Conference of Industrial Electronics Society of IEEE,* pages 45-49, 2005.

[19] Q. Xiao, X. Gao, X. Fu and H. Wang. New local path re-planning algorithm for unmanned combat air vehicle. In *Proceedings of the 6th World Congress on Intelligent Control and Automation,* pages 4033-4037, 2006.

[20] A. Sud, E. Andersen, S. Curtis, M. C. Lin, and D. Manocha. Real-time path planning in dynamic virtual environments using multi-agent navigation graphs. In *IEEE Transactions On Visualization And Computer Graphics,* pages 526-538, 2008.

[21] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation,* pages 500-505, 1985.

[22] Y. Kitamura, T. Tanaka, F. Kishino and M. Yachida. 3-D Path planning in a dynamic environment using an octree and an artificial potential field. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 474-481, 1995.

[23] K. H. Yong and N. Ahuja. A potential field approach to path planning. In *IEEE Transactions on Robotics and Automation*, pages 23-32, 1992.

[24] A. Poty, M. Melchior and A. Oustaloup. Dynamic path planning for mobile robots using fractional potential field. In *First International Symposium on Control, Communications and Signal Processing*, pages 557-561, 2004.

[25] I. Hasircioglu, H. R. Topcuoglu and M. Ermis. 3-D path planning for the navigation of unmanned aerial vehicles by using evolutionary algorithms. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 1499-1506, 2008.

[26] P. B Sujit and R. Beard. Multiple UAV path planning using anytime algorithms. In *American Control Conference*, pages 2978-2983, 2009.

[27] Y. Kuwata and J. How. Three dimensional receding horizon control for UAVs. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2004.

[28] N. Vandapel, J. Kuffner and O. Amidi. Planning 3-D path networks in unstructured environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4624-4629, 2005.

[29] G. Dudek and M. Jenkin. Computational principles of mobile robotics. Cambridge University Press, Cambridge, UK, 2000.

[30] F. Mitch, Z. Tu, L. Stephens and G. Prickett. Towards true UAV autonomy. In *Proceedings of the IEEE International Conference on Information, Decision and Control*, pages 170-175, 2007.

[31] S. Karim, C. Heinze, and S. Dunn. Agent-based mission management for a UAV. In *Proceedings of the IEEE International Conference on Intelligent Sensors, Sensor Networks & Information Processing*, pages 481-486, 2004.

[32] C. Hai. A survey of autonomous control for UAV. In *Proceedings of the IEEE International Conference on Artificial Intelligence and Computational Intelligence*, pages 267-271. 2009.

[33] S. Eric. Evolution of a UAV autonomy classification taxonomy. In *Proceedings of the IEEE International Conference on Aerospace*, 2007.

[34] S. M. LaValle. Planning Algorithms, Cambridge University Press, 2006.

[35] S. J. Russell and P. Norvig. Artificial intelligence: A modern approach, 2nd Edition, Prentice Hall, Upper Saddle River, N.J., 2003.

[36] K. Yang and S. Sukkarieh. An enhanced optimization approach for generating smooth robot trajectories in the presence of obstacles. In *Proceedings of the 1995 IEEE European Chinese Automation Conference*, pages 263–268, 2008.

[37] J. Zhang and A. Knoll. Real time continuous curvature path planning of UAVs in cluttered environments. In *Proceedings of the 5th IEEE International Symposium on Mechatronics and its Applications*, pages 1-6. 2008.

[38] S. Aydin and H. Temeltas. A novel approach to smooth trajectory planning of mobile robot. In *IEEE International Workshop on Advanced Motion Control*, pages 472–477, 2008.

[39] H. K. Sung and R. Bhattacharya. Multi-layer approach for motion planning in obstacle rich environments. In *Conference and Exhibit of AIAA Guidance, Navigation and Control*, 2007.

[40] M. Shanmugavel, A. Tsourdos, B. White and R. Z Bikowski. Co-operative path planning of multiple UAVs using dubin paths with clothoid arcs. In *Control Engineering Practice*, Elsevier, 2009.

[41] H. Choset, G. Kantor, W. Burgard, L. Kavraki and S. Thrun. Principles of robot motion: Theory, algorithms, and implementations, The MIT Press, 2005.

[42] R. Siegwart and I. R. Nourbakhsh. Introduction to autonomous mobile robots, Bradford Company, Scituate, MA, USA, 2004.

[43] A. B. Doyle. Algorithm and computational techniques for robot path planning. PhD Thesis, 1995.

[44] B. Faverjon. Object level programming using an octree in the configuration space of a manipulator. In *Proceedings of the IEEE International Conference* on *Robotics and Automation San Francisco*, pages 1406-1412, 1986.

[45] J- C. Latombe. Robot motion planning, Kluwer Academic Publisher, 1991.

[46] H. Samet. Neighbor finding techniques for images represented by quadtrees. In *Computer Graphics and Image Processing,* pages 35-57, 1982.

[47] J. Kim and D. Kim. Visibility graph path planning using a quadtree, In *Proceedings of The 9th POSTECH-KYUTECH Joint Workshop On Neuroinformation,* pages 37-38, 2009.

[48] D. Chen, R. Szczerba, and J. Uhran. Planning conditional shortest paths through an unknown environment: A framed-quadtree approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 33-38, 1995.

[49] Y. Alex, S. Anthony, S. Singh and B. L. Barry. Framed-quadtree path planning for mobile robots operating in sparse environments. In *Proceedings, IEEE Conference on Robotics and Automation,* 1998.

[50] T. K. Priva and K. Sridharan. An efficient algorithm to construct reduced visibility graph and its FPGA implementation. In *Proceedings of I7th International Conference on VLSI Design,* pages 1057-1062, 2004.

[51] Y. Wang, G. S. Chirikjian. A new potential field method for robot path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 977-982, 2000.

[52] D. Chen, L. Zhan, X. Chen, Mobile robot path planning based on behaviour information potential field in unknown environments. In *Proceedings of the IEEE International Conference on Robotics and Biomimetics,* pages 683–687, 2004.

[53] D. Parsons and J. Canny. A motion planner for multiple mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 8-13, 1990.

[54] M. Jun and R. D'Andrea. Path planning for unmanned aerial vehicles in uncertain and adversarial environments, cooperative control: Models, applications and algorithms, Kluwer, 2002.

[55] T. Arney. An efficient solution to autonomous path planning by approximate cell decomposition. In *Proceedings of the IEEE International Conference on Information and Automation for Sustainability*, pages 88-93, 2007.

[56] N. S. V. Rao. Algorithmic framework for learned robot navigation in unknown terrains. In *Journal of Computer*, 22(6), pages 37-43, 1989.

[57] T. Guang, M. Bertier and A-M Kermarrec. Visibility graph-based shortest-path geographic routing in sensor networks. In *Proceeding of the IEEE International Conference on Computer Communications*, pages 1719-1727, 2009.

[58] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Computer Science Dept., Iowa State University, October 1998.

[59] J. T. Schwartz and M. Sharir. A survey of motion planning and related Geometric algorithms. *Artificial Intelligence*, MIT Press, pages 157-169, 1988.

[60] S. Akishita, T. Hisanobu and S. Kawamura. Fast path planning available for moving obstacle avoidance by use of laplace potential. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pages 673-678, 1993.

[61] F. Lingelbach. Path planning using probabilistic cell decomposition. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 467-472, 2004.

[62] L. Zhang, J. Y. Kim and D. Manocha. A hybrid approach for complete motion planning. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7-14, 2007.

[63] J. O. Kim and P. K. Khosla. Real-time obstacles avoidance using harmonic potential functions. *IEEE Transactions on Robotics and Automation,* 8(3), pages 338-349, 1992.

[64] S. A. Bortoff. Path planning for UAVs. In *Proceedings of the American Control Conference Chicago Illinois*, pages 364-368, 2000.

[65] S. Garrido, L. Moreno, M. Abderrahim and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2376-2381, 2006.

[66] P. Khosla and R. Volpe. Superquadratic artificial potentials for obstacle avoidance and approach. In *Proceedings of the IEEE Conference on Robotics and Automation,*1988.

[67] L. E Kavraki, P. Svestka, J-C. Latombe and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE Transactions on Robotics and Automation,* 12(4) pages 566-580, 1996.

[68] B. J. Oommen, S. Iyengar, N. S. V. Rao and R. L. Kashyap. Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case. In *IEEE Journal of Robotics and Automation*, RA-3(6), pages 672-681, 1987.

[69] C. Godsil and G. Royle. Algebraic Graph Theory, Springer, 2001.

[70] A. Louchene, N-E. Bouguechal, A. Dahmani, S. Yahiaoui and S. Merrouchi. Automated guided vehicle path planning without obstacles expansion, In *Proceedings of the IEEE International Conference on Control Applications*, pages 1333-1337, 1998,

[71] K. Jiang, L .D. Seneviratne and S.W. E. Earle. Finding the 3D shortest path with visibility graph and minimum potential energy. In *Proceedings of the IEE/RSJ International Conference on Intelligent Robots and Systems*, pages 679-684, 1993.

[72] D. Wooden and M. Egerstedt. Oriented visibility graphs: low-complexity planning in real-time environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2354-2359, 2006.

[73] E. F. Moore. The shortest path through a maze. *Proceedings of an International Symposium on the Theory of Switching*. Cambridge: Harvard University Press, pages 285-292, 1957.

[74] H. P. Huang and S. Y. Chung. Dynamic visibility graph for path planning. In *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2813-2818, 2004.

[75] J.A Janet, R. C. Luo and M. G. Kay. The essential visibility graph: An approach to global motion planning for autonomous mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1958-1963, 1995.

[76] G. Song, S. Thomas and N. M. Amato. A general framework for PRM motion planning. In *Proceedings of the IEEE International Conference on Robotics & Automation*, pages 4445-4450, 2003.

[77] K. Belghith, F. Kabanza, Hartman and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2372-2377, 2006.

[78] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki. Sampling-based roadmap of trees for parallel motion planning, In *IEEE Transactions on Robotics*, 21(4), pages 597-608, 2005.

[79] T. Siméon, J-P. Laumond and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. In *Advanced Robotics* 14(6). Pages 477-494, 2000.

[80] P. O. Pettersson and P. Doherty. Probabilistic roadmap based path planning for an autonomous unmanned aerial vehicle. In *Proceedings of the Workshop on Connecting Planning and Theory with Practice. 14th International Conference on Automated Planning and Scheduling*, 2004.

[81] Y. Tian, L. Yan, G. Y. Park, S. H. Yang, Y. S. Kim and S. R. Lee, C-Y. Lee. Application of RRT-based local path planning algorithm in unknown environment. In *International Symposium on Computational Intelligence in Robotics and Automation*, pages 456-460, 2007.

[82] J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings of the IEEE International Conference on Robotics &Automation*, pages 2200-2205, 2003.

[83] N. A. Melchior and R. Simmons. Particle RRT for path planning with uncertainty. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1617-1624. 2007.

[84] S. Kamio and H. Iba. Random sampling algorithm for multi-agent cooperation planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1265-1270, 2005.

[85] R. Pepy and A. Lambert. Safe Path Planning in an Uncertain-Configuration Space using RRT. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5376-5381, 2006.

[86] M. Shao and J. Y. Lee. Development of autonomous navigation method for non-holonomic mobile robots based on the generalized voronoi diagram. In *Proceedings of the IEEE International Conference on Control Automation and Systems*, pages 309-313, 2010.

[87] Q. Zhang and X. Wang. Global path planning method in uncertain environment. In *Proceedings of the IEEE International Conference on Control Applications*, pages 2725-2730, 2006.

[88] P. Bhattacharya and M. L. Gavilova. Voronoi diagram in optimal path planning. In *Proceedings of the IEEE International Symposium on Voronoi Diagrams in Science and Engineering*, pages 38-47, 2007.

[89] R. Wein, J. P. Van and D. Halperin. The visibility-voronoi complex and its application. In *Computational Geometry: Theory Applications* 36(1), pages 66-87, 2007.

[90] R. Daily and D. M. Bevly. Harmonic potential field path planning for high speed vehicles. In *American Control Conference*, pages, 4609-4614, 2008.

[91] T. Ishida. Real-time search for autonomous agents and multiagent systems. In *Autonomous Agents and Multi-Agent Systems*. Kluwer Academic Publishers, pages 139-167, 1998.

[92] D. Glavaski, M. Volf and M. Bonkovic. Robot motion planning using exact cell decomposition and potential field methods. In *Proceedings of the WSEAS International Conference on Simulation, Modelling and Optimization*, pages 126-131, 2009.

[93] P. Broz. Path planning in combined 3D grid and graph environment. In *Proceedings of the 10th Central European Seminar on Computer Graphics*, 2006.

[94] M. N. Bygi and M. Ghodsi, 3D visibility graph. In *12th CSI Computer Conference,* 2006.

[95] C. H. Chung and G. N Saridis. Path planning for an intelligent robot by the extended VGraph algorithm. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pages 544-549, 1989.

[96] P. Hart. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, pages 100-107, 1968.

[97] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numerische Mathematik 1*, pages 269-271, 1959.

[98] R. J Szczerba, D. Z. Chen and K. S. Klenk. Minimum turns/shortest path problems: A framed-subspace approach. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 398-403, 1997.

[99] J. L. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artefacts. In *International Journal of Robotics Research*, 18(11), pages 1119-1128, 1999.

[100] K. G. Joll, R. S. Kumar and R. Vijayakumar. A Bezier curve based path planning in a multi-agent robot soccer system without violating the acceleration limits. In *Elsevier Journal of Robotics and Autonomous Systems*, 57(2009), pages 23–33, 2009.

[101] K. Yang and S. Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. In *IEEE Transactions on Robotic*, 26(3), pages 561-568, 2010.

[102] T. Kito, J. Ota, R. Katsuku, T. Mizuta, T. Arai, T. Ueyama and T. Nishiyama. Smooth path planning by using visibility graph-like method. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3770-3775, 2003

[103] G. Yang and V. Kapila. Optimal path planning for unmanned air vehicles with kinematic and tactical constraints. In *Proceedings of the IEEE Conference on Decision and Control*, pages 1301-1306, 2002.

[104] E. P. Anderson, R. W. Beard and T. W. McLain. Real-time dynamic trajectory smoothing for unmanned air vehicles. In *IEEE Transactions on Control Systems Technology*, 13(3), pages 471-477, 2005.

[105] A. Richards and J. P. How. Aircraft Trajectory Planning with Collision Avoidance Using Mixed Integer Linear Programming. In *Proceedings of the American Control Conference*, pages 1936-1941, 2002.

[106] L. Labakhua. Smooth trajectory planning for fully automated passengers vehicles–spline and clothoid based methods and its simulation. In *Proceedings of the International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 89-96, 2006.

[107] K. Yang and S. Sukkarieh. Planning continuous curvature paths for UAVs amongst obstacles. *In Proceedings of the Australasian Conference on Robotics and Automation (ACRA)*, 2008.

[108] J-W. Choi, R. E. Curry and G. H. Elkaim. Continuous curvature path generation Based on bezier curve for autonomous vehicles. In *IAENG International Journal of Applied Mathematics* 40(2), 2010.

[109]. R. H. Bartels, J. C. Beatty and B. A. Barsky. An introduction to splines for use in computer graphics and geometry modeling. M. Kaufmann Publishers, 1987.

[110] M. Bak, N. K Poulsen and O. Ravn. Path following mobile robot in the presence of velocity constraints. In *Technical Report, Informatics and Mathematical Modelling, Technical University of Denmark*, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2001.

[111] R. Omar and D-W Gu. Visibility line based methods for UAV path planning. In *Proceedings of the International Conference on Control, Automation and Systems (ICCAS-SICE)*, pages 3176-3181, 2009.

[112] R. Omar and D.-W. Gu. 3D path planning for unmanned aerial vehicles using visibility line based method. In *Proceedings of the International Control on Informatics in Control, Automation and Robotics*, pages 80-85, 2010.

[113] M. Kothari, I. Postlethwaite and D.-W. Gu. Multi-UAV path planning in obstacle rich environments using rapidly-exploring random trees. In *Proceedings of the IEEE Conference on Decision and Control, held jointly with the 28th Chinese Control Conference*, pages 3069-3074, 2009.

[114] P. Zarchan. Tactical and strategic missile guidance.In *Progress in Astronautics and Aeronautics (4th Ed.)*, 176, AIAA, 2002.

[115] Y. Baba and H. Takano. Robust flight trajectory tracking control using fuzzy logic. In *Proceedings of the 8th ISDG&A*, Maastricht, pages 68–75, 1998.

[116] A. G. Richards, J. P. How, T. Schouwenaars and E. Feron. Plume avoidance maneuver planning using mixed integer linear programming. In *Proceedings of Conference of AIAA Guidance, Navigation and Control*, 2001.

[117]Y. Kuwata and J. P. How. Robust cooperative decentralized trajectory optimization using receding horizon MILP. In *Proceedings of American Control Conference*, pages 522-527, 2007.

[118] I. K. Nikolos, N. C. Tsourveloudis and K. P. Valavanis. Evolutionary algorithm based off-line path planner for UAV navigation. *Automatika Journal*, 42(2001) 3-4, pages 143-150, 2001.

[119] D. Jia. Parallel evolutionary algorithms for UAV path planning. In *Proceedings of the AIAA 1st Intelligent Systems Technical Conference*, 2004.

[120] Y. Dadi, Z. Lei, R. Rong and X. Xiaofeng. A new evolutionary algorithm for the shortest path planning on curved surface. In *Proceedings of IEEE Conference on Computer-Aided Industrial Design and Conceptual Design*, pages 1-4, 2007.

[121] J. O'Rourke. Computational geometry in C (2nd Edition). Cambridge University Press, 1998.

[122] A. C. Kermode. Mechanics of flights (10th Edition). Prentice Hall, 1996.