Deanship of Graduate Studies

Al-Quads University

Finding the best Altitude of motion for a system of UAVs moving among 3D environments containing HDFPOs.

Amjad Ja'far Mustafa Khalil

M. Sc. Thesis

Jerusalem-Palestine

1443/2022

Finding the best Altitude of motion for a system of UAVs moving among 3D environments containing HDFPOs.

Prepared by:

Amjad Ja'far Mustafa Khalil

M.Sc.: Computer Science-Al-Quds University-Palestine

Supervisor: Dr. Abdullah Kamal

A thesis Submitted in Palestine Fulfillment of the Requirements for the Degree of Master of Computer / Department of Computer Science/ Faculty of Graduate Studies at Al-Quds University

1443/2022

Deanship of Graduate Studies

Al-Quads University

Computer Science Department

Thesis Approval

Finding the best Altitude of motion for a system of UAVs moving among 3D environments containing HDFPOs.

Prepared by: Amjad Ja'far Mustafa Khalil

Registration No.: 21312706

Supervisor: Dr. Abdullah Kamal

Master Thesis submitted and accepted, Date 10/03/2022

1- Head of committee: Dr. Abdullah Kamal Signature: .....

2- Internal Examiner: Dr. Nidal Alkafri Signature: .....

3- External Examiner: Dr. Iyad Hashlamon Signature: .....

Jerusalem-Palestine

1443/2022

Dedication

I am dedicating this thesis to my beloved people who have meant and continue to mean so much to me. My father who is no longer of this world, your memories continue to regulate my life, may Allah (SWT) grant you Jannah Firdaws. Amen.

Also, I dedicate this to my mother, my wife, my sister Reem and my family, who loved me, raised me, and taught me to seek knowledge. Last but not the least I am dedicating this to my daughter Shahd, whose love for me knew no bounds and, who taught me the value of a smile, I will make sure your smile goes on as long as I shall live.

Thanks for all

Declaration

I certify that this thesis submitted for the degree of master, is the result of my own work and research except for the acknowledged parts. This thesis or any part of the same has not been submitted to any university or institution.

Signed 

Amjad Jafar Mustafa Khalil

Date: 10/03/2022

Acknowledgments

First and foremost, I thank Allah (SWT) for letting me live to see this thesis through. I am forever indebted to Dr. Abdullah Kamal for his unwavering support, encouragement, and patience through this process.

Special mention for All my Doctors, I can never pay you back for all the help you have provided me, the experience you have helped me while studying my master's degree and the precious time you spent making sure my thesis is always on track. I hope you find some satisfaction in this modest paper.

I would like to give my sincere thanks and gratitude to prof. Dr. Eng. Barbel Mertsching, the director of Get Lab. At Paderborn University that gives assistance and support for my master research during my attendance at Paderborn University through DAAD scholarship.

Thank my committee members and colleges who were more than generous with their expertise and precious time.

I would like to give special thanks for the support and love of my family, who supported me financially and morally.

Finally, yet importantly thanks go to Al-Quds University, especially for the faculty of high studies, Computer science department.

Abstract

The problem of motion planning in **three-dimensional space** (**3D-space**) is still a big challenge for researchers and **Artificial Intelligent** (**AI**) scientists. A rapid increase in the numbers of aerial vehicles flying among a variety of environments. While the promising industry of drones and quadrotors systems, that interning many services and jobs indeed, makes a huge need to find new solutions that organize and autonomously control the movement of such systems. Our goal is to have a motion controlled over many spaces surrounding us like the big cities.

In this paper, we Examine how obstacle density affects the trajectories of aerial robots, to present a path planning approach in a 3D environment that contain **High Density Fixed Polygonal Obstacles** (**HDFPOs**). We generate different random maps in **2-dimensions** (**2D**), and then convert them to 3D-environment by distributing multi sized polygonal obstacles. Our simulation contains more than one target point and five aerial robots (quadrotors) under Unity 3D simulation tool. We optimized our 3D environment into several 2D intersection maps, and then apply the **A Star searching algorithm** (**A***) and weighted A* search method. It treats our 2D known maps generated from the intersection of many planes, with different altitudes among the 3D fixed polygonal obstacles. Our aim is to generate collision-free trajectories for the multiagent system on each experiment. By executing this approach for a large number of maps and on different elevation, obviously these planes intersect the generated 3D environment. Repeating this experiment by resizing the volume density of the polygonal obstacles added to our maps, we collect a huge amount of data. Then we analyze the collected data using a curve fitting approach to compare and find the best altitude that shorten the flying path. The results obtained will help a system of **Unmanned Aerial Vehicles** (**UAVs**) to reach their targets within the shortest distance and in less time. All experiments executed on different ratios of polygonal obstacle densities. We aim to predict the best altitude of flying in the case of **HDFPOs** environment that achieves the solution of shortest path with respect to the average height of all obstacles.

Keywords: Motion Planning Problem, Autonomous Aerial Robots, Altitude of Flying, High density Polygonal Obstacles, A star, Path Finding.

Content

List of Figures

List of Tables

List of Flow Charts

List of Codes

List of Acronyms

AI	**Artificial Intelligent**
AARs	**Autonomous Aerial Robots**
UAVs	**Unmanned Aerial Vehicles**
HDFPOs	**High Density Fixed Polygonal Obstacles**
2D	**Two Dimensions (refer to dimensional space)**
3D	**Three Dimensions (refer to dimensional space)**
SLAM	**Simultaneous Localization and Mapping**
GPS	**Global Positioning System**
DOF	**Degree of Freedom**
q	**Robot Configuration**
C-space	**Configuration Space**
A*	**A star searching algorithm.**
DWA	**Dynamic Window Approach**
RRTs	**Rapidly exploring Random Trees.**
PRM	**Probabilistic Road Map**
U	**A Potential Field**
BFS	**Breadth-First Search**
DFS	**Depth-First Search**
C#	**C sharp scripting language**
NRW	**North Rhine Westphalia**
DAAD	**German Academic Exchange Service**
IOSU	**impeded operating system unit**
IMU	**Inertial Measurement Unit**
API	**Application Programming Interface**
OAH	**Obstacles Average Height**

Chapter 1

Introduction

1.1. Introduction

In the area of motion planning research. A large number of researchers focus their efforts to answer the following question: How can we enable Ariel robots (quadrotors) to fly autonomously? The answer is clarified by two steps: Firstly, by estimating its state from sensor readings, and secondly, by generating suitable control commands that move it towards its goal [1]. In other words, to generate a collision free path from point A (Start Point) to point B (Target Point) continuously [2]. In a 3D environment that contains high density polygonal obstacles, it is important to determine at what altitude the quadrotors should fly toward its goal. In fact, we are looking for a method that allows us to know what altitude can give us a shorter path between point A and point B. The result of this finding led us to reduce cost of time and enhance performance for the motions. When the previously two mentioned steps are achieved, many systems and services will depend on aerial robots' systems to handle many tasks autonomously. In personal use and entertainment services, for example, imagine that you have an autonomous flying camera that can follow you automatically and take nice photos and videos for you, to examine the roof of your house for any kind of damages after a storm, or inspecting constructions such as bridges or high buildings which can be done easily by Autonomous Aerial Robots (AARs) that handle a camera. Using such a system can save time and money. It is very important in many critical cases where the intervention of a flying robot as a multi-agent system can make a difference. In these cases, such as the time of epidemics and infection spread, disasters or crash accidents, aerial robots could play as a real lifesaver. For example, after an earthquake or tsunami, you can quickly send a team of quadrotors to that place of calamity in order to inspect buildings from above and determine their status, or to go inside buildings in particular if we are not sure if buildings are still safe for humans. If the building could collapse, then it's better to send a flying robot that have advantage on wheel mobile robots that may be stuck very soon by stones or falling cables on the floor. AARs can get visual imagery which is very important as a guideline for rescue teams to come to the

victims and provide needed aid very soon, were many victims could be saved. Another application for autonomous aerial robots is remote farming [3]. where quadrotors can navigate and monitor fields in general, or to test if additional watering is needed or to make sure plants are growing healthy. In Addition, when detecting some diseases, an aerial robot system may spray certain substances on the plants very precisely and exactly with the right amount based on the visual inspections. There are many applications where quadrotors can play a major role such as mapping buildings externally or internally, or in transportation and delivering services. For example, Amazon started such a service where quadrotors deliver packages to customers, but all the current working systems still required skilled human pilot, and the motivating question is how we can increase the autonomy of flying robots in 3D-space if it contains high density obstacles (Figure 1.1).

Figure 1.1: New York city, a real representation of HDFPOs

In this introduction, we start by identifying the research problem statement and the substantial motivations to propose an approach to **AARs** motion planning problem, a 3D-enviornment that contain **High Density Fixed Polygonal Obstacles** (**HDFPOs**). We then describe the main objectives, advantages, and contributions to develop this work.

1.2. Problem definition and overview

Motion planning for autonomous aerial robots in cities that contains high buildings need to verify the suitable altitude for flying. We are motivated by applications relevant to the future of Unmanned Aircraft Systems and traffic management in low-altitude airspace [4]. When a system of cooperative **AARs** linked with a task, then each robot needs to travel from point A to point B. While in between the start point and the target point there are two classified spaces. One is the free space that contains no obstacles where the **AARs** can move freely. And two is the obstacle space where **AARs** are not allowed to move, since space close to lands is occupied by obstacles. Buildings within cities represent static obstacles that most of the cases have a polygonal shape [5]. For our research we take this obstacle space among the 3D static environment in the consideration. The path planning is referred to the generation of a feasible flight path from a start position to a destination, without consideration of dynamic feasibility in our case [6].

Controlling the movement of quadrotors each day become more accurate. It can fly horizontally or vertically or can fly in combination of horizontal and vertical axes at a time. For a 3D environment that has low obstacles density, it's clear that obstacle space to free space ratio will be low. In such a case, most of the motion paths from start points to targets will be straight line or chain of segments. When obstacles occupied more space, it becomes more complicated issue to find a short path between start point and target (Figure 1.2). Although AARs can fly over some obstacles to exceed there occupied unwalkable space,

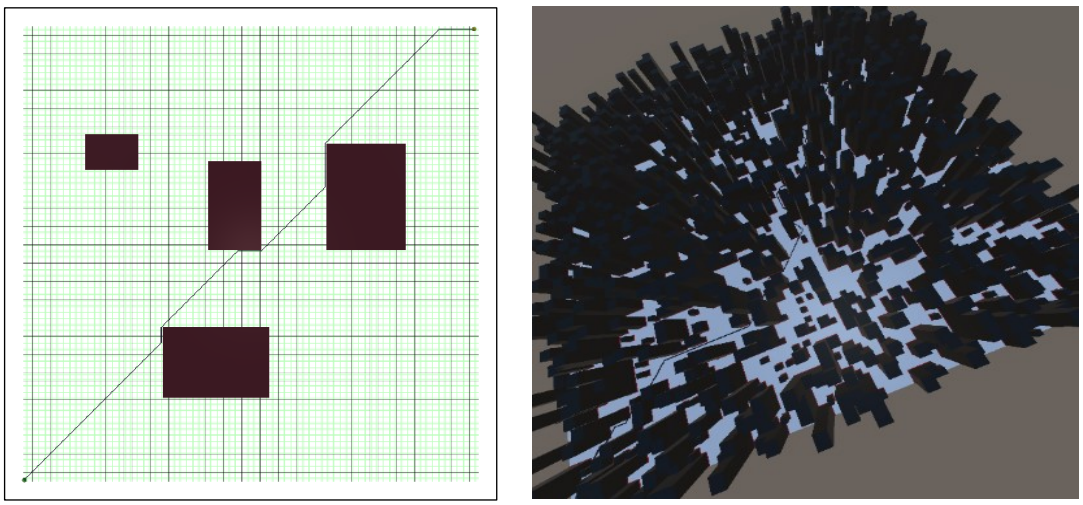


Figure 1.2: Low Density vs HD Obstacles Environment

higher altitude may cause longer paths instead of other paths at lower altitudes. Therefore, there must be acritical point (altitude) satisfies the shortest path we can find, the critical point gives us more information about the most suitable altitude for **AARS** motion in **HDFPOs** environment. For each environment state, we will look if there is a relation between the best altitude that contain the shortest path and the density of obstacles.

1.3. Introduction to path planning and formulation.

The main goal of many researchers in the field of path planning, is to enable flying robots to operate autonomously. Extensive approaches to deal with 3D environments using collected onboard data from sensors and cameras (Depth Cameras, RGB-D) [7]. This data led to control robots' motion [7,8,9]. In real world state, the motion planning problem can be divided to several subproblems to perform the navigation. Starting from localization to 3D mapping, which is a reconstruction of the workspace [9], to path calculation and path execution. We may go further for exploration and target following, or the avoidance of dynamic obstacles in case of dynamic environment. In computational geometry, combining **simultaneous localization And Mapping** (**SLAM**) [7], mean, the computational problem of knowing position and constructing or updating a map of an unknown environment [1], while simultaneously keep tracking of target location within it. Mapping the 3D environment will give a huge number of nodes depending on the resolution of the mapping process, each node represents a reference for **Global Positioning System (GPS)** in real world. The calculation of paths for AARs at these circumstances requires dealing with each node in the map, the complexity of huge amount of data (nodes) makes it a big challenge for researchers to achieve the autonomous motion of areal robots, many researchers concentrate on decomposition strategies to find methods for simplifying the 3D space[ref]. One method of simplification, using a multiresolution grid which greatly reduces memory usage and computation time [10]. Another simplification method in which the continuous map representation contains a set of infinite lines that approximate real-world environmental lines based on a 2D slice of the 3D world [6]. Basically, this transformation from the real world to the map representation is a filter that removes all non-straight data and furthermore extends line segment data into infinite lines that require fewer parameters,

in other words, any motion planning problem in 3D space is simplified to fulfill a 2D motion planning problem, due to the complexity of mapping the 3D space and the requirement of non-affordable resources to do that. Fundamental real-world features that mobile robot map representations do not yet represent well. These continue to be the subject of open research [11].

In other words, for any AARs, the ability to navigate in the environment is important. Avoiding collisions and safe flying comes first, and if AAR has a purpose that is related to specific goals in Robot's environment, it must find those targets. The robot's need to determine its own position in its work space and then to plan a trajectory towards some goal locations in order to navigate in the environment. AARs or any other mobile robot require to SLAM regarding the environment.

With dynamic environment there will be dynamic obstacles, in this case the motion planning consists of two sub problems, global planning problem when we explore a huge map within environment where AAR calculate a path to go from start point to target. And local planning problem while AAR follow that path, we need to be sure the trip will go safe, and no AAR collides with any dynamic obstacle or with other AAR.

Despite that the problem of motion planning is defined in the real-world space, we must recreate that space in simulators, we define it as the **configuration space** (**C-space**). The first step of motion planning problem is to determine the c-space, the workspace where the robot can move, some robots (like wheel robot) can move on **two dimensions** (**2D**) then there will be two axes define our space. A plane represents that space like the room floor, and the robot owns two **degrees of freedom** (**DOF**) for that movement. The robot wheels can move forwards or backwards, and robot may change its rotation or orientation angle, this mean wheel robots has 2DOF. Other robots (like quadrotors) can move on **three dimensions** (**3D**), at any moment the robot position in 3D space will be defined with three vectors (x,y,z). A configuration is usually expressed as a vector of positions and orientations, the robot may have 3DOF or higher, and we must plan the motion with respect to 3D space. Configuration space can be represented as a subspace of $\mathbf{R^d}$, where **d** is the number of dimensions ($\mathbf{C} \subset \boldsymbol{R^d}$). Workspace can be represented as ($\boldsymbol{W} = \boldsymbol{R^d}$), O is the set

of obstacles ($\mathbf{O} \in \boldsymbol{W}$). Robot configuration $\boldsymbol{q}$ is the set of the positions of all robot points relative to a fixed coordinate system where **A(q)** the robot in configuration ($\mathbf{q} \epsilon \mathbf{C}$), then we can define free space as ($\boldsymbol{C_{free}} = \{\boldsymbol{q} \in \boldsymbol{C} | \boldsymbol{A}(\boldsymbol{q}) \cap \boldsymbol{O} = \emptyset\}$), and obstacle region as ($\boldsymbol{C_{obs}} = \boldsymbol{C} / \boldsymbol{C_{free}}$) [12].

We can further define the start configuration as q_i and the target configuration as q_G. By giving this setting, then global motion planning amount to find a continuous path for the robot, assume it as a point in C-space. With a condition to avoid any collision with static obstacles, (Figure 1.3) [12].

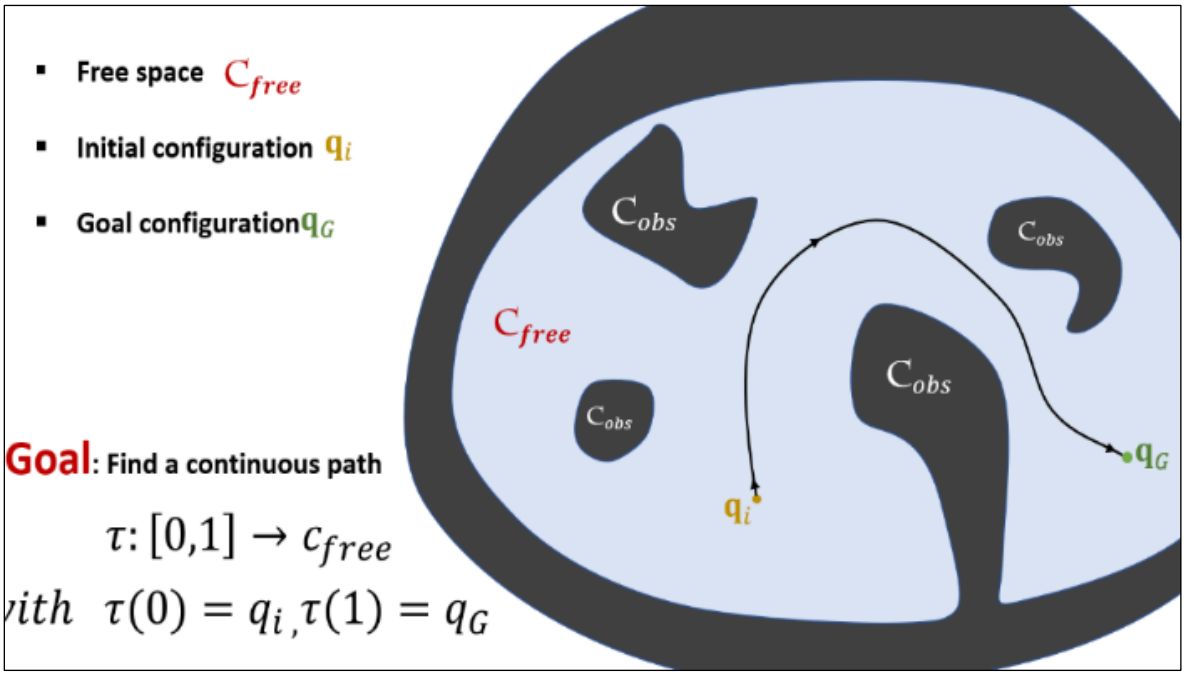


Figure 1.3: the basic of motion planning problem [12]

To achieve navigation, we need three fundamental components, those components represent the stages of motion planning for aerial robots which can be summarized as follows:

1. **Localization**: at this stage we must determine the robot position (Figure1.4) in the environment, the **GPS** (**Global Positioning System**) would inform the robot of its

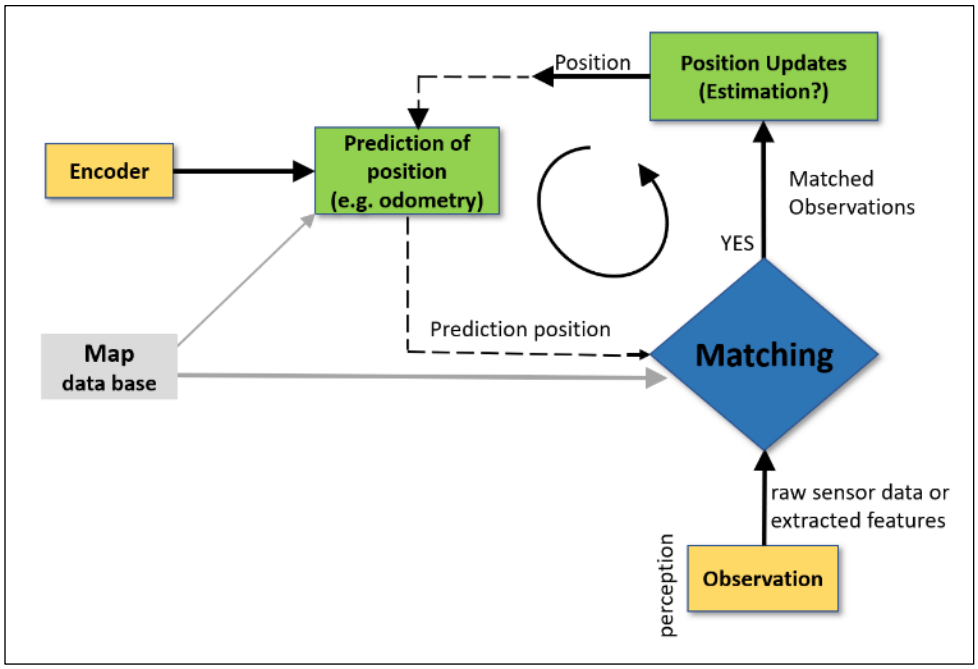


Figure 1.4: Mobile robot localization general schematic [6]

estimated position and orientation indoor and outdoor [6,11]. Noise and antialiasing can fault the determination of accurate position. Variations of SLAMs can be categorized based on their sensory system to laser-based [13], sonar-based, and vision-based systems. Other variations of sensorial sources include compasses, infrared, and global positioning system (GPS) [14]. Using efficient sensors can generate a good GPS system to reduce localization error to minimum.

2. **C-Space discretization or Decompositions**: there are three general approaches to discretise C-space:

    - First approach is Combinatorial Planning which is exact planning and contain different techniques such as Exact cell decomposition. Approximate cell decomposition, visibility graphs, and Voronoi diagrams [12, 16]. All these techniques produce a roadmap, the mathematical implementation of roadmap is a graph. The graph in workspace where each vertex is a configuration in C_{free} and the continuity of edges in between the vertices form a collision-free trajectory through

the free space. Combinatorial planning techniques are complete which means if there is a solution, they can find it, but they lose efficiency when the dimensionality of C-space increases and become intractable.

- Second approach is Sampling-Based Planning which is randomized or probabilistic planning and contain different techniques such as **Probabilistic Road Maps** (**PRM**) [12,16], and **Rapidly exploring Random Trees** (**RRTs**), the planning relay on a collision detection algorithm to investigate workspace and determine if it is belong to C_{free} or C_{obs} space [12,15,17]. Know that it is hard for practical planning problem in the real world to explicitly represent an extended C-space. It is faster and easier to check wither any given configuration belongs to C_{free} using the detection algorithm, this method can reduce planning time and guarantee finding a solution if there is one, which mean it is probabilistically complete.

- Third approach is the artificial Potential field: this approach depends on analyzing the connectivity of C_{free} and convert the wok space into equivalent graph needed to execute the path search. But the idea is to represent the robot as a point as the center of mass and modeled as a particle under the influence of **a potential field U**. The potential field is the combination of an attractive action toward the desired target and a repulsive action from C_{obs}, this technique could be used in many ways to control the robot orientation towards a goal specially for real-time motion planning, but the disadvantage of such a method is the misleading of local minima with global minimum.

(Figure 1.5) illustrate a classification of different regions that may found on world space. A and E is a free space, B is a narrow passage, C is a blocked passage, D is a cluttered area, and F is inaccessible space. Noting that the terms (free), (cluttered) and (narrow) are relative; if the robot is small and maneuverable compared to the environment, then the workspace appears more open [10, 18].

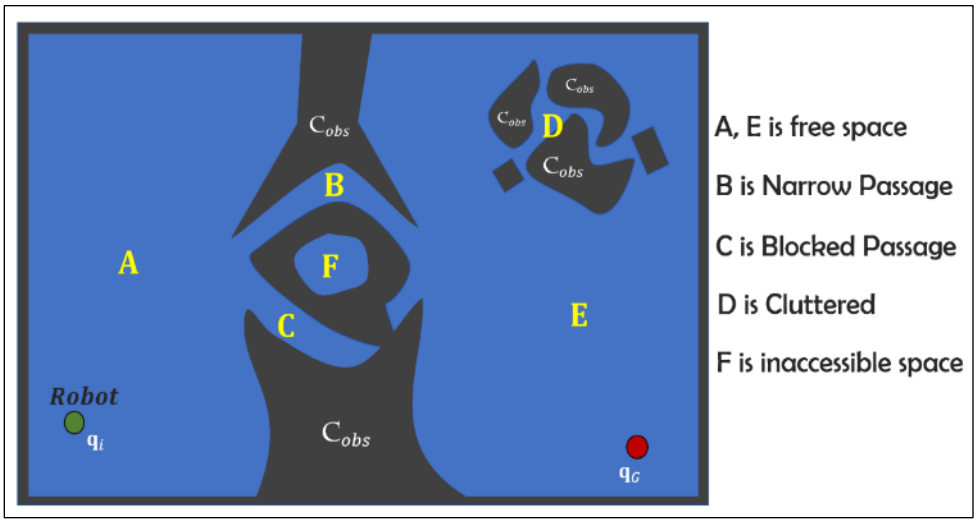


Figure 1.5: Classification of regions in world space for purpose of path planning [18]

3. **Planning method**: In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized [19]. Given a road map and a graph G consisting of vertices and edges with associated costs, then we would like to search out the shortest path between two vertices q_i and q_G. The foremost common approach for path planning is by executing uninformed or informed search algorithm to find the required trajectory over a 2D occupancy grid map, let us look at the most used searching algorithms:
    - Uninformed search algorithm: it's a general-purpose search algorithm which operates in brute force-way. Uninformed search algorithms don't have additional information about state or search space aside from how to traverse the graph, so it's also called blind search, the commonly known algorithms are:

- Breadth-First Search (**BFS**): An algorithm for traversing tree or graph. The searching starts at the tree root or some arbitrary vertex of a graph, marking the visited vertex and exploring all the neighbor vertices at the present depth prior to moving on to the vertices at the next

depth level, each out edge of a vertex is visited once. BFS is complete and optimal if action costs equal thus the total time complexity is $O(b^d)$. The space complexity is additionally $O(b^d)$. Any tree is characterized by an asymptotic branching factor b (where b is ratio of the number of child nodes to a parent node in the limit) and maximum tree depth d.

- Depth-First Search (DFS): An algorithm for traversing or searching tree or graph. The searching starts at the tree root, or by selecting some arbitrary vertex. Because from the root within the case of a graph we can explores all vertices along each branch before backtracking, but it may forget explored subtrees. DFS is not complete in infinite space and not optimal thus the time complexity is O(bd), and the space complexity is O(bd).
- Dijkstra's Algorithm: A greedy based algorithm finds the closest vertex at each step; the shortest path is consisting of the shortest (least weight) vertices at each step [19]. Dijkstra's algorithm is used to calculate the shortest paths for more general cases, where the graph has non-negative edges, the time complexity is $O(|V|^2+|E|)$→V is vertex and E is edge.
    - Informed search algorithm: suitable to used when large workspace explored and finds the most promising path. The concept is to expand further vertices based on evaluation function. At first, we explore the lowest value vertex. An evaluation function f(s) may often a combination of path cost, let us say g(s) and a heuristic function h(s). The heuristic function estimated distance to goal and used as part of the evaluation function to guide the search. It allows the algorithm to prune candidate paths and investigate the most promising paths first, hence, it can be more efficient than a less informed algorithm [18]. **F(s) = g(s) + h(s)**, h(s) add additional knowledge to planning. A widely used heuristic estimate for world space-based graphs is the Euclidean distance, **A Star** (**A***) also is an informed graph traversal and path search algorithm for the global search problem. It is one of the most known informed search algorithms with many applications in robotics. It is often used due to its completeness, optimality, and optimal efficiency. The space complexity is $O(b^d)$ which is drawback as it stores all graph nodes in memory. In our research we use A* and weighted A* algorithm for searching trajectories, there are several main criteria for motion planning problem we need to satisfy:

- Feasibility: extracting trajectory that end with reaching the goal state, irrespective of its efficiency.
- Optimality: If over one solution exists select the most effective one, that is to search out a feasible trajectory which is optimal in some manner (shortest, minimal time, smoothest path, Maximal safety, Min/Max …), knowing that achieving optimality can be considerably harder than feasibility.
- Completeness: If at least one feasible and admissible path exists, the path planning algorithm can find it, (Figure 1.6) shows an ideal architecture of flying robot [20].

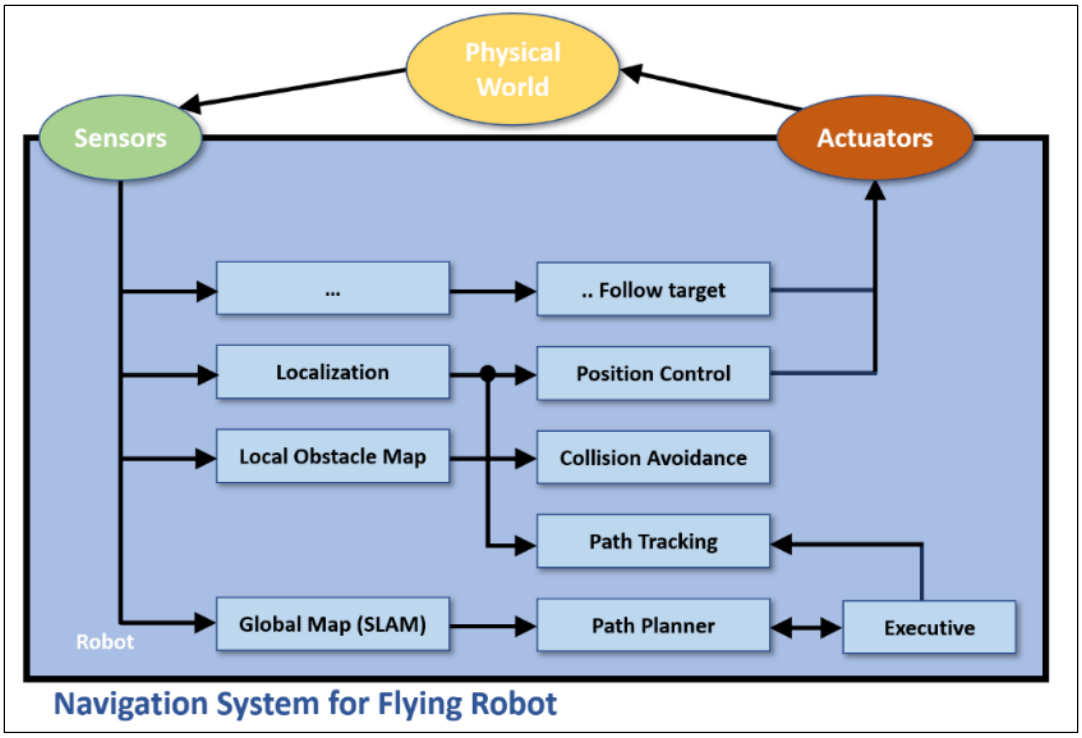


Figure 1.6: Robot Architecture [20]

1.4. Thesis objectives

The aim of this thesis is to get more knowledge about motion planning for Aerial Robots in 3D environment. And contribute to the work of building AARs system that can move and navigate among real world space considering the calculation and execution of obstacles collision avoidance. This research introduced a method to generate a 3D-enviornment that simulate a city model that contain **HDFPOs**.

In many research different solutions achieved for the path finding problem in 2D space, for the 3D space domain the problem is still facing many difficulties. The classical method is to process the work region by dividing it into three-dimensional mesh. Our research aims to optimize 3D workspace into different 2D workspaces by intersecting several planes in different altitudes, therefore we can apply the searching algorithm to construct the best trajectories between any two points connecting the 3D environment. Thus, forming a three-dimensional network diagram connecting the starting point and end point. If we discretize the 3D space by the regular method, for example one cube divided into 8 sub-cubes. Subdivisions will form 27 nodes, which will be involved in the process of searching. Imagine the extremely high number of nodes formed by processing real 3D space with the same concept. Our approach depends on optimizing the huge number of nodes by considering only several planes intersecting the HDFPOs environment within the 3D workspace. When we resample the three-dimensional space this way, then we still can apply the 2D searching path method on different altitude of 2D intersecting planes generated. Then, we compare the cost between the path finding process on zero ground and another path finding process on generated planes with different altitudes. By doing this we can compare the solutions obtained and its relationship with the obstacle density. And then we can find the best altitude that present shortest paths for AARs. Our robot system follows the motion concept of flying up vertically to a chosen altitude, then use the 2D solution to reach its goal reference point, then it flies again down to required goal position.

Our main objectives in this thesis are as follows:

- The primary main objective is to predict the best altitude for flying AARs system among HDFPOs and evaluate the benefit of such approach.
- The secondary objective of this research is to develop a new approach, which enables AARs system developers and operators to enhance autonomous motion planning methods, these methods will be capable of dealing with 3D-space environments containing HDFPOs in decomposing with a computationally efficient manner.

- completing the graduation requirements to get the master's degree in computer science insha' Allah.

1.5. Major contributions

The major contribution of this research is the resampling of the three-dimensional space model, by considering several intersecting 2D-plans on different altitudes that can lead to faster routing results. This three-dimensional motion planning method can incorporate multiple decision objectives such as safety, regulatory (rules of the C-space) and mission efficiency objectives. UAVs uses space variable to operate the motion and provide trajectory's planning with arbitrary selectable resolution. The HDFPOs variables and attributes are the key to path optimization under case of heavily crowded obstacles environment.

The pointed contributions of our work include the following:

- Creating a 3D HDFPOs model simulate a city and execute several experiments to Extrapolates how obstacle density affects the UAVs or AARs motion.
- To validate the approach and perform the simulation using Unity Real-Time Development Platform based on A* and weighted A* algorithm.
- Contribute to pathfinding that work in 3D state, and its respective representation for space, with the 2D projective representation as well.
- Enhancing the overall progress in motion planning approaches to involve more UAVs for daily life services and applications.

1.6. Thesis benefits

The main benefit for this research stem directly from the research objectives, given the primary research objective, the major benefit of the research is:

- The development of a suitable case study test scenarios. Our HDFPOs case study serves as a reference implementation for researchers in any future aerial robot operation and development. For our proposed motion planning approach, a major

component in the verification is the validation, both analytically and by simulations. It is important to disseminate and share the research outcomes.

This research achieves other benefits for AI domain in particular; some of them can be summarized as follows:

- Present an algorithm to generate a model of random HDFPOs workspace.
- The research answers the question of what the best altitude that AARs can use for flying in the case of HDFPOs workspace.
- It is a new methodology for creating enhanced motion planning missions.
- The research also develops an implementation example for AARs case study scenario by simulating a motion planning for HDFPOs environment.

1.7. Thesis constraints

There are many techniques for mapping. The state of the map decomposition in real world depends on the observation, the sensors reading, and data gathered by devices like depth camera attached to aerial robots. The implementation of this thesis case study could be done only by simulators, due to the lack of robotics laboratory infrastructure, many simulators are not free. We simulate our experiments using Unity Real-Time Development Platform, since the company offers a noncommercial free version, but with limited features. Part of this research work implemented at GET-lab cognitive systems in Paderborn University, where I was attending NRW-scholarship from DAAD association. At GET-lab researchers used **Robot Operating System** (**ROS**) for simulation, **ROS** works under UNIX. But our simulation experiments done by Unity since we used windows PC from the beginning. All lab workstations work under UNIX platform, there was many difficulties and technical problems in setting up Unity simulator platform on these workstations. For scripting and programming the classes, we used c-sharp (C#) scripting language. It was new language for me, I take some time to learn C# to complete all the function and algorithms needed. During the development of the thesis, we built the HDFPOs model and execute the discretization of a high-resolution maps using a laptop with specification of i5 processor and 8 gigabyte RAM. Therefore, many experiments execution collapsed and

crashed, many simulations repeated several times until the recording of maps and results done successfully.

Chapter 2

Background and related work

2.1 Quadrotors Basics

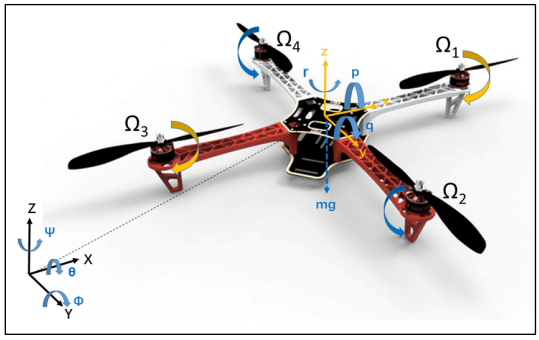


Figure 2.1: Quadrotor or Drone model

Quadrotors-Drones could be commercially manufactured for special purposes, but other quadrotors are designed to be used for research and developments. To understand the basic hovering mechanism of any quadrotor (Figure 2.1), we will discuss the main components on different platforms, also we will discuss the basic structure and control concept for any quadrotor. It consists of two main parts, the actuator, and the sensing system. The actuator components are the brushless motors and the controllers. The sensing system may contain one or more of gyroscope, accelerometer, magnetometer, ultrasonic height sensor, Pressure sensor, Visual odometry sensor, cameras [6, 21, 22]. A general procedure of developing the autonomous motion is to program the motion using a simulator. After that we need to test the experiment in real world. This implies to use impeded operating system unit (**IOSU**) to facilitate the process of development. IOSU include operating system such as Linux and ROS. Programing the motion of aerial robot can be simulated and visually tested, by using 3D visualization tools for debugging such as RVIZ, Gazebo, LibQGLViewer, and Unity.

The flying principle of a quadrotor depends on the torques of all four rotors that generate thrust against earth gravity. To test our understanding of the flight principle, assume that our system has no noise, so for generating a motion or to preserve a position in 3D space, we need to specify a sequence of motor commands for controlling movement such as ascend, descend, fly forward, fly left, fly two meter forward, Fly (blindly) through the track, and any other compound motions [22, 23]. And to keep the quadrotor stable at one position two condition must be satisfied:

1 Thrust compensates is equal to earth gravity.

2 Torques of all four rotors sum is zero.

The movement of quadrotor can be generated based on the speed of the four rotors, and cycling direction, each faced pair of rotors loop on opposite direction of the other pair. For ascending movement, we send speed up command to rotors. For descending movement, we send speed down command to rotors [23, 24, 25]. See (Figure 2.2), note that the thickness of the round arrow represents the fan speed.

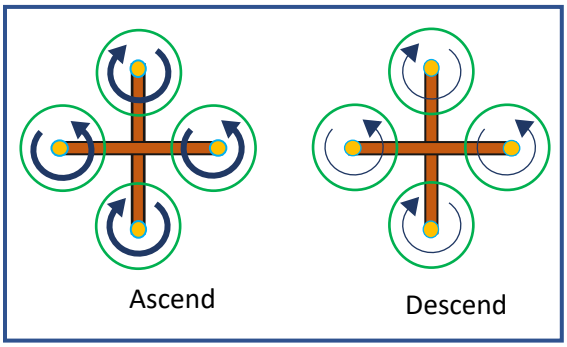


Figure 2.2: Ascend or Descend motion

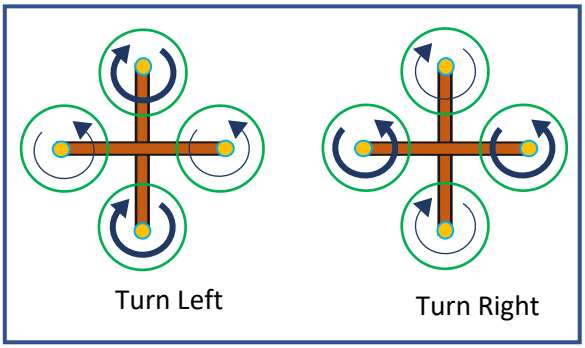


Figure 2.3: Turn Left or Right motion

For turning left movement, we send speed up command to left rotating rotors. For turning right movement, we send speed down command to right rotating rotors. And if we want to

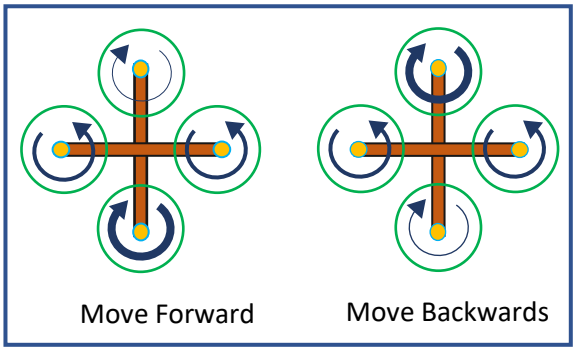


Figure 2.4: Move forward or backward motion

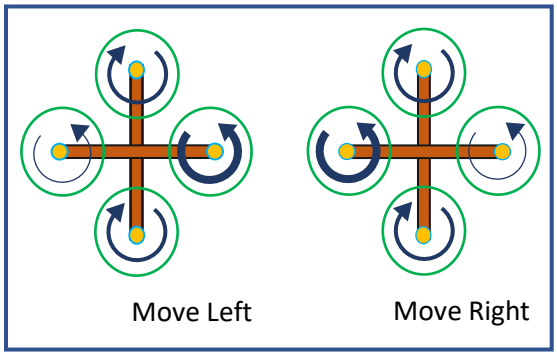


Figure 2.5: Move Left or Right motion.

keep the vertical position of quadrotor, then the overall thrust of the four motors must exactly equals earth gravity, see (Figure 2.3).

For forward movement, we send speed up command to forward rotor, and speed down command to backward rotor. For backward movement, we send speed up command to backward rotor, and speed down command to forward rotor, see (Figure 2.4). For left movement, we send speed up command to move left rotor, and speed down command to

move right rotor. For right movement, we send speed up command to move right rotor, and speed down command to move left rotor see (Figure 2.5). Finally, we can combine all these motions at the same time. For example, to move to the right and at the same time rotate right, or to ascend and rotate.

2.2 A* Algorithm Theory

3D Path planning is referred to the carrying out of a feasible flight path from a start position to a destination without consideration of dynamic feasibility. The most common 2D map searching algorithm is A*, it uses the heuristic function h(s) to estimate the distance of any point to the target point, in order to reduce the search space and improve the search efficiency [12, 26, 27]. We use A* and weighted A* to search the shortest path from the start node to target node over the 3D selected occupancy grid map. Weighted A* is a biased version of the algorithm towards the heuristic part, it may speed up the arrival for the destination, as A* algorithm is a simple, complete, fast algorithm. While A* traverses the map, it builds up a tree of partial paths. The leaves of this tree called the open set are stored in a priority queue that orders the leaf nodes by a cost function. The cost function F(s) is described by F(s) = g(s) + h(s) [12], where g(s) is the distance travelled from the initial node to node n and its value is tracked by the algorithm, h(s) is a heuristic estimation of the cost to reach the goal node T from node s while neglecting obstacles. The cost function F(s) is also modified to satisfy a weighted A* algorithm, which is a biased function of F(s) with a ratio between [1,2) and multiplied by h Cost), this means that we can focus more on the movement towards the target and accelerate our calculation with more biased for the target side instead of calculation from the start point side, (see Node class code Appendix D)

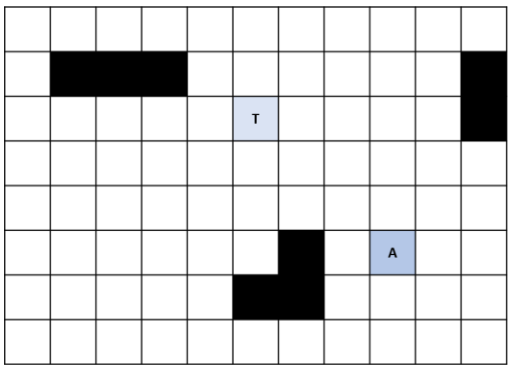


Figure2.6: Grid Map

Following the A* algorithm working steps:

1- Assume that we have 2D grid map contain node A as start node, and node T as target node, white nodes represent a walkable area, black nodes contain obstacles and represent unwalkable area, see Figure 2.6.

2- Initialization: Define start node A. Let A=s and g(s) = 0 and the cost function f(s) = g(s)+h(s), h(s) is the Euclidean distance from the start node A to the destination node T. Set the closed set consisting of all obstacle nodes in the occupancy grid map and node s. And set node s as its parent node.

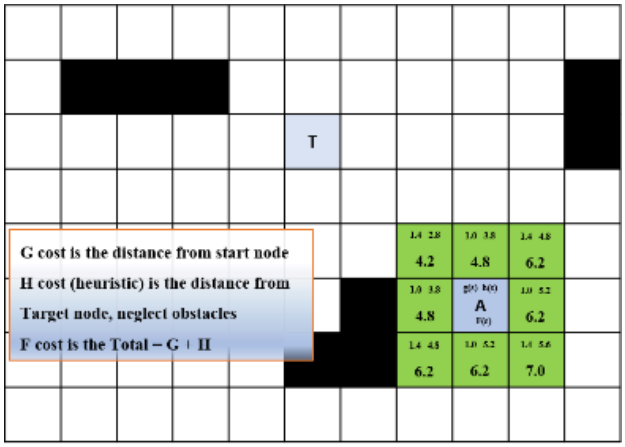


Figure 2.7: Grid map and A first calculations*

3- Node Expansion: Find free surrounding nodes of the node A (A = s at the start node) from its 8 surrounding nodes in case of 2D map, save the cost (g+h) and corresponding parent of these free surrounding nodes. Keep these free surrounding nodes in an open set in the order of cost value, see Figure 2.7.

4. Node Selection: If there is no node in the open set, display" No Path" and exit from this algorithm; otherwise, identify the node s + 1 with **minimum cost** in the open set. Delete this node from the open set and put it into the closed set and save it as parent node, see Figure 2.8.

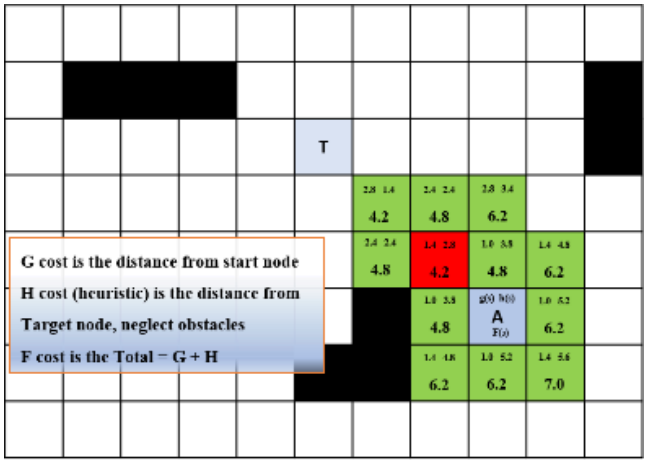


Figure 2.8: Gid map and Parent calculations

5. Stopping Criteria: If the node **s + 1** is the destination, go to step 7; otherwise, let **s = s+1** and go to step 4.

6. Short Cut: If there is no obstacle on the straight line between a node i (start from the destination node t) and its parent's parent, replace its parent with its parent's parent. Otherwise, let i = i+1 and repeat Step 4 until the start node is reached, see Figure 2.9.

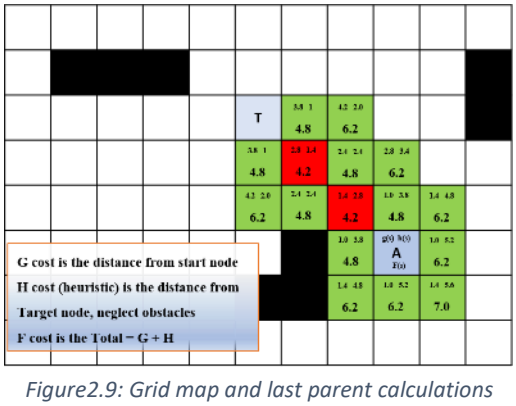
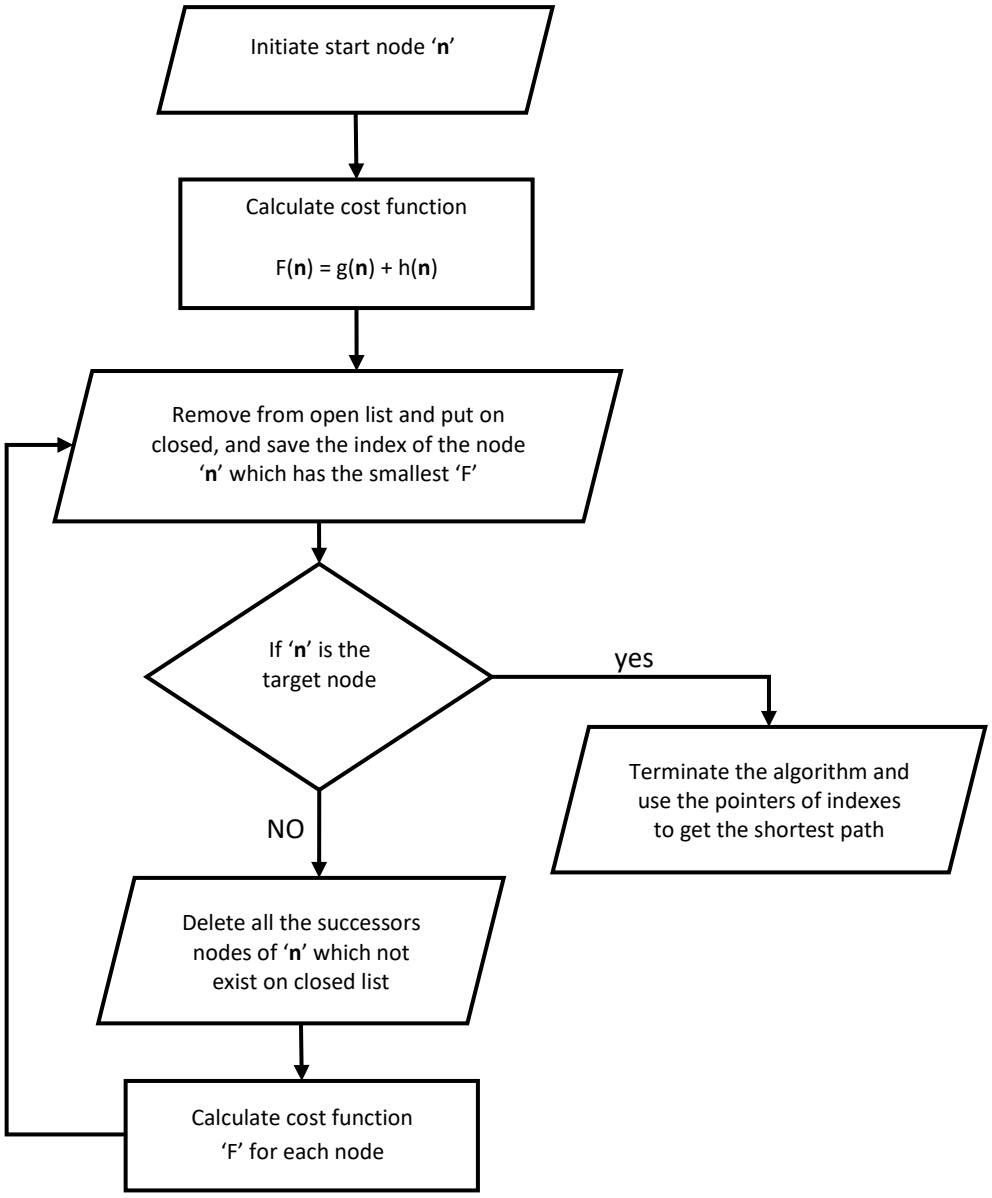


Figure2.9: Grid map and last parent calculations

7. Path Finding: From destination node and its parent and its parent's parent, and so on, by using backward induction, the shortest path from start node s to destination node T can be found, see Flow Chart1 of A* Algorithm [27].

   It is noticeable that the path generated by the A* algorithm may not be the shortest path on the global map since the A* algorithm works on the local occupancy grid map. However, the computational time will be reduced and the completeness of the A* algorithm will not be affected.

Flow Chart1 of A* Algorithm [27]



2.3 Literature Review

The exploration of 2D and 3D environments using Voronoi transform, and fast marching method discussed on reference [1]. This method begins with the calculation of the logarithm over the extended Voronoi transform (EVT) of the 2D a priori map or in case of 3D maps represent the environment. Each white point of the initial image represents free cells in the map, each point is associated with a level of grey that equals the logarithm of the 2D distance to nearest obstacles or the EVT in 3D. As a result of this process, a kind of potential proportional to the distance to the nearest obstacles to each cell is obtained, zero potential indicates that a given cell is part of an obstacle and maxima potential cells corresponds to cells located in the Voronoi diagrams[26], which are the cells located equidistant to the obstacles, And the function introduces a potential like a repulsive electric potential in 2D. the paper presents a new autonomous exploration strategy. The full exploratory method consists of the VFM motion planner applied to plan the trajectory towards the goal. A new exploratory strategy that drives the robot to the most unexplored region, and the SLAM algorithm build a consistent map of the environment. So, the proposed autonomous exploration method is a combination of the mentioned three tools which can completely construct consistent maps of unknown interior environment in an autonomous way, the method can be used in 2D or 3D environments [1].

Autonomous navigation of quadrotors in complex environments has been an active research topic for last decade [2], a closed-loop 3D path planning extracted based on A* search algorithm. The research proposed to generate collision-free path. The 3D motion planning method that controls UAVs to fly under GPS over unknown obstacle-rich environment such as forest and civil areas [2]. The implementation of building occupancy grid map, SLAM, 3D paths planning and generating 3D online trajectory are done under ROS. The path generated by the 3D path planning algorithm consists of a serial waypoint arranged in a pre-defined order. The local target generation algorithm selects the way points one by one as the local target input to Reflexes trajectory generator. To switch the local target from one waypoint to the next one, collision check of the trajectory from current

position to the next waypoint generated by Reflexes is carried out. If it is obstacle-free, the local target switches to the next waypoint; otherwise, the local target keeps unchanged [2].

Reference [4] discussed online path planning algorithm, in a low altitude dangerous environment that have density obstacles. The authors proposed a novel method to solve the problem to get a feasible and safe path. Paths developed based on rapidly exploring random tree (RRT). A local path planner is constructed by improving dynamic domain rapidly exploring random tree (DDRRT) to deal with complex obstacles. RRT is embedded into the planner to optimize paths, they construct the 3D global path at the end. By defining the orientation of the tangent of a tree node, the direction of the edge ending at the tree node. Authors used an approach to construct the 3D path directly by composite 2D paths. The idea is to generate a straight-line maneuver of the 3D path defining by the intersection of two planes, a start points with different consecutively subgoals comes after in between the obstacles edges, a rapidly planning while flying method that scan and generate local paths. A process of initiative and terminated pose is executed, the transited initiative pose is rotated about its tangent vector, to make the binormal vector of the initiative pose to be parallel to that of the terminative pose. The new initiative pose is coplanar with the terminative pose and the final curve can be constructed; the curve parameters can be calculated based on the differential geometry.

Chapter5 of the book Introduction to Autonomous Mobile Robots [6] discussed mobile Robot localization. It is clear that the decomposition stage Occupies high importance, the strategy of decomposition plays a main rule in optimizing the complexity of calculations. One of our main objectives in this research, is to optimize the 3D space into accessible 2D space that facilitate processing, map representation in 3D is complex. The key advantage of a continuous map representation is the potential for high accuracy and expressiveness with respect to the environmental configuration, as well as the robot position within that environment. The position of environmental features can be annotated precisely in continuous space. “The mobile robot implementations to date use continuous maps only in 2D representations, as further dimensionality can result in computational explosion”. One method of simplification, in which the continuous map representation contains a set of

infinite lines that approximate real-world environmental lines based on a 2D slice of the 3D world. Basically, this transformation from the real world to the map representation, is a filter that removes all unneeded data, and furthermore extends line segment data into infinite lines that require fewer parameters.

Reference [28] talks about a 3D path planning method for small Unmanned Aircraft Systems (UAS) navigating in Low-Altitude airspace. The proposed approach can trade-off a collision-free path versus the path length. The algorithm carried out in two steps. First, an initial suboptimal path is generated using A∗ search. Second, a chain of unit masses connected by springs and dampers evolves in a simulated force field, using the A∗ solution as an initial condition. A set of ordinary differential equations describes the chain that is driven by virtual forces to find the steady-state equilibrium. As a result, dynamic chain analogy used to smooth path into a flyable shape while maintaining a safe distance from intruders. One of the advantages of this approach is that timing information is specifically embedded in the future path representation of the vehicle, and it is also flexible, in that dynamic environment is incorporated.

A 3D motion planning using kino dynamically feasible motion primitives in unknown environments is a research present solution to two subproblems [29]. First, it uses motion planner that ignores kino dynamic constraints, to compute an initial collision-free path. Then, using the presented heuristic re-sampling and transformation algorithms, this collision-free path is transformed into a series of kino dynamically feasible motion primitives. Second, the transformation algorithm incorporates with a motion planner capable of calculating an initial candidate path, and then re-plan as necessary as the vehicle explores the environment, it is a modified version of the PRM planner with a dynamic A* search.

A hybrid method for path planning in 3D spaces present a standard approach in path planning to discretize a continuous space [30], by representing it as a visibility graph or as a discretized grid in 2D or 3D. To find the optimal path, algorithms should check for 26 edges between nodes that surrounds the starting node. A hybrid graph-based method proposes to use an improvement to a near-optimal 2.5D algorithm together with a logic-

based fuzzy decision-making component [30]. It keeps control of the rest of the system to fly in the partially known 3D environments. Path planning is performed off-line using topographical data in 3D about the flying area. The topographic data stored in a database (TDB), should include information about potential obstacles such as buildings, monuments, or towers, that classified by the classification component as semi or full obstacles depending on their dimensions.

2.4 Summary

The motion of aerial robots' system autonomously in real world is not fulfilled yet. Many researchers working on the development of avoiding collisions with obstacles in static or dynamic environment. Working on enhancing AARs motion regarding safety and performance is still in progress. The main goal is to achieve a solution that satisfy the shortest path between a start point and a destination.

Our focus is on real-world environment or 3D space that contain areas with high density obstacles. Different approaches seek and examine optimizing 3D space to reduce and simplify processing and calculation of finding trajectories. Our Solution based on reducing 3D space discretization data by dealing only with several 2D intersecting planes across that 3D space on different altitudes. Then we want to measure and experiment the relation between the density of obstacles and the best altitude that achieves the shortest paths among HDFPOs environment.

The configuration space or C-space is the space of all possible configurations. A configuration q may be an initial configuration (a start point), or it may be a goal configuration (end point), where C-space may have different dimensions and consists of free space and obstacle space. A robot needs to travel between start point to endpoint continuously with shortest way passing through free space and avoiding obstacle space.

In any Coordinate Systems the pose of a robot can be described by different DOF. A simple description is 2 DOF when the robot moves in 2D only with no orientation. A general case is 6 parameters (6DOF), three-dimensional cartesian coordinates [6, 9, 20], and three Euler

angles roll, pitch, yaw. Therefore, the state space of such a system is six-dimensional. $\mathbf{X}_t = (x, y, z, \phi, \theta, \psi)^\top$ [9, 20]

Implementing a motion model and simulating path finding experiments need a good knowledge of 2D and 3D geometry. It also needs the understanding of 2D and 3D representation and transformations. Note that in real world, it is very hard to read 3D orientations or rotations, no matter in what representation it will be implemented, but they are usually easy to visualize and can then be intuitively interpreted, the Advice is to use 3D visualization tools for debugging or simulations.

Chapter 3

Proposed Model

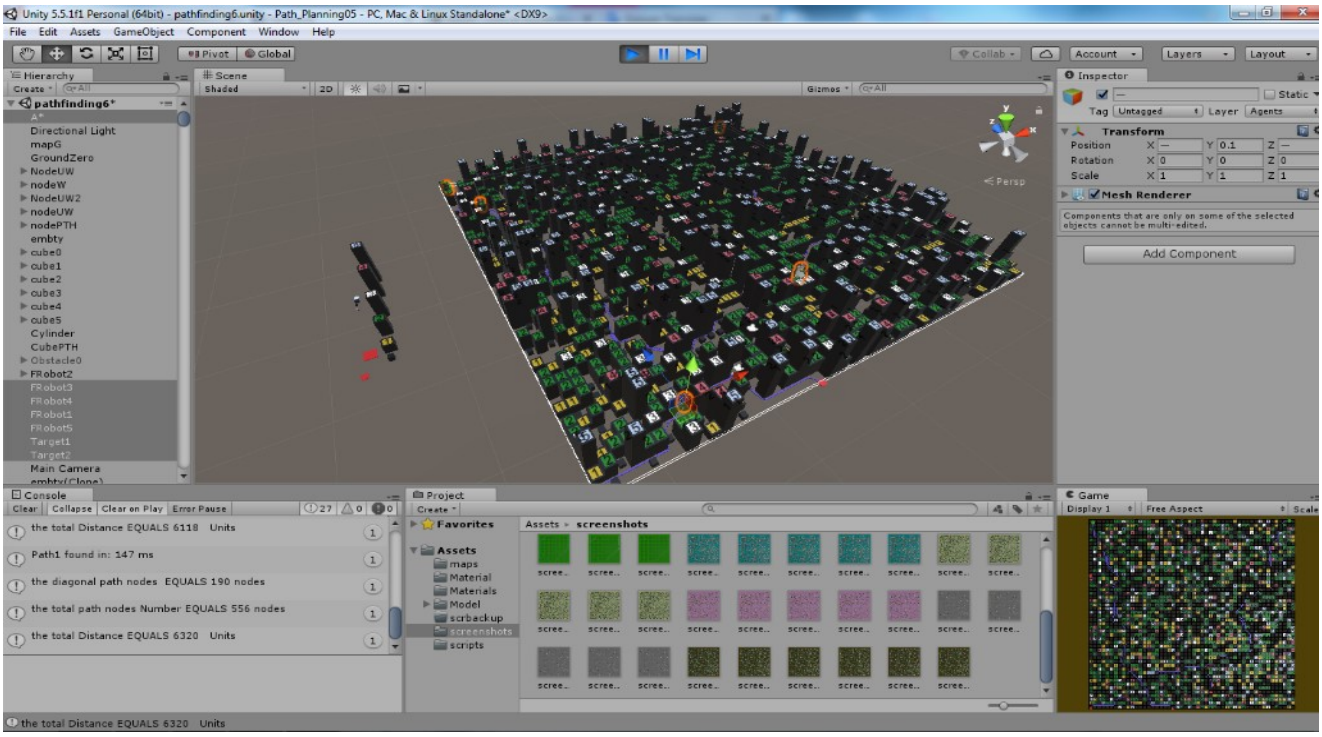


Figure 3.1 : scene of environnement contains HDFPOs in simulation.

3.1 Introduction

The implementation of our simulated motion model under the HDFPOs environment and performing the experiments divided into four stages. We start the first stage by generating a random 3D space, the 3D space simulates the environment of cities that contain HDFPOs, see Figure 3.1. On second stage we consider the mapping and decomposition, when the 3D model scene is built and ready, a decomposition of the environment is executed, that is to define the free space area and the obstacle space area. The third stage is the path searching stage, we search for the path using A* and weighted A* algorithms. The path will find the shortest distance between different start points and related targets. Under the created scene, we can create more than one AAR object, and more than one target. To perform the simulation, we need to assign one target to each AAR, we can execute the algorithm on different altitudes, and it's obvious that the execution will give different results depending on the density of interacting obstacles at each altitude. At last, the fourth stage were for

each simulation performed, we record all measurements and results on one log file, we record a visual snapshot of maps and results, finally, results are compared.

3.1.1 Proposed Simulation model

The methodology of the proposed model contains four stages where each stage depends on the previous one:

1. create area as 2D plane then convert it to random 3D HDFPOs environment.
2. On a specified altitude, decompose 2D map of free space and obstacle space.
3. Perform simulation to find the shortest paths for AARs and Targets created.
4. Record Results and maps of our experiments and then compare the results.

3.1.2 The Simulator

Unity 3D is a game development tool and simulator [31]. It offers rich visuals and developer-friendly toolkits that not only increases the ability to make changes but also gets ultimate performance. It had been developed by Unity Technologies and extended to support many platforms. The engine can be used to create 2D or 3D environments for virtual reality, and augmented reality, as well as simulations and other experiences. The script under Unity must be attached to an Object in the scene in order to be called, scripts are written in a special language such as UnityScript (Java) and C-sharp (**C#**). We used **C#** for scripting our model, (see Figure 3.2). All the languages that Unity operates with are object-oriented scripting languages. Scripting languages have syntax, or parts of speech, and the primary parts are called variables, functions, and classes. Unity engine has been adopted by many industries beside the video gaming, it is used also in film industry, automotive, architecture, engineering, and construction. In the process of production most of the projects will experience the problem of finding the path or seeking the road, a path to connect a starting point to the target point in a map. Complex projects also need to consider the file structure of the map and the passable of target location [30].

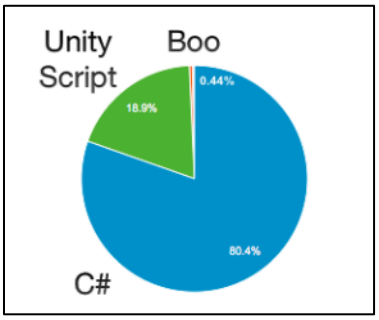


Figure 3.2: Unity scripting language

In this research, we focus on building a common path-finding algorithm A* under Unity, and then put forward the improvement and application of algorithm in the special 3D model.

The Five Features Unity Engine Affords [31]:

1- Flexibility:
- The flexibility of Unity 3D makes it ready to adapt changing needs.
- The C# scripting system with extensive **API** (Application Programming Interface), and documentation that make Unity extensibility and possibilities.

2- Quick creation:
- The extraordinary work in real-time helps in creating and launching prototypes.
- The user interface and available tools are intuitive, save programming time.

3- Quick implementation on many platforms:
- The task of publishing project on different platforms is simplified.
- The game can be deployed on more than 25 console and computer platforms, even on mobile devices.

4- Integrations: Integrations with other systems are possible because of well-documented APIs.

5- Graphics performance:
- User interface responsiveness is maintained when scaling for various devices.
- Creating high quality and flexible graphics is possible due to high-resolution rendering pipeline.

3.2 Building random HDFPOs simulation environment

For establishing the environment, we create a simulation 3D scene under Unity, the scene model contains a plane that represent the working space at altitude zero. Then we generate a random 3D Polygonal Obstacles, a C# script class named *RandomPlacement* contain a method for generating a random integer number between zero and hundred, this range of numbers assorted into seven scopes, six of those scopes returns a specific sized obstacle (cube0, cube1, cube2, cube3, cube4, cube5). Cube0 is the smallest obstacle with a volume of 0.5 unit, cube5 is the largest and highest obstacle with a volume of 5 unit. To control the

randomness of the environment generated, If-Else statement control the zero-fifty numbers generated and the ratio of the returned obstacle, the other numbers between 51 and 100 will return an empty space which represent a walkable area, see (code 3.1, Appendix B) .

The generated data is saved to maz.txt file, so it will be called to generate the 3D random HDFPOs environment when we need to execute a specific experiment. Certainly, the file size of the maze depends on the grid size and the node radius, maximum resolution of maps depends on hardware performance, for earlier version of maps (Figure 3.3).

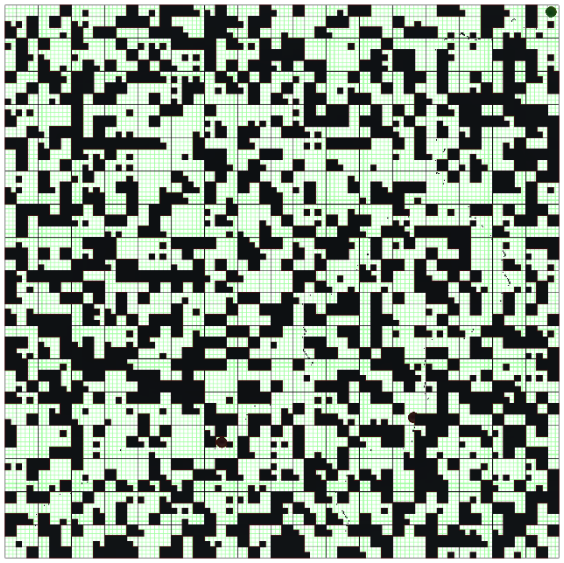


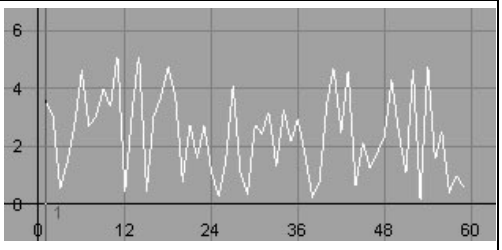
Figure 3.3: Random obstacles Map

The randomization method may have different behaviors, many functions or methods are used to generate randomness. Functions such as noise returns a random number from -1 to 1 according to a Perlin noise field generator, while gauss function returns a random floating-point number or vector, the number returned falls within a Gaussian (bell curve) distribution with mean value 0. Even though seed function sets a seed value that other previous functions use to generate random numbers, if we assign a value to the seed then execute the gauss, rand, or sph-rand function repeatedly, an identical sequence of random numbers will be generated.

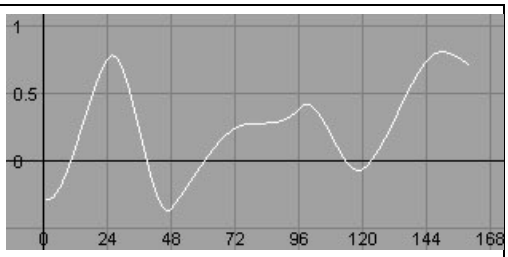
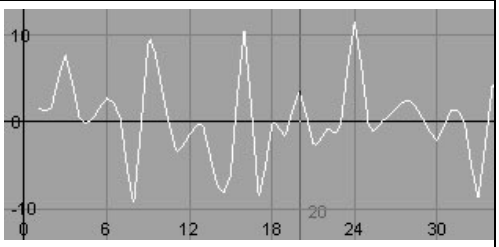
A summary of randomness methods [32] shown in table 3.1 below:

***Table 3.1: Randomness Functions* [32]**

Randomness function	Description
Random	Returns a random floating-point number or vector within a range of your choice. Float rand(float minnumber, float maxnumber) vector rand(vector maxvector) seed can control the random values returned by this function. rand( ) results in a very jagged curve; every step has an entirely different value from the last. Rand( ) can be used effectively for "jitter" by using a small range. Another good use for rand( ) might be when selecting a number periodically, such as in a conditional statement.



translateY = rand(5);

Noise	Returns a random number from -1 to 1 according to a Perlin noise field generator. Float noise(float xnum, float ynum) float noise(vector vector) If we execute this function with the same argument value repeatedly, the function returns the same random value each time it executes.  Noise function provides a smoother random motion over time, but you must use a steadily increasing value, in this case the predefined variable "time". Noise always returns a value between -1 and 1, so you should multiple the result and/or add an offset to get the range you need.
Gauss	Returns a random floating-point number or vector. The number returned falls within a Gaussian (bell curve) distribution with mean value 0. Example is vector gauss(float XstdDev, float YstdDev)stdDev specifies the value at which one standard deviation occurs along the distribution. This gives a one-dimensional Gaussian distribution. 
Seed	an identical sequence of random numbers is generated. For Example: if (frame == 1) seed (1); Translate = rand (5); Every step the second line is executed, including when the execution rewinds to step 1 again. The first line is a conditional statement that explicitly sets the random seed to 1, thereby reproducing the exact same random stream of numbers each time the execution or run is played. For more details visit Ohio-State University website on the link below: (www.asc.ohio-state.edu/price.566/courses/694/random.htm)

The reason why we used random method instead of any other methods is that other method is not totally randomness, but it follows a pattern somehow and it will give the same answer when initial state conditions are similar.

3.3 Scripting Node and Grid Class

Next, we create an empty A* game object, this game object will hold the grid script constructor and the pathfinding algorithm. For testing purpose, we add several cubes that represent the fixed polygonal obstacles occupying the unwalkable area, these obstacles inserted into a layer called Unwalkable, so it will be masked later for collision calculations. Also, we add two 3D-objects, first object represents the robot or the seeker, the second object represents the goal or the target. To prepare the grid for the execution of the

searching path algorithm, we create two C# script files, a new class “Node” contains the information of storage node, and the call of the **fcost** function, see (code 3.2, Appendix B).

The other class “Grid” will construct the grid in the scene environment at every execution. This class define the radius of the node, the world coordinates, the parent node, and create a constructor for transfer value and a call of the CreateGrid method. The CreateGrid method consists of 2 for loops and collision check method that set each node in the grid to be walkable or not. On real world the collision check performed by odometry or laser sensors, see (code 3.3, Appendix B).

This method selects the desired walking ground, also it determines the number and size of the grid according to the size of the ground and the radius of each Node (see Figure 3.4). Determine the grid where the AAR will use the world coordinates begins at the starting

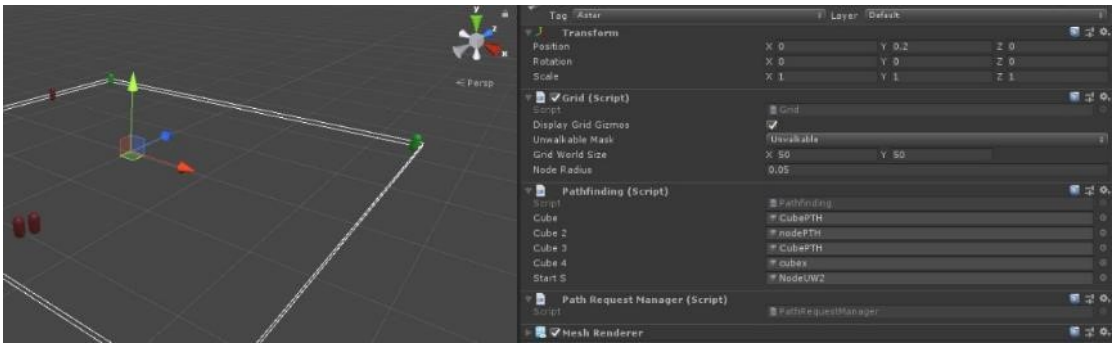


Figure 3.4: Grid Attributes

point of the Robot and declaring an open set and a closed set. Then we add the start point on grid to the beginning of the collection, and start the searching process that will discussed in section 3.4 below.

3.4 A* Implementation

We create a new C# script called pathfinding and attach it to A* game object. The script represents the algorithm that will calculate the shortest path between the start node and the target node. The first thing we need to do is to convert a world position (start and target positions) into nodes, we already create that method in the Grid class, we make a reference for start position and target position in pathfinding class and pass these variables to grid

class for obtaining nodes position (check Grid class code. Appendix B). Referring to the flowchart of A*, first steps is to create an open set and a close set, where we need to be able to add and remove nodes to those sets, we also need to check if a specific node is in these sets, and the open set is the place where we need to search the node with lowest fcost in node neighbor's, which is the most expensive part of the algorithm. Know that at first stage of implementation the data structure we use to build it was a hash set (list<node>), but later we optimize the algorithm by using a heap data structure (see section 3.5), the calculations time of the path decreased dramatically to almost half of the time. The process of finding the path start by adding the start point to the open set, then a loop continue to check the lowest fcost between the current node and all of the nodes in the open set, if the node fcost in processing is less than the current node fcost, then this node will be the current node, note that if they are equals, we compare which node is closer to the target by comparing the hcost. As discussed before, fcost is equale to gcost + hcost, and hcost is a heuristic estimation of the cost to reach the goal node. A list of Node GetNeighbours is also implemented in Grid Class that will be called each time we process a current node and its neighbors. Know that neighbor's node could be eight nodes at most in 2D scale, or if we process a node on the corners or a node located at an obstacle edge, it will be less than eight nodes , see (code 3.4, Appendix B).

To calculate the distance between any two nodes, let's say node A and Node B, a GetDistance method created, and it will return an integer regarding on how many nodes we will move vertically or horizontally. Based on the example on (Figure 3.5) GetDistance method concept to reach node B starting from node A, we move Five nodes horizontally and two nodes vertically. To calculate this movement, we subtract the lower number, which is 2 that represent the y axis, from the higher number which is 5 represent the x axis. By assuming that each node is one unit square, so the diagonal length of one node = $\sqrt{2}$ or 1.4, if we move diagonally then we multiply the

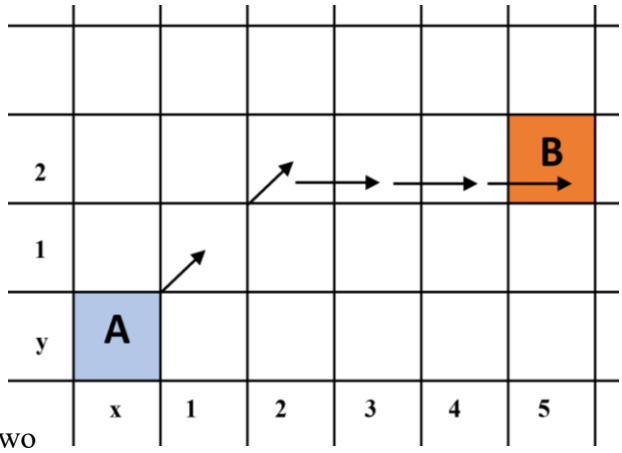


Figure 3.5: GetDistance method Concept

number of nodes by 1.4, then the extracted equation for obtaining the distance is (1.4y+(x-y)), this is only in the case when x is greater than y, if y is greater than x, then the equation becomes (1.4x+(y-x)), and if the case is x=y, it is clear that we only move diagonally, see (code 3.5, Appendix B).

The GetDistance method is used to check if the current path with current node is shorter to any of its neighbors' paths. If one neighbor gives a shorter path, it will be the current node, assign as a parent, and add it to the open set if it is not in there, then we keep the comparison processing until we reach the target node. After that another method created named RetracePath, it will retrace the nodes in the open set to get the path between the starting node and the target node, it will be called at last before we exit the loop, other methods were written to draw the paths or document the results.

3.5 Optimizing the iterations

If a node radius of 0.01 for example, then the scale of our maps will be large enough so that we can process 250000 nodes in one execution. And since we will develop the model to have more than one robot, let us say we can have five AARs with five starting points and two Targets; so, when we execute one experiment to find the trajectories for these AARs, the iteration calculation part is the slowest. The high processing happened when we keep searching for the node with the lowest Fcost in the open set, this loop calculation needs to be optimized to stop the simulator crashing.

3.5.1 Heap Data Structure:

we will use a Min-heap data structure instead of a list or the array we use before to represent the open set. A heap is a special tree-based data structure in which the tree is a complete binary tree. Generally, heaps can be of two types, first type is Max-Heap where the value at the root node must be greatest among the values at all its children [31]. The same property must be recursively true for all sub-trees in that binary tree. The second type is Min-Heap where the value at the root node must be minimum among the values at all its children. The same property must be recursively true for all sub-trees in that binary tree.

Giving any node in the heap, we can find its parent node, or we can find the left or the right child, by using the formula as described in (Figure 3.6).

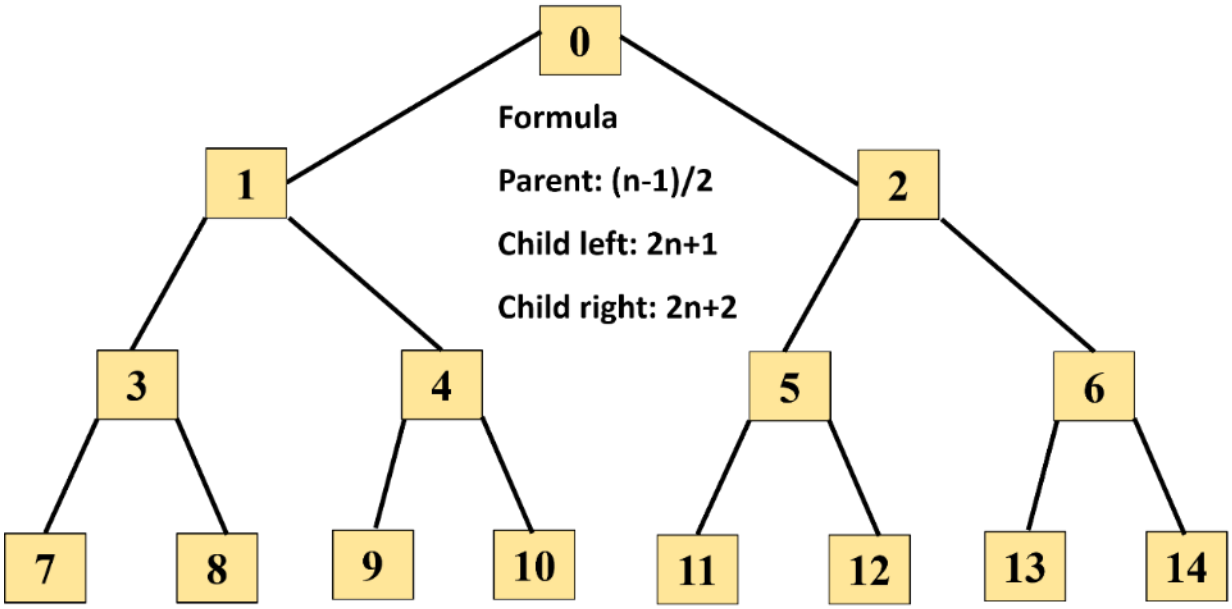


Figure 3.6: Min-Heap and its formula [33]

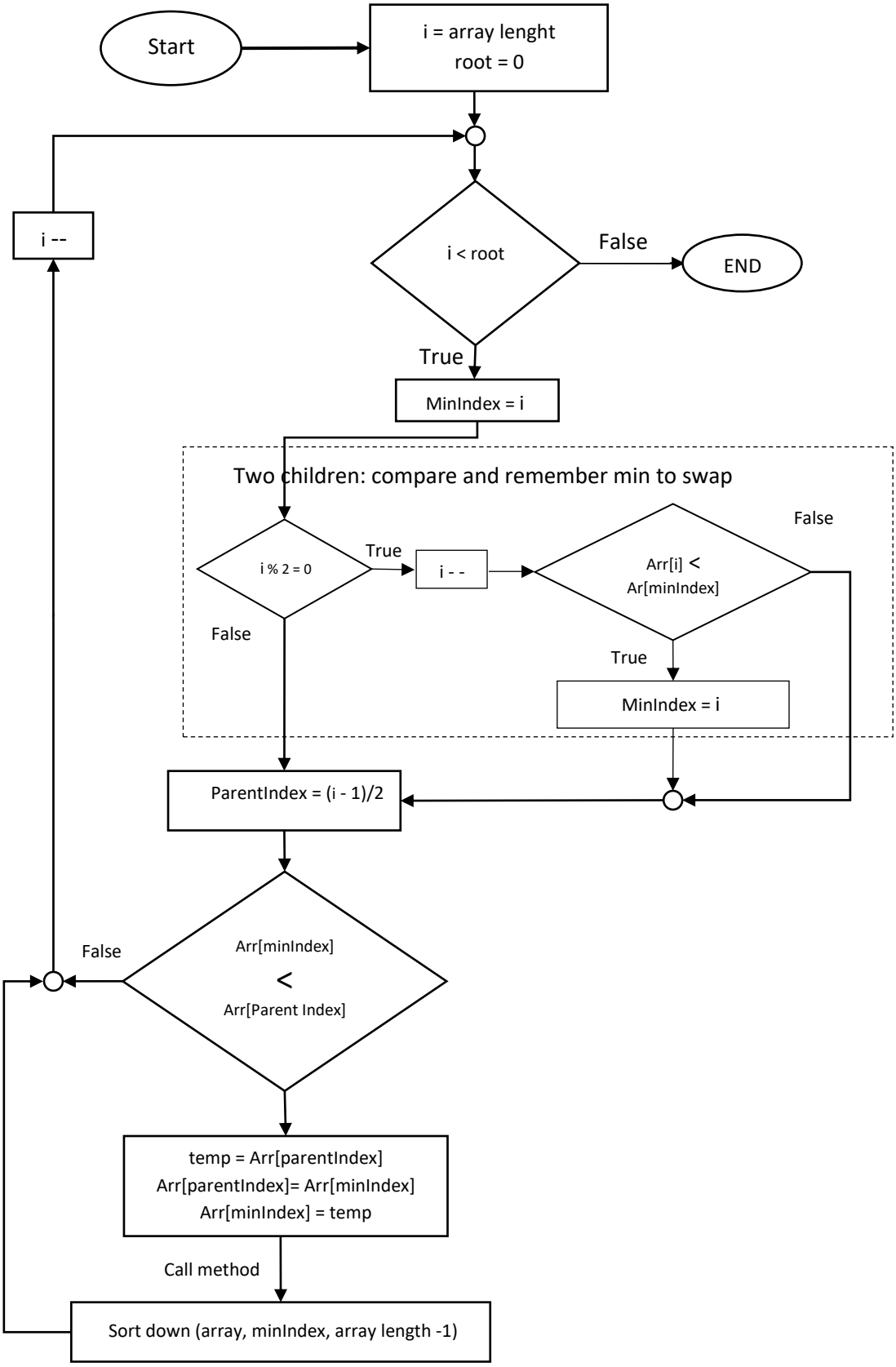
3.5.2 Heap Implementation:

The two arrangements of the heap could be a min or a max heap [33, 34]. we just saw that the heap representation of a random array is not a min or max heap. Therefore, we need to rearrange the elements in the array, so its tree representation is a minimum heap [33], (see flowchart2).

The implementation of the heap class in our model done by creating a heap with a general type such as a heap with an item type T. Then we create a public interface that will contain a heap index, heap index is the value we used for comparison. At the beginning we need to specify the heap size since it's difficult to resize the array construct our heap.

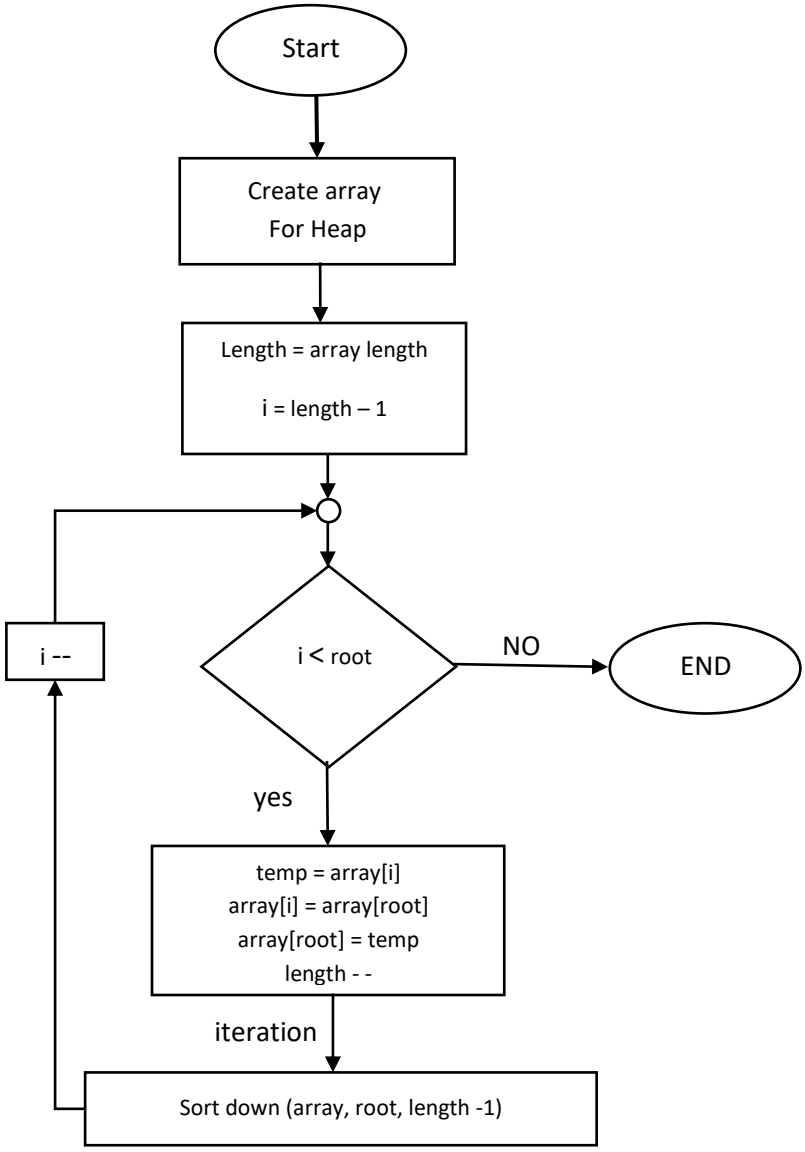
Also, we need to keep tracking of the current item count, then we build different methods in the heap class that will be used for adding or removing nodes or items.

Flow Chart2, Min-heap arrangement Algorithm [33, 34]



Another method for sorting up items see (Code 3.6, Appendix B), when we add a new item, we will keep checking the current item with its parents until it set where it belongs regarding other items. An important method that will swap items when the comparison result give that the current item value is lower than its parent item, of course the item value here means the fcost for the nodes. The swap method takes two attributes such as item A and item B. Another sort down method is also constructed see (Code 3.6, Appendix B).

Flow Chart3, Min-heap sort down Algorithm [33, 34]



Since in the case of removing an item, we remove the first item of the tree (root), then we take the item at the end of the tree (leaf) and make it the first item. At this state we make the sort down to resort our heap items (see flowchart3). Three are more methods for checking if the heap contains a specific item, or a method that return the current item count, and finally the method for updating the heap, since in some cases we can find a new path to a node that already added to the open set but with lower Fcost, this mean that the new path is shorter and we have to update or change the priority of the items. After we apply the heap data structure, we measure the time needed to find the path, the time of execution depends on the map and node size, and also depends on the obstacle complexity. So, for one case, the time of execution was 39ms when we use array or list data structure, but it takes only 7ms average time when we apply the heap data structure on the same environment (map and obstacle complexity), hence the time consumed get down from 39ms to 7ms for that case because of the way the heap works. Surely, for bigger maps and more complex environment more performance gain we can get as a result.

3.6 Multi-Path Management

At this stage we developed a system where a multi AARs can request paths from the path finding script. One thing to keep in mind if many AARs request the path at the same time, it might cause the system or the simulator to freeze for a few moments or to crash while it's doing all the calculations. Ideally, we want to make the system take all those requests and manage the processing for finding the solutions.

3.6.1 Path Request Manager:

We create a new class called a path request manager, and add this script to A* object, we use a queue to store all the requests. A RequestPath method constructed to receive all path requests. Another TryProcessNext method that call the StartFindPath method that will run the FindPath coroutine each time the path request manager enqueue a new path. For any current path request the attributes we need always to pass is the path Start and path End positions. To start the processing a StartFindPath method created in the pathfinding class and will be called inside TryProcessNext method. The priority of processing the requests done based on FIFO strategy. Other methods also created such as FinishedProcessingPath

method to return the result of processing if succussed or not, and to hold the processing if there is no request Enquire. Note that we always need to check if the start and target position are located in the free space or walkable area, this is important for optimizing matter of our processing, in other wards no need to process a request if one of its start or target points are unreachable, see (Code 3.7, Appendix B).

3.6.2 Moving AARs to follow paths:

In order to move the robots' objects created on the simulator towards their targets, we need to define a constant speed for any robot on the system and link each robot with a target. User should Know that we may connect more than one robot to one target. Then any robot should follow the path generated, the path connect it's start point position and the linked target position specified for this robot. A Unit class is the last script created for our model; we define a speed attribute that will represent the speed of the robots moving towards targets. On running we need to call the path request manager, the call should happen after the grid is generated, so we set the call in the unit class in an Update method start when we press the "keypad5" on keyboard, passing parameters of the start and end position with any new path request. With new OnPathFound method that will return a boolean set to true if we found the requested path. The next step after finding any path is to call the coroutine FollowPath, this coroutine will work only if the path request is successful and a path is found. It will use MoveTwards method to move the robot towards its target following its trajectory with the speed defined before, see (Code 3.8, Appendix B).

3.7 Testing the Model

The last class we created is the Unit class, we should attach this class to any robot created inside the simulated scene, and then we should link the specified target to the attached robot before we run any execution of the system. Of course, we can duplicate the AAR as many copies as we like, taking in consideration the hardware performance of our computer. Let's say we make Four AARs and two targets, in any new running the first step will be done is generating the environment on the workspace, the obstacles will be spread on the environment by reading the maze date file. Remember that the library of maze files generated by using the randomness method discussed before. After the environment or the

obstacles are ready the grid map will be built to decompose the working area. This map gives the guidance so we can know the walkable or nonweldable area. As we mentioned before starting the request manager is done when pressing a keybad5 key, it has been delayed in order to give the previous stages the time to be ready. So, when press the path request manager starter the paths or trajectories will be calculated passed on A* algorithm and the ARRs object will start to move toward the targets in the simulation. notice that for

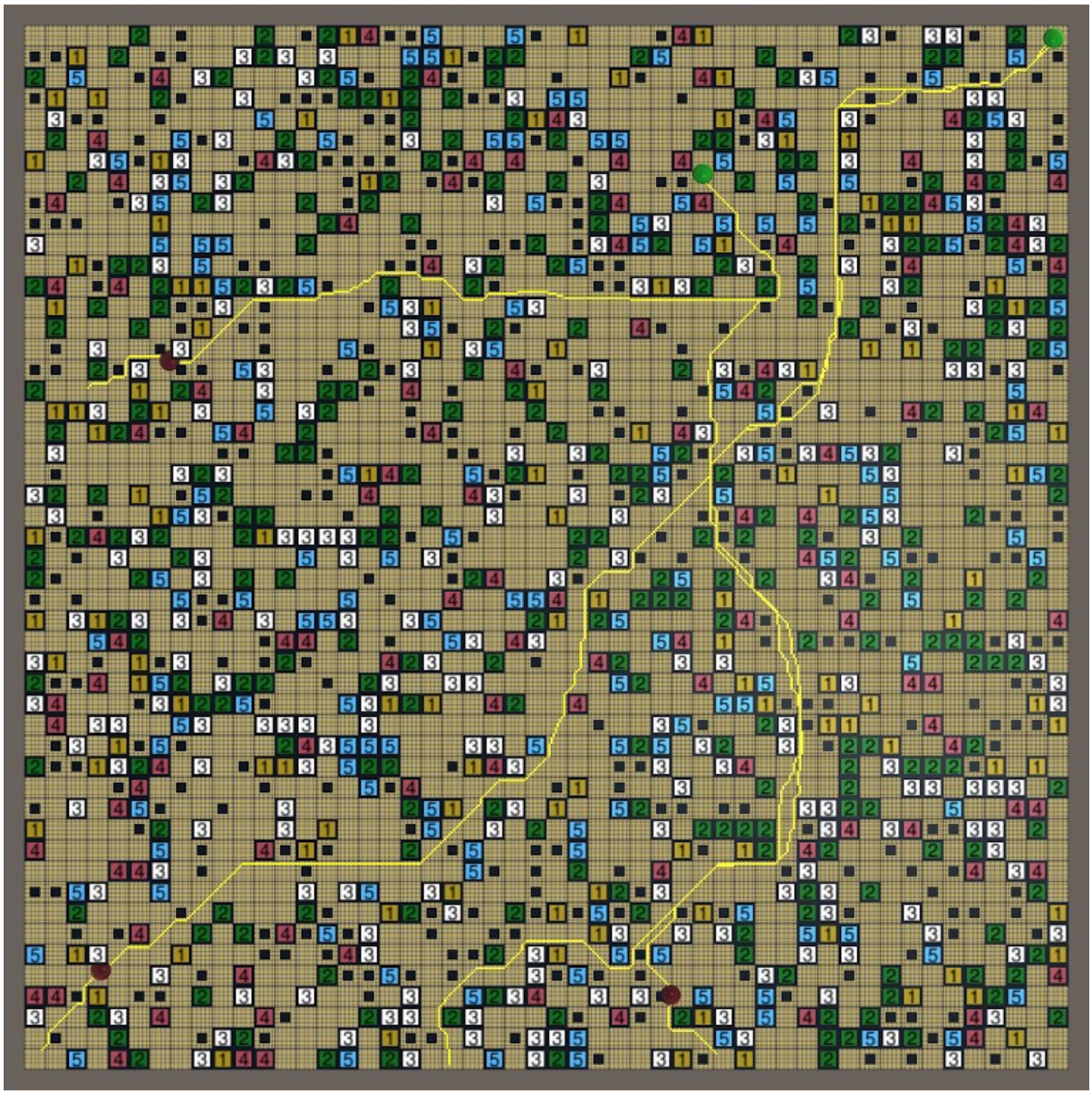


Figure 3.7: Snapshot of map taken on one of the experiments

research purpose and documentation, more methods are created such as OnDrawGizmo, this method is used to draw the results such as to draw the paths generated. Other methods

are written to measure the time of executions, or to write results to text files, or to take snapshots of high-resolution maps (see figure 3.7) for the environment and the resulted trajectories, see full script code at the Appendix B section page 62.

3.8 Summary

Before we summarize our implementation, I would like to clarify that the work of the implementation we did at GetLab laboratory at Paderborn University in Germany, this was during a DAAD fellowship for NRW Program researching purpose, the laboratory is equipped with good workstations (HP and Dell) Under Unix OS. You will see some codes specially the reading and writing to file may look strange or give error on execution, so it may need to modify in order to work under windows environment.

3.8.1 Summary of The Model:

The simulation Model we built under Unity simulator consists of the environment objects and the scripted classes. The environment objects such as the A* object that define the work area and its attributes. Work area contain the environment with its obstacles. Also, the AARs and targets. Obstacle objects of our model is classified into five different scales, the number on the top of each obstacle represent its hight. A library of maps generated randomly and stored on txt files so it can be used later when we do our experiments. Work area or maps can be placed at any altitude crossing the obstacle objects in our environment. Remember that our main aim of this research is to find the best altitude AARs can take to achieve a shortest trajectory between two points within HDFPOs. The overall gathering will be the minimum consuming of different resources.

Other objects in our model are the AARs, we can duplicate these objects and spread them on the environment, linking any AAR with A* and attach it to a specific target should be done before running the system or execute any experiments. We should always set the start point of any AARs at a walkable area or the free space, if we set the starting point on obstacle space or unwalkable area there will be no path found towards the specified target, the same concept should be applied also on our chosen target object, it should be also set or placed at a free space area too.

3.8.2 Summary of the Code:

To run the system correctly, we must understand the classes we built. Many methods call other methods, the procedure and the flow of data and information while processing is very important, some methods is delayed, and others work inside a wake function, and it can be run after pressing specified key. The nine classes that form the model scripts can be ordered in importance priority as follows:

Grid Class: The grid class composed of different methods; the main method is the CreateGrid method. It will create the work area at every execution, it will determine the free space and the obstacle space based on random generated obstacles.

Node Class: Node class define the node attributes; node is the basic constructor of our grid map. Node class also contain the fCost method that will call the sum of gCost and hCost methods, a comparison method that return the next lowest cost node.

Pathfinding Class: The pathfinding class is the main class of processing. It will call all methods and subroutine that will find a shortest path from start point to target point. It contains other methods for drawing the paths, or writing data and results to data files, or process calculation of distances that simplifying reading the results.

PathRequestManager Class: This class will manage all the requests of finding path during the processing. Our model allows to deal with more than one AAR and one target. In easy steps we can duplicate the seeking robot or the desired target in the simulator.

Heap Class: The need of the heap data structure method become urgent, after the collapse of system hardware, while process the searching based on array data structure. In the first version of our model the iteration was very huge, on such scale of node density. Finding the node with the lowest cost, heap is the optimization solution.

Unit Class: The unit class main role is for visual observation of the results. We can see the resulted paths, and we can see AARs moving towards the targets because of methods scripted in the unit class.

Other classes such as the **GridOverlay class** is for generating the random map data. **RandomPlacmentT class** is for building the 3D obstacle environment. And finally, the **HiResScreenShots class** that process taking high resolution square snapshots for every map and its solution if demand only. The screenshots class controls the coloring and implemented just for observation and documentation.

Chapter 4

Results

4.1 Introduction

In the previous chapter, we presented our proposed model for finding trajectories for a simulated system of AARs over randomly generated HDFPOs environment. The model is implemented under Unity simulator platform, and it's ready for performing the experiments needed for our research. On this chapter we will discuss the procedure we take for carrying out seventy-one experiments, the data we used, the results we extracted and documented, and the comparison of all results.

To extract a solution for the problem of finding the best altitude that a system of AARs can take. Where this system of AARs need to travel between different starting and end points in a HDFPOs environment. The methodology we used to find the solution after we build the simulation model for the problem is by executing many experiments, applying the 2D implemented solution on different altitude on the 3D generated environment Model. We consider ten different random maps. All results collected are documented and compared later, to find the extremal values such as the minimum value that fulfil a solution. A curve fitting performed on data collected to generate the equations that help us on finding the solutions we are looking for. As we know the obstacles minimum and maximum hight is known and finite, so, we run these experiments among many levels between the ground level and the maximum hight level. We take in consideration the average height of all the environment obstacle.

At first stage we produce the random map data files where 10 different density and distribution obstacle maps are generated and saved. On the second stage we run five path searches for each map on five different altitudes. For each map a method pre calculates the average height of all obstacles, then we select the altitudes of interest around this Obstacles Average height **OAH**. Next for the built environment inside the simulator we used 5 objects that represent 5 AARs and 2 objects that represent two targets for every experiment we carry out. On the last stage, the results printed out on text data files, it contains data such

as the grid size and the node radius, the obstacles volume and the calculated average height, the ratio between obstacle and free workspace. It also contains results about the distance each robot will travel to reach its target and the time it consumes for finding the shortest path for each robot. More data such as the number of diagonal node and the vertical or horizontal node in each path found also calculated (Figure 4.1).

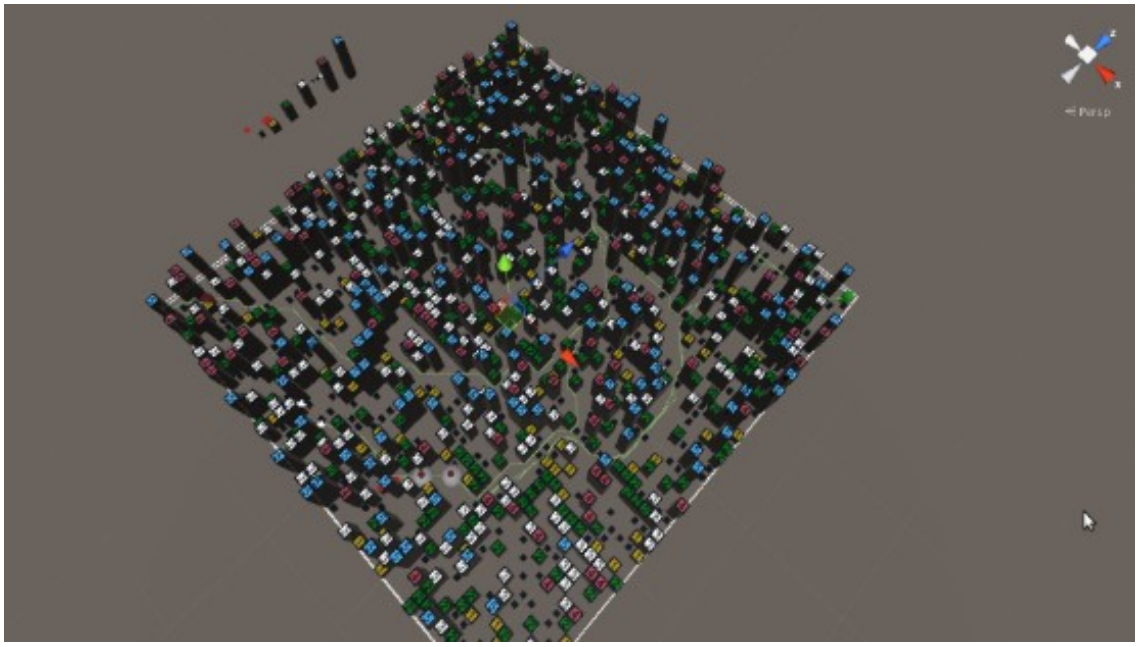


Figure: 4.1: Result data file

The average height is calculated by summing the volume of all obstacles, then we divide the result by the obstacle area. For example: on one of the 3D environments maps, the average high of obstacles is 4.0275 unit, then we execute five experiments on five different altitudes. These altitudes are chosen to cover elevation a above and under the average hight of that experiment obstacles. Consider one more point, when calculating the path length for AAR, we assume that any start or end point will be placed on the ground altitude or the zero elevation. So, for the 2d path solutions on other than the ground altitude we must add the vertical distance from start point to reach that level and add the vertical distance from that altitude back to target point on ground altitude.

4.2 The procedure for running the simulator:

Before we run any experiment, there are few attributes we should set or prepared. For example: Let us perform a testing experiment to verify the execution steps. At first, we define the map file path on the hard drive, the file that we need to read data of the map from it. Then we set some values for our experiment such as the dimension of our map. For this testing experiment we used a 50 by 50 grid with 0.5 node radius, the total number of map nodes will be 25000. Then we set the speed for any AARs to be a constant value such as 10 unit/s. We also set the file path of result file on the hard drive too. Then define the resolution dimensions of the result screen shot to be (3000X3000 pixels), this is an output image file that will be saved after running the experiment, it will represent an imaginary result data for the map, obstacles, and the solutions, this attribute can be set in the script (HiResScreenShots) attached to the camera Object, see Figure 4.2.

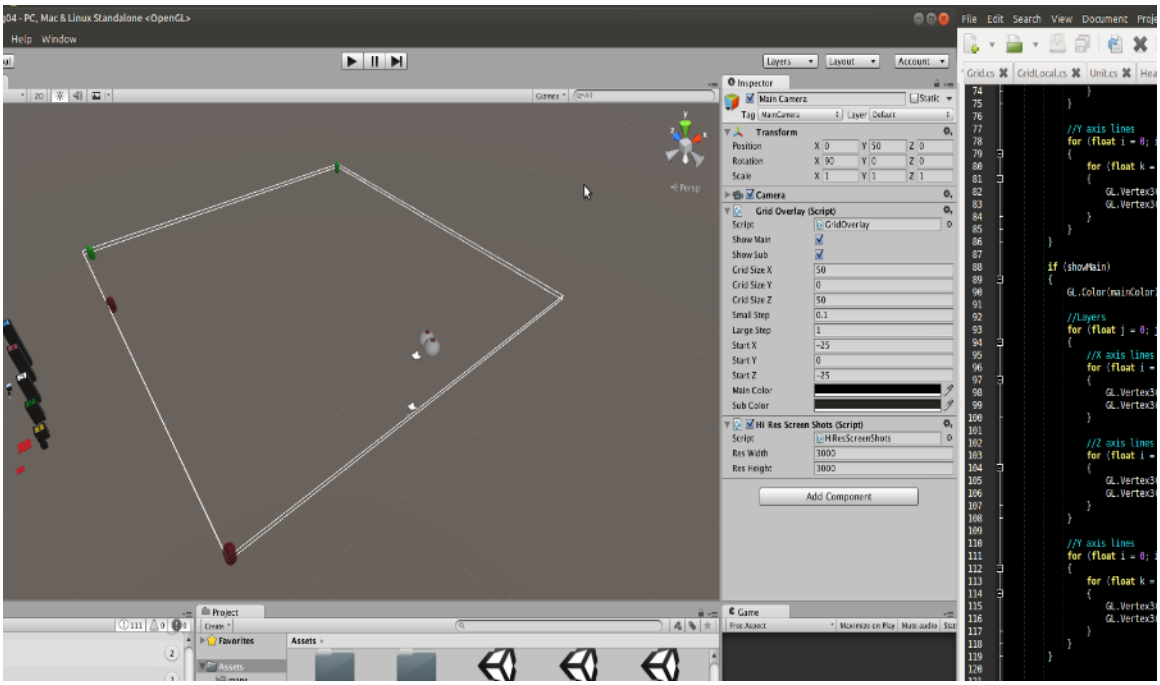


Figure 4.2: Screenshot setting

After we prepare and modify all the settings needed to perform the execution, we press the run button on the simulator interface, in the first three seconds we must press the key on keyboard that assigned for generating the 3D map, in our case we assign the arrow down key where the Update method code can be modified on RandomPlacmentT class. After the map is generated, another Update method in the Unit class waiting to start the request of

the path's solution, we assign the keybad5 input key to start the processing data we mentioned before. Results will start to show up on the console and saved on the result data file when we press keybad5 on keyboard. The last assigned key is the "k" key that take a high-resolution screenshot for the map and solution, the image file will be saved on screenshots folder inside the asset of the project folder. After we test the model and discuss the procedure of running any experiment, know we ready to expand our experiments and collect all the data needed for comparisons. In the next section we will explain the data of two parts of experiment we perform.

4.3 Performing experiment and data collection

4.3.1 Result of Experiment One:

10 maps generated and prepared for testing; 5 executions made on each map to find path solution for 5 distributed AARs.

Table 4.1: Experiment Part1, Data and results

Input Data (Experiment Number)					Output data: Path Length for the 5 robots					Output Data: Time of Processing in milliseconds				
Experiment Number	MAP	Obstacle Average High (OAH)	Altitude of Flight	Obstacle/Free Ratio	PATH1	PATH2	PATH3	PATH4	PATH5	PATH1	PATH2	PATH3	PATH4	PATH5
1	map01	1	0.1	4.608 %	5374	5682	4900	5044	5246	14	8	8	7	29
2	map01	1	0.55	4.608 %	5374	5682	4900	5044	5246	15	9	10	9	29
3	map01	1	1.2	0.00%	5374	5682	4900	5044	5246	13	8	7	7	27
4	map01	1	1.8	0.00%	5374	5682	4900	5044	5246	12	8	7	7	29
5	map01	1	2.1	0.00%	5374	5682	4900	5044	5246	12	8	7	7	29
6	map02	3.124	0.3124	6.2304 %	5374	5682	5068	5044	5246	112	84	74	6	8
7	map02	3.124	1.7182	4.952 %	5374	5682	5068	5044	5246	122	100	79	7	8
8	map02	3.124	3.7488	2.8184 %	5374	5682	4908	5044	5246	79	69	42	7	8
9	map02	3.124	5.6232	0.00%	5374	5682	4900	5044	5246	15	8	8	30	7
10	map02	3.124	6.5604	0.00%	5374	5682	4900	5044	5246	15	8	31	6	7
11	map03	2.85	0.285	14.0948 %	5374	5682	5140	5086	5262	128	151	90	20	28
12	map03	2.85	1.5675	11.186 %	5374	5682	5052	5052	5246	130	160	71	23	21
13	map03	2.85	3.42	4.776 %	5374	5682	4908	5044	5246	102	14	37	7	11
14	map03	2.85	5.13	0.00%	5374	5682	4900	5044	5246	16	8	8	7	7
15	map03	2.85	5.985	0.00%	5374	5682	4900	5044	5246	15	8	8	29	6
16	map04	2.994	0.2994	33.8396 %	5436	5730	5204	5094	5298	102	109	87	129	127
17	map04	2.994	1.6467	25.4916 %	5388	5682	5068	5076	5270	123	19	50	122	94
18	map04	2.994	3.5928	14.6628 %	5380	5682	5020	5044	5246	135	58	56	129	101
19	map04	2.994	5.3892	0.00%	5374	5682	4900	5044	5246	12	8	8	7	7
20	map04	2.994	6.2874	0.00%	5374	5682	4900	5044	5246	14	8	8	7	7
21	map05	2.871	0.2871	62.218 %	12762	7130	7704	6152	6316	343	200	186	102	131
22	map05	2.871	1.57905	47.572 %	11468	6048	5884	5668	5812	835, 628	173, 20	147, 64	147, 43	188, 42
23	map05	2.871	3.4452	25.3404 %	5482	5688	5144	5056	5246	408, 29	164, 12	101, 9	95, 21	72, 36
24	map05	2.871	5.1678	0.00%	5374	5682	4900	5044	5246	15	8	8	7	7
25	map05	2.871	6.0291	0.00%	5374	5682	4900	5044	5246	15	8	8	7	7
26	map06	1	0.1	32.8032 %	5486	5818	4900	5160	5350	323	347	33	277	311
27	map06	1	0.55	32.8032 %	5486	5818	4900	5160	5350	303	349	31	288	315
28	map06	1	1.2	0.00%	5374	5682	4900	5044	5246	15	9	8	8	29
29	map06	1	1.8	0.00%	5374	5682	4900	5044	5246	15	8	8	7	29
30	map06	1	2.1	0.00%	5374	5682	4900	5044	5246	15	9	9	8	31
31	map07	2.927	0.2927	62.7344 %	8160	7514	7386	6804	6930	149	188	170	142	121
32	map07	2.927	1.60985	48.158 %	7632	6642	7108	6254	6386	248	301	319	278	278
33	map07	2.927	3.5124	26.2516 %	5464	5694	5504	5152	5306	120	157	101	275	257
34	map07	2.927	5.2686	0.00%	5374	5682	4900	5044	5246	15	8	7	7	28
35	map07	2.927	6.1467	0.00%	5374	5682	4900	5044	5246	15	8	8	7	7
36	map08	2.331	0.2331	62.8464 %	12938	6464	7096	6118	6320	266	95	66	153	133
37	map08	2.331	1.28205	47.7336 %	7638	6166	6784	5466	5644	329	128	110	167	153
38	map08	2.331	2.7972	19.58 %	5406	5690	5172	5066	5274	231	59	126	192	271
39	map08	2.331	4.1958	8.3764 %	5374	5690	4932	5044	5246	161	37	16	28	47
40	map08	2.331	4.8951	8.3764 %	5374	5690	4932	5044	5246	160	36	17	21	49
41	map09	3.006	0.3006	34.9324 %	5494	5982	5504	5152	5438	101	161	127	223	222
42	map09	3.006	1.6533	27.6532 %	5452	5744	5264	5152	5438	113	177	109	299	284
43	map09	3.006	3.6072	14.7172 %	5374	5682	5028	5044	5246	137	100	100	128	166
44	map09	3.006	5.4108	0.00%	5374	5682	4900	5044	5246	14	8	8	7	7
45	map09	3.006	6.3126	0.00%	5374	5682	4900	5044	5246	15	9	9	8	8
46	map10	2.703	0.2703	14.3768 %	5374	5746	4964	5044	5246	109	66	38	51	40
47	map10	2.703	1.48665	10.4412 %	5374	5682	4932	5044	5246	135	17	26	44	32
48	map10	2.703	3.2436	3.7104 %	5374	5682	4916	5044	5246	85	10	14	13	40
49	map10	2.703	4.8654	2.1592 %	5374	5682	4916	5044	5246	96	10	14	17	28
50	map10	2.703	5.6763	0.00%	5374	5682	4900	5044	5246	15	8	7	7	7

Each execution performed on different altitude. We try to make the altitude of interest around the average height of obstacles on each specific map. Our goal from the beginning is to find the best altitude of travelling for a system of AARs in HDFPOs environment. The altitude in the first two execution will be under the average height of obstacles while the other three execution will be above the average height of obstacles OAH. Of course, on every five executions that represent one map experiment, we modify our map data file every next five executions. Finally, the data of 50 execution is collected and gathered in the result files, we summarized all results as shown in table 4.1.

Before we explain the table data, we just clarify that these experiments performed using maps with 50 by 50 dimensions and a node radius of 0.05 unit, which means that in our map we have 250000 nodes involved in the processing. For each map, 5 different altitudes generated by multiplying a factor with OAH, with 10 different maps and five executions for each map, the total result give us output data for 50 executions. Column 2 shows the map data file we read to construct our environment, column 3 represent the OAH, the average height of all obstacles calculated as clarified before in section 4.1. Remember that the highest obstacle value is 5-unit, column number 4 represent the altitude of flight for each experiment. That means where we put the intersecting plane for processing A* algorithm to find the solution located at the 3D environment. Column number 5 contain the ratio between obstacle space to free space at that particular altitude, this ratio changed and decreases whenever we go up in our flight, it's obviously clear that the highest ratio will be at zero level, since all obstacles involved in this ratio for calculations. The next five columns contain the solution data for the path's length for each Robot in the experiment. The number in these columns represent the total number of nodes the robot will travel from the start point to the target node. And the last five columns contain the time of processing to find the path solution for every robot in each experiment.

If we look at the data of our maps, we can observe that map number one contain a distribution of one size obstacle, the obstacle with 1 unit volume, which mean it has 1 OAH, and the ratio of obstacle space to free space at any level under altitude of 1 unit is very low and equals 0.46%, and whenever we go above that altitude the ratio of obstacle

will be zero. Another map is common and has value of 1 OAH is map number 6; but at map 6 the ratio of obstacle space to free space under 1 altitude is 32.8%. We can also see that the maximum OAH for all experiments is the one of map2, the maximum value is 3.124 and the obstacle to free ratio is about 62.3% at any altitude less than 0.5 unit, this is because the smallest size of the obstacles is 0.5 volume unit. The obstacle to free space ratio decreased dramatically at every step up of raising the altitudes value, at this map (map2) particularly, if we observe the length of the paths or distance the robot will travel, we can see that whenever the density of obstacles decreased, the path will go shorter until some level of density measurements. This observation will be clarified when we will apply the curve fitting method in order to observe more about the results.

Timing is calculated starting when we call the request of any path and end when the solution is found. From the results in the last five columns, we can see that there is expanded raising of processing time whenever the obstacle to free space is rise. For example: a high ratio of obstacle density on experiment of map5 was 47.5%, the time of processing to find path1 was 835ms, it is a very long time in compare with average time of all execution that equal to 73ms (see sheet average time of execution in model excel file)

4.3.2 Experiment Part2:

In experiment execution part two, we give more focus on one map, but expand the execution on wide range of altitude of flight, we take map8 as our focus map sample. The range of altitude changed in between 21 different values, it starts from 0.1 unit, then slightly incremented at ten levels. The other next eleven increments extend higher altitude that at end exceeds all obstacles; the reason for this is to check if the flying over all obstacles can make the trajectories of AARs shorter. Remember that we take into account the distances the robots will travel to reach these elevations or altitudes. Table 4.2 shows the five robots path finding results on 21 executions on map 8 using a wide range value of altitudes as discussed. On column three we can observe the change of obstacle to free space ratio, the

starting value is 62.85% and decreased at every increment of altitude, while when we reach executions 15, this ratio become zero since we skip up all the obstacles. In other words, the altitude of executions number 15 to 21 is done above all the obstacles in the map,

Table 4.2: Data of Experiment Part2 and results

Input Data (Experiment Number, Map, O\F Ratio, Altitude, OAH = 2.33)				**Output data: Path Length for the 5 robots (2D)**					**Output Data: Distance robots will Travel (3D)**				
Experiment Number	MAP	O/F Ratio	Altitude of Flight	PATH1	PATH2	PATH3	PATH4	PATH5	Robot1	Robot2	Robot3	Robot4	Robot5
1	map08	62.8464 %	0.1	12938	6464	7096	6118	6320	12958	6484	7116	6138	6340
2	map08	62.8464 %	0.2	12938	6464	7096	6118	6320	12978	6504	7136	6158	6360
3	map08	62.8464 %	0.4	12938	6464	7096	6118	6320	13018	6544	7176	6198	6400
4	map08	58.8864 %	0.6	12892	6426	7028	6134	6312	13012	6546	7148	6254	6432
5	map08	58.8864 %	0.9	12892	6426	7028	6134	6312	13072	6606	7208	6314	6492
6	map08	47.7336 %	1.3	7638	6166	6784	5466	5644	7898	6426	7044	5726	5904
7	map08	47.7336 %	1.8	7638	6166	6784	5466	5644	7998	6526	7144	5826	6004
8	map08	19.58 %	2.3	5406	5690	5172	5066	5274	5866	6150	5632	5526	5734
9	map08	19.58 %	2.6	5406	5690	5172	5066	5274	5926	6210	5692	5586	5794
10	map08	19.58 %	2.9	5406	5690	5172	5066	5274	5986	6270	5752	5646	5854
11	map08	13.7104 %	3.3	5374	5690	5092	5044	5246	6034	6350	5752	5704	5906
12	map08	13.7104 %	3.8	5374	5690	5092	5044	5246	6134	6450	5852	5804	6006
13	map08	8.3764 %	4.2	5374	5690	4932	5044	5246	6214	6530	5772	5884	6086
14	map08	8.3764 %	4.7	5374	5690	4932	5044	5246	6314	6630	5872	5984	6186
15	map08	0.00%	5.2	5374	5682	4900	5044	5246	6414	6722	5940	6084	6286
16	map08	0.00%	6	5374	5682	4900	5044	5246	6574	6882	6100	6244	6446
17	map08	0.00%	7	5374	5682	4900	5044	5246	6774	7082	6300	6444	6646
18	map08	0.00%	9	5374	5682	4900	5044	5246	7174	7482	6700	6844	7046
19	map08	0.00%	12	5374	5682	4900	5044	5246	7774	8082	7300	7444	7646
20	map08	0.00%	15	5374	5682	4900	5044	5246	8374	8682	7900	8044	8246
21	map08	0.00%	18	5374	5682	4900	5044	5246	8974	9282	8500	8644	8846

and by looking on all distances travelled by robots, we can observe that flying over all obstacles is not giving the shortest distances. The last point will be clarified in the next section when we apply the curve fitting method to analyse our results.

4.4 Curve Fitting Approach:

Curve fitting is the process of constructing mathematical formula or a curve, that has the best fit to a series of data points, possibly subject to constraints, and it is one of the most widely and most powerful analysis tools. Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a smooth function is constructed that approximately fits the data, despite that curve fitting examines the relationship between one or more predictors variable which might be independent variables and a response variable which might be dependent variable, with the goal of defining a best fit model of the relationship, the relationship might be linear or nonlinear, if its linear then we will get a linear fit, in other relations we might get a polynomial fit or a multiple linear regression, so its origin provides tools for linear, polynomial, and nonlinear curve fitting along with validation and goodness of fit tests.

Fitting curves as a tool can be used as an aid for data visualization, to infer values of a function where no data are available, and to summarize the relationships among two or more variables. Extrapolation refers to the use of a fitted curve beyond the range of the observed data and is subject to a degree of uncertainty since it may reflect the method used to construct the curve as much as it reflects the observed data.

We can summarize and present our results with customized fitting reports. There are many time saving techniques such as a copying operation feature which allows us to copy just completed fitting operation to another curve or data packet or columns. Curve fitting operations can also be part of analysis template, allowing us to perform batch fitting operations on any number of data files or data columns.

4.5 Curve Fitting of Experiment part1:

The result of table 4.1 can be analyzed so we can formulate the relation between the altitude of flight and the distance the robot travels in a certain density of obstacles on any known

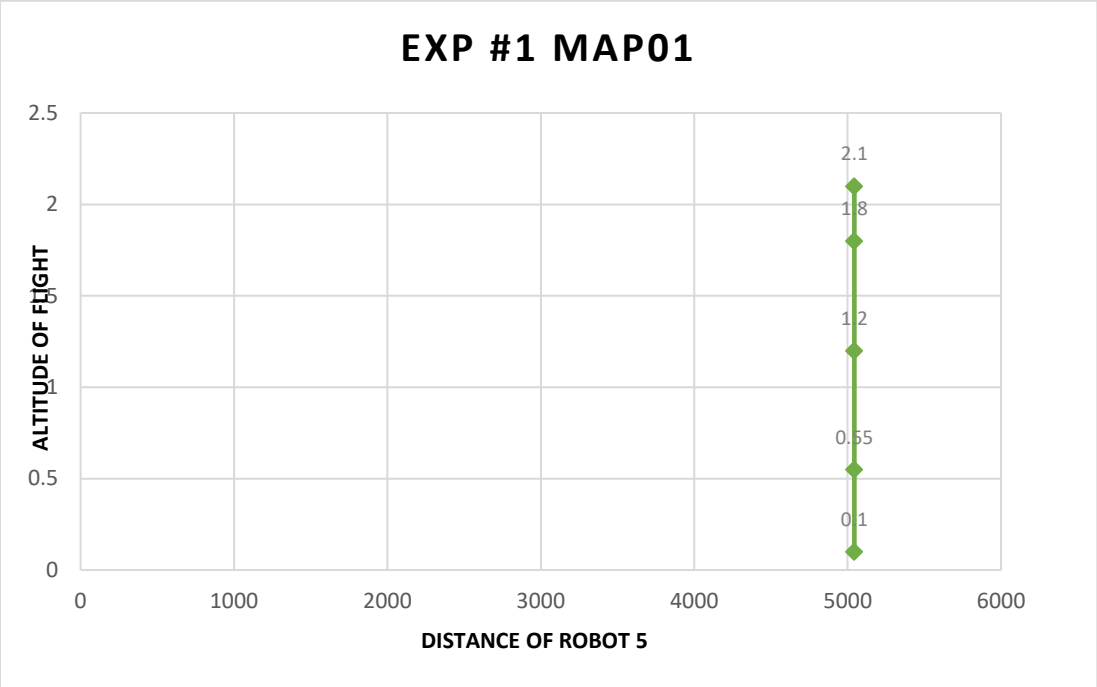


Figure 4.3: Curve Fitting first experiment

HDFPOs environment. This relation can lead us to the extremal values where we can find the lowest distances that achieves the shortest paths for our AARs, in experiment number 1 map01, the effect of altitude of flight to the distance the robot will travel is zero, the distance set to same value despite the altitude increment is, this is because the map contain a small ratio of obstacles, with a value of 0.4608% ratio of obstacle space to free space, it's clear that the density of obstacle at this map is very low and neglected, on figure 4.3, we can see the relation between altitude and distance of travel is linear and a constant value that unchanged.

On experiment 2 map 2, only distance of robot 3 from the five robots decreases when we increment the altitude of flight, here we can see also that the obstacle to free space ratio is still low and under 10%, see Figure 4.4

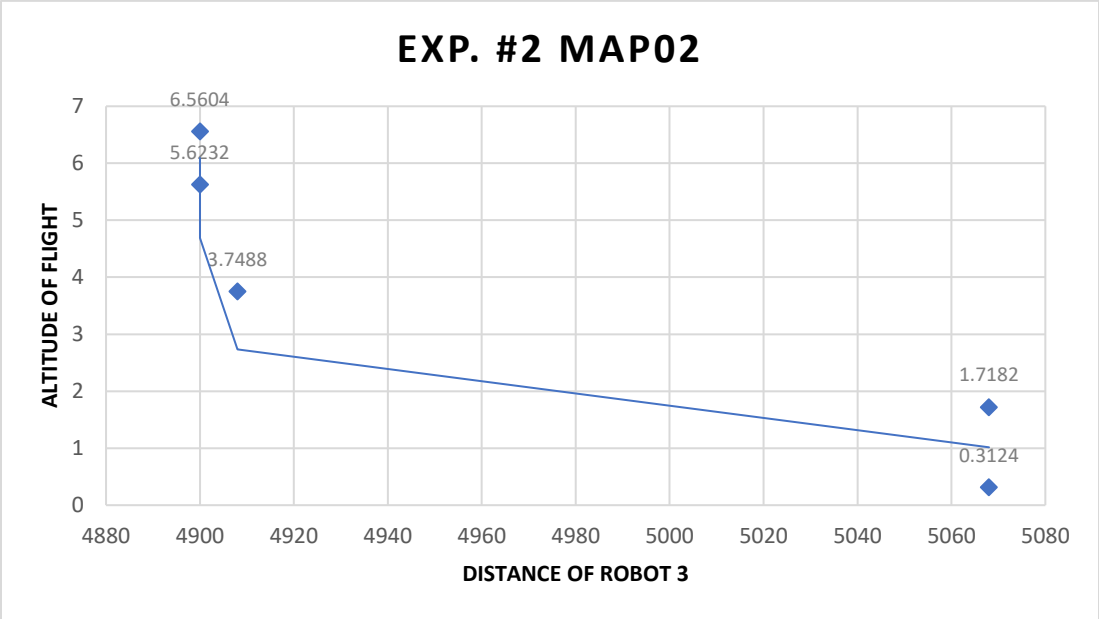


Figure 4.4: data of Exp.#2-Map02

If we draw a chart for the result of map 07 where the density of obstacles decreased from

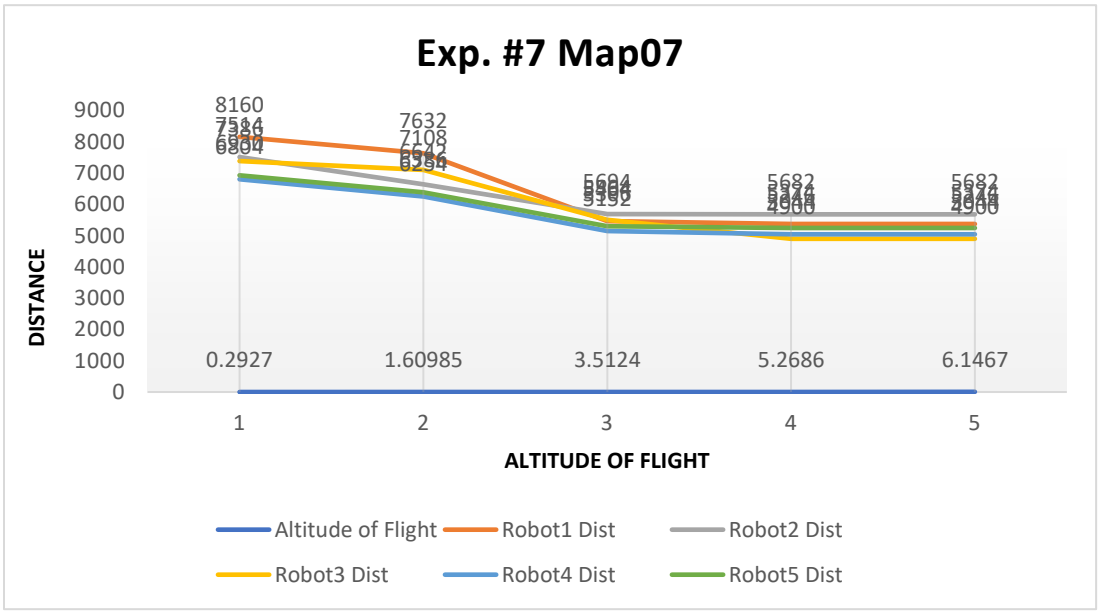


Figure 4.5: Result of Exp.#1 - Map07

62.7% to zero when we increment the value of altitude of flight, then we will notice the decrease of the value of distance of flight for the five robots, see figure 4.5

4.6 Result of Experiment two:

In Part1 experiments, we can see that the data we collect give us a little indication about the relation between altitude of flight and the distance the robot travels within changeable density obstacle environment, results show that in some cases the distance decreased when we increase the altitude of flight over the zero level, then it goes constant for some margin, so we decide to expanded the data of one map to see more about the effect of changing altitude over a HDFPOs map and how it affect the path distance for each Robot, on part two of experiment we take map08 as our environment, we did a 21 execution on different altitudes, the increments made a big variation of the obstacle density, it changed from 62.7% to zero over the 21 executions, the results show that there is a critical extreme (minimum value) and the distance decreased for all robots as show in data of table 2, the five red columns represent the path length for each robot at any altitude, the next five columns represent the distance the robot will travel from a start point to destination point, this distance is equal to the path length added to a double of altitude distance value, the double altitude value represent the vertical distance of taking off and landing, since our start point and target represent a position on the map at ground. The curve fitting formula

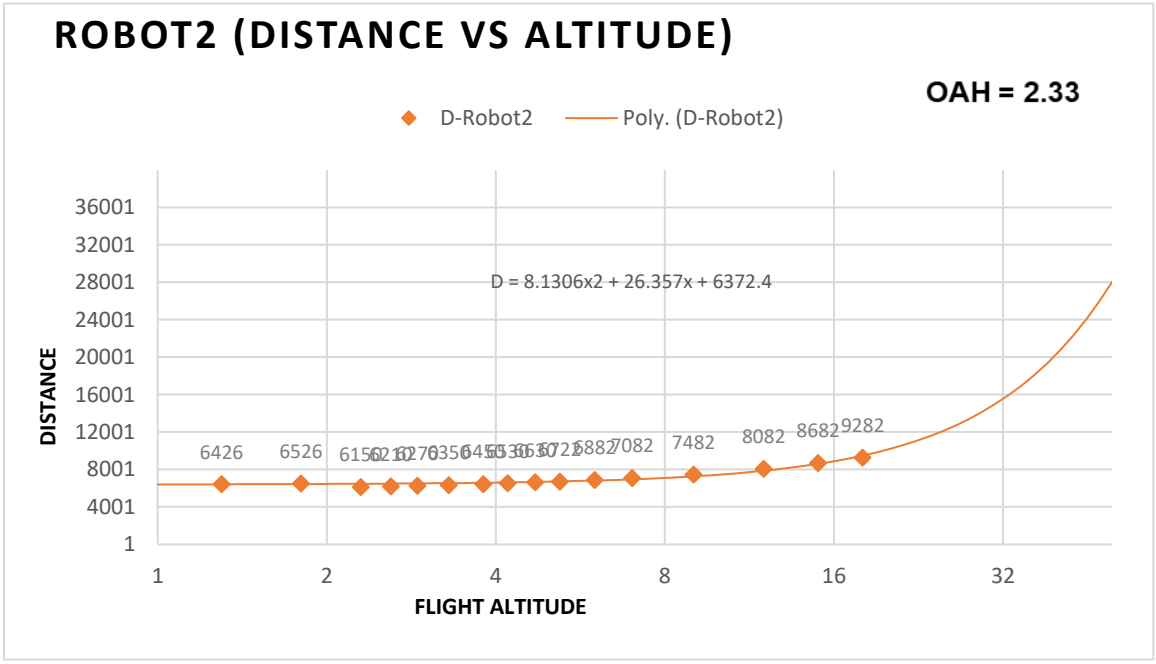


Figure 4.6: Exp. 2 Robot2 Results

of robot #2 is shown on Figure 4.6, the most accurate fit for the data generated is a

polynomial formula, even if we predict or forecast data forwards or backwords, it still gives the same fit, an equation is generated and the formula from the data of distance and altitude is predicted as ($\mathbf{D = 8.1306x^2 + 26.357x + 6372.4}$), x is the altitude variable, the polynomial is concaved up so it has a minimum value that represent the shortest distance, the minimum value of distance from table 2 for the second robot is 6150, this minimum value occurs at execution 8, the altitude of flight is 2.3, remember that the obstacle average hight is 2.33, this is an indication about which altitude of flight can give the shortest distance robots might travel.

If we compare all results of part2 experiment, we can find that all the critical value of the formula fitted is near the value of obstacles average high, if we look back on table 2, then we can observe that execution number 8 where the altitude is 2.3 gives a minimum distance value for the five robots, Figure 4.7 contain all result of experiment 2, for more analysis and curve fitting for each execution, you can check the excel data file with the name Model.

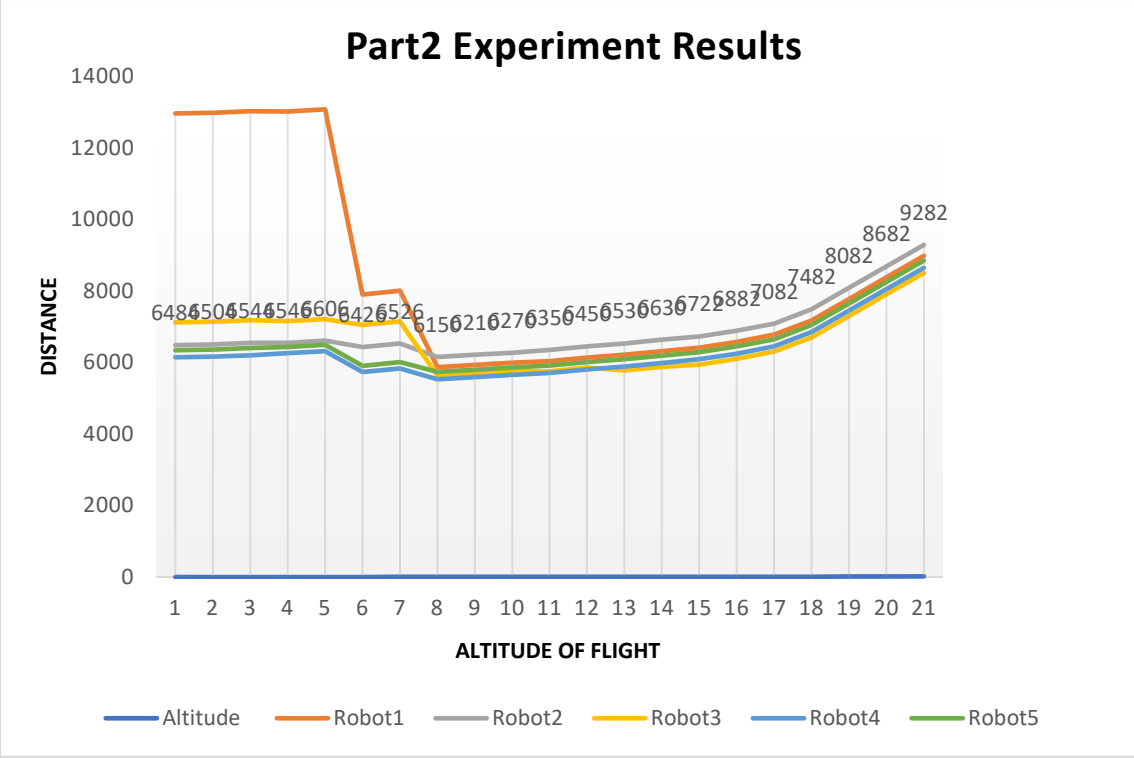


Figure 4.7: Result of part2 Experiment

One more execution done on part1 experiment, this execution is to evaluate the effect of changing the data structure from array or list to a heap, as we discussed before, heap data

structure minimizes the time of processing very rapidly, as we can observe from table one, the last five columns contain data about execution time in milli second, we can see that the time increases when the complexity of environment is high, in other words, whenever the ratio of obstacle is increased the time of processing also increased, the processing was worked out on Personal computer of a specification (i5 cpu, 512 Ram), in table one, the two yellow rows is a comparison of execution time between array and heap data structure

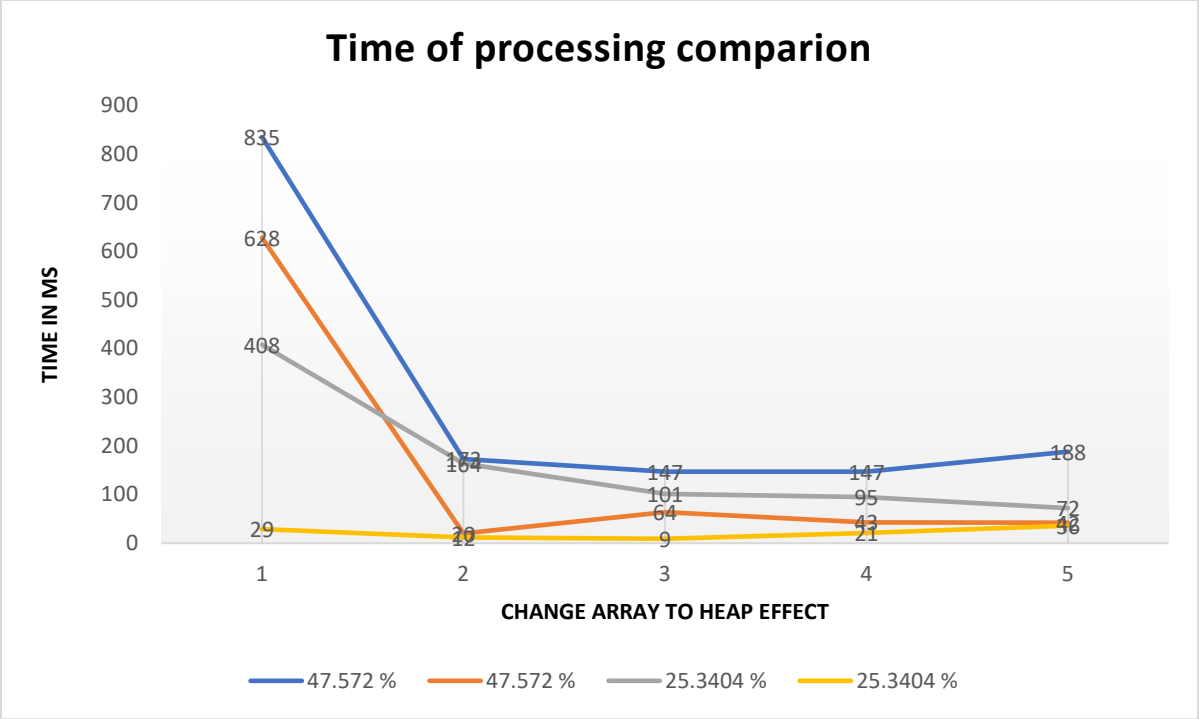


Figure 4.8: time processing comparison

for the same test, the first number is the time needed to find the solution when array data structure is used, the second number is when a heap data structure used, obviously, time decreased rapidly when heap is used, (Figure 4.8) represent the comparison and relation between obstacle complexity and processing time.

4.7 Results Summary:

For known HDFPOs environment, a question about what is the best altitude that drones, or quadrotors should flight at, the best altitude that achieves the shortest distance traveled by a system of AARs, we build a simulation model and execute two parts of experiments, the first results indicate that the shortest distance is not the ground level or the zero altitude when the obstacle density is high, expanding data in part 2 experiments give us minimum value for the shortest distance, when we compare the result with the attributes in our model, we found that its very close to the obstacle average Hight, (Figure 4.9) summarize results,

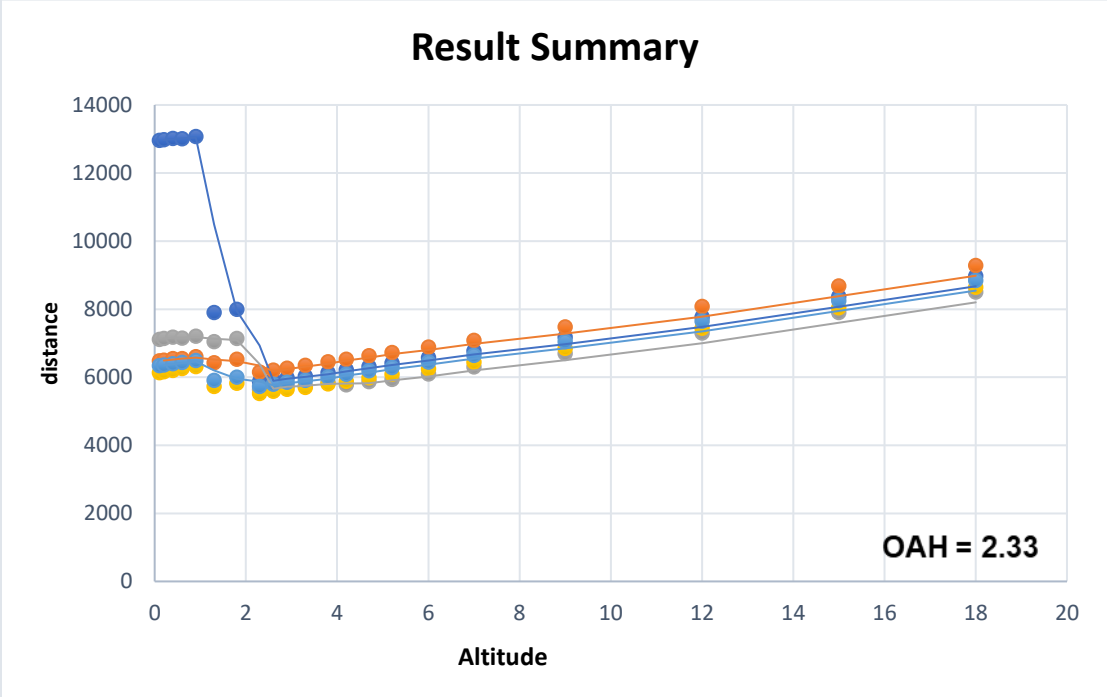


Figure 4.9: Result Summary

a relation between altitudes and distance travelled by robots shows that there is minimum value located at very close value of the OAH.

Chapter 5

Conclusion and Future Work.

5.1 Introduction

This chapter concludes the thesis and give a review of the importance of results and the proposed model. Focusing on the thesis contributions, results, limitation, and future work. As the problem of producing a system of aerial robots that can fly autonomously is still a challenge, a variety in research fields such as the vision, the odometry readings, the motion planning, and other fields are still under progress. In this thesis, we take special case when the environment contains a HDFPOs, that simulate a city with high buildings, and we ask the question, what is the best altitude for flying that a system of autonomous aerial robots can take and make the path or distance between two points is the shortest? To answer this question, we built a simulation model under unity simulator, a random HDFPOs were generated, a system of AARs is then involved in a searching experiment over several altitudes, the search done based on A* algorithm to find the shortest distance between start and target points. Results shows that the best solution comes when the altitude overcome the average high of all obstacles, which means that a system of AARs need to overcome more than half of obstacle density in order to plan the motion with the least distance. As in part 2 experiment, the density of obstacle at ground was 62.85%, when we exceed the average high of obstacle, we decrease the obstacle ratio more than the half, and then the free space ratio reaches about 80%. Our curve fitting analysis shows that the altitude that achieves the critical minimum value is a value near the obstacle average height.

5.2 Contribution

In our research, we have three main contributions:

1) Extracting the problem of motion planning for AARs for a special case when the environment contains a high-density polygonal obstacle.

2) Building a simulation model for researchers who need to plan motion over HDFPOs.

3) Finding the best altitude of flying for AARs in a HDFPOs environment that simulate a system of quadrotors or drones doing a task or a service in large cities.

5.3 Results

Our proposed model is evaluated, and experiments result are tested using simulation. The experiment data that collected then is curve fitted and the results give us polynomial equation (**$D = 8.1306x^2 + 26.357x + 6372.4$**) that represent a solution with a minimum value that achieved the shorted distance, where x is the altitude of flight attributes.

5.4 Limitation

The limitation of our research is as follows:

1) The creation of the dataset was not simple; collecting and building more data set need to expand experiments and manipulate many variables over the environment and the simulated model.
2) Limited working laboratory environment, the simulation crashed many times when we try to expand experiments and manipulate many variables, all experiment done on a personal computer.
3) Limitation of testing the simulated model over a real-world environment.

5.5 Future Work

In our research we still have future work of finding a solution to prevent collision of robot that might share the same path and meat at one point at a moment, we think of giving a unique number for each robot with priority, so the robot with higher priority may fly over the robot with lowest priority number at point of collision, another future work is to do our simulation on an environment that contain obstacle with many shape, in our research we use only a polygonal obstacles, but the real world contain obstacles with different shapes, this step can be done easily on the simulated model, since we can build any obstacle shape and add it to the obstacle mask layer. Finally, I read an article in scientific magazine, a paragraph of the article says:" A woman unloads merchandise sent by a drone during a media show in a large city. The Drone's Initiative has launched the third pilot phase of delivery of orders with drones. The two-year project aims to create a national network for

the use of commercial drones to send goods, including medicine, food, medical equipment, and other things, to customers who order goods via the Internet.", these wards mean that the future of AARs that do services are coming, yet a lot of work need to be done from researchers to achieve autonomously traveling for robots. (Figure 5.1) shows a woman loading goods to drone in service for delivery. And (Figure 5.2) shows a system of drones in a working task for delivering goods, both photos were taken from the article mentioned before.

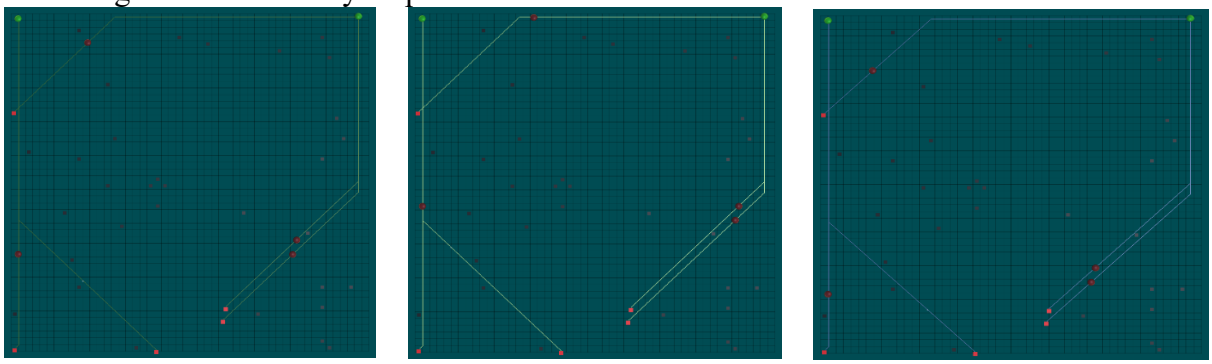
Figure 5.1: woman loading goods

Figure 5.2: a system of drones in service

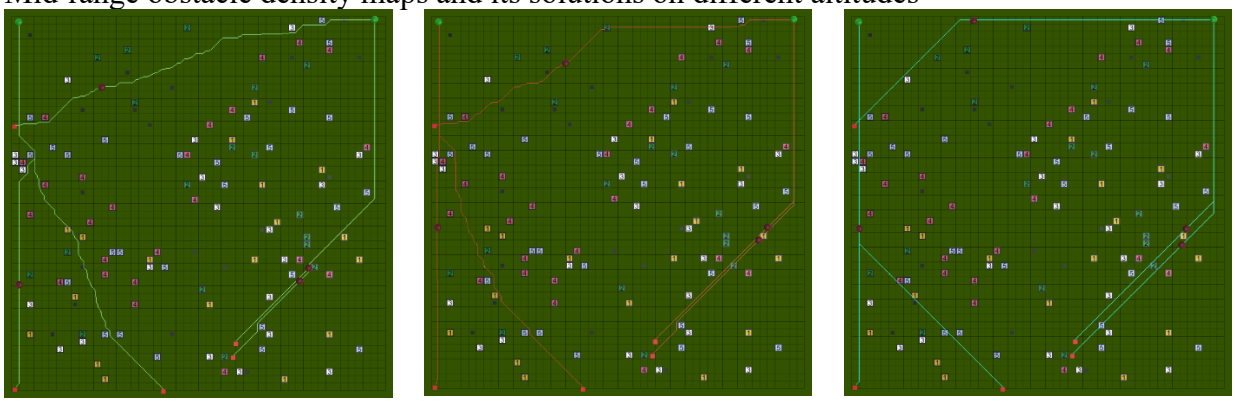
Appendix A

The Maps generated may contain low or mid or high density of obstacles. We can observe that the trajectories less affected whenever the density of obstacles goes down**.** On low obstacle density maps results are the same despite the altitudes. On high density trajectories results changed at every altitude, see maps and solutions on the three levels below:

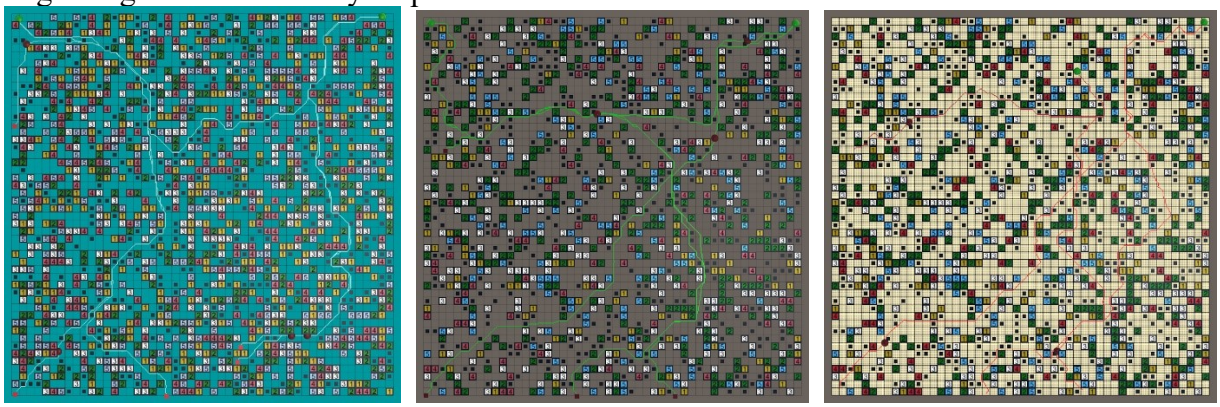
Low-range obstacle density maps and its solutions on different altitudes



Mid-range obstacle density maps and its solutions on different altitudes



High-range obstacle density maps and its solutions on different altitudes

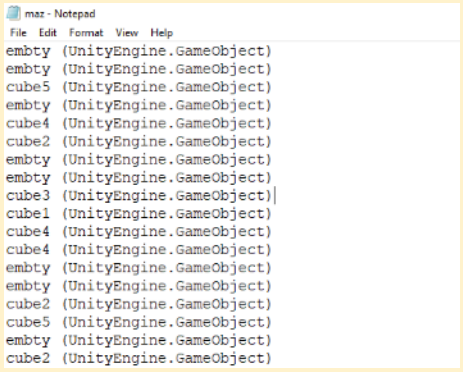


Appendix B

The Main Codes and Methods written in C# scripting language:

```
//// code 3.1: GetRandomCube + maz.txt file generated //////

private GameObject GetRandomCube()
{
   var change = (int)Random.Range(0, 100);
                                    if (change <= 15)
   {return cube0;}
                                    if (change <= 20)
   {return cube1;}
                                    if (change <= 30)
   {return cube2;}
                                    if (change <= 40)
   {return cube3;}
                                    if (change <= 45)
   {return cube4;}
                                    if (change <= 50)
   {return cube5;}
                                    return embty;
}
```




```
                                    //// code 3.2: Node Class //////
public class Node : IHeapItem<Node> {      public bool walkable; public Vector3 worldPosition; public int gridX;
public int gridY; public float gCost; public float hCost; public Node parent; int heapIndex;
            public Node(bool _walkable, Vector3 _worldPos, int _gridX, int _gridY) {
                        walkable = _walkable;
                        worldPosition = _worldPos;
                        gridX = _gridX;
                        gridY = _gridY;}
            public float fCost {
                        get {       return gCost + hCost;
                        //return gCost + 1.5f*hCost;
                        } }
            public int HeapIndex {
                        get {
                        return heapIndex;
                        }
                        set {
                                    heapIndex = value;
                        } }
                                                //// code 3.2: Node Class //////
public class Node : IHeapItem<Node> {      public bool walkable; public Vector3 worldPosition; public int gridX;
public int gridY; public float gCost; public float hCost; public Node parent; int heapIndex;
            public Node(bool _walkable, Vector3 _worldPos, int _gridX, int _gridY) {
                        walkable = _walkable;
                        worldPosition = _worldPos;
                        gridX = _gridX;
                        gridY = _gridY;}
            public float fCost {
                        get {       return gCost + hCost;
                        //return gCost + 1.5f*hCost;
                        } }
            public int HeapIndex {
                        get {
                        return heapIndex;
                        }
                        set {
                                    heapIndex = value;
                        } }
```

```
////  code 3.3: CreateGrid  method  //////
void CreateGrid() {
        float nodeRad = nodeRadius + 0.005f;
        grid = new Node[gridSizeX,gridSizeY];
Vector3 worldBottomLeft = transform.position - Vector3.right * gridWorldSize.x/2 - Vector3.forward * gridWorldSize.y/2;

        for (int x = 0; x < gridSizeX; x ++) {
                for (int y = 0; y < gridSizeY; y ++) {
Vector3 worldPoint = worldBottomLeft + Vector3.right * (x * nodeDiameter + nodeRadius) + Vector3.forward * (y *
nodeDiameter + nodeRadius);

bool walkable = !(Physics.CheckSphere(worldPoint,nodeRad,unwalkableMask));
                grid[x,y] = new Node(walkable,worldPoint, x,y);
                        }}}
```

```
////  code 3.4: the list GetNeighbours, part of Grid class //////

public List<Node> GetNeighbours(Node node) {
                List<Node> neighbours = new List<Node>();
                for (int x = -1; x <= 1; x++) {
                        for (int y = -1; y <= 1; y++) {
                                if (x == 0 && y == 0)
                                        continue;
                                int checkX = node.gridX + x;
                                int checkY = node.gridY + y;

                                if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY) {
                                        neighbours.Add(grid[checkX,checkY]);
                                }       }       }
                return neighbours;                                      }
```

```
//////  code 3.5: GetDistance method //////

int GetDistance(Node nodeA, Node nodeB) {
                int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);
                int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);

                if (dstX > dstY)
                        return 14*dstY + 10* (dstX-dstY);
                return 14*dstX + 10 * (dstY-dstX);
        }
```

////// code 3.6: SortDown & SortUp methods //////

```
void SortDown(T item) {
                while (true) {
                        int childIndexLeft = item.HeapIndex * 2 + 1;
                        int childIndexRight = item.HeapIndex * 2 + 2;
                        int swapIndex = 0;
                        if (childIndexLeft < currentItemCount) {
                                swapIndex = childIndexLeft;

                if (childIndexRight < currentItemCount) {
                        if (items[childIndexLeft].CompareTo(items[childIndexRight]) < 0) {
                                                swapIndex = childIndexRight;
                                        }}
                                if (item.CompareTo(items[swapIndex]) < 0) {
                                        Swap (item,items[swapIndex]);
                                }
                                else {  return;
                                }}
                        else {  return;
                        }       }       }
        void SortUp(T item) {
                int parentIndex = (item.HeapIndex-1)/2;
                while (true) {
                        T parentItem = items[parentIndex];
                        if (item.CompareTo(parentItem) > 0) {
                                Swap (item,parentItem);
                        }
                        else {  break;
                        }
                        parentIndex = (item.HeapIndex-1)/2;
                }       }
```

////// code 3.7: class of PathReqManager //////

```
public class PathReqManager : MonoBehaviour {
	Queue<PathReq> pathReqQueue = new Queue<PathReq>();
	PathReq currentPathReq;
	static PathReqManager instance;
	Pathfinding pathfinding;
	bool isProcessingPath;
	void Awake() {
		instance = this;
		pathfinding = GetComponent<Pathfinding>();
	}
	public static void ReqPath(Vector3 pathStart, Vector3 pathEnd, Action<Vector3[], bool> callback) {
		PathReq newReq = new PathReq (pathStart,pathEnd,callback);
		instance.pathReqQueue.Enqueue(newReq);
		instance.TryProcessNext();}
	void TryProcessNext() {
		if (!isProcessingPath && pathReqQueue.Count > 0) {
			currentPathReq = pathReqQueue.Dequeue();
			isProcessingPath = true;
			pathfinding.StartFindPath(currentPathReq.pathStart, currentPathReq.pathEnd);}}
	public void FinishedProcessingPath(Vector3[] path, bool success) {
		currentPathReq.callback(path,success);
		isProcessingPath = false;
		TryProcessNext();
	}
	struct PathReq {
		public Vector3 pathStart;
		public Vector3 pathEnd;
		public Action<Vector3[], bool> callback;

		public PathReq(Vector3 _start, Vector3 _end, Action<Vector3[], bool> _callback) {
			pathStart = _start;
			pathEnd = _end;
			callback = _callback;
		}		}
}
		TryProcessNext();
	}
	struct PathReq {
		public Vector3 pathStart;
		public Vector3 pathEnd;
		public Action<Vector3[], bool> callback;

		public PathReq(Vector3 _start, Vector3 _end, Action<Vector3[], bool> _callback) {
			pathStart = _start;
			pathEnd = _end;
			callback = _callback;
		}		}	}
Vector3 pathStart;
		public Vector3 pathEnd;
		public Action<Vector3[], bool> callback;

		public PathReq(Vector3 _start, Vector3 _end, Action<Vector3[], bool> _callback) {
			pathStart = _start;
			pathEnd = _end;
			callback = _callback;
		}}}
```

////// code 3.8: Unit class //////

```
public class Unit : MonoBehaviour {
        public Transform target;
        public GameObject Cube;
  public GameObject NodeUW;
  public GameObject NodePTH;
        float speed = 1.0f;
        Vector3[] path;
        int targetIndex;

void Update() {     if (Input.GetKeyDown(KeyCode.Keypad5)){
                     PathRequestManager.RequestPath(transform.position,target.position, OnPathFound);
                    }
public void OnPathFound(Vector3[] newPath, bool pathSuccessful) {
                    if (pathSuccessful) {

                                path = newPath;
                                StopCoroutine("FollowPath");
                                StartCoroutine("FollowPath");
                    }
        }
        IEnumerator FollowPath() {
                    yield return new WaitForSeconds(0.2f);
                    Vector3 currentWaypoint = path[0];

                    while (true) {
                                if (transform.position == currentWaypoint) {
                                            targetIndex ++;

                                            if (targetIndex >= path.Length) {
                                                        yield break;
                                            }

                                            currentWaypoint = path[targetIndex];
                                }

        transform.position = Vector3.MoveTowards(transform.position,currentWaypoint,speed * Time.deltaTime);
                                yield return null;

                    }
}
```

Appendix C

Part of this work was done at GET Lab Cognitive Systems Engineering at Paderborn University, throw the NRW DAAD scholarship Program. During my internship I completed three months at GET Lab, one of the most advanced research centers working on robotics, and one of the lab teams concentrate their researches work on developing rescue robots, and they awarded a 3rd place in the World Robo Cup competition this year. At Get-Lab, I learned more than I ever imagined, and it was so much fun and interesting to get involved in the practical. I spend the first month practicing on ROS (Robot Operating System) using two simulators, first simulator was SIMORE with R-VIZ for recognizing the results. I executed many experiments by programing motion for the Robot Get-Bot, to walk in a predefined path or to walk in an obstacle environment avoiding any collisions based on many attributes and depending on the observed data from odometry such as camera or laser sensor and by publishing the movement data to wheels. Other experiments were executed to control a quadrotor for both internal and external scene environment by using ROS with GAZIBO simulator and the control command inputs come throw gamepad joystick handle. Of course, algorithms and codes were written using C++ at this stage. Next and after reading many articles related to motion planning problem for aerial robots, I spend the next two months on developing a decentralized multi-agent motion planning method for quad-rotors moving among high-density 3D polygonal obstacles, but this time the 14 classes algorithm was written using C sharp scripting language under Unity engine environment as a simulator, in step by step and under the advice of my instructor prof. Dr. Eng. Barbel Mertsching, the director of Get Lab, the final work was recommended and found very valuable to this domain for further research purpose at the lab. (See recommendation letter on the next page).

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

UNIVERSITÄT PADERBORN | 33095 PADERBORN

FAKULTÄT FÜR ELEKTROTECHNIK, INFORMATIK UND MATHEMATIK

GET Lab

Prof. Dr.-Ing.
Bärbel Mertsching

Pohlweg 47 - 49
D-33098 Paderborn
Raum P 16.02
Fon +49. 52 51. 60-52 93
Fax +49. 52 51. 60-32 38
E-Mail mertsching@upb.de
Web getwww.upb.de

20. October 2016

To Whom It May Concern

Mr. Amjad Khalil (born May 09, 1978) worked in GET Lab – Cognitive Systems Engineering – at the Department of Electrical Engineering at the University of Paderborn, Germany, **from August 5, 2016 until October 26, 2016** as part of an internship program (DAAD).

In GET Lab, Mr. Khalil developed a decentralized multi-agent motion planning method for aerial robots moving among high-density 3D polygonal obstacles. His work is valuable for further research purposes in our lab. The algorithm was implemented under the Unity Engine environment using C# scripting.

Mr. Khalil is an excellent student, a quick learner, and possesses a natural ability for problem solving. He solved the given tasks with little supervision in a well-organized fashion. His contribution is valuable for ongoing research in this domain. Throughout his work he demonstrated perseverance and initiative. As a person, Mr. Khalil is well-organized and hard-working student with a pleasant personality.

If we can be of any further assistance or provide you with additional information, please do not hesitate to contact us.

Sincerely,

Prof. Dr.-Ing. Bärbel Mertsching
Director, GET Lab

Muhannad Mujahed, M. Eng.
Responsible Supervisor

Appendix D

Data to download or upon request by email:

1. A Folder contain all reference files in pdf format (References)
2. Data of random map generated (txt file).
3. Figure design file contain the design of all figures (.pptx file)
4. Model code and methods written in C# scripting language (pdf file)
5. Model Source project working under Unity simulator (version 2019 or higher).
6. Excel sheets contain Model Result Analysis and curve fitting (excel file).
7. Log file for data result and measurement of All experiments (txt file).
8. Amjad Khalil Thesis file (docx and pdf format)

This link is a Dropbox folder contain the described data files.

Biography

Amjad Khalil, born in Jerusalem-Palestine, holds a BA in Electronic Engineering and he complete his master's degree in computer science from Al-Quds University. Amjad is a junior research scientist, he also received a NRW scholarship as a DAAD grant for scientific research at Paderborn University - Germany at the end of 2016. He is interested in research fields such as the UAV path planning, image and video signals processing, multimedia, and media technologies. He is currently working as a lecturer at Palestine Technical University-Kadoori in media technology and multimedia specializations.

(Contact: amjad.pal@gmail.com)

Dr Abdullah Kamal from Jerusalem-Palestine is a research scientist, Dr Abdullah worked at Al-Quds university at computer science department. He taught the theoretical computer science courses, he studied mathematics on the topic of graph theory, and the topic of geometric graphs. He gets his PHD from the Hebrew University of Jerusalem, his PhD. thesis in Geometric graphs of Turán type. (Contact: abdullah.kamal@staff.alquds.edu)

References

[1] S. Garrido, "Exploration of 2D and 3D Environments using Voronoi Transform and Fast Marching Method," *Journal of Intelligent and Robotic Systems volume ,* vol. 55, p. 55–80, 2009.

[2] F. Liao, "3D Motion Planning for UAVs in GPS-Denied Unknown Forest Environment," *IEEE Intelligent Vehicles Symposium ,* vol. IV, pp. 84-91, 2016.

[3] T. Jiang, "Unmanned Aircraft System Traffic Management: Concept of Operation and system architecture," *International Journal of Transportation Science and Technology,* vol. 5, no. 3, pp. 123-135, 2016.

[4] N. Wen, "UAV Online Path Planning Algorithm in a Low Altitude Dangerous Environment," *IEEE/CAA JOURNAL OF AUTOMATICA SINICA,* pp. 173-184, 2015.

[5] M. N. a. S. Behnke, "Hierarchical Planning with 3D Local Multiresolution Obstacle Avoidance for Micro Aerial Vehicles," in *nternational Symposium on Robotics (ISR) and 8th German Conference on Robotics*, Munich, 2014.

[6] Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza. Introduction to autonomous mobile robots. MIT press, 2011.

[7] A. Das, "Mapping, Planning, and Sample Detection Strategies for Autonomous Exploration," *Journal of Field Robotics,* vol. 31, no. 1, p. 75–106, 2014.

[8] J. Cunha, "Using a Depth Camera for Indoor Robot Localization and Navigation," Aveiro, Portugal, 2011.

[9] R. Szeliski, Computer Vision Algorithms and Applications, London, New York: Springer, 2011.

[10] Wu, Paul Pao-Yen. Multi-objective mission flight planning in civil unmanned aerial systems. Diss. Queensland University of Technology, 2009.

[11] E. I. N. R. Siegwart, "Mobile Robot Localization", ch5. in *Autonomous Mobile Robots*, CMU, 2017, pp. 59-229.

[12] S. M. LaValle, PLANNING ALGORITHMS, Illinois: Cambridge University Press, 2006.

[13] Hähnel, Dirk, Dirk Schulz, and Wolfram Burgard. "Mobile robot mapping in populated environments." Advanced Robotics 17.7 (2003): 579-597.

[14] Atyabi, Adham, and Samia Nefti-Meziani. "Applications of computational intelligence to robotics and autonomous systems." HANDBOOK ON COMPUTATIONAL INTELLIGENCE: Volume 2: Evolutionary Computation, Hybrid Systems, and Applications. 2016. 821-863.

[15] M. ŠEDA, "Roadmap Methods vs. Cell Decomposition in Robot Motion Planning," in *International Conference on Signal Processing, Robotics and Automation*, Corfu Island, Greece, 2007.

[16] A. Gasparetto, "Path Planning and Trajectory Planning Algorithms: A General Overview," *Motion and operation planning of robotic system,* pp. 3-27, 2015.

[17] S. J. R. a. P. Norvig, Artificial Intelligence A Modern Approach, New Jersey: Pearson Education, Inc, 2010.

[18] Chen, Peiyi. 3D Motion Planning using Kinodynamically Feasible Motion Primitives in Unknown Environments. MS thesis. University of Waterloo, 2011.

[19] X. Z. Wang, "The Comparison of Three Algorithms in Shortest Path Issue," *First International Conference on Advanced Algorithms and Control Engineering,* vol. 1087, no. 2, pp. 1-7, 2018.

[20] D. Eynard, "Real time UAV Altitude Attitude and Motion Estimation from Hybrid Stereovision.," *Auton Robot ,* vol. 33, p. 157–172, 2012.

[21] Erra, Ugo, and Giuseppe Caggianese. "Real-Time Adaptive GPU Multiagent Path Planning." GPU Computing Gems Jade Edition. Morgan Kaufmann, 2012. 295-306.

[22] K. Nonami, Autonomous Flying Robots, Unmanned Aerial Vehicles and Micro Aerial Vehicles, Tokyo: Springer, 2010.

[23] S. Piskorski, AR.Drone Developer Guide, SDK 2.0, 2012.

[24] Prado, Igor Afonso Acampora, and Davi Antônio dos Santos. "A Model Predictive Guidance Strategy for a Multirotor Aerial Vehicle." Journal of Aerospace Technology and Management 9 (2017): 116-128.

[25] Zhang, Heng, "RGB-D SLAM combining visual odometry and extended information filter." Sensors 15.8 (2015): 18742-18766.

[26] Z. He, "Research and Application of Path-finding Algorithm Based on Unity 3D," *IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS),* vol. 16251845, no. 3, pp. 5090-0806, 2016.

[27] Zidane, Issa Mtanos, and Khalil Ibrahim. "Wavefront and a-star algorithms for mobile robot path planning." International Conference on Advanced Intelligent Systems and Informatics. Springer, Cham, 2017.

[28] L. R. Sahawneh, "3D Path Planning for Small UAS Operating in Low-Altitude Airspace," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, 2016.

[29] P. Chen, 3D Motion Planning using Kinodynamically Feasible Motion Primitives in Unknown Environments, Ontario, Canada: University of Waterloo Library, 2011.

[30] D. Ortiz-Arroyo, A Hybrid 3D Path Planning Method for UAVs, Mexico: Workshop on Research, Education and Development of Unmanned Aerial Systems, 2015

[31] DocFX, "Unity User Manual," Unity Technologies, 3 2021. [Online]. Available: https://docs.unity3d.com/Manual/UnityManual.html. [Accessed 24 6 2021].

[32] A. Price, "Experimental Scripting for Animation in Maya course," ohio-state University, December 2002. [Online]. Available: https://www.asc.ohiostate.edu/price.566/courses/694/random.htm. [Accessed 09 May 2020].

[33] T. H. C. e. al, "Introduction to Algorithms," in Chapter 6 Heapsort, Cambridge, Massachusetts Institute of Technology MIT, 2022, pp. 230-249.

[34] McMillan, Michael. "Data Structures and Algorithms Using C sharp," in Chapter 14 Cambridge University Press, 2007, pp. 249-260.

إيجاد أفضل ارتفاع للحركة لنظام من الطائرات بدون طيار تتحرك بين بيئات ثلاثية الأبعاد تحتوي على عوائق ثابتة ذات كثافة مرتفعة

إعداد: أمجد جعفر مصطفى خليل

إشراف: د. عبد الله كمال

ملخص:

لا تزال مشـــكلة تخطيط الحركة في الفضـــاء ثلاثي الأبعاد (3D space) تمثل تحديًا كبيرًا للباحثين وعلماء الذكاء الاصـطناعي. هناك زيادة مضـطردة في عدد المركبات الجوية التي تحلق على شكل نظام ضـمن بيئات منوعة، وفي حين أن الصـــناعة الواعدة مســتقبلا هي للطائرات بدون طيار أو محلقات رباعية المراوح (quadrotors)، وتطويرها للعمل ضـمن العديد من الخدمات والوظائف، هو ما يدفع الحاجة الملحة لإيجاد حلول جديدة تنظم حركة هذه الأنظمة وتتحكم فيها بشكل مستقل. هدفنا الرئيسـي في هذه الدراسـة هو تطوير جانب من الحركة خلال العديد من المسـاحات المحيطة بنا مثل المدن الكبرى.

في هذه الرســـالة البحثية، ندرس كيف تؤثر كثافة العوائق على مســـارات الروبوتات الهوائية، لتقديم طريقة لتخطيط مســـار الحركة في بيئة ثلاثية الأبعاد تحتوي على عوائق ذوات الشـــكل المضـــلع الثابتة والعالية الكثافة أو المزدحمة (HDFPOs) في بيئة الحركة. في البداية نقوم بإنشاء خرائط عشوائية مختلفة في بعدين (2D)، ثم نقوم بتحويلها إلى بيئـة ثلاثيـة الأبعـاد من خلال توزيع عوائق متعـددة الأضـــــلاع. تحتوي محـاكـاتنـا على خمســـــة روبوتـات جويـة (quadrotors) وأكثر من نقطة هدف للوصـول إليها، تمت المحاكاة باسـتخدام أداة محاكاة ثلاثية الأبعاد Unity. لقد قمنا بتحســـين بيئتنا ثلاثية الأبعاد في عدة خرائط تقاطع ثنائية الأبعاد، ثم طبقنا خوارزمية بحث (A*) A Star وطريقة البحث A Star المرجحة. يتم تطبيق عمليات البحث عن المسار المناسب للحركة باستخدام خرائطنا المعروفة ثنائية الأبعاد التي تم إنشـاؤها والتي تتقاطع مع العديد من الارتفاعات في البيئة ثلاثية الابعاد الثابتة والمحتوية عوائق متعددة الأضـلاع. هدفنا هو إنشـاء مسـارات خالية من الاصـطدام للنظام والروبوتات الطائرة في كل تجربة. من خلال تنفيذ هذا النهج لعدد كبير من الخرائط وعلى ارتفاعات مختلفة، من الواضـح أن هذه الطائرات تتقاطع مع البيئة ثلاثية الأبعاد التي تم إنشاؤها. بتكرار هذه التجربة عن طريق تغيير نسبة كثافة حجم العوائق متعددة الأضـلاع المضـافة إلى خرائطنا، نقوم بجمع كمية هائلة من البيانات. ثم نقوم بتحليل البيانات التي تم جمعها باســتخدام نهج ملائمة المنحنى (Curve fitting) للمقارنة والعثور على أفضـل ارتفاع يختصـر مسـافة الطيران. سـتسـاعد النتائج التي تم الحصـول عليها نظام المركبات الجوية ذاتية الحركة (UAVs) على الوصـول إلى أهدافها باتباع أقصـر مسـافة وضـمن أقصـر زمن. نفذت جميع التجارب على نسب مختلفة من كثافات العوائق متعددة الأضـلاع. نهدف إلى التنبؤ بأفضـل ارتفاع للطيران في حالة بيئة HDFPOs التي تحقق أقصـــر مســـار حركة ولإيجاد نقطة مرجعية لذلك مثل متوســـط ارتفاع جميع العوائق.