1-21-2021 11:30 AM

Developing a Resource and Energy Efficient Real-time Delivery Scheduling Framework for a Network of Autonomous Drones

Gopi Gugan, *The University of Western Ontario*

Supervisor: Haque, Anwar, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

The use of unmanned aerial vehicles (UAV) or drones appears to be a viable, low-cost solution to problems in many applications. However, the limited onboard computing resources and battery capacity make it challenging to deploy drones for long-distance missions.

Path planning capabilities are essential for autonomous control systems. An autonomous drone must be able to rapidly compute feasible, energy-efficient paths to avoid collisions. We first evaluate existing sampling-based algorithms' performance and present a hybrid sampling-based algorithm to generate a solution quicker, using less memory. We then introduce the notion of a layered graph, which accurately and efficiently models the search environment. Simulations show that when applying a modified A* algorithm on the layered graph, paths can be generated at least twice as fast, using significantly less memory than the sampling-based algorithm.

Finally, we propose a novel cell-based model that uses a network of drones to perform long-range tasks such as last-mile deliveries. Drone charging stations are strategically placed to ensure that drones can replenish their batteries. The genetic algorithm was implemented to solve the scheduling problem for multiple drones using this model. We show that this model can be used to deliver many packages within a short amount of time.

Keywords

UAV, Autonomous Drones, Path Planning Problem, Drone Delivery

Summary for Lay Audience

Unmanned aerial vehicles (UAV), or drones, have gained a lot of popularity over the last decade. The versatility of drones makes them an ideal solution for many applications. But the battery-powered drones often have limited flight time. Moreover, due to drones' weight restrictions, they do not have many computing resources available to them during flight. To deploy drones for commercial purposes, a robust path planner needs to be developed, and the flight range of drones needs to be extended for tasks such as drone deliveries.

In this thesis, we go over the path planning problem for autonomous drones. First, we investigate sampling-based algorithms, which compute a path based on randomly sampled points from the search space. Specifically, we look into Randomly Exploring Random Trees (RRT) and variants of this algorithm and evaluate its performance in environments with many obstacles. We then introduce a hybrid sampling-based algorithm to generate a solution quicker, using less memory. A layered graph is also proposed for path planning. With this approach, information about the obstacles in the search space can be efficiently represented as a graph. Using a graph traversal algorithm (A* search algorithm), we show that the algorithm can generate an energy-efficient path must faster than the sampling-based algorithm using fewer computing resources.

Finally, we propose a novel cell-based model that uses a network of drones to perform long-range tasks such as last-mile deliveries. Drone charging stations are strategically placed to ensure that drones can replenish their batteries. To evaluate the impact of the cell-based model, we assessed the time it would take for a set of drones to deliver packages using this model. The genetic algorithm, which is commonly used for optimization problems, was implemented to optimize a schedule for drone deliveries. We show that this model can be used to deliver many packages within a short amount of time.

Co-Authorship Statement

Chapter 4 of this thesis forms a technical paper submitted for publication, co-authored by Nichole Chow and Dr. Anwar Haque. Nichole Chow formulated the original algorithm. I extended the algorithm and prepared the manuscript with supervisory advisory.

Chapter 6 in this thesis forms an original technical paper submitted for publication, co-authored by Fu Chi Chen, Dr. Anwar Haque and Dr. Roberto Solis-Oba. Under the supervision of Dr. Haque, I collaborated with Dr. Solis-Oba and Fu Chi Chen and participated in developing the algorithm. I was also responsible for the implementation of the sampling-based algorithm, which was used for comparative purposes.

Acknowledgments

First, I would like to express my deepest gratitude to my supervisor and mentor, Dr. Anwar Haque. He has supported and guided me through this endeavour. As an advisor, he has provided feedback, which has been invaluable for this thesis. I would also like to thank Dr. Haque for being involved and providing a lab to get hands-on experience building drones and to work towards developing autonomous drones. It was a pleasure to work with him and to learn from him.

I would also like to thank Dr. Roberto Solis-Oba for his contribution to the path planning algorithm proposed in Chapter 6 and for his support throughout the process of completing this thesis. He has continuously provided feedback to improve the algorithm and generate meaningful results. I learned a lot from him and enjoyed collaborating with him and his graduate student Fu Chi Chen to develop the path planning algorithm.

Table of Contents

List of Tables

List of Figures

Chapter 1

1 Introduction and Motivation

Unmanned aerial vehicles (UAV), commonly referred to as drones, are defined as aerial vehicles that operate without the need for an onboard human pilot. According to the Federal Aviation Administration, there will be 828,000 commercial drones registered in the United States of America by 2024 – more than double the number of commercial drones registered in 2019 [1]. Drones are particularly beneficial since they can perform high-risk missions without endangering human lives. Since drones are a safer alternative to manned military aircraft, they have been widely used for defence. However, due to advancements in technology and decreases in drones' production costs, there has been a lot of interest in developing drones for commercial applications. The versatility of drones makes them ideal for applications such as surveillance, precision agriculture, search and rescue missions, inspections and the delivery of goods.

Although drones can be controlled by an operator at the ground control station, developing drones with high autonomy levels makes them very useful for completing tasks that must be performed beyond the visual line of sight. There are many challenges involved in the development of a control system for autonomous drones. Several factors, including decision-making capabilities, path planning, trajectory generation and fault-tolerant redundant management (in the instance of disturbances and failures), are required for drones to navigate autonomously [46]. To achieve this, drones must have the ability to sense and perceive the environment and compute a path using on-board computers and sensors. Additionally, flight navigation controlled by an on-board computer reduces the frequency with which an operator needs to interact with a drone; thus, limiting the opportunity for human error. Although much progress has been made in the field of path planning for drones, there are no current solutions that propose a robust path planner that addresses all the limitations mentioned in the following section. Thus, our focus has been on investigating path planning algorithms that are efficient and computationally inexpensive.

Over the last decade, large companies such as Amazon, UPS and DHL have been looking for alternative approaches to reduce the cost of last-mile deliveries. Last-mile deliveries cost

companies the most since packages must be delivered individually. The use of autonomous vehicles such as drones could significantly reduce the cost of deliveries and help deliver packages faster. Amazon [47] recently announced that they were developing a hybrid drone with the ability to travel 15 miles to deliver packages under 5 pounds. Even though drones have limited flight time, for drones to be used for delivery applications, they must be able to perform long-distance missions. Thus, in this thesis, we were interested in developing a comprehensive drone-only delivery solution.

1.1 Limitations of Drones

Apart from current regulations that prohibit drones' operation beyond the visual line of sight and safety concerns in instances of mechanical failure or poor weather conditions such as strong winds, other limitations need to be addressed – notably, the limited battery performance and on-board computational capabilities of drones.

1.1.1 Energy Constraints

Although rotor-wing drones are highly maneuverable and versatile, they are still limited in terms of endurance. Thus, a single rotor-wing drone cannot be used for missions requiring long flights. Many factors can affect the battery performance of drones. Maneuvers such as hovering, horizontal and vertical movements, payload, speed, and flying with respect to the direction of the wind can all impact drones' battery performance at varying levels [51].

Improvements have been made to reduce the energy consumption of drones. The overall weight of rotor-wing drones has been reduced by using carbon-fibre airframes. There have also been improvements in brushless DC motors' power-to-weight ratios, which contributes significantly to energy consumption [32]. Since there are weight constraints to drones, additional batteries or batteries with larger capacities cannot be placed on the drone to improve its endurance. In [52], it has been shown that there is an optimal value of mass on a rotorcraft. Increasing the drone's weight after such a point begins to reduce the endurance of the drone and causes the drone to become unstable.

1.1.2 On-board Computational Capabilities

Due to weight restrictions, on-board computers placed on drones usually have limited memory and on-board computational power. Due to extensive requirements of memory and computational power, certain path planning algorithms and image processing algorithms cannot be executed directly by the on-board hardware. Moreover, since the environment is continually changing, drones must be able to rapidly compute a path for the drone to avoid a collision. Although an optimal path in terms of path length is desirable to improve battery performance, such paths often require heavy computations and a lot of time, which is not feasible for drones.

1.2 Thesis Contribution

Many path planning algorithms have been suggested for drones. The contributions of this research aim to address the energy limitations that were mentioned above. The contributions of this thesis are given below:

- Since sampling-based algorithms are by far one of the most utilized algorithms for 3D planning, we investigate how various factors such as nearest neighbour search radius and number of iterations impact the length of the generated paths.

- We introduce a hybrid sampling-based path planning algorithm for path planning in cluttered environments. Experimentation was performed to evaluate the algorithms effect on the path length, running time and memory consumption

- We introduce the idea of a layered graph to reduce the overall computational complexity of path planning algorithms. By applying the A* algorithm, a path can be rapidly generated using limited computational resources. This approach's effectiveness is demonstrated by comparing the performance of the algorithm to the hybrid sampling-based path planning algorithm.

- We propose a comprehensive solution for drone-only deliveries. A cell-based design is introduced. This model allows for multiple drones to work together to complete missions cooperatively. We then apply evolutionary algorithms to assign jobs to drones in this cell-based design.

1.3 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 provides background information related to drones, the path planning problem, and the algorithms used in this thesis. Chapter 3 provides a literature review of path planning methods and drone delivery solutions. Chapter 4 outlines a preliminary study done to bound search spaces with the hope of reducing the overall memory usage of the algorithm. Chapter 5 provides an analysis of sampling-based algorithms and introduces a hybrid algorithm. Chapter 6 presents a study that introduces the notion of a layered graph to compute a path efficiently. In chapter 7, we propose a cell-based design for delivery applications to address the limited battery capacity of drones. A complete solution with a scheduling algorithm is presented to demonstrate the effectiveness of this approach. Finally, in chapter 8, we conclude the thesis by summarizing our findings and provide an insight into potential future studies that can be performed.

Chapter 2

2 Background

This chapter provides background information about drones, drone path planning problems and a brief overview of the algorithms used in this thesis.

2.1 Drone Classification

Generally, drones can be categorized as either rotary-wing, fixed-wing or flapping-wing drones. Rotary-wing drones are often preferred due to their VTOL (Vertical Take-off and Landing) capabilities. Many studies model multi-rotor drones such as quadcopters for path planning applications due to their high maneuverability and hovering capabilities. The maneuverability makes multi-rotor drones suitable for tasks such as surveillance and search and rescue operations. However, the main issue with rotary-wing drones is their limited flight time. Figure 1 depicts the maneuverability and endurance of the different types of drones.

Even though fixed-wing drones have better battery performance than multi-rotor drones, they have many kinematic and dynamic constraints that need to be considered for practical applications. Moreover, it has been shown that rotary-wing drones are less susceptible to air turbulence than fixed-wing drones of similar dimensions [49]. However, in rural areas, fixed-wing drones have proven to be reliable for delivery applications. The drone delivery company Zipline [48] has been using fixed-wing drones to deliver blood in Rwanda and Ghana. Take-off and landing are a challenge with fixed-wing drones, but they are valuable for rapid delivery at speeds up to 100 km/h.

Flapping-wing systems, which are inspired by birds, have potential benefits over rotary-wing and fixed-wing drones. However, there are many design challenges that affect the stability of the drone [50]. To benefit from the endurance of fixed-wing drones and the maneuverability of rotary-wing drones, more recently, hybrid models such as tilt-rotor drones have been proposed for drone delivery applications [47]. Tilt-rotors have vertical takeoff and landing capabilities and can cruise at high speeds. However, there are design challenges related to transitioning between vertical and horizontal configurations [50].

A detailed review of the different types of drones and design challenges can be found in [50]. For this thesis, we are particularly interested in quadcopter drones. Quadcopters can rapidly change their flight direction and make sharp turns when necessary, unlike fixed-wing drones. Therefore, quadcopter drones have the potential to navigate in cluttered environments easily.

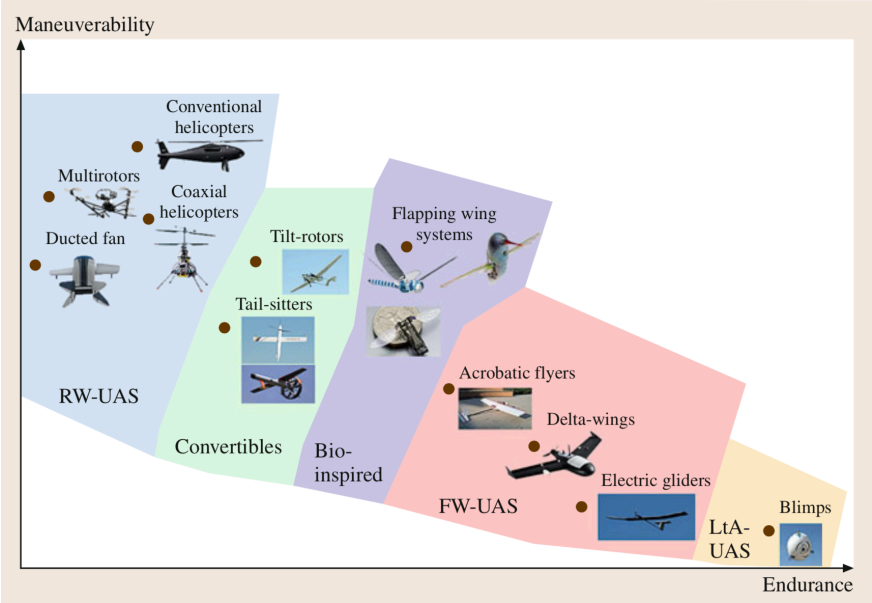


Figure 1. Classification of aerial robotics based on their endurance and maneuverability properties. Lighter-than-air unmanned aerial system (LtA-UAS), fixed-wing unmanned aerial system (FW-UAS) and rotary-wing unmanned aerial system (RW-UAS) [31]

2.2 Drone Path Planning Problem

In this thesis, the *drone path planning problem* refers to generating a minimum cost, collision-free path between the start and destination point. In many studies, the objective is to minimize total path length. However, some studies also minimize flight time [32], flight altitude [15] and drone speed [3].

Path planning algorithms can be categorized as either offline path planners or online (real-time) path planners. Offline path planners compute a path for the drone before takeoff. Thus, these path planners require information about the environment in advance. Information regarding obstacles and no-fly zones are made available to the path planner to generate an accurate path. An advantage of offline path planners is that they can employ models of the drone dynamics to

ensure that the path is feasible. However, these types of algorithms are not able to respond to changes in the environment dynamically. The environment is continually changing. Moving obstacles such as birds, wind and other aircraft may hinder the original path, and a robust path planner must be able to re-compute a new path to avoid such obstacles.

Online path planners utilize sensor data to detect obstacles and, therefore, react to changes in the environment. However, online path planners are often not able to ensure near-optimum paths are generated. Some online path planners function by generating an initial feasible path and modify their path as dynamic obstacles are detected [4]. Other online path planners utilize probability functions to create a path as new information regarding the environment becomes available [5]. Depending on the specific use case, one of the approaches may be beneficial over the other. For example, in drone delivery applications, the first approach would allow the drone to travel a relatively shorter overall distance than the second approach. However, in target tracking applications, pre-defined paths cannot be utilized. Therefore, the path planner must incrementally generate a path.

2.3 Shortest Path Algorithms

In this section, two shortest path algorithms, Dijkstra's and the A* algorithm, are explained in detail. Both algorithms have been applied in various sections of this thesis.

2.3.1 Dijkstra's Algorithm

Dijkstra's algorithm, introduced by Edsger W. Dijkstra in 1959 [72], is a well-known algorithm used to compute the shortest path between two points in a weighted, directed graph. Given a graph $G = (V, E)$, from the source s to destination t, where $s, t \in V$, the Dijkstra's algorithm computes the sequence of edges with the smallest weight to reach t from s. As shown in Algorithm 1, Dijkstra's algorithm determines the shortest path from s to any vertex in the graph by iteratively traversing through all vertices and updating the distances of its adjacent nodes.

Algorithm 1: Dijkstra(G, w, s)

Input: Graph $G = (V, E)$, function w indicating the weights of the edge between two vertices and the source vertex s

```
for each vertex v ∈ V do
    v.dist ← ∞
    v.prev ← ∅
s.dist ← 0
Q ← Set of all vertices in G
while Q is not empty do
    u ← vertex in Q with the minimum distance
    Remove u from Q
    for all edges (u, v) do
        if v.dist > u.dist + w(u, v) then
            v.dist ← u.dist + w(u, v)
            v.prev ← u
```

2.3.2 A* Algorithm

The A* algorithm [73], introduced in 1968 by Hart et al., prioritizes the expansion of vertices with lower costs. Although the implementation of Dijkstra's algorithm can be optimized to perform quickly, in large environments where there are many vertices in the graph, the A* algorithm is often desired since the A* algorithm focuses on searching for the shortest path towards the destination point. Dijkstra's algorithm computes the shortest path from the initial point to every vertex in the graph. The vertex is evaluated using the following equation:

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

where $g(v)$ is the cost to reach the vertex v, and $h(v)$ is a heuristic used to estimate the cost of reaching the destination. As long as the value of $h(v)$ is lower than or equal to the actual distance of v to the destination point, the heuristic is considered permissible and can compute an optimal path. However, if the distance from v to the destination point is overestimated, then the A* algorithm cannot guarantee that an optimal solution will be computed. The value of $h(v)$ can have an impact on the time complexity of the algorithm.

2.4 Evolutionary Algorithms

Evolutionary algorithms are population-based metaheuristics commonly used in combinatorial optimization problems. The general idea of these evolutionary approaches is to start with an initial set of solutions, called a population, and have multiple generations where the set of

solutions evolve. Offspring solutions (new populations) are created from the parents. Figure 2 depicts the general structure of an evolutionary algorithm. Generally, the initial population can be generated randomly. Variation operators such as mutations (modifying a single parent structure to create the child) and hereditary (modifications as a result of combining two parent structures) are required to generate the new population [58].

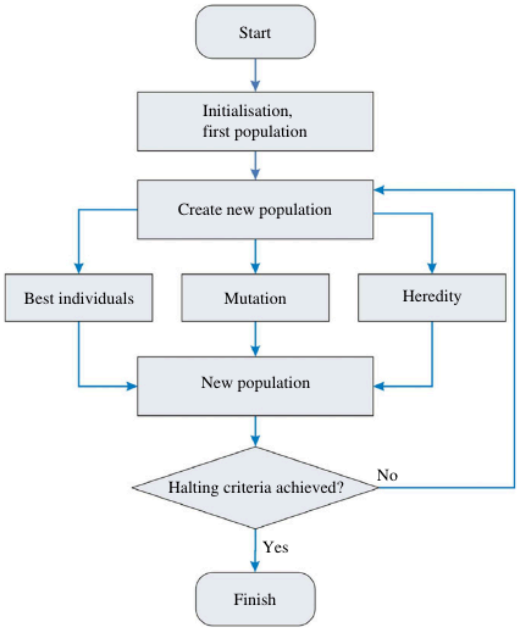


Figure 2. General Structure of Evolutionary Algorithms [58]

2.4.1 Genetic Algorithm

The genetic algorithm (GA), based on natural evolution, was initially developed by Holland [69]. Over the last two decades, genetic algorithms have been applied to multiple fields, especially in path planning and scheduling problems. GA uses the idea of the survival of the fittest among solutions and information exchange to improve solutions generation after generation until a near-optimal solution has been computed.

2.4.2 Particle Swarm Optimization

Particle swarm optimization (PSO), proposed by Kennedy and Eberhart [75], is an optimization technique inspired by bird flocking and fish schooling's social behaviour. The PSO algorithm functions by having a population (swarm) converge towards the optimal solution by moving

around the search space and exchanging information with its neighbours. The position and velocity of each solution (particle) are adjusted dynamically based on its own best-known position and the best-known position of its neighbouring particles. The updated velocity (v) and position (x) of each particle can be obtained from 2.1 and 2.2.

$$v_i^{k+1} = wv_i^k + c_1 r_1 \left(p_{best}{}_i^k - x_i^k\right) + c_2 r_2 \left(g_{best}{}^k - x_i^k\right) \tag{2.1}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{2.2}$$

where i is the particle number and k is the iteration number. p_{best} is the local best solution generated by a particle, and g_{best} is the global best solution among all particles in the swarm. c_1 and c_2 are positive constants (learning factors) that move the particle towards the p_{best} and g_{best} positions. r_1 and r_2 is a random number between 0 and 1. w is the inertia weight, which affects a particle's global and local searching ability. An adaptive inertia weight was used in this thesis where the value of w changes as the number of iterations increase (equation 2.3).

$$w = w_{max} - \left[\frac{(w_{max} - w_{min})\, k}{K}\right] \tag{2.3}$$

where K is the maximum number of iterations the algorithm performs.

Chapter 3

3 Literature Review

This chapter provides a review of path planning algorithms and drone delivery solutions. A thorough literature review of recent 2D and 3D path planning algorithms is performed, and future research directions are suggested. Finally, current studies investigating drone delivery solutions are discussed.

3.1 Drone Path Planning Problem

Research into the path planning problem for drones has been growing since 2000. The number of published proceeding papers and articles related to path planning in the Web of Science database has been growing exponentially since 2000 (Figure 3). Roughly 10% of all published drone studies between 2000-2018 were related to the path planning problem. Computing the shortest path between two points in a 3D environment with polyhedral objects is NP-hard [2]. Thus, most drone path planning algorithms use heuristic and metaheuristics to generate a near-optimal path.

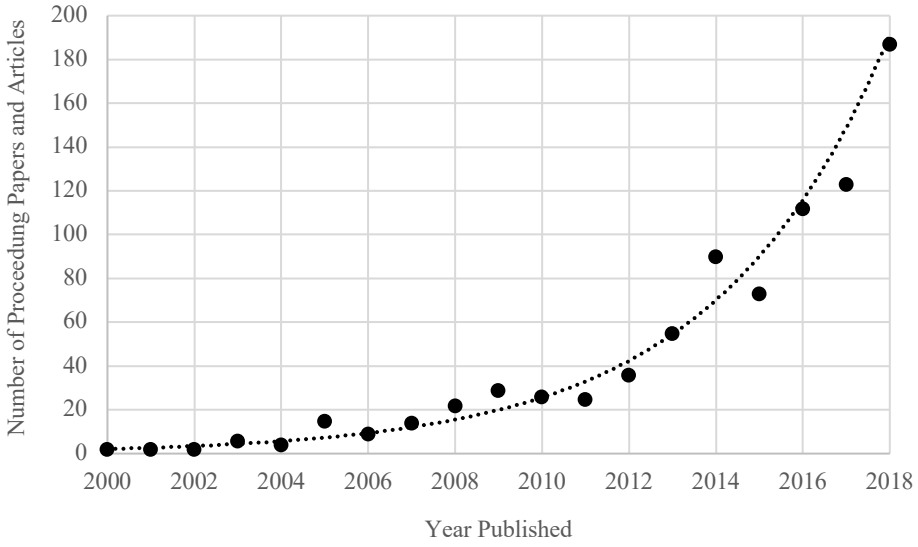


Figure 3. Drone path planning papers published between 2000-2018

3.1.1 Environmental Representation

Before searching for a path, the search space must be mapped for the algorithm to understand the location of obstacles. Figure 4 summarizes some of the most commonly used techniques to represent the environment. Cell decomposition techniques have been extensively used in many studies [24, 33, 35]. The environment can be divided into a uniform grid (Figure 4a) with fixed-size cells or voxels to simplify the task of computing paths. Any cell containing a portion of an obstacle is marked as occluded, and any cell without an obstacle is marked as free. However, even if a small portion of a particular cell contains an obstacle, the entire cell is marked as being occluded with this approach. Adaptive cell decomposition techniques such as quadtrees (Figure 4b) have been proposed to represent obstacle information effectively. There are more voxels (smaller in size) when close to an obstacle and fewer (larger) voxels when away from an obstacle. Although such approaches can determine whether there is a solution for any given input, in a finite amount of time, these approaches require a large amount of memory and cannot function in real-time to solve problems.

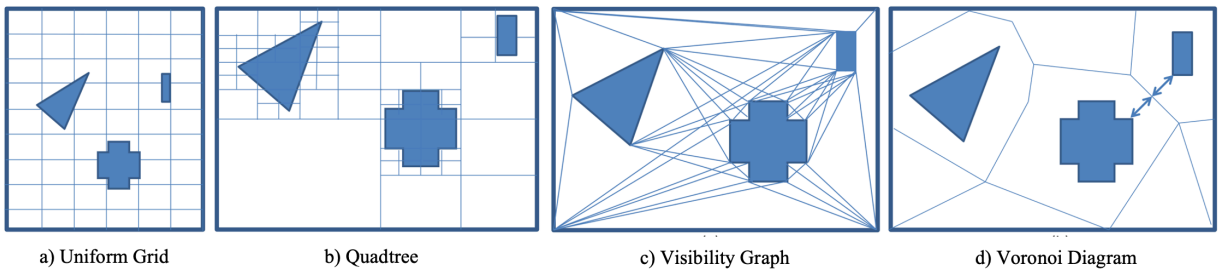


Figure 4. Representation of an environment. Adapted from [71].

To mitigate the computationally intensive process of the cell-decomposition technique, some studies have designed solutions that limit the number of voxels used to generate a path. In [36], a wavelet transform was used to get a multi-resolution cell decomposition of a 2D environment, where a high-resolution map was created in the drone's vicinity, and a lower resolution one was produced for places far away from the drone. An adaptive cell decomposition method for 3D environments was suggested in [37] that reduces the amount of memory required to compute a path by continuously performing a cell decomposition of the environment; however, this approach may require a large amount of time.

Voronoi diagrams (Figure 4d) have also been used to model the environment. This method involves partitioning a plane with a set of objects into convex polygons, where each polygon contains a single object. Any point in each polygon is closer to the object in its polygon than any other object. Moreover, each point on the edge of the Voronoi diagram is equidistant from its two nearest neighbours. However, using the edges of the Voronoi diagram to generate a path may lead to sub-optimal solutions. Voronoi graphs are often used to compute a path away from a threat or radar [74]. In recent studies, Voronoi diagrams were constructed to improve the quality of paths generated by other algorithms. For example, in [10], Voronoi diagrams were created to increase the genetic algorithm's convergence rate by selecting members on Voronoi vertices to be part of the initial population.

Visibility graphs can also be used to represent the search space. Visibility graphs function by adding nodes corresponding to the vertices of obstacles into the visibility graph. An edge connects nodes in the graph if the edge connecting the two nodes lie in an obstacle-free region (Figure 4c). Visibility graphs have been used extensively in 2D path planning because of their simplicity and since they can be used to compute global shortest paths for a point [38]. However, a large amount of time and memory is needed to generate visibility graphs in 3D, and only near-optimum solutions can be generated when only considering obstacle vertices. To address this issue, some studies have constructed reduced visibility graphs using only a subset of obstacles in the environment [39-41, 44]. The bounded energy space [40] and approximation with visibility line [41] algorithms reduce the number of obstacles by only considering obstacles in the straight path from the start to the destination point. However, such approaches can generate poor solutions (roughly 8% larger than the optimal path) and cannot guarantee that a solution will be found in complex environments with many obstacles [41]. Moreover, these algorithms have a high time complexity [44]. Other studies have transformed the 3D space into 2D space to reduce the problem's complexity [42, 43]. Although there are limitations in using this approach for cluttered environments, in use cases where there are a few obstacles, visibility graphs have proven beneficial [45].

3.1.2 Path Generation

Many studies attempt to reduce the three-dimensional path planning problem to two dimensions [33-34]. By limiting maneuvers of the drone to a horizontal plane (at a fixed altitude), the

problem's complexity can be significantly reduced, and algorithms proposed for other ground vehicles can be directly applied to drones. However, there are many benefits to utilizing the vertical translational movement of drones. One particular advantage is in obstacle avoidance and navigation.

Many shortest path algorithms have been suggested for drone path planning. After modelling the environment, Dijkstra's algorithm [34, 43] has been proposed to generate the shortest path for drones. Although the implementation of Dijkstra's algorithm can be optimized to perform quickly, in large environments where there are a lot of vertices in the graph, the A* algorithm is often desired. The A* algorithm focuses on searching for the shortest path towards the destination point instead of the shortest path to every vertex in the graph. Thus, it has often been used for drone path planning [14, 29]. Variation of the A* algorithm, such as the D* [24] algorithm and Theta* [14], have been proposed for online path planning. However, such approaches may be slower than the A* algorithm due to the increase in the number of calculations.

Sampling-based path planning algorithms have been extensively studied due to their low computational complexity and the ability to solve complex, high-dimensional problems. Probabilistic Road Maps (PRM) and Randomly Exploring Random Trees (RRT) [53] are common sampling-based path planning algorithms that compute a path using a set of randomly chosen points from the configuration space. PRMs can be inefficient when obstacle geometry is not known beforehand [55]. The original RRT algorithm [53] generated solutions very rapidly in dynamic environments; however, the solution was not optimal. The RRT* algorithm [54] was later proposed to improve the RRT algorithm by using the nearest neighbours to refine the overall path. The RRT* was proved to be asymptotically optimal. Thus, the cost of the path will almost surely converge to the optimum. However, the convergence rate at which the algorithm reaches the optimal solution is very slow. Although RRT* can compute optimal paths, the slow convergence rate makes them unsuitable for real-time applications such as drone path planning. Many variants of the RRT and RRT* have been proposed to improve the rate at which the algorithm converges to an optimal solution [54, 55]. RRT and RRT* based solutions have been commonly proposed for drone path planning [4, 5, 8, 17, 21, 26]. Many studies use RRT for online path planning where sensors are used to detect either stationary or moving obstacles, and

RRT is used since they are effective at re-planning paths [4, 5, 8]. Other studies have attempted to integrate RRTs with other path planning techniques, such as potential fields [26] or particle swarm optimization [21]. Some studies target fixed-wing drones and only sample points which satisfy a minimum turning radius [17]. A limitation with RRT based approaches is that if a solution does not exist, the algorithm may run infinitely.

Artificial potential fields have also been proposed for path planning in 3D environments [3, 26, 28]. A gravitational field of attraction is placed around the destination, and repulsive forces are placed around the obstacles. The combination of potentials creates a force field, which is then used to guide the drone towards the destination while avoiding obstacles. A major issue with potential fields is that they can quickly converge to a local minimum due to the location and obstacles' geometry. Once the drone is trapped at a local minimum, it can no longer reach its destination point. Moreover, artificial potential fields cannot guarantee that the resulting path is optimal [3]. Several approaches have been proposed to solve these issues. In [3], the problem is converted into an optimization problem and solved using the optimal control method to move a vehicle out of traps successfully. In [57], a pre-determined or randomly selected guided point was introduced to allow the UAV to continue searching for the destination when it reaches a local minimum. Artificial potential fields have been demonstrated on a network of micro-drones to successfully avoid obstacles while maintaining formation [26].

Many metaheuristic algorithms based on biological models have also been proposed as solutions to the path planning problem. Genetic algorithm [6, 11, 19], ant colony optimization (ACO) [7, 9, 12], particle swarm optimization (PSO) [13, 21], fruit fly optimization [20, 30], wolf pack search [25] and differential evolution [15] are all examples of such evolutionary algorithms which use bio-inspired properties for optimization.

A limitation of genetic algorithms is that they can be computationally expensive. However, implementing the genetic algorithm on field-programmable gate arrays (FPGA) and GPUs has been shown to quickly generate feasible solutions in small environments [6, 11]. Another major limitation with evolutionary algorithms, in general, is premature convergence. To overcome this issue, increasing the diversity of the population through periodic mutation applications has resulted in fast convergence for the genetic algorithm [19]. Moreover, combining PSO with an

adaptive decision operator has helped overcome premature convergence [13]. This improved PSO was able to generate solutions that were superior to the original genetic algorithm, PSO and firefly algorithms. To resolve the local minimum and premature convergence in the ACO algorithm, adding a disturbance by applying a particular chaos factor has been shown to work [12]. The use of multiple colonies that communicate with each other [10] has also been shown to improve the quality of ACO's solutions.

More recently, learning-based methods have been suggested for drone path planning [23, 27]. Reinforcement learning can be used to make sequential decisions. Thus, information about the environment does not necessarily need to be made available [27]. An adaptive and random exploration approach based on Q-learning has been proposed for drones to navigate and avoid obstacles in real-time [23]. During path planning, there is a learning module that derives a strategy for action by looking at historical data of the drone's action. In [27], the classical Q-learning algorithm is improved by combining action selection strategies to avoid local minima and improve the convergence rate. However, these approaches may be computationally expensive in large environments since the environments are represented as grids.

3.1.3 Analysis of Recent Studies

For this review, path planning papers were retrieved from the Web of Science database. Highly cited studies between 2010-2019 were investigated. Preference was given to more recent studies and studies that developed path planning algorithms for cluttered environments. Since we were mainly interested in path planning algorithms for commercial applications, we did not investigate coverage path planning algorithms or target tracking path planning algorithms. Coverage path planning algorithms are necessary for applications such as search and rescue and surveillance, which require a drone to compute the best path to traverse the entire environment.

The critical review of drone path planning papers were based on the following criteria:

- *Dimension*: Spatial dimension for which the path planning algorithm was proposed. Either the algorithm was suggested for 2D or 3D environments.

- *Energy Constraints*: Studies that analyze factors other than path length in their cost function. Factors such as payload, weight, wind, motor thrust, drone maneuvers can affect the battery performance of drones [51]

- *Path Optimization*: Flight trajectory is essential – particularly for fixed-wing drones. Since fixed-wing drones have limited turning angles, often Dubins curves or Bezier curves are used to ensure the path is feasible. Path smoothening techniques have also been applied to rotor-wing drones to allow for continuous motion and prevent the drone from having to come to a complete stop

- *Drone Type*: Fixed-wing drones (F), Rotary-wing drones (R) or the paper did not specify which type of drone the algorithm was intended for (N)

- *Path Planner*: Algorithm was developed as an online path planner to compute paths in real-time (ON) or as an offline path planner (OFF)

- *Obstacle Type*: Algorithm can generate a path in an environment with stationary obstacles (ST) or moving obstacles (MV)

- *Environment Type*: Small-sized environment with 0-50 obstacles (S), Medium-sized environment with 51-500 obstacles (M) or a large-sized environment with more than 500 obstacles (L)

- *Computer Simulations*: Indicates whether computer simulations were used to validate the algorithm

- *Drone Implementation*: Indicates whether the algorithm was tested on physical drones

Table 1. Summary of 28 Published Drone Path Planning Papers between 2010-2019

Ref	3D	Energy Constraints	Path Optimization	Drone Type (F/R/N)	Path Planner (ON/OFF)	Obstacle Type (MV/ST)	Environment Type (S/M/L)	Computer Simulations	Practical Implementation (I/O)
[3]	✓	✓	×	R	OFF	ST	S	✓	×
[4]	✓	×	✓	R	ON	ST/MV	S	×	I
[5]	✓	×	✓	R	ON	ST	M	✓	×
[6]	✓	×	×	R	OFF	ST	S	✓	×
[7]	×	×	×	N	OFF	ST	S	✓	×
[8]	✓	×	✓	R	ON	MV	S	✓	I
[9]	×	✓	✓	F	ON	ST	S	✓	×
[10]	×	×	×	R	OFF	ST	S	✓	×
[11]	✓	×	✓	F	OFF	ST	S	✓	×
[12]	×	×	✓	N	OFF	ST	S	✓	×
[13]	✓	×	×	N	OFF	ST	S	✓	×
[14]	✓	×	×	N	OFF	ST	S	✓	×
[15]	✓	×	✓	F	OFF	ST	S	✓	×
[16]	✓	×	✓	R	OFF	ST	S	✓	×
[17]	✓	×	✓	F	OFF	ST	S	✓	×
[18]	×	×	×	F	OFF	ST	S	✓	×
[19]	✓	×	✓	N	OFF	ST	M	✓	×
[20]	✓	×	✓	N	OFF	ST	S	✓	×
[21]	✓	×	✓	N	ON	ST	M	✓	×
[22]	✓	×	✓	N	OFF	ST	S	✓	×
[23]	×	×	×	F	ON	ST	S	✓	×
[24]	✓	×	×	R	ON	ST/MV	M	✓	×
[25]	✓	×	✓	R + F	OFF	ST	S	✓	×
[26]	×	×	×	N	OFF	ST	M	✓	×
[27]	×	×	×	N	ON	ST	S	✓	×
[28]	✓	×	×	R	OFF	ST	S	✓	O
[29]	×	×	✓	R	ON	ST/MV	S	✓	×
[30]	✓	×	✓	N	OFF	ST	S	✓	×

Table 1 summarizes the results from reviewing 28 published papers related to the drone path planning problem between 2010 and 2019. Although a few 2D solutions were proposed, 68% of the reviewed papers focused on 3D drone path planning. Moreover, most of the reviewed studies incorporated a smoothing technique to ensure the path was feasible; however, only [3] considers motor thrust and [9] considers wind energy to improve drones' energy consumption.

Since path planning algorithms were being developed for combat operations, many studies previously focused on developing solutions for fixed-wing drones. However, since drones are being applied for civilian applications in recent years, many studies have been developing solutions for rotary-wing drones.

Furthermore, when reviewing the types of algorithms commonly studied for the drone path planning problem, bio-inspired algorithms and RRT based approaches were used for 61% of the reviewed studies (Figure 5). However, in recent years, reinforcement learning approaches are being investigated for real-time path planning.

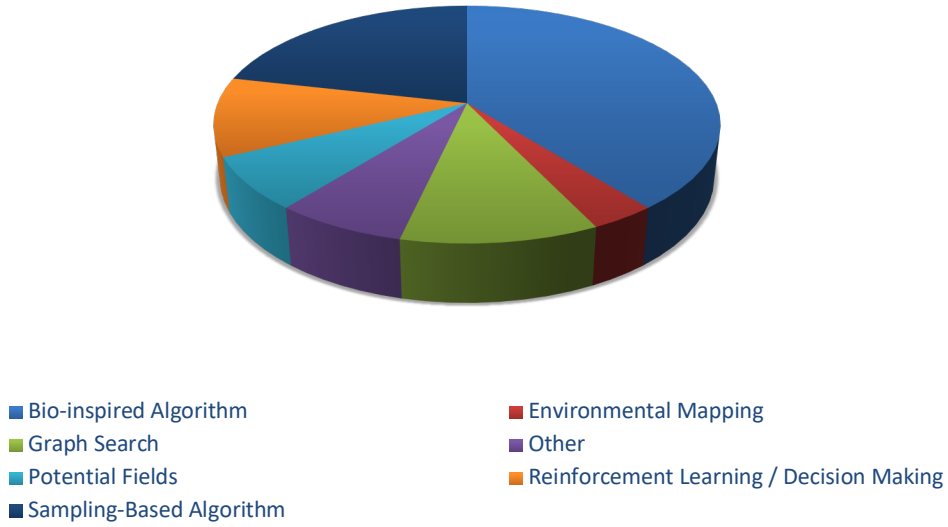


Figure 5. Algorithm Types

Although some of the proposed algorithms have great potential to be used as path planners for drones, limitations exist in terms of the quality of the solutions they produce or the overall computational resources they need. In the future, path planning solutions must consider multiple factors that affect the battery performance of drones. For example, changes in altitude or the number of times a drone needs to alter its flight direction can be considered. Moreover, to determine the effectiveness of proposed solutions for drone delivery applications, studies must test the algorithm in large-sized cluttered environments. Most studies only report the quality of the algorithm's path and running time in small-sized environments. Furthermore, although mathematical models can be used to simulate the dynamics of drones, more studies need to implement the algorithm on physical drones for validation. Finally, a detailed comparative

analysis that compares multiple types of algorithms based on memory consumption, running time and path cost is necessary. Although some studies compare their proposed algorithm's performance to other solutions, many of the reviewed studies do not perform a comparative analysis (Appendix A).

3.2 Drone Delivery

In this section, we review studies that have proposed the use of drones for delivery applications. Generally, there are two schemes relating to drone delivery that have been proposed: hybrid models and drone-only model.

Although companies can benefit from drone deliveries, the battery capacity constraints of drones limit their coverage range. To address this challenge, a few different approaches have been proposed. First, a drone-truck hybrid model [76] has been suggested where a truck leaves a depot with a set of parcels and drones. The drones depart from the truck and deliver to some customers while the truck delivers packages to other customers. Another study suggested using public transportation to make deliveries using drones [77]. A drone lands on the roof of a mobile vehicle and leaves the vehicle near the destination point. Although this would be a low-cost solution for delivering packages, it may be hard to scale with many deliveries. The approaches mentioned above are cost-effective and faster than traditional truck-only deliveries; however, there are still many limitations. For example, traffic congestion could lead to unexpected delays. Moreover, in remote areas and crisis regions, it may be too dangerous or time-consuming to rely on trucks to deliver essential items.

A few studies have suggested drone-only delivery solutions. Some studies focus on scheduling drones [78, 79]. A fleet of drones depart from a depot or warehouse, drop off a package and return to the warehouse, and the problem is to optimize the route of the drones to deliver packages efficiently. The mentioned problem is often referred to as the vehicle routing problem for drones. Although algorithms have been suggested to optimize drone delivery time, these studies often assume that the drone has the battery capacity to complete the delivery or assume deliveries will be made within a limited flight range. Another strategy has been to focus on optimizing the placement of recharging stations to extend the range of a drone [80]. In this model, the goals were to primarily limit the number of charging stations for drone refuelling

while ensuring that most of the area of customer demand was covered. Charging stations were placed strategically so that a drone can either recharge its battery or swap its battery and continue its mission.

In our proposed model in Chapter 7, we contribute to the development of the drone-only strategy. As opposed to having the same drone continue the delivery, we suggest that the drone hands off its parcel to another available drone and return to its original charging station to increase the efficiency of drone deliveries. Moreover, in our study, we propose having multiple drones at charging stations. With this approach, drones that are idle in nearby charging stations can assist with moving a parcel from one point to another.

Chapter 4

4 Preliminary Path Planning Study

In this chapter, we describe a path planning algorithm for drone navigation in fixed regions. We describe a method to bound the search space and generate a graph representative of obstacle-free areas of the environment. Finally, we apply a well-known shortest path algorithm to generate the shortest path between two points in the environment.

4.1 Algorithm Description

In the previous chapter, we reviewed many approaches used to compute a path. Most of the solutions required a lot of time and memory to generate a near-optimal solution. In the graph-search based algorithm proposed in this study, we reduce the number of nodes stored in memory to compute the shortest path between two points by limiting the overall search area.

The objective of the proposed offline algorithm is to determine the shortest path for the drone to travel from a source point *S* to a destination point *D*. The algorithm has been designed to accept an aerial image of the environment and pre-plan the shortest path of flight from the source point. An aerial image of the environment (i.e. snapshot) provides information about static obstacles that the drone must avoid during flight. Static obstacles include plants, trees and any commercial, residential or industrial infrastructure.

During the development of our proposed algorithm, the following properties and assumptions were made about the elements in this algorithm:

1. There exists a third-party service to track all obstacles in a snapshot. Images and information about the environment will be gathered from separate sources and consolidated to populate the snapshot.

2. Battery capacity is not a limiting factor. Thus, we assume that the drone will always have sufficient energy to navigate from *S* to *D*.

3. The drone will always take off at *S*. During drone navigation, Global Positioning System (GPS) technology will capture its current location.

4. Snapshots are rectangular and include all obstacles present at a specified altitude.
5. There is at most one *S* and one *D* in each snapshot.
6. Both *S* and *D* are always in obstacle-free, *F* grids. This ensures the possibility of finding an optimal path.
7. Path $P = \{F_1, F_2, \ldots, F_l\}$ which consist of a set of *l* adjacent *F* nodes.
8. There may be multiple unique paths from *S* to *D*, but *P* will store the shortest feasible path.
9. The drone can navigate in any direction on a horizontal plane.
10. The drone does not have any minimum turning angle requirements that need to be considered when computing the path.

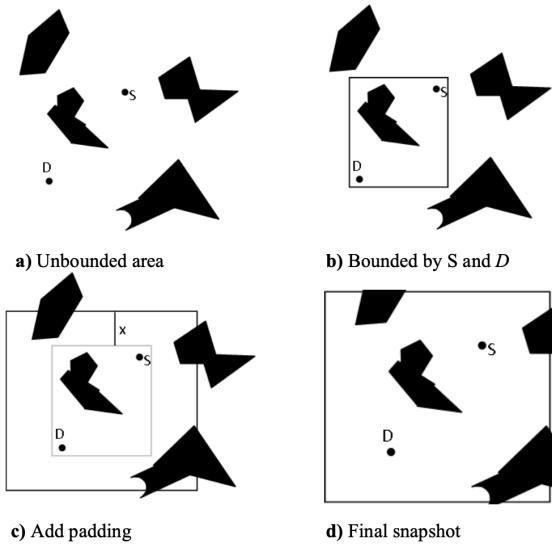


Figure 6. Processing the Snapshot [34]

Searching for an optimal path through the entire snapshot of the environment (Figure 6a) may be computationally expensive and unnecessary if the source and destination points are located in a portion of the snapshot. Thus, the snapshot should be processed in a manner where the algorithm can limit its search. The premise of determining the snapshot boundaries involves creating a two-

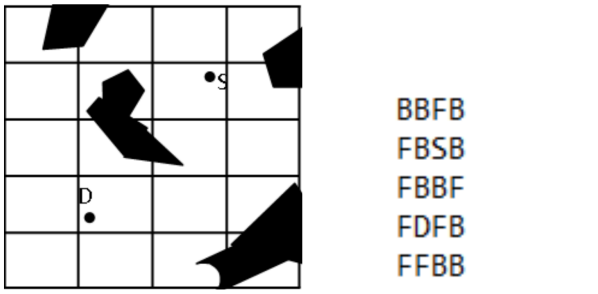
dimensional rectangle that places *S* and *D* in opposite diagonal corners (Figure 6b). To increase the success of finding the shortest path (by allowing the drone to travel around obstacles), the search area should be greater than the snapshot bounded by *S* and *D*. The amount of padding that should be included around the bounded image (Figure 6c) is proportional to the linear distance *x*, between the source and the destination. Therefore, the final snapshot (Figure 6d) will provide an optimal search area to determine the shortest path.

Algorithm 2: CellDecomposition(w, x)

Input: Frame size w, expansion value x, where x is at most the value of straight distance between the initial point and destination point
Output : Two-dimensional array C with a value of either F, B, S or DT

```
1  L ← −1; W ← −1
2  while L ∉ ℕ and W ∉ ℕ do
3      U ← GetBoundedSnapshot(x)
4      snapshotLength ← Length of U
5      snapshotWidth ← Width of U
6      L ← snapshotLength / w
7      W ← snapshotWidth / w
8      x ← RecomputeExpansion(x)
9  U* ← Partition U into L × W cells
10 for i ← 0 to W do
11     for j ← 0 to L do
12         if U*[i][j] contains the initial point then
13             C[i][j] ← S
14         else if U*[i][j] contains the destination point then
15             C[i][j] ← DT
16         else if U*[i][j] contains an obstacle then
17             C[i][j] ← B
18         else
19             C[i][j] ← F
20 return C
```

If most of the snapshot contains obstacles, it will be more challenging to determine the shortest path. Thus, to further improve the success of finding an optimal path, a threshold *y* is introduced to evaluate whether the proportion of obstacles is acceptable for the snapshot. If the snapshot is deemed unacceptable, a new snapshot (with a different amount of padding) will be created until the snapshot is acceptable. In Algorithm 2, the *getBoundedSnapshot()* function is responsible for interacting with third-party services to get a new snapshot of the environment. In this function, the snapshot is processed and adjusted to ensure that the proportion of obstacles is within the acceptable range.



a) Partitioning the snapshot into grids **b)** Label for each cell

Figure 7. Environmental Modelling Process [34]

Following the processing of the snapshot in Algorithm 2, the snapshot of the environment is modelled by partitioning the image into grid cells (Figure 7a). Each grid's size will be designed to ensure that the drone can physically fit in each cell. Thus, the number of grids on the length *L* of the snapshot and the number of grids on the width *W* of the snapshot must be a natural number to avoid creating a grid that would not be accessible by the drone. In Algorithm 2, the *recalculate()* function will determine a new padding value for the bounded snapshot until *L* and *W* are a natural number. After successfully partitioning the snapshot, each grid is labelled in a data structure with their respective grid cell types; *F* for obstacle-free cell, *B* for blocked cell, *S* for the cell with the source point, *D* for the cell with the destination point (Figure 7b).

Dijkstra's algorithm is traditionally applied to a graph with nodes to find the shortest path. To model the snapshot into a graph, we decided to implement a network, *N*, with the following properties:

1. A node *u* is created for each grid cell labelled *F, S* or *D*
2. A node *u* is adjacent to node *v* if and only if
    a. *u* and *v* are both *F* cells, and
    b. *u* is immediately next to, above, below or diagonal to *v*
3. A link connects two adjacent nodes
4. All links are bidirectional

5. All links have equal weights

To apply Dijkstra's algorithm to find the shortest path, each node in *N* must store specific attributes. The *distance* variable will be used to store the current distance from the source node. *prevNode* will store the parent node to keep track of the path. Finally, *id* stores the grid cell number. Each grid cell is assigned a number from 1…*q*, where *q* represents the total number of grid cells. Links are formed between adjacent obstacle-free cells, as shown in Figure 8.

Provided with a network *N* (Figure 8), Dijkstra's algorithm can now be used to determine the shortest path between *S* and *D* if both nodes lie in the same connected component. The path *P* will be returned as an empty structure if a path does not exist (*S* and *D* are in different connected components). If there is a path, the returned *P* will store each node's grid cell numbers from the source to the destination. Initially, the distance of all nodes except the source node is initialized to be a value greater than the size of the network to ensure that the algorithm starts traversal from the source node. The *distance* of each of a nodes' neighbours is changed to be min{neighbour node distance, current node's *distance* + 1}, to ensure that the shortest path from the source node is always maintained. Finally, once the destination node is determined, the path can be determined by tracing back the path with *prevNode*. The drone will contain a mapping of the environment with grid cell numbers (Figure 8). Thus, with information from *P*, the drone will be able to traverse to the destination without colliding into obstacles.

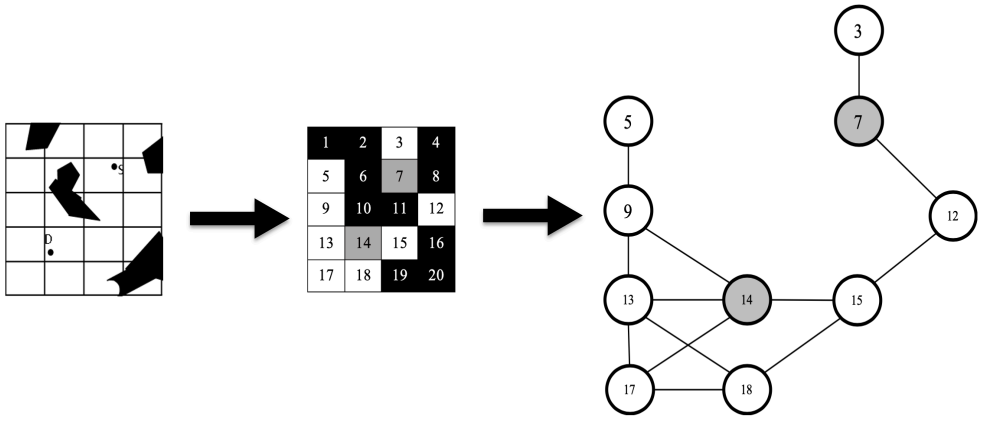


Figure 8. Process of converting approximate cells into a graph. Adapted from [34]

4.2 Discussions

In this study, we proposed an algorithm that can be used by surveillance drones to determine the optimal flight path between two points. We provide insight into how the drone should process a snapshot of the environment and decompose it into grid cells. Finally, we convert the modelled environment into a graph to determine the optimal path.

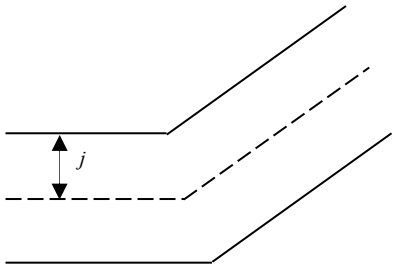


Figure 9. Proposed "tube" for the flight path. The dashed line represents the flight path. [34]

The “optimal” path can be determined on several criteria such as minimum distance travelled, flight time and fuel consumption. In our study, we determined the shortest distance from the source node to the destination node. However, in this study, we did not consider other criteria, such as fuel consumption and other flight constraints. In [82] and [83], they consider various constraints when proposing their path planning algorithm. Using a linear utility function, the authors in [83] developed a decision-maker that chooses the best path between targets and then the best overall tour. In [82], the authors propose using a path planner to evaluate and adjust the path according to geometric constraints (maximum deviation angle) when moving from one cell to another. In the future, we could further investigate the approach taken in [82]. We will visualize the flight path as a tube (Figure 9). The radius *j* will provide space for the drone to safely adjust its path or position due to disturbances like wind gusts. Moreover, this setup will ensure the passage of the drone without collision into obstacles when moving to a grid cell located diagonal to the current cell. To ensure that collisions do not occur due to geometric constraints, we could also make all cells next to an impermeable cell impermeable – as described in [82]. The advantages and disadvantages of these approaches will need to be investigated in the future. Finally, there are many different types of drones, such as fixed-wing and multirotor drones. When designing this algorithm, we assumed a multirotor would be used during the

implementation of the algorithm. Multirotor drones have better maneuverability and vertical takeoff and landing capabilities; thus, kinematic constraints were not considered when developing our path planning approach.

Currently, the developed algorithm assumes the environment is static. However, there are many dynamic constraints such as the weather, animals and other aerial vehicles that need to be considered when the flight path is being evaluated. This is an issue that needs to be addressed since the predetermined optimal path may contain a blocked grid cell due to an introduced obstacle. Thus, in the future, we will make changes to the algorithm to ensure that the optimal path is calculated each time the drone moves to the next grid cell in the flight path. This way, even if the destination point is modified during the flight path, the drone will be able to re-route its path to move to another location. Although this may be computationally complex, if the number of nodes created from the snapshot increase exponentially, since surveillance drones are only surveilling a fixed area, the maximum number of nodes (grid cells) is limited.

Furthermore, alterations can be made to our proposed strategy to improve the drone's efficacy for surveillance purposes. The current assumption is that links in the network N have equal weights. However, when surveilling an area, there may be portions of the area which have a greater probability of detecting a threat. Thus, we will need to determine how assigning greater weights for these links in N will affect the generation of the optimal path. Introducing weights also introduces the possibility of using the A* algorithm to improve execution time; however, developing the correct heuristic function will be important in determining the optimal shortest path. Finally, we are also interested in how this study can be applied to multiple drones – in order to survey the perimeter of a predefined area thoroughly. Thus, future studies will address how the proposed strategy can be applied to a swarm of drones and how they can cooperate as a team to surveil the desired area.

With the algorithm's current design, we propose using two important methods that we have not implemented in this study; *recalculate()* and *getBoundedSnapshot()*. The *recalculate()* method will recalculate the amount of padding added to the snapshot when some generated grids are inaccessible to drones. The maximum proportion of padding that should be allowed to be added to the snapshot is yet to be determined. Moreover, the *getBoundedSnapshot()* assumes that

the snapshot is being processed in a manner where the proportion of obstacles does not exceed a certain threshold, y. However, further investigations will be needed to determine a threshold that can be applied to various scenarios.

Although the algorithm currently returns an empty P if no path exists, we plan to alter the algorithm to improve the success of finding a path in future implementations. Even though the generated snapshot may not have a path, when expanding the snapshot using *recalculate()*, there may be an opportunity for the drone to traverse around an obstacle, resulting in an optimal path. Thus, the algorithm will be modified, so it will not be able to conclude that there is no path until the snapshot has been expanded to search the entire environment.

Furthermore, to improve the success of finding the optimal path, we need to consider path planning in 3D environments. Currently, it is assumed that the drone can only traverse in the same horizontal plane. Though, in reality, the drone may pass an obstacle and even produce a more optimal path by moving above or below the current plane. However, to map a 3D environment, our environmental modelling approach would need to be slightly altered. Similar to the approach taken in [84], we could consider constructing a 2D search space based on the 3D space by introducing a “virtual terrain” above the real terrain, enabling path planning on a 2D surface. Thus, in future studies, we will try to determine whether the authors' approach taken in [84] can be integrated with our approach to determine a path in a 3D environment.

Finally, in the future, we plan on implementing the complete proposed strategy on a physical drone to determine its efficacy in surveilling a real environment. The proposed approach requires further investigation to account for the assumptions made in this study and to improve its usefulness; however, it provides a base strategy that can be used to determine an optimal path in fixed-areas – making it ideal for surveillance applications.

Chapter 5

5 Analysis of Sampling-Based Path Planning Algorithms

In this chapter, we investigate the performance of existing path planning algorithms, in particular sampling-based algorithms. We examine properties of RRT, RRT* and variants of the RRT* algorithm.

5.1 Problem Formulation

Let a set $X \subset \mathbb{R}^n$ represent the environment in which the drone operates (i.e., workspace), where n is the number of dimensions used to describe the workspace. In this chapter, we analyze the performance of select sampling-based algorithms in two-dimensional (2D) and three-dimensional (3D) space. Obstacles are depicted as polygons in 2D spaces and as a right parallelepiped (where the angle between any edge is 90°) in 3D spaces. To transform the drone into a point in space, obstacles in the workspace were expanded (Figure 10). The x and y dimensions of each obstacle were increased by an amount equivalent to the frame size of the drone w and the height of the drone d. This ensures that the drone can navigate around obstacles without collision.

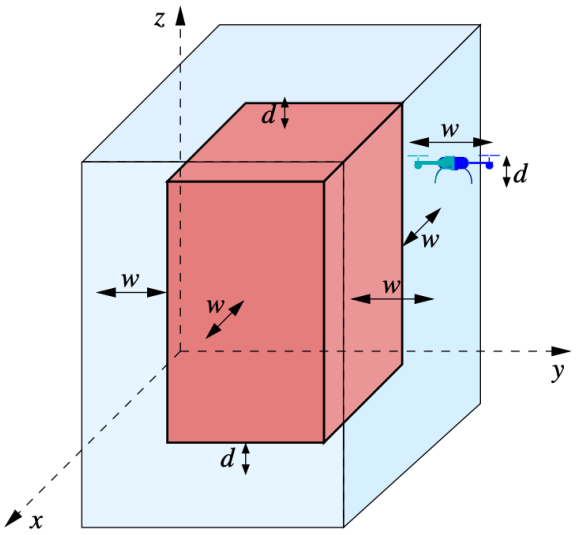


Figure 10. Obstacle Expansion [62]

Let $O = \{o_1, o_2, ..., o_K\}$ be the set of expanded obstacles, where K is the total number of obstacles in the environment. The set of points occupied by all obstacles are denoted as X_{obs}, where $X_{obs} \subset X$. Obstacle free regions in X are denoted as X_{free}, where $X_{free} = X \setminus X_{obs}$. To generate a feasible path, we assume that the initial point p_{init} and the destination point p_{dest} belong to X_{free}. $X_{goal} \subset X_{free}$, defines the goal region, which represents the set of points in the vicinity of the destination point. A cost of a path $\sigma(p_a, p_b)$ is the amount of energy spent by the drone to navigate from p_a to p_b. Let $\phi(p_a, p_b)$ represent the computed path between p_a to p_b. In this chapter, we assume that the amount of energy spent by the drone to navigate between the two points is equivalent to the total distance travelled. In general, distance between the points p_a and p_b is calculated using equation 5.1, where $p_a = (a_1, a_2, ..., a_n)$ and $p_b = (b_1, b_2, ..., b_n)$. Assuming that the line connecting two points, p_a and p_b lies in X_{free}, $\sigma(p_a, p_b) = l(p_a, p_b)$. We use the Cartesian coordinate system on the region X to specify the obstacles' positions and describe the path taken by a drone.

$$l(p_a, p_b) = \|p_a - p_b\| = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{5.1}$$

Sampling-based algorithms aim to address three main problems in path planning [54-56]:

1. Find a feasible path: If there is a path connecting p_{init} and p_{dest}, the algorithm will output the path. Otherwise, a failure will be reported.

2. Generate an optimal solution: Minimize the cost of the path connecting p_{init} and p_{dest}

3. Convergence to an optimal solution: Compute an optimal path in the least possible time.

5.2 Sampling-Based Algorithms

By far, sampling-based algorithms are one of the dominating methods used for offline and online path planning for drones and other robotic systems. Sampling-based path planning algorithms are particularly beneficial for drones since they can compute a path very fast. Moreover, many studies have used RRT and RRT* variants for path planning in cluttered environments [59-61],

indicating that these algorithms can rapidly generate feasible paths in urban environments with many obstacles.

Sampling-based algorithms are not c*omplete* planners. Complete planners can guarantee that they will find a solution if one exists. But these planners have been shown to suffer from computational complexity for even the basic version of the motion planning problem [54]. However, RRT, RRT* and its variants are probabilistically complete. Thus, provided that a solution exists, the probability of computing a feasible path approaches one as the number of samples inserted into the tree approaches infinity.

5.2.1 Randomly-Exploring Random Trees (RRT)

RRT is a randomized planning technique introduced by LaValle [53]. The RRT algorithm builds a tree $T = (E, V)$ by repeatedly sampling points from X_{free}. Each node in the tree contains information about its state as well as its parent node. The tree is initialized from $p_{init,}$ and in each iteration, the tree is extended until $p_{dest} \in X_{goal}$ is reached. The RRT algorithm has an interesting property referred to as Voronoi bias, which is responsible for the rapid exploration of X. When considering the Voronoi diagram of the RRT vertices, the probability that a node is randomly selected is proportional to the volume of its Voronoi region. Therefore, exploration is biased towards the nodes with the largest Voronoi regions, which represent unexplored regions of X [54]. The general structure of the RRT algorithm is provided in Algorithm 3.

Algorithm 3: RRT

```
T ← InitializeTree()
T ← InsertNode(∅, p_init, T)
for i ← 0 to N do
    p_rand ← Sample()
    p_nearest ← Nearest(T, p_rand)
    p_new ← Steer(p_nearest, p_rand, η)
    if ObstacleFree(p_nearest, p_new) then
        T ← InsertNode(T, p_nearest, p_new)
return T
```

In an environment with obstacles, there are four methods that are used in the process of adding a node into the tree: sampling, nearest neighbour selection, steering and collision checking. In each iteration up until a value N, a sampling technique such as uniform sampling is used to select a random point p_{rand} from X_{free}. Using equation 3.1 (Euclidean distance), the point with the

minimum distance to p_{rand} (i.e. nearest vertex) is selected from T. The steer method (Figure 11) then returns a point p_{new}, such that p_{new} is closer to p_{rand} than it is to the nearest point $p_{nearest}$ in the tree, by moving $p_{nearest}$ a pre-specified step value $\eta > 0$ towards p_{rand}. Finally, an attempt is made to connect $p_{nearest}$ and p_{new}. If the edge between p_{new} and $p_{nearest}$ lies in X_{free}, then p_{new} is inserted into the vertex set and ($p_{nearest}$, p_{new}) is added to the edge set.

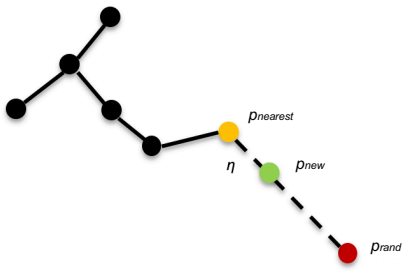


Figure 11. Adding a new sample to the tree

5.2.2 RRT*

Karaman and Frazzoli [54] show that RRT almost always converges to a non-optimal solution and instead propose an alternative solution, the RRT* algorithm. The RRT* algorithm was proved to be asymptotically optimal by showing that as the number of samples inserted into the tree approaches infinity, the probability that the minimum cost path in RRT* converges to an optimal solution is one.

Algorithm 4: RRT*

```
T ← InitializeTree()
T ← InsertNode(∅, p_init, T)
for i ← 0 to N do
    p_rand ← Sample()
    p_nearest ← Nearest(T, p_rand)
    p_new ← Steer(p_nearest, p_rand, η)
    if ObstacleFree(p_nearest, p_new) then
        X_near ← Near(T, p_new, r)
        p_min ← ChooseBestParent(T, X_near, p_nearest, p_new)
        T ← InsertNode(T, p_min, p_new)
        T ← Rewire(T, X_near, p_min, p_new)
return T
```

The details of the RRT* algorithm are shown in Algorithm 4. The RRT* grows a tree T from p_{init} and inserts p_{new} to the tree in a similar manner as RRT. But the RRT* algorithm has two major

additions. Instead of always connecting p_{new} to the nearest vertex in the tree, the algorithm looks at vertices in $X_{near,}$ which represents the set of vertices in the tree that are located within a specific radius r around $p_{new.}$ The value of r is determined using equation 5.2:

$$r = \min\left\{ \gamma \left(\frac{\log |V|}{|V|}\right)^{1/n}, \eta \right\} \tag{5.2}$$

where $|V|$ is the number of vertices in the tree and γ is a planning constant. To ensure asymptotic optimality, Karaman and Frazzoli proved that the value of γ must satisfy the following:

$$\gamma > \left(2\left(1+\frac{1}{n}\right)\right)^{1/n} \left(\frac{\mu(X_{free})}{\zeta_n}\right)^{1/n} \tag{5.3}$$

where $\mu(X_{free})$ is the area (n = 2) or volume (n = 3) of the free space in the environment and ζ_n is the volume of the unit sphere in a dimension, $n \geq 2$.

As shown in Algorithm 5, the `ChooseBestParent` method iterates through the entire set of vertices to determine the vertex with the minimum cost. Once the vertex with the minimum cost p_{min} is found, the edge (p_{min}, p_{new}) is added to the edge set.

Algorithm 5: ChooseBestParent $(T, X_{near}, p_{nearest}, p_{new})$

```
p_min ← p_nearest
c'_min ← σ(p_init, p_nearest) + σ(p_nearest, p_new)
for all p_near ∈ X_near do
    if ObstacleFree(p_near, p_new) then
        if σ(p_init, p_near) + σ(p_near, p_new) < c'_min then
            p_min ← p_near
            c'_min ← σ(p_init, p_near) + σ(p_near, p_new)
return p_min
```

Finally, the RRT* performs a "rewiring" operation to ensure that a path with the minimum cost from p_{init} to any vertex in the tree can be generated. The `Rewire` method (Algorithm 6) modifies the edge of vertices in X_{near} if the cost of reaching the vertex is lower through $p_{new.}$ To maintain the property of a tree, the edge (p_{parent}, p_{near}) must be removed. Although this improves the path quality of the RRT* when compared to RRT, it requires more iteration to optimize the path, resulting in a slow rate of convergence.

Algorithm 6: Rewire $(T, X_{near}, p_{min}, p_{new})$

```
for all p_near ∈ X_near \ {p_min} do
    if ObstacleFree(p_near, p_new) and σ(p_init, p_near) > σ(p_init, p_new) + σ(p_new, p_near) then
        p_parent ← Parent(p_near)
        E ← E \ {(p_parent, p_near)}
        E ← E ∪ {(p_new, p_near)}
return T
```

5.2.3 Intelligent Bidirectional RRT* (IB-RRT*)

Qureshi and Ayaz [55] proposed the IB-RRT* algorithm, a variant to the RRT* algorithm, which improves the rate of convergence to an optimal solution. The IB-RRT* algorithm (Algorithm 7) utilizes a bidirectional tree to generate a path. Two trees, $T_a = (V_a, E_a)$, grows from p_{init} and $T_b = (V_b, E_b)$ grows from p_{dest}. Until the termination condition is met, the respective trees grow, and an attempt is made to connect the trees in each iteration.

Algorithm 7: IB-RRT*

```
T_a ← InitializeTree(); T_b ← InitializeTree()
T_a ← InsertNode(∅, p_init, T_a); T_b ← InsertNode(∅, p_dest, T_b)
σ(p_init, p_des) ← ∞
for i ← 0 to N do
    Connection ← true
    p_rand ← Sample()
    {X^a_near, X^b_near} ← Near(T_a, T_b, p_rand, r_a, r_b)
    if X^a_near = ∅ and X^b_near = ∅ then
        {X^a_near, X^b_near} ← Nearest(T_a, T_b, p_rand)
        Connection = false
    else if X^a_near = ∅ then
        X^a_near ← Nearest(T_a, p_rand)
        Connection = false
    else if X^b_near = ∅ then
        X^b_near ← Nearest(T_b, p_rand)
        Connection = false
    L_a ← GetSortedList(p_rand, X^a_near)
    L_b ← GetSortedList(p_rand, X^b_near)
    {p_min, flag} ← GetBestTreeParent(L_a, L_b)
    if Connection then
        if σ(p_init, p_des) > σ(p_init, p_rand) + σ(p_dest, p_rand) then
            Update φ(p_init, p_des)
    if flag then
        T_a ← InsertNode(p_rand, p_min, T_a)
        T_a ← Rewire(p_rand, L_a, T_a)
    else
        T_b ← InsertNode(p_rand, p_min, T_b)
        T_b ← Rewire(p_rand, L_a, T_b)
return T
```

Since collision checking is one of the costliest operations in the RRT* algorithm, in the IB-RRT* algorithm, the authors enhance the computational efficiency by limiting the number of collision checking operations that need to be performed. In the `GetSortedList` method, a list is created for near vertices from T_a and T_b. The list $L = \{(p_1, C_1, \phi_1), (p_2, C_2, \phi_2), \ldots, (p_k, C_k, \phi_k)\}$, where k is the number of vertices in X_{near}, C_k is the cost of the path from p_{init} to p_k in addition to the distance from p_k to p_{rand} and ϕ_k is the concatenation of the path from p_{init} to p_k and the path from p_k to p_{rand}. The vertices in the list belong to X_{near} and are sorted in increasing order by C. The `GetBestTreeParent` method replaces the `ChooseBestParent` method in the RRT*. Since L_a and L_b are sorted, the `ObstacleFree` method does not need to be called for the entire set of vertices in X_{near}^{a} and X_{near}^{b} to find the vertex with the minimum cost from each tree; thus, improving the algorithm's performance. The best vertex p_{min} is chosen to be inserted into either T_a or T_b, and a Boolean variable *flag* is used to indicate the tree in which the vertex can be inserted into with the minimum path cost (Line 19).

Finally, if the Boolean variable *Connection* is true, an attempt is made to connect the trees (Line 20). The value of *Connection* is only true when there are a set of near vertices around p_{rand} from both T_a and T_b. The global path from the initial point to the destination point $\phi(p_{init}, p_{dest})$ is computed if the concatenation of the path to p_{rand} from T_a (ϕ_a) and the path to p_{rand} from T_b (ϕ_b) has a path cost that is less than $\sigma(p_{init}, p_{dest})$. Similar to the RRT* algorithm, the IB-RRT* also modifies edges in the trees to ensure that a path with the minimum cost from p_{init} (in T_a) or p_{dest} (in T_b) to any vertex in their respective tree can be generated (Lines 24 and 27).

5.2.4 RRT*-Adjustable Bounds (RRT*-AB)

The RRT*-AB algorithm [56] has been shown to generate an optimal path in less time than the RRT* algorithm by limiting the exploration region in the environment and using path optimization techniques. The RRT*-AB algorithm is outlined in Algorithm 8.

The RRT*-AB algorithm initializes a tree T, rooted at p_{init} and randomly samples points from a limited region C_{Region} (Line 6), where $C_{Region} \subset X_{free}$. C_{Region} defines a space between p_{init} and p_{dest} using an expansion distance scale D_{scale}:

$$D_{scale} = \frac{E}{m} \tag{5.3}$$

where E is the size of the environment map, and m is an expansion factor. In order to sample points from at least half of X_{free}, the default value of m is 4. In each iteration, since points are sampled from C_{Region}, sampling is biased towards the goal. When the tree is occluded and cannot expand any further in C_{Region}, the value m is modified in the `ConnectivityRegion` method to gradually expand C_{Region} until a path has been discovered or until any point in X_{free} can be sampled (Lines 18-19).

Algorithm 8: RRT*-AB

```
1  T ← InitializeTree()
2  T ← InsertNode(∅, p_init, T)
3  pathFound ← false
4  C_Region ← ConnectivityRegion(p_init, p_dest, T)
5  for i ← 0 to N do
6      p_rand ← BoundedSample(C_Region)
7      p_nearest ← Nearest(T, p_rand)
8      p_new ← Steer(p_nearest, p_rand, η)
9      if ObstacleFree(p_nearest, p_new) then
10         X_near ← Near(T, p_new, r)
11         p_min ← ChooseBestParent(X_near, p_nearest, p_new)
12         T ← InsertNode(p_nearest, p_new, T)
13         T ← Rewire(T, X_near, p_min, p_new)
14         if p_new ∈ X_goal then
15             pathFound ← true
16         if pathFound then
17             C_Region ← ConnectivityRegion(p_init, p_dest, T)
18         else if CompleteScan(C_Region) then
19             C_Region ← ConnectivityRegion(p_init, p_dest, T)
20 φ(p_init, p_dest) ← PrunePath()
21 return φ(p_init, p_dest)
```

Once a path has been discovered, optimization techniques are used to improve the path further. The RRT*-AB algorithm further limits the points that can be sampled. C_{Region} is modified to ensure that points can only be sampled around the initial path. This results in a lot of rewiring operations around the initial path and reduces the path cost. Moreover, a node rejection technique is used. The algorithm considers the path cost between p_{init} to $p_{new,}$ and the straight distance between p_{new} to p_{dest} is used to determine whether p_{new} can be inserted into the tree. Since inserting p_{new} with a higher cost would not benefit in optimizing the initial path, the vertex is

discarded. Finally, the path pruning is performed to shorten the path further (Line 24). The path pruning process is described in detail in section 5.3.4.

5.2.5 Bidirectional RRT*-Adjustable Bounds (BiRRT*-AB)

Since the objective is to rapidly generate a near-optimum path using limited resources, we decided to implement a hybrid algorithm BiRRT*-AB, which combines properties of the IB-RRT* and RRT*-AB to generate a feasible path. A bidirectional tree, with two trees growing from p_{init} from $p_{dest,}$ was implemented. Bidirectional trees have been shown to perform better than a single tree in the RRT* algorithm. Moreover, restricting the search space to a defined region between p_{init} and p_{dest} has proved to reduce the large memory requirements of the RRT* algorithm.

Finally, an additional pruning function was added to the BiRRT*-AB to reduce the path cost. Pruning the path also reduces the number of turns required to complete a path (Figure 12). Studies have shown that the power consumption of a rotorcraft can be reduced by decreasing the number of turns [68]. This is in part because a rotorcraft needs to slow down before altering its flight direction. The frequent acceleration and deceleration can impact the battery performance of a drone.

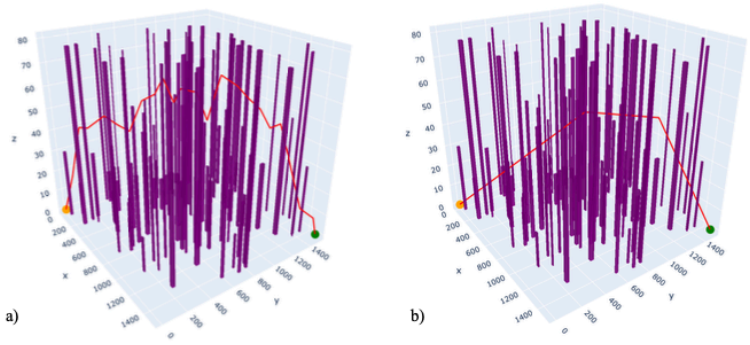


Figure 12. (a) Original Path (b) Pruned Path

We define a turn in the path as follows:

$$10° < \theta = \cos^{-1}\left(\frac{u \cdot v}{\|u\|\|v\|}\right)$$

where θ is the angle formed between two directed line segments. Experimental studies have shown that pruning the path for the IB-RRT* algorithm only improves the path cost by up to 1%. But there is up to a 75% decrease in the number of turns in the drone's flight path.

5.3 Implementation Details

In this section, the implementation details of important methods are described. Various methods were analyzed to determine factors that affect the performance of the algorithms.

5.3.1 Sampling Method

Sampling a point from X_{free} is one of the most critical steps in sampling-based algorithms. There are many different types of sampling strategies. For example, for online path planning, to avoid having to maintain a full discrete map, in [5], a Gaussian process occupancy map was created to predict the probability of there being a collision along the path. Based on this information, the algorithm would decide on whether it should insert vertices into the tree. Another approach is to apply a heuristic to bias sampling. Goal biasing is often used to sample points towards the destination point. The RRT*-AB algorithm uses this methodology. Local biasing [63] can also be used to sample points near a current path to improve the path cost towards a locally optimal one.

For consistency, the RRT, RRT* and IB-RRT* algorithms were implemented using a uniform sampling method. Thus, every point in X_{free} had an equal probability of being sampled. For the RRT*-AB algorithm, every point within C_{Region} had an equal probability of being sampled.

5.3.2 Nearest Neighbour Search

Nearest neighbour searches and collision detection between two points are arguably the costliest operations in sampling-based algorithms. The computational complexity of near neighbour searches grows as the tree rapidly expands. To overcome this issue, some studies have limited the number of possible points that can be inserted into the tree [66]. Another study uses a box approach where the configuration space is partitioned into *n*-dimensional boxes [67]. With this approach, only specific boxes need to be searched for the nearest neighbour. Other data structures such as quadtrees and *k*-d trees have also been suggested for nearest neighbour searches [67].

In this study, to improve the performance of searching for neighbours around a sampled point, a *k*-d tree was implemented. A *k*-d tree recursively subdivides the *k*-dimensional space into two half-spaces. Unlike binary search trees, which use a single key throughout the tree, a *k*-d tree alternates between *k* keys at each tree level. In the average case, the *k*-d tree can perform a nearest neighbour search in sublinear time [54]. Since near neighbour and nearest neighbour searches are performed frequently in these algorithms, utilizing this data structure can effectively reduce the algorithm's run time.

5.3.3 Collision Detection

Since collision detection frequently occurs in sampling-based algorithms, an efficient method to detect collisions may improve the algorithm's running time. In fact, some studies have shown that collision detection queries account for approximately 90% of the time taken to compute the path [65]. In the analysis of the 2D implementation of the algorithms (5.4.1), a naïve approach was taken. For the line connecting p_a and p_b, an intersection point was calculated for each edge of an obstacle in O to determine if the path was feasible. This consequently increased the running time of these algorithms. However, for the algorithm's 3D implementations, an improved technique was used to reduce the number of obstacles that needed to be checked. Only obstacles within the polygon (or plane) generated by p_a and p_b were evaluated for collisions (Figure 13).

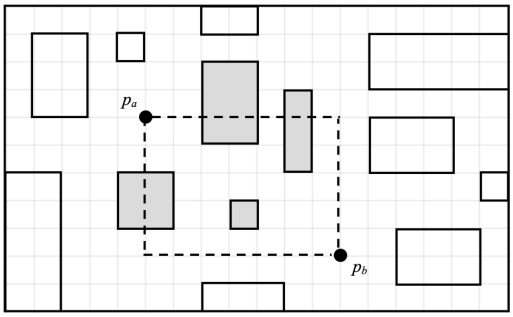


Figure 13. Collision detection technique

5.3.4 Path Pruning

Once the algorithm generates an initial path $\phi(p_{init}, p_{dest})$, the `PrunePath` method optimizes the generated path by applying a path pruning technique. Using the principle of Triangular Inequality, the initial path can be shortened. As shown in Figure 14, an iterative process starting

from p_e continuously checks previous points in the path until the line connecting the two points are no longer visible, or p_a is reached.

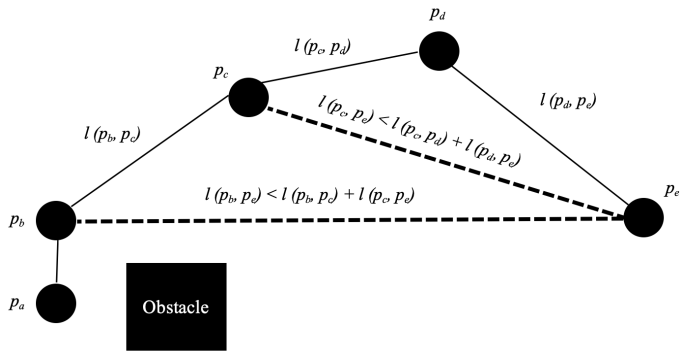


Figure 14. Path Pruning Technique

5.4 Experimental Results

Initially, a comparative analysis was performed to evaluate the performance of the algorithms in two-dimensional spaces. Later, we compare the performance of BiRRT*-AB and the other RRT* variants in cluttered three-dimensional spaces with various number of obstacles.

5.4.1 Performance Comparison in 2D Space

Experiments were performed on an Ubuntu 19.04 system with 4 GB RAM and an Intel Core 2 Duo E7500 processor @ 2.93 GHz. For comparison purposes, the size of the search area was kept constant at 500 × 500. Due to the random nature of these algorithms, each experiment was repeated 20 times.

The algorithms were evaluated in three different environments (Figure 15). We define E1 as a *simple environment*, which depicts an environment with less than 50 obstacles. E2 represents a *cluttered environment*, where many obstacles of various sizes are randomly placed in the environment. For this study, we use a cluttered environment with 250 obstacles. Finally, E3 is a *complex environment*, requiring exploration of a large portion of the environment to generate a feasible path.

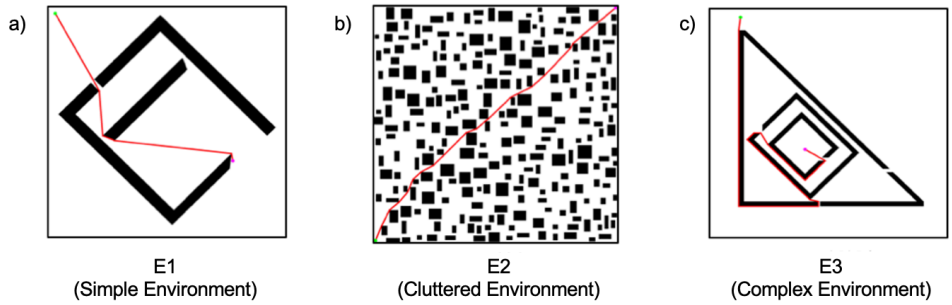


Figure 15. Environment Types for 2D Experiments

Although RRT* and IB-RRT* ensure that a near-optimum solution can be generated as the number of iterations approach infinity, this is impractical for drones. Therefore, we set the termination condition such that the algorithm would continue to iterate until a finite number of nodes have been inserted into the tree after an initial path was generated. This allows the initial path to improve by further exploration and rewiring. For the RRT* and IB-RRT* algorithms, 15,000 nodes had to be inserted since the entire environment needed to be explored. However, since the RRT*-AB performs concentrated sampling around the initial path, we set this value to be 1,500 nodes. Finally, the value of the planning constant γ was set to 200 for all experiments.

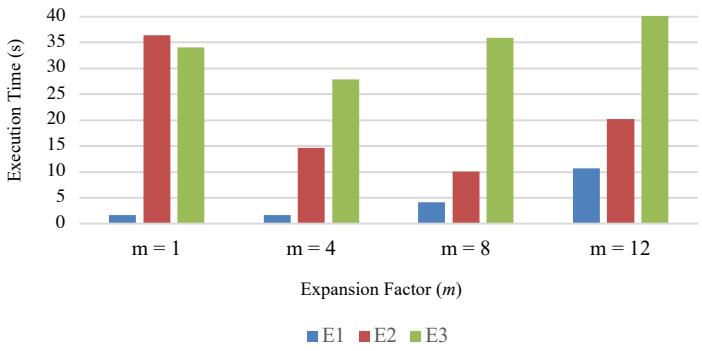


Figure 16. Effect of the expansion factor on the execution time of RRT*-AB

Since RRT*-AB relies on the expansion factor m to define the limited search region C_{Region}, we evaluated the algorithm's performance when modifying the value of m in all three environment types. As the value of m increases, C_{Region} decreases by narrowing the search region. Although there were very minimal changes in the path cost for E1 and E3, generally, as the value of m

increases from 4 to 12, the average execution time increases (Figure 16). When *m* is 1, the entire space is searched for a path, which results in an increase in execution time (comparable to RRT*) for E2 and E3. Furthermore, improvements in path cost were observed for E2. As the value of *m* increased from 1 to 12, the average path cost for E2 decreased by 5%. When the value of *m* was 8, the RRT*-AB performed the best in terms of execution time while having a path cost that was not significantly different from the path generated when $m = 12$. Therefore, the expansion factor can have an impact depending on the type of environment. More narrow search spaces improve the path cost in cluttered environments, whereas greater search spaces improve the path cost in complex environments.

In this comparative study, the original RRT, RRT* and the IB-RRT* algorithms were modified to include the path pruning technique described in 5.3.4. As shown in Figure 17, since the original RRT algorithm does not have any rewiring operations to optimize its path, it benefited the most from pruning its path. However, the running time of the RRT algorithm also increased by an average of 50% in E1, E2 and E3 as a result of pruning the path. This is mainly due to the increase in the number of collision checking operations required. There was also a 1% to 3% improvement in the average path cost for RRT* and IB-RRT*; however, this impacted the average running time by around 1%. Although minor changes were observed for the RRT* and IB-RRT*, the impact these changes could have on a drone's energy consumption requires further investigation.

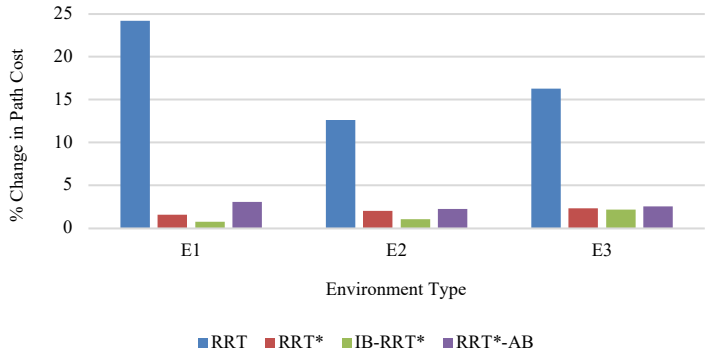


Figure 17. Improvement in path cost after pruning the initial path

In this study, we used the average execution time (running time of the algorithm), the number of nodes required to generate a path (to estimate memory usage) and the path cost $\sigma(p_{init}, p_{dest})$ as performance metrics to evaluate each algorithm. Table 2 summarizes the results from the experiments with the minimum, maximum and average path cost (σ), running time of the algorithm (t) and the number of nodes in the tree ($|V|$).

Table 2. Experimental results comparing path cost, running time and the number of nodes in each tree

Environment	Algorithm	σ_{min}	σ_{max}	σ_{avg}	t_{min} (s)	t_{max} (s)	t_{avg} (s)	$\|V\|_{min}$	$\|V\|_{max}$	$\|V\|_{avg}$
E1	RRT	836.7	1128.7	969.6	0.1	1.0	0.3	699.0	11439.0	4915.1
	RRT*	821.3	959.8	835.8	2.9	3.2	3.0	15141.0	16096.0	15432.3
	IB-RRT*	819.1	835.9	827.6	2.0	2.2	2.1	15006.0	15226.0	15060.2
	RRT*-AB	826.4	1061.7	865.0	3.2	5.0	4.2	3851.0	9826.0	6690.0
E2	RRT	1111.4	1417.0	1243.3	0.4	4.0	1.6	1975.0	17767.0	9136.6
	RRT*	1028.5	1073.8	1047.1	17.5	33.9	23.4	6017.0	10372.0	8175.0
	IB-RRT*	1038.5	1077.1	1053.2	8.2	9.2	8.7	5063.0	5582.0	5232.3
	RRT*-AB	1010.1	1059.3	1034.7	6.2	13.1	10.1	1116.0	2428.0	1670.1
E3	RRT	1622.1	2170.7	1738.6	11.9	68.3	35.8	28780.0	57377.0	42566.2
	RRT*	1552.8	1654.7	1583.8	17.5	36.0	27.1	38651.0	69213.0	54794.0
	IB-RRT*	1558.8	1596.9	1574.6	4.2	7.3	5.2	17277.0	27672.0	20613.9
	RRT*-AB	1550.5	1650.0	1588.2	27.0	45.8	36.0	44355.0	74453.0	57643.5

For comparison purposes, we constructed a visibility graph and applied Dijkstra's algorithm to generate the optimal path for each experiment. The vertices in each tree and the near-optimal paths generated by the RRT, RRT* and RRT* variants are illustrated in Figure 18.

In the simple environment, the RRT algorithm generates a path the fastest with an execution time of 0.3s while utilizing the least number of vertices to find a viable path. Although the RRT algorithm randomly generates a path close to the optimum (as shown in Figure 18a), there is a large variance between the minimum path cost and maximum path cost. Even though the RRT* produces a better solution, it requires more time to execute all the rewiring operations and requires many more vertices in the tree. Moreover, in simple environments, the IB-RRT* algorithm outperforms RRT*-AB in terms of running time and path cost. Once the RRT*-AB initially discovers a path, it only samples vertices near the initial path. Therefore, it can no longer detect if a better solution is available following a path further away from the initial path. As a result, there is a large variance in path cost.

In the cluttered environment (E2), although both IB-RRT* and RRT*-AB generate a path similar to the optimal solution, RRT*-AB generates a better path with fewer vertices in its tree. Since the

search space is limited, RRT*-AB requires much less memory than the other algorithms. However, in complex environments (E3), when a larger area of the environment needs to be searched, RRT*-AB does not perform well. Since sampling is restricted in C_{Region} until the tree cannot expand any further, there are more nodes in the tree compared to the other algorithms. Even though RRT*, IB-RRT* and RRT*-AB generate similar paths (Figure 18c), the IB-RRT* generates the path with the smallest tree (on average), smallest variance in path cost, and requires the least amount of time to generate the solution.

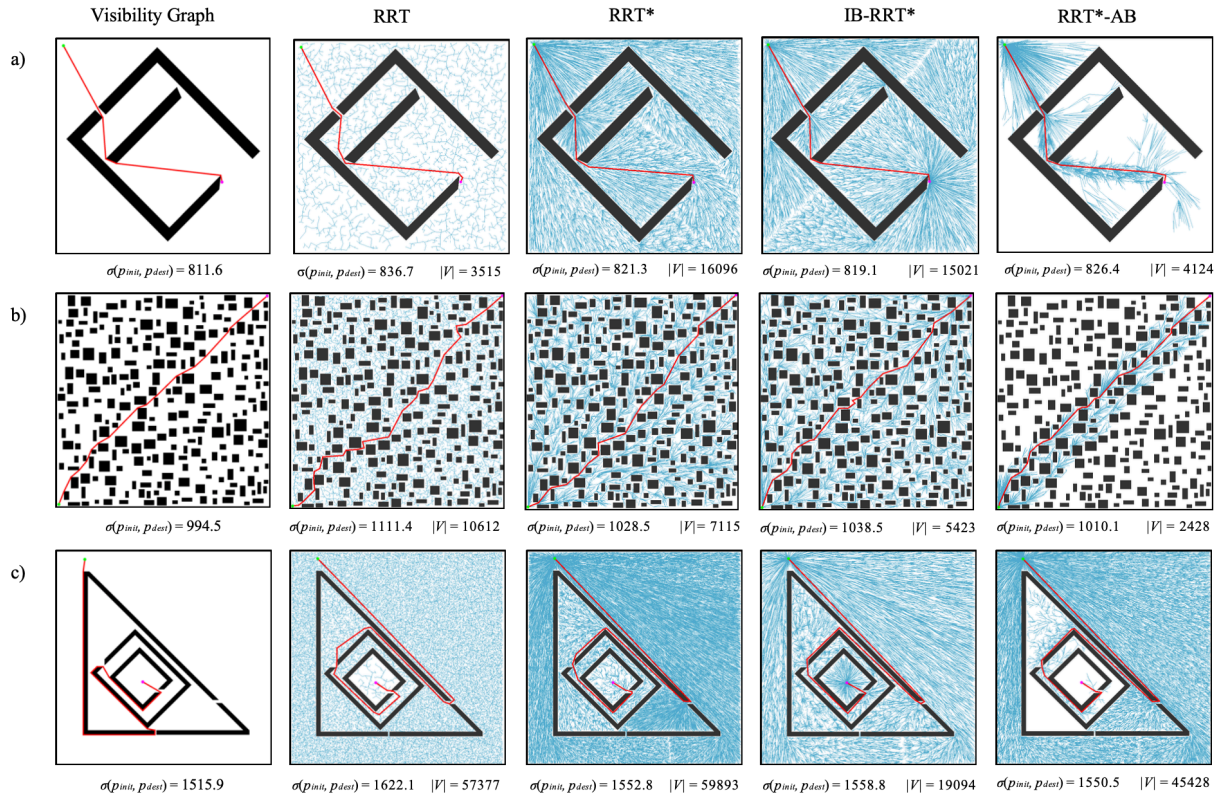


Figure 18. Best paths generated by RRT, RRT*, IB-RRT* and RRT*-AB in (a) E1 (b) E2 and (c) E3 from p_{init} (green) to p_{dest} (magenta)

5.4.2 Performance Comparison in 3D Space

Experiments were performed on a system with an Intel Core i5-2400 processor running at 3.10 GHz and 8 GB of RAM. For this comparative study, we focused on cluttered environments,

particularly since we were interested in applications requiring drone navigation in urban environments.

Several sets of experiments were performed, varying the number and sizes of the obstacles. A box B of size $X \times Y \times Z$ containing the starting point and destination was used as the test environment. In all the experiments, the starting location was one of the corners of the rectangular base R of B. The destination was the opposite corner along the diagonal of R.

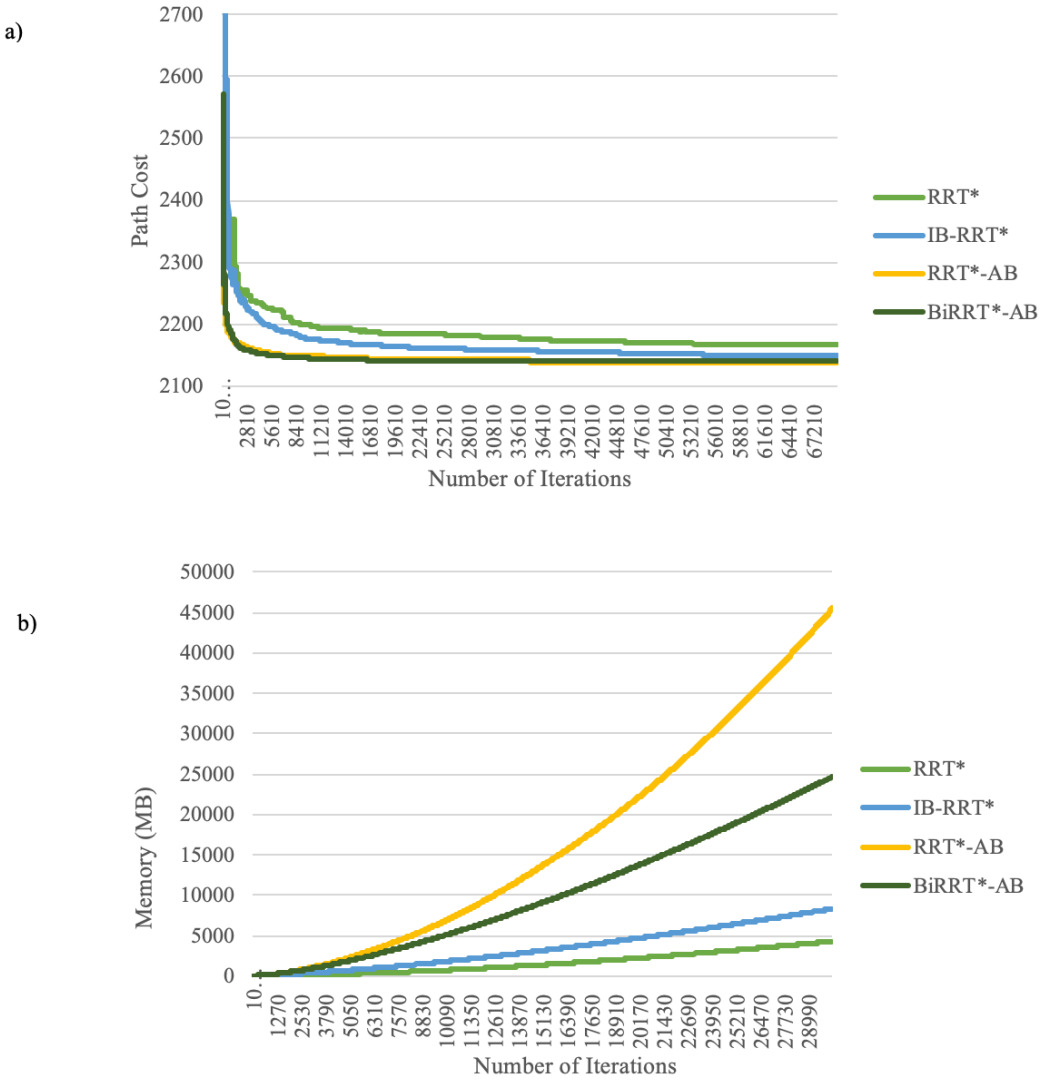


Figure 19. Graphs plotting (a) Path cost (b) Memory consumption as the number of iterations increase

For proper comparison, multiple parameters were tuned to ensure that the algorithms were performing at their best. Experiments were performed on 1500×1500×100 sized environments with 1000 and 550 obstacles. For each algorithm, the tree kept expanding for 70,000 iterations, and each experiment was repeated 50 times to compute the average rate of convergence. Similar to the termination condition used in 5.4.1, we wanted to determine the number of iterations required for each algorithm to generate a near-optimal solution. As shown in Figure 19, after a certain number of iterations, there are only minor improvements to the path cost; however, there is a significant increase in the running time and memory consumption. Therefore, we investigated the number of iterations required for each algorithm to generate a path within 1.5% of the best path generated by RRT*, IB-RRT*, RRT*-AB and BiRRT*-AB.

The effect of the value of γ was also investigated. As previously mentioned, the value of γ must be greater than $\left(2\left(1+\frac{1}{n}\right)\right)^{1/n}\left(\frac{\mu(X_{free})}{\zeta_n}\right)^{1/n}$ to ensure asymptotic optimality. Multiple factors v were applied to the equation mentioned earlier to test the algorithm's performance. When v = 0.5, even though it generated a solution very quickly, the path cost was 20% greater than the best solution generated.

Table 3. Effect of γ on generating near-optimal solutions for experiments with 1000 obstacles. The best overall results have been bolded

	Best Path Cost	*v*											
		1			2			3			4		
		# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)
RRT*	2157.6	-	-	-	**49340**	**26.01**	**9.49**	38380	35.87	12.02	30330	40.84	12.64
IB-RRT*	2148.8	46310	12.90	4.81	**14150**	**10.22**	**2.90**	14170	24.02	5.69	11680	36.25	7.08
RRT*-AB	**2137.5**	5920	7.15	2.22	2960	2.86	0.99	**1690**	**1.68**	**0.46**	1840	2.53	0.72
BiRRT*-AB	2138.7	**2380**	**0.63**	**0.31**	1680	1.03	0.44	1390	1.70	0.56	1290	2.90	0.74

Table 4. Effect of γ on generating near-optimal solutions for experiments with 550 obstacles. The best overall results have been bolded

	Best Path Cost	*v*											
		1			2			3			4		
		# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)	# Iterations	t (s)	Memory (GB)
RRT*	2128.2	-	-	-	7830	1.51	0.45	3490	0.89	0.22	**2500**	**0.85**	**0.19**
IB-RRT*	2126.7	23450	4.66	1.48	1180	0.43	0.09	**540**	**0.38**	**0.08**	580	0.65	0.11
RRT*-AB	**2125.9**	35020	33.45	10.47	2740	1.79	0.32	460	0.36	0.08	**400**	**0.36**	**0.07**
BiRRT*-AB	2126.0	650	0.16	0.06	**240**	**0.15**	**0.06**	180	0.19	0.06	200	0.28	0.07

Experiments were run with 1000 obstacles, and the number of iterations required to generate a solution that was 1.5% from the best path cost when v is 1, 2, 3 and 4 (Table 3). Generally, as the

value of v increased from 1 to 4, the number of iterations required to generate a near-optimum solution decreased; however, the running time and memory required increased. Interestingly, when a similar experiment was run with 550 obstacles, a near-optimal solution was generated with fewer iterations compared to the experiments with 1000 obstacles. The best result was generated with larger values of v (Table 4). Therefore, the ideal settings to generate near-optimal solutions are dependent on the number of obstacles in the environment.

The value of m was once again investigated for the RRT*-AB and BiRRT*-AB algorithms. Experiments were run in an environment with 1000 obstacles, and the value of m varied between 2 to 12, in intervals of 2. As the value increases from 2 to 12, the path cost improves, but the running time and the memory consumption increases as well. Table 5 shows the percent difference from $m = 2$. Since there was only a 0.45% improvement in the path cost for BiRRT*-AB when increasing m from 2 to 4, but an increase of 19.18% in running time and 17.80% in memory consumption, we set the value of m to 2 for BiRRT*-AB experiments. However, since there was over 1% improvement in the path cost when increasing m from 2 to 4 for RRT*-AB, the value of m was set to 4.

Table 5. Percent difference in the path cost, running time and memory as the value of *m* increases from 2 to 12

	m														
	4			6			8			10			12		
	Δ σ (%)	**Δ***t* (%)	**Δ** *memory* (%)	**Δ** σ (%)	**Δ***t* (%)	**Δ** *memory* (%)	**Δ** σ (%)	**Δ***t* (%)	**Δ** *memory* (%)	**Δ** σ (%)	**Δ***t* (%)	**Δ** *memory* (%)	**Δ** σ (%)	**Δ***t* (%)	**Δ** *memory* (%)
RRT*-AB	-1.12	+5.90	+4.79	-1.36	+12.24	+14.79	-1.51	+17.88	+22.31	-1.56	+20.26	+21.46	-1.61	+35.04	+41.24
BiRRT*-AB	-0.45	+19.18	+17.80	-0.67	+33.98	+32.91	-0.80	+42.63	+43.23	-0.89	+48.84	+50.26	-0.98	+59.10	+54.91

Multiple experiments were performed with 100, 250, 400, 550, 700, 850 and 1000 obstacles. For each number of obstacles, 50 different environments with randomly generated obstacles were generated. The width and the length of each obstacle was an integer randomly selected from the interval [5, 20] and the height was an integer randomly selected from the interval [1, 80]. The obstacles were randomly positioned in an environment B of size 1500 × 1500 × 100. Due to the random nature of the sampling-based algorithms, each experiment was repeated 50 times. For all the experiments in this study, to ensure that the value of η was large enough, η was set to l (p_{init}, p_{dest}) for all algorithms. The parameters used for the RRT* and RRT* variant algorithms are

presented in Table 6. In Table 6, x represents the number of iterations required to generate an initial path.

Table 6. Parameters for RRT* and RRT* Variant Algorithms

Parameters	RRT*	IB-RRT*	RRT*-AB	BiRRT*-AB
Number of Iterations	x + 50,000 samples	x + 15,000 samples	x + 2,000 samples	x + 2,000 samples
v	2	2	3	1
m	-	-	4	2

For comparative purposes, the RRT algorithm was also implemented with the `PrunePath` method. Since RRT does not have any cost function associated with the algorithm, the generated path will not be optimum. Therefore, experiments were run using the path pruning method mentioned in 5.3.4.

As expected, the RRT algorithm generated the best results in terms of memory consumption and running time for each experiment. As shown in Figure 20, although the RRT algorithm generated paths within 1% of the other algorithms when the path was pruned for the 100 obstacle experiments, the path lengths and the variance in the results increased as the number of obstacles increased.

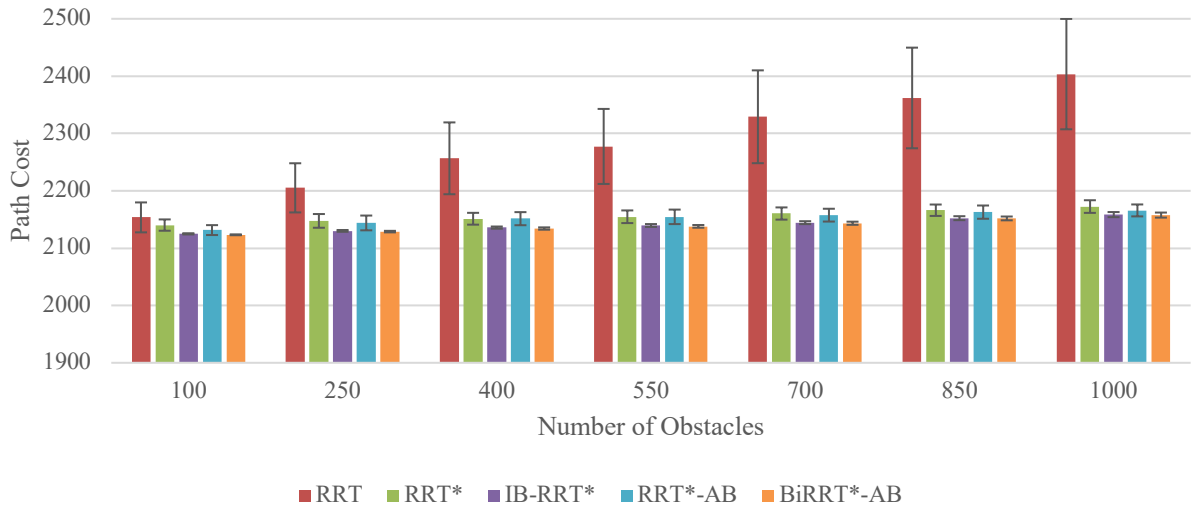


Figure 20. Comparison of the lengths of the paths generated by each algorithm

RRT* and IB-RRT* generated similar paths to RRT*-AB and BiRRT*-AB; however, the running time for IB-RRT* to generate these paths was more than five times greater than BiRRT*-AB for each experiment (Figure 21). Since the running time for RRT* was greater than 20 seconds for each experiment, the results have been omitted from Figure 21. Moreover, as shown in Figure 22, the IB-RRT* algorithm also required more than five times the amount of memory as BiRRT*-AB to generate a similar path. RRT* and IB-RRT* require more time and memory to generate a path because the termination condition was set so that they needed to complete more number of iterations than RRT*-AB and BiRRT*-AB to converge to a similar solution. Since RRT*-AB and BiRRT*-AB limit the search space and utilize the path pruning method, the algorithms can effectively reduce the length of the path rapidly.

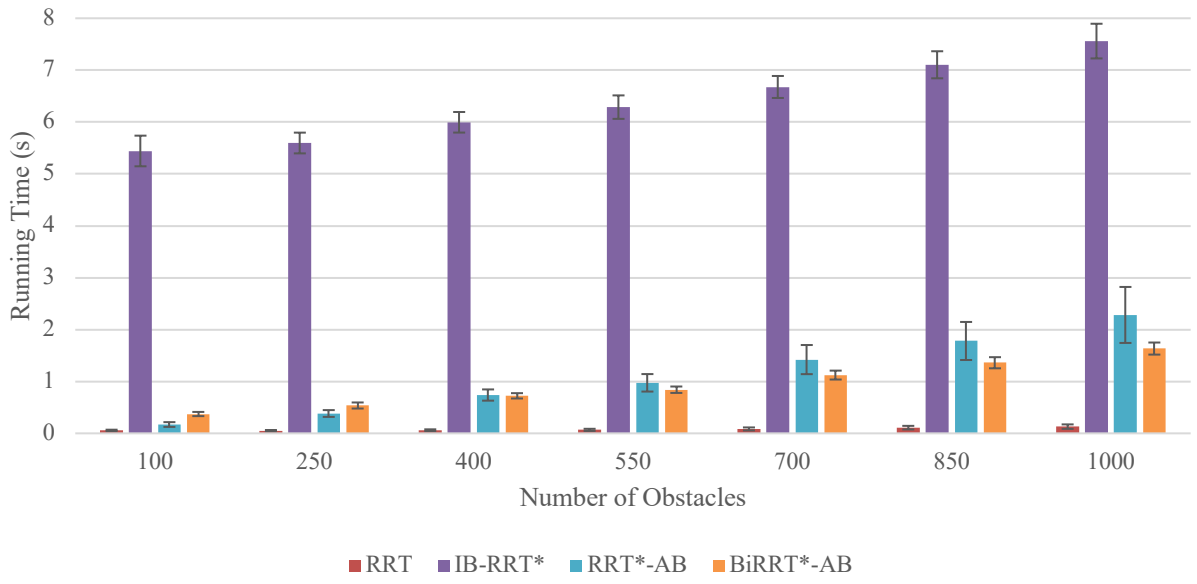


Figure 21. Comparison of the running times for each algorithm

For the BiRRT*-AB algorithm, there was less than 1% improvement in all experiments' path cost compared to RRT*-AB. However, in the experiments with more than 400 obstacles, there was (on average) a 30% improvement in the running time. This may result from a smaller value required for v, meaning that the near neighbour radius around each sampled point is less than RRT*-AB. Moreover, RRT*-AB must grow its tree from p_{init} to p_{dest}, whereas in BiRRT*-AB, the tree grows from both p_{init} to p_{dest}; therefore, generating a path very rapidly.

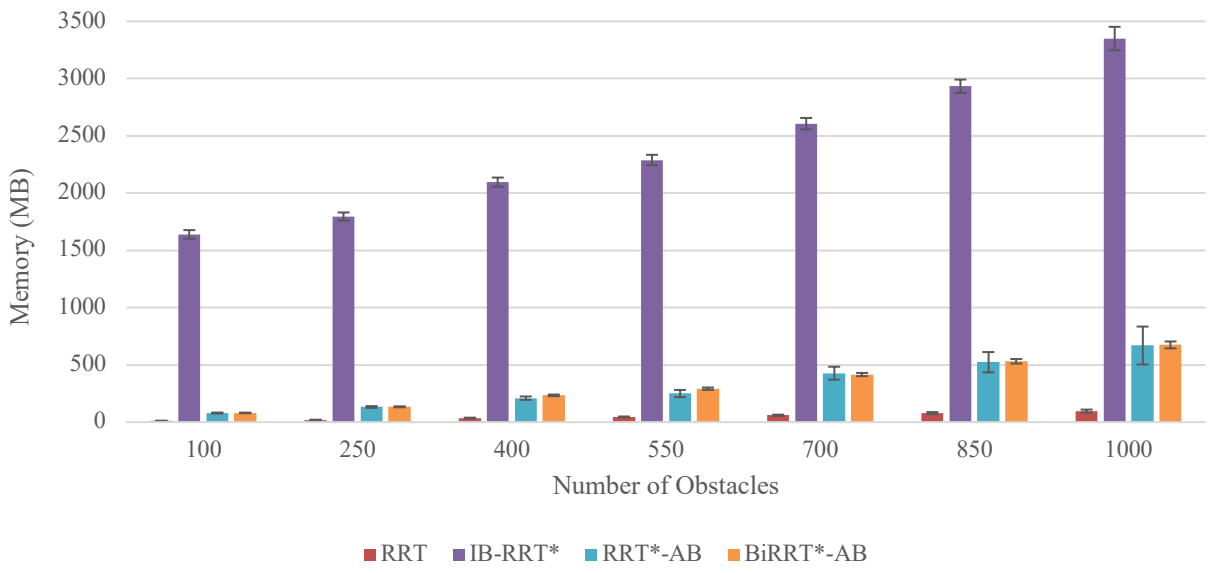


Figure 22. Comparison of the memory usage by each algorithm

Compared to RRT*-AB, the BiRRT*-AB algorithm generates more consistent results. As shown in Figures 20-22, in each experiment, the variance in path cost, running time and memory usage is much lower than RRT*-AB. On average, there was a ±11.24 difference in the path length generated by the RRT*-AB algorithm for each run; however, there was only a ± 2.42 difference between each path generated by the BiRRT*-AB algorithm. Moreover, there was a ± 52.72 MB difference in memory usage on average for each experiment, whereas a ± 12.66 MB difference in memory usage for BiRRT*-AB.

Developing a computationally inexpensive, efficient and robust path planner for drones can be challenging. In this chapter, we investigate the performance of the RRT, RRT*, IB-RRT*, and RRT*-AB in two-dimensional space to determine their robustness in different types of environments. The study is extended in three-dimensional space to determine the performance of these algorithms in addition to the hybrid BiRRT*-AB algorithm in cluttered environments. Results show that different algorithms perform best in different types of environments. IB-RRT* performs well in simple and complex environments, whereas RRT*-AB performs the best in cluttered environments. In three-dimensional space, the BiRRT*-AB algorithm reliably generates better paths much faster than the RRT*-AB algorithm in cluttered environments with many obstacles present.

Chapter 6

6 Energy-Efficient Path Planning Algorithm

In this chapter, we introduce an offline 3D path planning algorithm for drones. The main objective is to develop an algorithm that computes near-optimal paths without extensive computational power and memory requirements. Graph-based path planning algorithms are generally considered impractical for solving the 3D drone path planning problem due to their large memory requirement and high time complexity [65]. However, in this chapter, we show that the proposed algorithm can overcome the limitations of graph-based approaches. The notion of a layered graph is introduced to represent the workspace accurately. Using the layered graph in combination with the shortest path algorithm, a collision-free path that minimizes the energy required for a drone to fly from a starting point to the destination point can be generated.

6.1 Graph Construction

For details regarding notations and the problem description, please refer to Chapter 5.1. Each obstacle in O is identified by four points and two values: (x_1, y_1), (x_2, y_2), (x_3, y_3) and (x_4, y_4) represent the coordinates of the base of the obstacle projected on the xy plane; h_a and h_b are the two values that specify the z coordinate of the base and the top of the obstacle, respectively. Let $C = \{(x_{p_{init}}, y_{p_{init}}), (x_{p_{dest}}, y_{p_{dest}}), (x_1, y_1), (x_2, y_2), \dots, (x_r, y_r)\}$ be projections on the xy plane of p_{init}, p_{dest}, and the corners of all obstacles. Let $h_0 \leq h_1 \dots \leq h_t$ be the sorted set of different z coordinates for p_{init}, p_{dest}, and the bottoms and tops of all obstacles.

The environment is represented as a *layered graph* $\hat{G} = (\hat{V}, \hat{E})$. A layered graph is composed of several subgraphs, each representing the obstacles' positions at different heights. We assume that the bottom of the region X is flat at a height of zero. Consider the planes P_0, P_1, P_2, … , P_{t+1}, parallel to the xy plane at heights h_0, h_1, h_2, …, h_t, Mh_t, where M is a value greater than 1.

The layered graph consists of $t + 2$ subgraphs $G_0, G_1, G_2, \dots, G_t, G_{t+1}$. To build graph $G_i = (V_i, E_i)$, we create a grid g_i on plane P_i by tracing a horizontal and vertical line on P_i crossing at (x_j, y_j, h_i) for each point (x_j, y_j) of C. A node is added to V_i for each point where two lines of g_i cross (Figure 23).

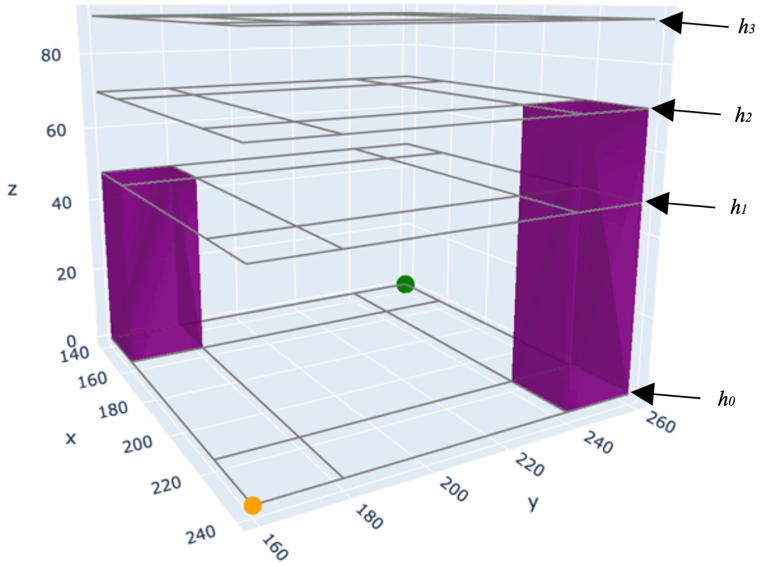


Figure 23. Environmental Representation (Layered Graph)

As for E_i, consider a grid rectangle r formed by two adjacent horizontal lines and two adjacent vertical lines of grid g_i. Let u_1, u_2, u_3, u_4 be the nodes corresponding to the vertices of r. We add an edge between any pair of these vertices u_j, u_k if a line connecting the grid's points corresponding to u_j and u_k do not intersect with any obstacles. $\hat{E}$ also includes edges between adjacent subgraphs. Let r' be the rectangle of grid g_{i+1} on P_{i+1} corresponding to r, given that $i < Mh_t$. Let r'' be a rectangle of the grid g_{i-1} on plane P_{i-1} corresponding to r, given that $i > 0$. As shown in Figure 24, for every node u corresponding to the vertices of r and every node v corresponding to the vertices of r' and r'' an edge is inserted in the layered graph and subgraph if the line connecting the points corresponding to u and v do not intersect with any obstacles.

Notice that we can build a layered graph that accurately models all kinds of environments – even ones with bridges or other structures that allow the drone to navigate under the objects.

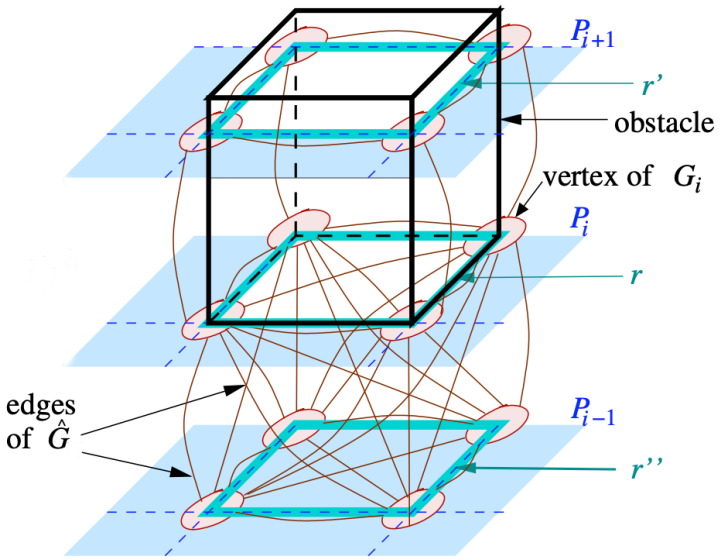


Figure 24. Edges of the layered graph between vertices corresponding to the rectangles *r*, *r′* and *r″* [62]

A path from p_{init} to p_{dest} is specified by the coordinates of the points corresponding to the nodes of the layered graph that the path visits:

$$(x_{p_{init}}, y_{p_{init}}, h_{p_{init}}), (x_1, y_1, h_1), (x_2, y_2, h_2), \dots, (x_{k-1}, y_{k-1}, h_{k-1}), (x_{p_{dest}}, y_{p_{dest}}, h_{p_{dest}})$$

where $(x_{p_{init}}, y_{p_{init}}, h_{p_{init}})$ and $(x_{p_{dest}}, y_{p_{dest}}, h_{p_{dest}})$ are the coordinates of the initial point and the destination point, respectively. Alternatively, the path ϕ can be specified by line segments. Each pair of adjacent nodes in the path defines a segment s and the drone navigates directly from the first endpoint of a segment to its second endpoint. Therefore, $\phi = s_1, s_2, \dots, s_k$.

The proposed algorithm computes a path from the starting point p_{init} to the destination point p_{dest} in two stages. First, a modified A* algorithm is used to compute the minimum path cost in the layered graph from the node corresponding to p_{init} to the node corresponding to p_{dest}. Notice that the minimum path cost in the layered graph between the nodes corresponding to p_{init} and p_{dest} is not necessarily a minimum path cost between p_{init} and p_{dest}. This is because the optimal path may not traverse through the vertices in the layered graph. Therefore, in the second stage of the algorithm, the pruned path adds shortcuts to the path computed in the first stage and reduces its cost.

For each node u explored in the layered graph, the cost of reaching node u from the starting node to u and the length of the last segment in the path are stored. The square of the Euclidean distance from u to the destination was used as the heuristic function in the implemented A* algorithm to decide which order to explore the nodes of the layered graph.

As previously mentioned, in the second stage of the path planning algorithm, shortcuts are added to the path computed in the first stage to reduce its cost. The path pruning method utilized in this algorithm is described in detail in 5.3.4. The path pruning algorithm repeatedly tries to add shortcuts between adjacent segments until no additional shortcuts can be added that reduce the cost of the path.

6.2 Experimental Results

We performed a series of experiments on environments with a set of randomly generated and randomly positioned obstacles to compare the proposed algorithm's performance with the BiRRT*-AB algorithm. To evaluate the performance of the algorithms, we compared the length of the paths computed, the amount of memory utilized and the running time.

All experiments were performed on a computer with an Intel Core i7-8750H processor running at 2.20 GHz and 16 GB of RAM. The programs were implemented in Java, and they were run using Java SE 14.0.1. Several sets of experiments were performed, varying the number and sizes of the obstacles. Similar to the experiments performed in 5.4.2, a box B of size $X \times Y \times Z$ containing the starting point and destination was used as the environment. In all the experiments, the starting location was one of the corners of the rectangular base R of B. The destination was the opposite corner along the diagonal of R.

For each experiment, the objects were generated as follows. Each obstacle's length and width were chosen randomly from an interval $[b_{min}, b_{max}]$, and the height of each obstacle was selected randomly from an interval $[h_{min}, h_{max}]$. The bases of all obstacles were placed at height 0. For each obstacle, the coordinates (x, y) of one of the corners of its base were selected randomly within the rectangular base R of B. If placing an obstacle in the selected location would cause it to overlap with another obstacle, then a new position for the corner was chosen randomly. This process was repeated until a position was found for the obstacle that did not cause overlaps.

The parameters used for BiRRT*-AB in section 5.4.2 were used for this set of experiments since the algorithm was evaluated in a similar environment with the same number of obstacles. For consistency, a layered graph was constructed, and in each iteration of the BiRRT*-AB algorithm, a node from the layered graph was chosen. For each experiment, we evaluated the proposed algorithm's performance when there were five different obstacle heights.

For the first set of experiments, 100, 250, 400, 550, 700, 850 and 1000 different obstacles were used. For each number of obstacles, 100 different environments with randomly placed obstacles were generated. Each obstacle's width was an integer selected randomly from the interval [5,20], and the height was an integer randomly selected from the interval [1,80]. The obstacles were randomly positioned in an environment B of size $1500 \times 1500 \times 100$. Due to the randomized nature of the BiRRT*-AB, each experiment was repeated 50 times.

Since computing a minimum path between two points in 3D space is NP-hard, we computed visibility graphs and applied Dijkstra's algorithm to compute near-optimal paths. Since constructing a visibility graph with each node from the layered graph was very complex, only the eight vertices of each polyhedral obstacle were inserted into the visibility graph. Constructing the visibility graph was a time-consuming process in large environments, even when attempting to build the visibility graph by performing collision detection operations in parallel.

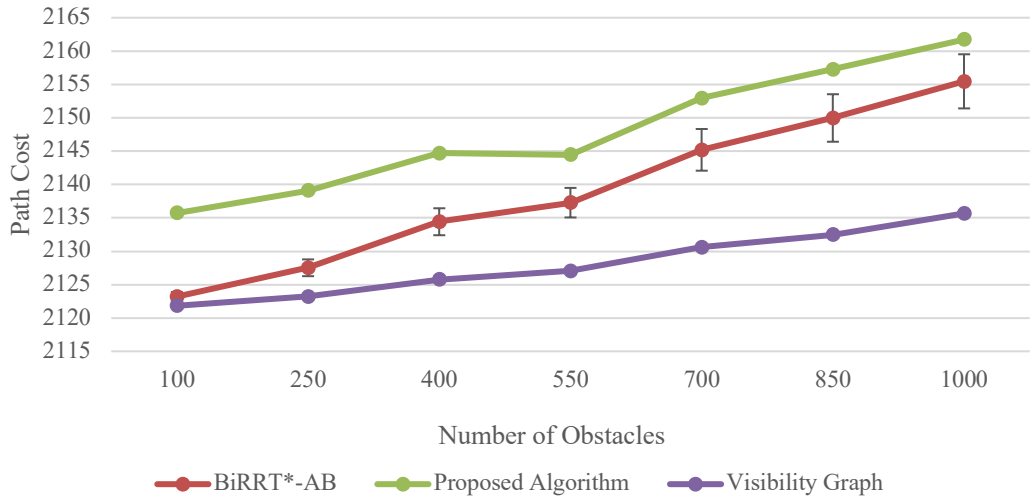


Figure 25. Comparison of the length of the path

Figure 25 compares the length of the paths computed by BiRRT*-AB, the proposed algorithm and the path computed by constructing a visibility graph. Both the BiRRT*-AB and the proposed algorithm generate paths within 1% of those computed by the visibility graph. Although the BiRRT*-AB generated a solution very close to the one computed by using visibility graphs for 100 obstacles, as the number of obstacles increased, the difference between the near-optimal solution generated by the visibility graph and the solution generated by BiRRT*-AB grew. The proposed algorithm always generated paths that were marginally greater than the BiRRT*-AB.

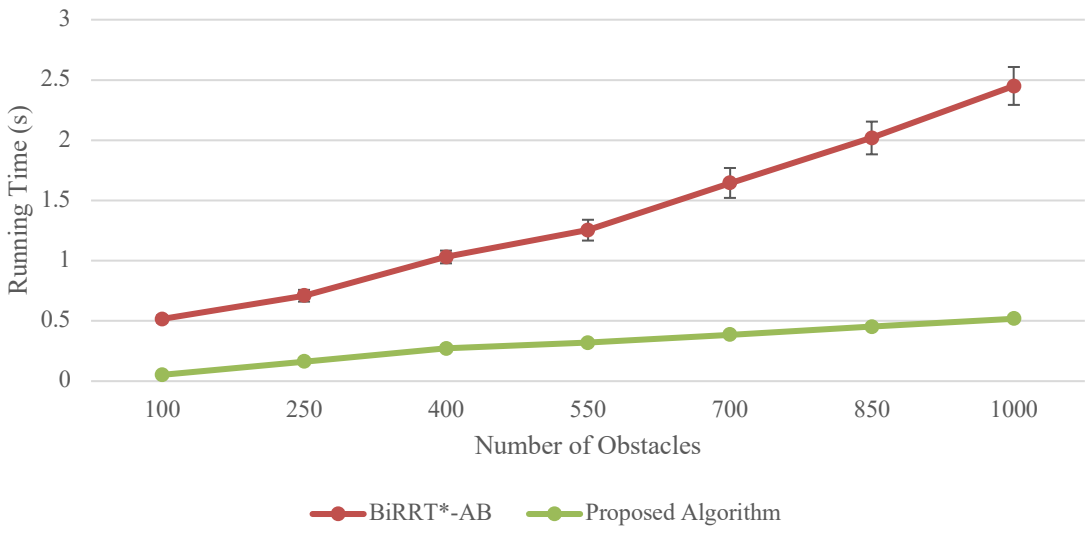


Figure 26. Comparison of the running time of the algorithms

Figure 26 shows that the proposed algorithm is more than two times faster than the BiRRT*-AB algorithm. This is partly due to the heuristic used in our implementation of the A* algorithm that allows it to expand and find a path with a short length quickly. The running time of the BiRRT*-AB algorithm seems to grow nearly linearly with the number of obstacles and at a faster rate than the running time of the proposed algorithm. As the number of obstacles grows, there are more nodes in the layered graph; thus, there are much more near neighbours for each sampled point and more rewiring operations, which may explain this increase in running time.

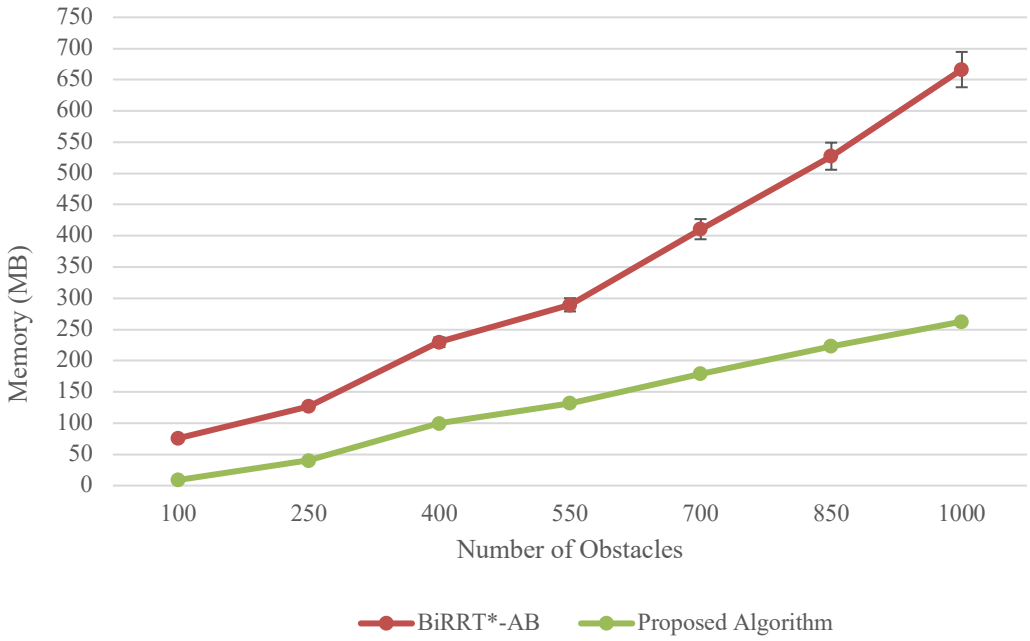


Figure 27. Comparison of the memory usage for each algorithm

Finally, Figure 27 shows that the proposed algorithm uses less than one-third of the amount of memory used by the BiRRT*-AB algorithm. This shows the effectiveness of the layered graph. Even in cluttered environments with many compact obstacles next to one another, the layered graph can accurately model the environment without consuming as much memory.

Another set of experiments were performed by placing many more obstacles in the simulated environment. Twenty different environments with 2,000-10,000 randomly generated obstacles were placed in the environment. However, for these experiments, the size of the environments varied between 1500 × 1500 × 100, 3000 × 3000 × 100, 4000 × 4000 × 100, and 5500 × 5500 × 100. Since the BiRRT*-AB requires a lot of time to generate a path, each experiment was repeated 20 times. The results for these experiments have been summarized in Table 7. Although the BiRRT*-AB and the proposed algorithm generate similar paths, the BiRRT*-AB required significantly more memory and time to compute the paths as the number of obstacles increased from 2,000 to 10,000.

Table 7. Comparison of the algorithms in an environment with many obstacles. Standard deviations are in parentheses.

Number of Obstacles	Running Time (s)		Path Length		Memory (MB)	
	BiRRT*-AB	Proposed Algorithm	BiRRT*-AB	Proposed Algorithm	BiRRT*-AB	Proposed Algorithm
2000	5.08 (0.46)	0.77 (0.03)	2209.58 (8.26)	2180.78 (8.88)	1505.02 (78.36)	359.24 (3.46)
4000	9.94 (0.82)	1.90 (0.02)	4308.97 (7.48)	4279.63 (9.61)	2684.69 (106.71)	140.67 (10.16)
6000	18.31 (1.68)	2.96 (0.05)	5717.38 (7.77)	5699.14 (12.57)	4165.79 (163.76)	258.28 (10.38)
8000	36.42 (3.96)	3.60 (0.07)	5750.33 (12.13)	5711.53 (15.04)	6012.87 (382.34)	276.04 (6.88)
10000	40.84 (4.64)	5.54 (0.14)	7847.80 (9.96)	7813.73 (8.97)	7678.98 (346.17)	512.16 (12.56)

6.3 Cost Function

The cost between p_a and p_b defined as $\sigma(p_a, p_b)$ is not necessarily the Euclidean distance between the two points. Although we focused primarily on the length of the path in this study, in the future, we intend on investigating other possible factors that negatively impact the battery performance in drones. Particularly, the number of turns or number of times the drone is required to change its direction of flight and the number and magnitude of changes in the altitude during the entire path. Studies have shown that a decrease in air density results in an increase in power consumed at higher altitudes [70].

We denote the length of a segment s_i as $l(s_i)$, and the difference in the height between two endpoints of s_i as $\Delta h(s_i)$. The angle between the segments s_i and s_{i+1} is denoted as $\theta(s_i, s_{i+1})$. This represents the change in the direction of the flight path so that the drone can navigate along s_{i+1} once it has travelled through segment s_i. The cost function that our algorithm will use to compute a path p from p_{init} to p_{dest} is the following:

$$\sigma(p_a, p_b) = \sum_{s_i \in p} [(l(s_i)w_1 + \Delta h(s_i)w_2 + \theta(s_i, s_{i+1})w_3] \tag{5.3}$$

where $\theta(s_i, s_{i+1}) = 0$ for the last segment s_k of p. The coefficients w_1, w_2, and w_3 are constants that vary depending on the drone. w_1 is the energy used by the drone to navigate a distance of one unit, where a unit can be measured in kilometres or meters. w_2 is the energy used to change a drone's altitude by one unit, where units can be measured in meters. Finally, w_3 is the energy used by the drone to make a turn of one unit of magnitude. A turn can be defined in degrees or radians. A pre-specified threshold can be defined to ignore minor changes in direction that do not greatly impact the drone's battery performance.

Chapter 7

7 Cell-Based Model for Drone Delivery

Utilizing drones can be cost-effective when incorporated in the planning of last-mile deliveries. Delivering a 2 kg package within a 10 km radius in America by ground transport costs $2-8, compared to just $0.10 using a drone [64]. Compared to traditional truck deliveries, drones also benefit from having lost cost maintenance, less required labour and a high-efficiency rate [79]. Drones are also beneficial for the environment since they can reduce carbon dioxide emissions compared with trucks [81]. This chapter introduces a cell-based design that enables a network of autonomous drones to work together to travel long distances cooperatively. Furthermore, we design a scheduling algorithm using evolutionary approaches to minimize the time it takes to complete a set of jobs.

7.1 Drone Delivery Model

As mentioned in Chapter 2, drones' limited flight duration (as a result of battery capacity constraints) is a significant drawback when using drones for commercial applications. A single drone cannot be used to complete jobs that require navigating long distances. A potential solution would be to strategically place drone charging stations around the mission's region so that a drone can travel further after replenishing its battery. In this study, this idea is extended to a network of drones. The model has been designed so that drones unable to reach the intended destination point can hand off the necessary information required to complete the mission to another drone located at the charging station with sufficient battery capacity. In this study, we focus on delivery applications where a job represents a delivery mission. Drones are used to pick up a package from a specified location (depot) and cooperatively hand off the delivery package to one another in order to reach the destination point.

In the proposed model (Figure 28), the entire region is divided into a fixed number of cells H. In this study, we divide the region into regular hexagonal cells. Dividing the region into cells is in itself a problem that needs to be investigated. Many factors, such as network coverage and the minimum size of each cell, need to be considered. In this study, each cell is designed so that the drone has enough battery capacity to travel around its perimeter on a single charge. This ensures

that the drone can navigate to any location within a cell and safely return to a charging station without completely draining its battery. Each hexagon's vertices represent a shared charging station (SCS), which is capable of simultaneously charging *k* drones at any given time.

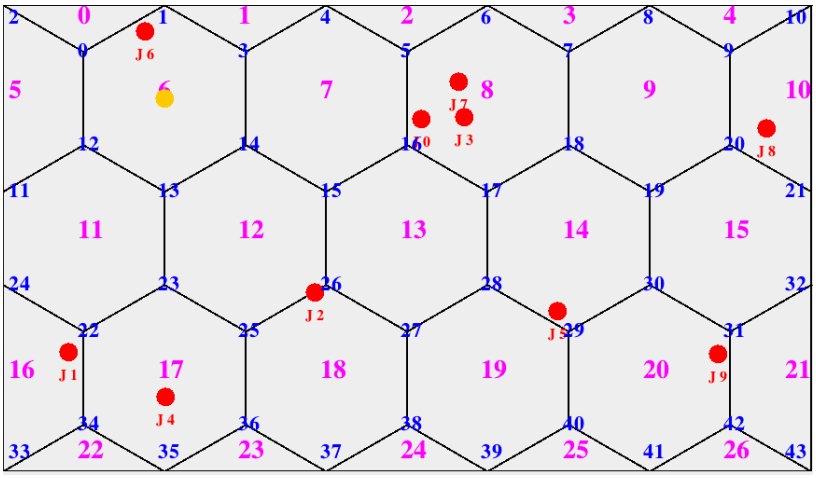


Figure 28. Cell-Based Delivery Model depicting the depot (yellow), cell number (magenta), SCS number (blue) and job destination locations (red)

Apart from the benefit of extending the flight range, there are many other benefits of using this model with a network of drones. Compared to a single drone operation, the proposed model increases the reliability of using drones to complete jobs. In the event of a single point failure where a drone cannot navigate any further, there are many nearby drones that can potentially take over and complete the job. Moreover, since each drone at an SCS is required to navigate within a limited region, computation of paths can be done rapidly using fewer resources because each drone does not need to store information related to the entire delivery region. As shown in Chapter 6, increasing the number of obstacles in the delivery region results in an increase in the memory consumption and running time of graph-based algorithms and sampling-based algorithms. Therefore, when limiting the search space, a path can be computed very rapidly, making it ideal for situations where the path must be re-computed frequently due to dynamic obstacles being introduced.

Although this model has been developed for delivery applications in this study, there are many different applications where this type of model could be used to complete tasks. For example, this model could benefit first responders. Since there are multiple drones located around a region,

an available drone close to an incident can be rapidly deployed. This model can also benefit search and rescue teams. Drones at each SCS can cooperatively search designated cells in parallel without needing to swap or recharge their battery. Therefore, search and rescue operations can be completed much quicker than it would take for human search teams.

7.2 Multi-Drone Job Scheduling Problem (MDJSP)

This section introduces the Multi-Drone Job Scheduling Problem (MDJSP), which employs the model described above to complete a set of jobs.

7.2.1 Problem Formulation

Provided a set of n jobs $J = \{J_1, J_2, \ldots, J_n\}$ and h drones $D = \{D_1, D_2, \ldots, D_h\}$ the objective is to minimize F, which represent the maximal time it takes all drones to complete all the jobs. Each job can consist of multiple operations $O_j = \{O_{j,1}, O_{j,2}, \ldots O_{j,l}\}$, where each operation a of a job j (denoted by $O_{j,a}$) must be scheduled in order. In this study, an operation refers to one of the following: assigning a drone to pick up a package from the depot station to a specified location (destination location or an SCS), assigning a drone to move a package from one SCS to another, or assigning a drone to navigate from an SCS to the destination location. The MDJSP can be decomposed into two sub-problems. The first problem is assigning each operation to a drone. The second one is to schedule the operations to minimize F. Each drone at an SCS has a predefined flight time C and recharge time R to ensure that the drones are capable of completing each operation that they are assigned.

The following assumptions were made when solving the problem:

1) When a drone begins an operation, it will reach its destination without any interruptions
2) Each drone can only perform one operation at a time
3) The time it takes to load a package onto the drone at the depot and the time it takes to transfer a package from one drone to another drone is negligible
4) All jobs have the same priority level
5) Each drone must return to the SCS that it originated from

6) The time required to complete an operation is equivalent to the distance travelled by the drone (i.e. the drone is travelling at a constant speed)

7) Each drone's battery recharges at a constant rate

7.2.2 Genetic Algorithm (GA)

Scheduling jobs to available machines is an optimization problem that is NP-hard. Therefore, meta-heuristics such as the genetic algorithm and particle swarm algorithm can be used to generate a schedule in a feasible amount of time. In general, there are three steps in GA – selection, crossover and mutation. In each generation, a set of individual solutions are selected from a population so that crossover and mutation operations can be applied. Crossover and mutation operations help to diversify the population to prevent getting trapped in local minima.

Individual solutions (chromosomes) of a feasible schedule are represented (encoded) as a sequence of operations. This represents the order in which the operations are inserted into the schedule. The initial population was initialized by randomly generating a set of chromosomes. Since the sequence of operations must follow a precedence constraint, any sequence of operations that did not satisfy the constraint was repaired before applying any evolutionary operations. An additional chromosome was inserted into the initial population where the order of the sequence of operations in the solution corresponded to the order of the furthest job from the depot. A further explanation for this reasoning is provided in 7.2.3. Since the initial population can impact the convergence rate of the GA, we decided to include an additional chromosome with a solution that was closer to the optimal solution.

The fitness $f(i)$ of each individual indicates the quality of each solution. For this study, the fitness is calculated as follows:

$$f(i) = 1/F_i$$

where F_i is the maximal time it takes to complete all operations in schedule i. Therefore, the genetic algorithm favours solutions with greater fitness values. To calculate $F_{i,}$ each operation was assigned to a drone using the algorithm presented in Algorithm 9.

Algorithm 9: ScheduleOperation (o, j, a, b)

Input: Operation number o, Job number j, Pickup Location a (if exists), Drop off Location b (if exists)

```
if o s the first operation or last operation for j then
    X ← Set of SCS in the cell containing the depot (if o is the first job) or destination location (if o is the last job)
else
    X_a ← Set of SCS in the cells containing a
    X_b ← Set of SCS in the cells containing b
    X ← X_a ∩ X_b
bestDrone ← ∞
bestTime ← ∞
for x ∈ X do
    for each drone d charging in x do
        t ← find the earliest time d can start the operation
        if d has enough battery at time t to complete the operation then
            currentFinishTime ← t + TimeToCompleteOperation(o)
        else
            currentFinishTime ← t + getRequiredRechargeTime() + TimeToCompleteOperation(o)
        if currentFinishTime < bestTime then
            bestDrone ← d
Assign o to bestDrone
```

Note that the processing time for each operation in the schedule is dependent on the drone it chooses to complete the operation. In this study, as per assumption 7, the `TimeToCompleteOperation` method uses the Euclidean distance to estimate the units of time required to complete an operation, where a unit of time can be measured in any time unit such as minutes. As shown in Algorithm 9, individual operations are attempted to be scheduled for every drone at an SCS. In situations where completing an operation would exceed the predefined flight time *C,* the `getRequiredRechargeTime` method calculates the amount of time the drone must wait at its SCS and recharge its battery to generate sufficient energy to complete the operation.

In each iteration (generation) of the GA, parents are selected for recombination to create individuals for the next generation. Parent selection is critical since the selection of extremely fit solutions in each generation can result in poor diversity in the generated solutions, resulting in premature convergence. In our GA implementation, we used the roulette wheel selection method to select parents in each generation. In this method, every individual has the potential to be chosen as a parent, but the probability at which it is selected is proportional to its fitness. Therefore, solutions with a larger fitness value have a greater probability of being chosen as a parent for the next generation. Moreover, the GA has *elite chromosomes,* which represent the number of chromosomes that are selected to be a part of the population in the next generation.

Crossover (recombination) is when two parent chromosomes or individuals are modified to generate a new child (individual). This is a necessary step to ensure that fitter solutions can be generated. Combining the order of operations from each parent may result in a much better schedule than its parents. In this study, a random position in the sequence of operations is chosen, and a random number of operations are selected from the first parent to be moved directly to the offspring. The rest of the operations are then added in the order from the second parent into the offspring. The crossover rate in the GA is the probability of recombination occurring in each generation.

Finally, mutation is another process added in the GA to prevent premature convergence. Within each chromosome, two operations are randomly swapped (without violating the order constraints of operations) to create a new chromosome for the next generation. Although swapping an operation may result in a poorer solution, it is required to introduce diversity in the population. The mutation rate, which ranges between 0 and 1, determines the probability of which a mutation operation occurs for each chromosome in the population.

7.2.3 Experimental Results

Experiments were performed scheduling 10, 50, 100 and 200 jobs for drones. Each experiment had jobs randomly placed in a 45 × 40 environment such that the job locations do not correspond to the depot location or any SCS locations. Each experiment had 37 different hexagonal cells, with 65 SCS in total. Due to the random nature of the GA, each experiment was repeated 50 times.

For comparison purposes, the first come first serve (FCFS) algorithm was implemented to depict the completion time generated by scheduling operations in the order that they are given to the program. As expected, Table 10 shows that the FCFS algorithm does not necessarily generate a near-optimum solution. The other algorithm used for comparison is the furthest job first (FJF). As shown in Figure 29, the furthest jobs away from the depot usually require multiple handoffs to reach their destination, consequently requiring a lot of time to complete. Therefore, if such jobs are scheduled before jobs that require less time to complete, then the overall completion time of the jobs can potentially be better than the FCFS approach.

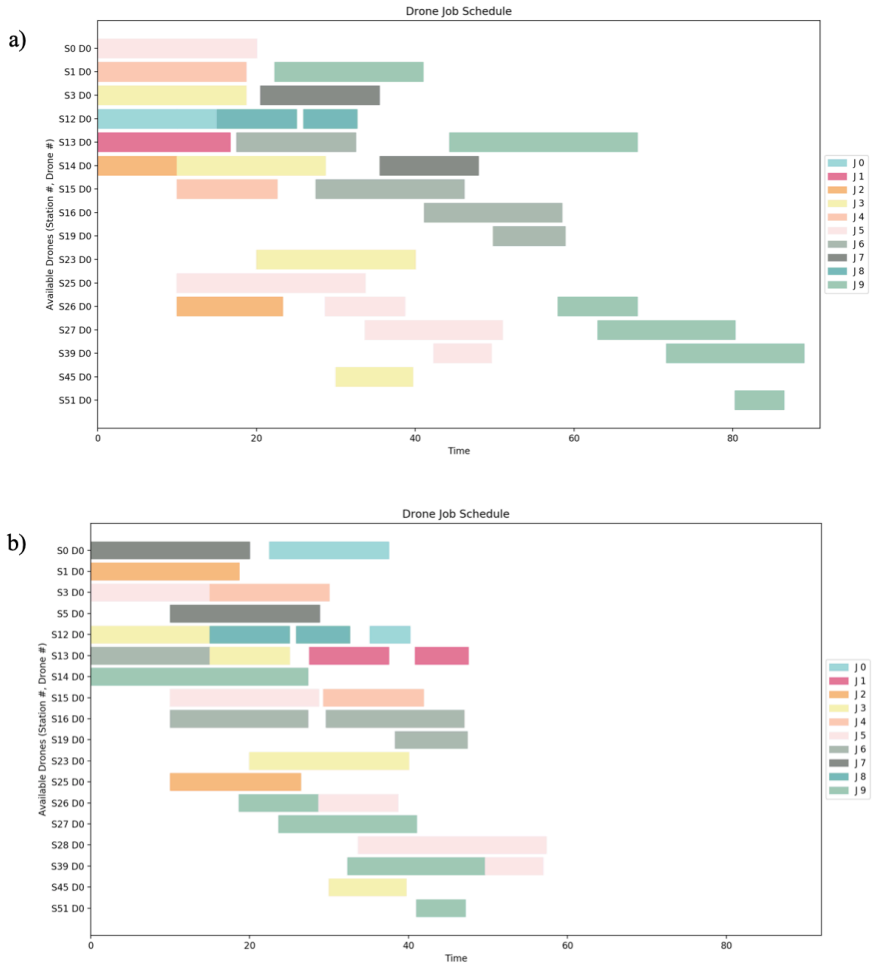


Figure 29. Schedule for ten jobs using (a) FCFS and (b) FJF method

Finally, another well-known evolutionary algorithm, Particle Swarm Optimization (PSO), was implemented to compare the performance of our implementation of the GA for scheduling drone delivery jobs. In each generation of the PSO algorithm, new particles are produced based on the previous generation's velocity and position.

The parameter settings for the GA and PSO used in these experiments are shown in Table 8 and Table 9. A sensitivity analysis was performed to ensure that these values would generate the best results for each algorithm. A summary of the experiments performed to generate these settings has been presented in Tables 11 and 12 in the Appendices.

Table 8. Parameters for the Genetic Algorithm

Parameters	Values
Number of Generations	600
Number of Chromosomes	150
Mutation Rate	0.01
Crossover Rate	0.9
Elite Chromosomes	60

Table 9. Parameters for the Particle Swarm Optimization Algorithm

Parameters	Values
Number of Iterations	400
Number of Particles	60
w_{max}	0.9
w_{min}	0.5
c_1	2
c_2	2

Table 10 shows the results generated by scheduling 10-200 jobs using FCFS, FJF, PSO and GA. The number of drones k at each SCS varied between 1 and 15 drones to evaluate the improvement in completion time by adding additional drones to each SCS. Although FJF generated comparable schedules to FCFS when k was 1, when increasing the number of drones in each SCS from 1 to 15, there was a 32% decrease (on average) in the completion time. For the experiments with fewer jobs (10 and 50 jobs), when more drones were available at each SCS, the FJF generated near-optimal solutions (similar to PSO and GA).

Our implementation of the PSO always generated better solutions than the FCFS method. However, compared to FJF, the PSO sometimes generated worse solutions. This may be due to the premature convergence to a solution. GA always generated a solution that was better than the FJF method. This is because the initial population always had one chromosome with operations ordered with the furthest job first.

As Table 10 shows, GA always generated schedules with a better completion time than FCFS, FJF and PSO. As expected, compared to FCFS, GA always generated a schedule that was more than 42% better. For each experiment, even though the PSO and GA required a similar amount of time to generate a schedule, GA generated solutions that were on average 10% better than PSO.

Table 10. Results when there are *k* drones at each station

	FCFS			FJF			Particle Swarm Optimization			Genetic Algorithm		
Number of Jobs	***k* = 1**	***k* = 5**	***k* = 15**	***k* = 1**	***k* = 5**	***k* = 15**	***k* = 1**	***k* = 5**	***k* = 15**	***k* = 1**	***k* = 5**	***k* = 15**
10	76.88	48.86	48.58	63.61	48.58	48.58	58.29	48.58	48.58	56.60	48.58	48.58
50	303.85	89.16	58.75	281.22	61.40	54.77	208.96	65.48	54.77	179.64	56.51	54.77
100	579.98	149.63	73.02	570.43	114.79	55.37	398.89	101.86	57.17	352.30	82.28	55.23
200	1154.59	274.11	115.41	1151.33	240.82	83.19	808.10	177.86	82.20	734.09	150.03	68.62

These experiments show that GA is effective at scheduling many jobs. Moreover, these results support the idea that drones can be effectively used to deliver many packages. Another observation was that the time required to complete each set of jobs decreases significantly when increasing the number of drones at each SCS. When there are 15 drones at each station, it requires roughly one-tenth of the amount of time it takes when there is only one drone at each station.

Chapter 8

8 Conclusions

The convenience and economic benefits of drones have resulted in many companies considering integrating drones into their daily operations. However, many factors complicate the use of autonomous drones for commercial applications – particularly the limited battery capacity of drones and the limited onboard resources available to the drone.

In this thesis, we first investigated the path planning problem. Initially, an algorithm that bounds a search space in 2D is proposed to compute a path effectively. We then evaluated recently proposed sampling-based algorithms to determine the algorithm which generates a near-optimal path in different types of environments. From the initial comparison of RRT, RRT*, IB-RRT* and RRT*-AB in two-dimensional space, the IB-RRT* algorithm performed well (in terms of running time) in simple and complex environments, while RRT*-AB performed the best in cluttered environments. We later investigated a hybrid sampling-based algorithm BiRRT*-AB, which combines properties of the IB-RRT* algorithm and the RRT*-AB algorithm to generate near-optimal paths. Results showed that the BiRRT*-AB generated a consistent path in cluttered environments using less memory than RRT*-AB. An alternative algorithm was later proposed to further reduce the memory required to generate a path to effectively compute paths in large environments with over 2000 obstacles. We introduced the notion of a layered graph to represent the workspace accurately. This approach is practical since it generated a similar path using significantly less memory than BiRRT*-AB.

Furthermore, in this thesis, we focused on developing a comprehensive solution for a practical problem using drones. We proposed a model and a scheduling algorithm to be used for drone delivery applications. First, we introduced a cell-based model for a network of drones to cooperatively complete jobs that require travelling long distances since current drones do not have the capability of travelling long distances due to their limited battery capacity. Using the proposed model, we implemented the genetic algorithm to generate an optimal schedule for jobs that are randomly placed in a simulated environment. Results showed that drones could be used

to effectively deliver packages in a relatively short amount of time with this model. Moreover, the results support that the genetic algorithm can be used for such a scheduling application.

8.1 Limitations and Future Work

Although the proposed algorithm in Chapter 6 of the thesis can generate a path very rapidly using few resources, there is a particular limitation that can potentially impact the algorithm's performance. With the way the algorithm has been designed, when obstacles are arranged in the environment in a certain position, the generated layered graph would be equivalent to a regular grid, causing the algorithm to be computationally expensive. To overcome this issue, in the future, instead of generating a plane for each height of the obstacle, there could be a fixed number of planes generated in the environment to prevent such scenarios.

In Chapter 6, we showed that the proposed layered graph could be used to accurately represent cluttered environments using less memory. Similarly, compared to regular 3D grids, octrees significantly reduce the amount of memory required to represent an environment – especially in sparse environments [86]. Since the octree is able to accurately represent the environment using fewer nodes, it may be beneficial for graph-search based path planning algorithms. In the future, studies can be done to compare the two methods to determine the most effective approach for representing cluttered environments.

Although we focus on path length in this thesis, there are many other factors such as payload, drag, the direction of wind and communication, which all impact the battery performance in drones. As mentioned in Chapter 6, in the future, a cost function considering the impact of additional factors can be integrated into the algorithm to realistically evaluate the performance of this algorithm in drones.

Since we have demonstrated that the proposed path planning algorithm in Chapter 6 only utilizes few resources, in the future, computations of paths can be performed onboard drones in dynamic environments. In this thesis, we proposed an offline path planning algorithm. However, in reality, a path must be recomputed frequently when the drone is in motion. As a result of wind disturbances, a drone may deviate from its path. Thus, the drone must be able to recompute its flight path rapidly. Moreover, when moving obstacles are introduced in the drone's pre-

determined flight path, the drone must modify its path to avoid a collision. By retaining nodes from a previously computed layered graph, a new path can be generated more rapidly. Therefore, in the future, the algorithm can be extended to an online algorithm by only generating a new graph when a new obstacle is introduced in the environment.

There is room for improvement in the cell-based design that was proposed. Currently, we suggested that there should be k drones at every shared charging station. However, as shown in Table 11, this may result in many drones being idle in the schedule. Therefore, in the future, machine learning can be used to learn which stations often require more drones to complete tasks, and this information can then be used to determine how many drones should be placed at each station.

Table 11. Percent of drones idle in the schedule generated by the Genetic Algorithm

	Idle Drones (%)		
Number of Jobs	**$k = 1$**	**$k = 5$**	**$k = 15$**
10	60.84	90.97	97.02
50	20.95	60.49	84.31
100	7.86	44.40	70.14
200	1.50	31.08	55.79

Some improvements can be made to the scheduling algorithm mentioned in Chapter 7. In Chapter 7, we made many assumptions to solve the scheduling problem. First, we assumed that every drone must return to its original shared charging station. This was to ensure that there were enough drones distributed around each cell at any given time and to ensure that a drone was always charging when it was idle at a shared charging station. However, further investigations can be made into having the drone land at the nearest available charging station in the future. Moreover, we assumed that each job (delivery) had the same priority level, where in reality, there are usually express delivery services where some deliveries have precedence over other deliveries. Furthermore, we assumed that the straight Euclidean distance between two points is equivalent to the time required to complete an operation. However, in reality, there are many factors such as speed, path cost, and the package's weight, which can all affect the time required to move a package from one point to another. Finally, as mentioned in Chapter 2, currently, many studies investigate drone-truck hybrid solutions for deliveries. In the future, a comparative

analysis can be done to determine the benefit of using the drone-only solution proposed in this thesis as opposed to these hybrid solutions.

To further explore the practicality of the discussed path planning algorithms and the cell-based drone delivery model, in the future, experimentation should be done in more realistic environments. In our simulations, we generated obstacles and randomly placed them in an environment. However, resources such as ArcGIS [85] can be used to obtain geographical information and elevation information for buildings. With this information, an accurate map of a known environment can be generated. Moreover, a prototype should be developed to evaluate the performance of these algorithms. The implementation of these algorithms on physical drones can help validate the energy benefits of the proposed path planning algorithm and the efficacy of the cell-based model.

Bibliography

[1] Federal Aviation Administration, “FAA Aerospace Forecasts,” [Online]. Available: https://www.faa.gov/data_research/aviation/aerospace_forecasts/media/Unmanned_Aircraft_Systems.pdf

[2] J. Canny and J. Reif, "New lower bound techniques for robot motion planning problems," *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, Los Angeles, CA, USA, 1987, pp. 49-60

[3] Chen, Yong-Bo, et al. “UAV Path Planning Using Artificial Potential Field Method Updated by Optimal Control Theory,” *International Journal of Systems Science*, vol. 47, no. 6, 2014, pp. 1407–1420

[4] Y. Lin and S. Saripalli, "Path planning using 3D Dubins Curve for Unmanned Aerial Vehicles," *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, Orlando, FL, 2014, pp. 296-304

[5] K. Yang, S. Keat Gan, S. Sukkarieh, “A Gaussian process-based RRT planner for the exploration of an unknown and cluttered environment with a UAV,” *Advanced Robotics*, vol. 27, no. 6, 2013, pp. 431–443

[6] J. Kok, L. F. Gonzalez and N. Kelson, "FPGA Implementation of an Evolutionary Algorithm for Autonomous Unmanned Aerial Vehicle On-Board Path Planning," in *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 2, April 2013, pp. 272-281

[7] J. Chen, F. Ye and T. Jiang, "Path planning under obstacle-avoidance constraints based on ant colony optimization algorithm," *2017 IEEE 17th International Conference on Communication Technology (ICCT),* Chengdu, 2017, pp. 1434-1438

[8] Y. Lin and S. Saripalli, "Sampling-Based Path Planning for UAV Collision Avoidance," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, Nov. 2017, pp. 3179-3192

[9] W. H. Al-Sabban, L. F. Gonzalez and R. N. Smith, "Wind-energy based path planning for Unmanned Aerial Vehicles using Markov Decision Processes*," 2013 IEEE International Conference on Robotics and Automation, Karlsruhe*, 2013, pp. 784-789

[10] U. Cekmez, M. Ozsiginan and O. K. Sahingoz, "Multi colony ant optimization for UAV path planning with obstacle avoidance," *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, 2016, pp. 47-52

[11] V. Roberge, M. Tarbouchi and G. Labonté, "Fast Genetic Algorithm Path Planner for Fixed-Wing Military UAV Using GPU," in *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 5, Oct. 2018, pp. 2105-2117

[12] D. Zhang, Y. Xian, J. Li, G. Lei and Y. Chang, "UAV Path Planning Based on Chaos Ant Colony Algorithm," *2015 International Conference on Computer Science and Mechanical Automation (CSMA)*, Hangzhou, 2015, pp. 81-85

[13] Liu Y, Zhang X, Guan X, et al, “Adaptive sensitivity decision based path planning algorithm for unmanned aerial vehicle with improved particle swarm optimization,” *Aerospace Science and Technology*, vol. 58, no. 1, 2016, pp. 92–102

[14] L. De Filippis, G. Guglieri and F. Quagliotti, “Path Planning Strategies for UAVS in 3D Environments,” *Journal of Intelligent & Robotic Systems*, vol. 65, 2012, pp. 247–264

[15] X.Y. Zhang, H.B. Duan, “An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning,” *Applied Soft Computing*, vol. 26, 2015, pp. 270–284

[16] Y. Chen, J. Yu, Y. Mei, et al., “Modified central force optimization (MCFO) algorithm for 3D UAV path planning,” *Neurocomputing*, vol.171, no.1, 2016, pp. 878–888.

[17] D. Lee, H. Song and D. H. Shim, "Optimal path planning based on spline-RRT* for fixed-wing UAVs operating in three-dimensional environments," *2014 14th International Conference on Control, Automation and Systems (ICCAS)*, Seoul, 2014, pp. 835-839

[18] E. Dolicanin, I. Fetahovic, E. Tuba, R. Capor-Hrosik and M. Tuba, “Unmanned Combat Aerial Vehicle Path Planning by Brain Storm Optimization Algorithm,” *Studies in Informatics and Control*, vol. 27, no. 1, 2018, pp. 15-24

[19] Y. V. Pehlivanoglu, “A new vibrational genetic algorithm enhanced with a voronoi diagram for path planning of autonomous uav,” *Aerospace Science and Technology*, vol. 16, 2012, pp. 47–55

[20] X. Zhang, X. Lu, J. Songmin and X. Li, “A novel phase angle-encoded fruit fly optimization algorithm with mutation adaptation mechanism applied to UAV path planning,” *Applied Soft Computing*, vol. 70, 2018, pp. 371–388

[21] K. Wu, T. Xi and H. Wang, "Real-time three-dimensional smooth path planning for unmanned aerial vehicles in completely unknown cluttered environments," *TENCON 2017*, Penang, 2017, pp. 2017-2022

[22] X. Liang, G. Meng, Y. Xu, et al., “A geometrical path planning method for unmanned aerial vehicle in 2D/3D complex environment,” *Intelligent Service Robotics*, vol. 11, 2018, pp. 301–312

[23] Z. Yijing, Z. Zheng, Z. Xiaoyi and L. Yang, "Q learning algorithm based UAV path learning and obstacle avoidence approach," *2017 36th Chinese Control Conference (CCC), Dalian*, 2017, pp. 3397-3402

[24] H. Kim, J. Jeong, N. Kim and B. Kang, "A Study on 3D Optimal Path Planning for Quadcopter UAV Based on D* Lite," *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, Atlanta, GA, USA, 2019, pp. 787-793

[25] Y.B. Chen, Y.S. Mei, J.Q. Yu, et al., “Three-dimensional unmanned aerial vehicle path planning using modified wolf pack search algorithm,” *Neurocomputing*, vol. 266 no. 29, Nov. 2017, pp. 445-457

[26] H. Yang, Q. Jia and W. Zhang, "An Environmental Potential Field Based RRT Algorithm for UAV Path Planning," *2018 37th Chinese Control Conference (CCC)*, Wuhan, 2018, pp. 9922-9927

[27] C. Yan and X. Xiang, "A Path Planning Algorithm for UAV Based on Improved Q-Learning," *2018 2nd International Conference on Robotics and Automation Sciences (ICRAS)*, Wuhan, 2018, pp. 1-5

[28] R. L. Galvez et al., "Obstacle avoidance algorithm for swarm of quadrotor unmanned aerial vehicle using artificial potential fields," *TENCON 2017*, Penang, 2017, pp. 2307-2312

[29] C. Yin, Z. Xiao, X. Cao, X. Xi, P. Yang and D. Wu, "Offline and Online Search: UAV Multiobjective Path Planning Under Dynamic Urban Environment," in *IEEE Internet of Things Journal*, vol. 5, no. 2, April 2018, pp. 546-558

[30] X. Zhang, S. Jia, X. Li and M. Jian, "Design of the fruit fly optimization algorithm based path planner for UAV in 3D environments," *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*, Takamatsu, 2017, pp. 381-386

[31] Leutenegger S. et al., "Flying Robots," In *Springer Handbook of Robotics*, Springer Handbooks, 2016, pp. 623-670

[32] F. Morbidi, R. Cano and D. Lara, "Minimum-energy path generation for a quadrotor UAV," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, 2016, pp. 1492-1498

[33] Z. Lv, L. Yang, Y. He, Z. Liu and Z. Han, "3D environment modeling with height dimension reduction and path planning for UAV," *2017 9th International Conference on Modelling, Identification and Control (ICMIC)*, Kunming, 2017, pp. 734-739

[34] N. Chow, G. Gugan and A. Haque, "RADR: Routing for Autonomous Drones," *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Tangier, Morocco, 2019, pp. 1445-1450

[35] X. Song and S. Hu, "2D path planning with dubins-path-based A∗ algorithm for a fixed-wing UAV," *2017 3rd IEEE International Conference on Control Science and Systems Engineering (ICCSSE)*, Beijing, 2017, pp. 69-73

[36] D. Jung and P. Tsiotras, "Multiresolution on-line path planning for small unmanned aerial vehicles," *2008 American Control Conference*, Seattle, WA, 2008, pp. 2744-2749

[37] F. Samaniego, J. Sanchis, S. García-Nieto and R. Simarro, "UAV motion planning and obstacle avoidance based on adaptive 3D cell decomposition: Continuous space vs discrete space," *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, Salinas, 2017, pp. 1-6

[38] K. Jiang, L.D. Seneviratne and S.W.E. Earles, "A Shortest Path Based Path Planning Algorithm for Nonholonomic Mobile Robots," *Journal of Intelligent and Robotic Systems*, vol. 24, 1999, pp. 347–366

[39] Y.-H. Liu and S. Arimoto, “Path Planning Using a Tangent Graph for Mobile Robots Among Polygonal and Curved Obstacles: Communication,” *International Journal of Robotics Research*, vol. 11, no. 4, 1992, pp. 376–382

[40] Z. Ahmad, F. Ullah, C. Tran, and S. Lee, “Efficient Energy Flight Path Planning Algorithm Using 3-D Visibility Roadmap for Small Unmanned Aerial Vehicle,” *International Journal of Aerospace Engineering*, vol. 2017, 2017, pp. 1-13

[41] G. Frontera, D.J. Martín, J.A. Besada et al., “Approximate 3D Euclidean Shortest Paths for Unmanned Aircraft in Urban Environments,” *Journal of Intelligent & Robotic Systems*, vol. 85, 2017, pp. 353–368

[42] S. Huang and R. S. H. Teo, "Computationally Efficient Visibility Graph-Based Generation Of 3D Shortest Collision-Free Path Among Polyhedral Obstacles For Unmanned Aerial Vehicles," *International Conference on Unmanned Aircraft Systems (ICUAS)*, Atlanta, GA, USA, 2019, pp. 1218-1223

[43] P. Maini and P. B. Sujit, "Path planning for a UAV with kinematic constraints in the presence of polygonal obstacles," *International Conference on Unmanned Aircraft Systems (ICUAS)*, Arlington, VA, 2016, pp. 62-67

[44] A. Majeed and S. Lee, “A Fast Global Flight Path Planning Algorithm Based on Space Circumscription and Sparse Visibility Graph for Unmanned Aerial Vehicle,” *Electronics*, vol. 7, no. 12, 2018, pp. 1-27

[45] M. Naazare, D. Ramos, J. Wildt and D. Schulz, "Application of Graph-based Path Planning for UAVs to Avoid Restricted Areas," *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Würzburg, Germany, 2019, pp. 139-144

[46] J. D. Boskovic, R. Prasanth and R. K. Mehra, "A multilayer control architecture for unmanned aerial vehicles," *2002 American Control Conference*, Anchorage, AK, USA, vol. 3, 2002, pp. 1825-1830

[47] J. Wilke, “A drone program taking flight,” *The Amazon blog: dayone* [Online]. Available: https://blog.aboutamazon.com/transportation/a-drone-program-taking-flight

[48] Zipline, “Safe, Fast, Precise, & Magical,” [Online]. Available: https://flyzipline.com/how-it-works/

[49] S. Norouzi Ghazbi, Y. Aghli, M. Alimohammadi, and A. Akbari, “Quadrotors unmanned aerial vehicles: A review.” *International Journal on Smart Sensing & Intelligent Systems*, vol. 9, no. 1, 2016, pp. 309-333

[50] M. Hassanalian and A. Abdelkefi, “Classifications, applications, and design challenges of drones: a review,” *Progress in Aerospace Sciences*, vol. 91, 2017, pp. 99–131

[51] H. V. Abeywickrama, B. A. Jayawickrama, Y. He and E. Dutkiewicz, "Comprehensive Energy Consumption Model for Unmanned Aerial Vehicles, Based on Empirical Studies of Battery Performance," *IEEE Access*, vol. 6, 2018, pp. 58383-58394

[52] A. Abdilla, A. Richards and S. Burrow, "Power and endurance modelling of battery-powered rotorcraft," *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, 2015, pp. 675-680

[53] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” *Computer Science Department*, *Iowa State University*, Ames, IA, TR 98-11, Tech. Rep., 1998

[54] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no.7, June 2011, pp. 846-894

[55] A. H. Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, June 2015, pp. 1-11

[56] I. Noreen, A. Khan, H. Ryu, N. L. Doh and Z. Habib, "Optimal path planning in cluttered environment using RRT*-AB," *Intelligent Service Robotics*, vol. 11, no. 1, Jan. 2018, pp. 41-52

[57] X. Chen and J. Zhang, "The Three-Dimension Path Planning of UAV Based on Improved Artificial Potential Field in Dynamic Environment," *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics,* Hangzhou, 2013, pp. 144-147

[58] A. O Lyakhov, A. R. Oganov, H. T. Stokes and Q. Zhu, “New developments in evolutionary structure prediction algorithm USPEX,” *Computer Physics Communications*, vol. 184, 2013, pp. 1172–1182

[59] L. Yang, J. Qi, Y. Cao, Y. He, J. Han, J. Xiao, “UAV Path Planning Framework Under Kinodynamic Constraints in Cluttered Environments,” *Intelligent Robotics and Applications*, vol. 9245, 2015, pp. 248-259

[60] Kwangjin Yang and S. Sukkarieh, "3D smooth path planning for a UAV in cluttered natural environments," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 794-800

[61] J. Meng, V. M. Pawar, S. Kay and A. Li, "UAV Path Planning System Based on 3D Informed RRT* for Dynamic Obstacle Avoidance," *IEEE International Conference on Robotics and Biomimetics (ROBIO),* Kuala Lumpur, Malaysia, 2018, pp. 1653-1658

[62] F. C. Chen, G. Gugan, A. Haque, R. Solis-Oba, “Simple and Efficient Algorithm for Drone Path Planning,” Accepted to the *IEEE 2021 International Conference on Communications (ICC)*

[63] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Francisco, CA, 2011, pp. 2640-2645

[64] PwC. “Clarity from above” [Online] : https://www.pwc.pl/pl/pdf/clarity-from-above-pwc.pdf

[65] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2, 2014, pp. 56–77

[66] O. Adiyatov, and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning", *IEEE International Conference of Mechatronics and Automation (ICMA)*, Japan, 2013, pp. 354-359

[67] M. Svenstrup, T. Bak and H. J. Andersen, "Minimising computational complexity of the RRT algorithm a practical approach," *2011 IEEE International Conference on Robotics and Automation*, Shanghai, 2011, pp. 5602-5607

[68] M. Torres et. al., "Coverage path planning with unmanned aerial vehicles for 3D terrain reconstruction," *Expert Systems with Applications*, vol. 53, no. 02, 2016, pp. 441-451

[69] J.H. Holland, "Genetic Algorithms and Adaptation," *Adaptive Control of Ill-Defined Systems*, vol. 16, 1984, pp. 317-333

[70] J. Lee and K. Yu, "Optimal Path Planning of Solar-Powered UAV Using Gravitational Potential Energy," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 53, no. 3, 2017, pp. 1442-1451

[71] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger and P. Feyzeau, "Path planning: A 2013 survey," *International Conference on Industrial Engineering and Systems Management (IESM)*, Rabat, 2013, pp. 1-8

[72] E. W. Dijkstra, "A note on two problems in connection in with graphs," *Numerische Mathematik*, vol. 1, 1959, pp. 269-271

[73] Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." *IEEE transactions on Systems Science and Cybernetics*, 1968, pp. 100-107

[74] X. Chen and X. Chen, "The UAV dynamic path planning algorithm research based on Voronoi diagram," *The 26th Chinese Control and Decision Conference (2014 CCDC)*, Changsha, 2014, pp. 1069-1071

[75] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39-43

[76] C. C. Murray, A. G. Chu, "The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery," *Transportation Research Part C: Emerging Technologies*, vol. 54, 2015, pp. 86-109

[77] H. D. Yoo and S. M. Chankov, "Drone-delivery Using Autonomous Mobility: An Innovative Approach to Future Last-mile Delivery Problems," *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Bangkok, 2018, pp. 1216-1220

[78] M. Torabbeigi, G. J. Lim and S. J. Kim, "Drone delivery schedule optimization considering the reliability of drones," *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, Dallas, TX, 2018, pp. 1048-1053

[79] K. Dorling, J. Heinrichs, G. G. Messier and S. Magierowski, "Vehicle Routing Problems for Drone Delivery," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 1, 2017, pp. 70-85

[80] I. Hong, M. Kuby and A. Murray, "A range-restricted recharging station coverage model for drone delivery service planning," *Transportation Research Part C: Emerging Technologies*, vol. 90, 2018, pp. 198-212

[81] A. Goodchild and J. Toy, “Delivery by drone: An evaluation of unmanned aerial vehicle technology in reducing CO_2 emissions in the delivery service industry,” *Transportation Research Part D: Transportation and Environment*, vol. 61, 2018, pp. 58-67

[82] K. S. Yakovlev, D. A. Makarov, and E. S. Baskin, “Automatic path planning for an unmanned drone with constrained flight dynamics,” *Scientific and Technical Information Processing*, vol. 44, no. 5, 2015, pp. 347 – 358

[83] D. Tezcaner, and M. Köksalan, “An interactive algorithm for multi-objective route planning,” *Journal of Optimization Theory and Applications*, vol. 150, no. 2, 2011, pp. 379-394

[84] Z. Qi, Z. Shao, Y. S. Ping, L. M. Hiot and Y. K. Leong, “An improved heuristic algorithm for UAV path planning in 3D Environment,” *Second International Conference on Intelligent Human-Machine Systems and Cybernetics*, 2010, pp. 258-261

[85] ArcGIS, “ArcGIS Online” [Online]. Available: https://www.arcgis.com/index.html

[86] Y. Kitamura, T. Tanaka, F. Kishino and M. Yachida, "3-D path planning in a dynamic environment using an octree and an artificial potential field," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Pittsburgh, PA, USA, vol. 2, 1995, pp. 474-48

Appendices

Appendix A: Literature Review

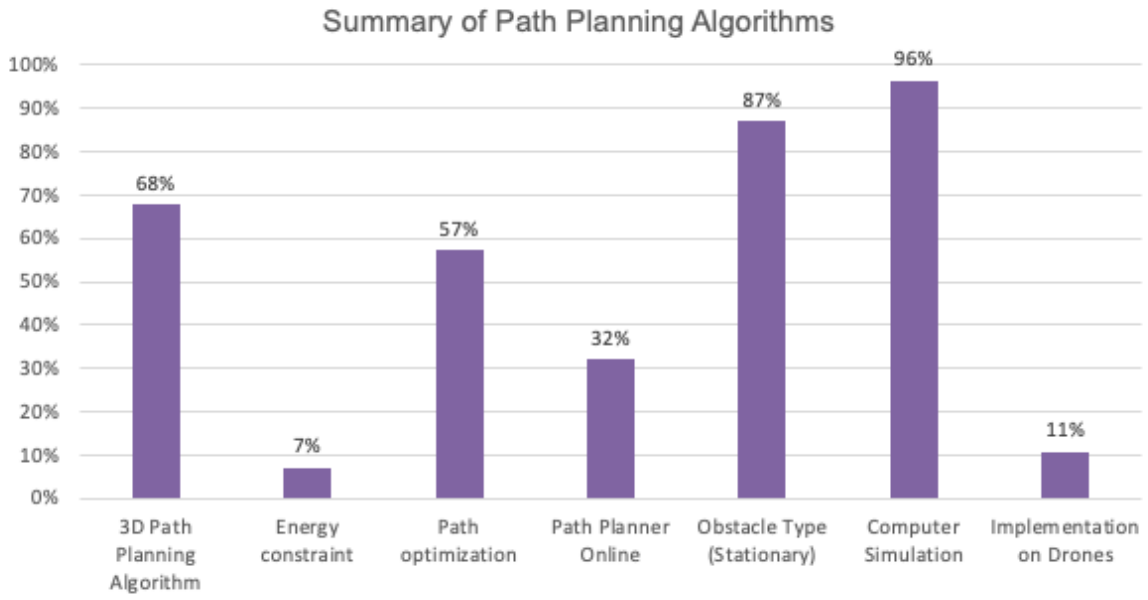


Figure 30. Summary of Reviewed Papers

Table 12. Further details of the reviewed papers

Paper	Algorithm Type	Details	Performance Comparison
[3]	Potential Fields	Artificial Potential + control force	Original Artificial Potential Field
[4]	Sampling-Based Algorithm	RRT + Dubins Curve	-
[5]	Sampling-Based Algorithm	RRT + Gaussian process Map	-
[6]	Bio-inspired Algorithm	Modified Genetic Algorithm	-
[7]	Bio-inspired Algorithm	Ant Colony Optimization	-
[8]	Sampling-Based Algorithm	Modified Closed Loop RRT	-
[9]	Reinforcement Learning / Decision Making	Markov Decision Process (Wind)	-
[10]	Bio-inspired Algorithm	Multi-Colony (Ant Colony Optimization)	Original Ant Colony Optimization
[11]	Bio-inspired Algorithm	Parallel Genetic Algorithm	-
[12]	Bio-inspired Algorithm	Chaos Ant Colony Algorithm	Original Ant Colony Optimization
[13]	Bio-inspired Algorithm	Improved Particle Swarm Optimization	Genetic Algorithm, Particle Swarm Optimization, Firefly Algorithm
[14]	Graph Search	Graph-Search Algorithms (A* and Theta*)	Theta*, A*
[15]	Bio-inspired Algorithm	Improved Differential Evolution (optimization Algorithms)	PSO, Differential Evolution Variations

[16]	Other	Modified Central Force Optimization	Original Central Force Optimization, Firefly Algorithm, Particle Search Optimization, Random Search
[17]	Sampling-Based Algorithm	Spline-RRT*	-
[18]	Other	Brain Storm Optimization	GA, DE, Firefly Algorithm, Bat Algorithm, Variations of GA, Firefly Algorithm and Bat Algorithm
[19]	Bio-inspired Algorithm	Multi-vibrational genetic Algorithm	Genetic Algorithm
[20]	Bio-inspired Algorithm	Variation Of Fruit Fly Algorithm	Fruit Fly, Improved Fruit Fly, Multi Swarm Fruit Fly, Artificial Bee Colony, Differential Evolution, PSO, Imperialist Competitive Algorithm
[21]	Sampling-Based Algorithm	RRT*-based Waypoint generation + PSO for smoothing	-
[22]	Environmental Mapping	Geometrical Path Planning	Artificial Potential Field, A* in 2D, GA, interfered fluid dynamical system in 3D
[23]	Reinforcement Learning / Decision Making	Random Exploration and Q-Learning	Traditional Q-Learning
[24]	Graph Search	D* Lite Algorithm	-
[25]	Bio-inspired Algorithm	Improved Wolf Pack Search	Original Wolf Pack Search, Genetic Algorithm, Random Search Way
[26]	Sampling-Based Algorithm	RRT + Potential Field	Original RRT
[27]	Reinforcement Learning / Decision Making	Improved Q-Learning	Traditional Q-Learning
[28]	Potential Fields	APF for Multiple Drones	-
[29]	Graph Search	Joint Offline, and Online planning using a Safety Index Map and A*	-
[30]	Bio-inspired Algorithm	Fruit fly Optimization	Genetic Algorithm

Tables 12-13 show the experiments performed to justify the parameters chosen for the Genetic Algorithm and the Particle Swarm Optimization algorithm. The standard deviations are noted in parenthesis in each table, and the best result has been bolded.

Table 13. Tuning the parameters for the Genetic Algorithm

Number of Jobs	**Crossover Rate**				
	0.6	**0.7**	**0.8**	**0.9**	**1.0**
10	73.83 (8.85)	73.85 (8.83)	73.76 (8.85)	73.79 (8.83)	**73.71 (8.83)**
50	207.41 (4.11)	207.50 (3.97)	207.69 (3.94)	207.14 (4.22)	**207.08 (4.14)**
100	391.01 (14.20)	390.77 (13.94)	**390.48 (14.10)**	390.60 (14.56)	390.61 (13.85)
200	789.46 (20.65)	790.92 (21.21)	789.42 (21.47)	**788.97 (19.96)**	789.11 (20.85)

Number of Jobs	**Mutation Rate**				
	0.01	**0.05**	**0.1**	**0.2**	**0.3**
10	**73.83 (8.85)**	73.72 (8.81)	73.86 (8.81)	73.75 (8.81)	73.83 (8.82)
50	207.41 (4.11)	**207.36 (4.46)**	207.48 (3.90)	207.59 (4.29)	207.92 (4.22)
100	391.01 (14.20)	391.08 (13.96)	**390.93 (13.88)**	391.47 (14.23)	391.85 (14.44)
200	**789.46 (20.65)**	790.01 (21.03)	791.19 (21.52)	790.78 (21.78)	790.07 (21.05)

Number of Jobs	Population Size			
	30	**50**	**100**	**150**
10	74.18 (8.98)	73.83 (8.85)	73.70 (8.83)	**73.52 (8.84)**
50	208.47 (3.60)	207.41 (4.11)	206.38 (4.36)	**205.55 (3.80)**
100	391.76 (13.52)	391.01 (14.20)	389.44 (14.19)	**388.05 (14.48)**
200	790.00 (21.40)	789.46 (20.65)	787.38 (21.11)	**785.38 (21.55)**

Number of Jobs	Generation Number		
	200	**400**	**600**
10	73.83 (8.85)	73.66 (8.83)	**73.52 (8.86)**
50	207.41 (4.11)	206.20 (4.37)	**205.70 (4.11)**
100	391.01 (14.20)	389.20 (14.32)	**388.24 (13.98)**
200	789.46 (20.65)	786.79 (20.79)	**784.94 (20.56)**

Number of Jobs	Elite Chromosomes						
	0	**1**	**5**	**10**	**20**	**30**	**40**
10	73.96 (8.84)	73.82 (8.84)	**73.68 (8.94)**	73.76 (8.98)	73.87 (8.98)	74.11 (9.09)	74.23 (9.16)
50	207.42 (3.94)	203.42 (4.62)	197.59 (4.19)	195.33 (4.68)	**192.58 (4.57)**	193.47 (4.03)	197.40 (4.32)
100	391.01 (14.19)	385.11 (14.14)	375.62 (13.29)	371.48 (12.96)	**368.09 (12.21)**	368.57 (13.62)	375.12 (14.42)
200	790.28 (20.52)	778.62 (21.18)	762.09 (18.24)	755.37 (17.47)	**750.69 (18.19)**	751.40 (17.95)	763.91 (20.40)

Table 14. Tuning the parameters for the Particle Swarm Optimization Algorithm

Number of Jobs	Value of C_1 and C_2					
	2.25	**2**	**1.75**	**1.5**	**1.25**	**1**
10	**74.42 (8.93)**	75.01 (8.88)	75.64 (8.68)	76.44 (7.97)	76.98 (7.79)	77.40 (7.35)
50	**208.67 (4.01)**	211.77 (3.99)	214.22 (4.05)	216.87 (3.37)	218.57 (4.19)	220.03 (3.71)
100	**390.31 (13.92)**	393.80 (13.59)	398.71 (14.74)	401.08 (13.58)	402.43 (14.14)	403.94 (14.63)
200	**788.53 (21.39)**	794.43 (21.80)	800.36 (21.94)	804.41 (22.34)	806.55 (22.92)	808.27 (24.01)

Number of Jobs	Population				
	20	**30**	**40**	**50**	**60**
10	75.01 (8.88)	74.79 (8.99)	74.70 (9.03)	74.56 (8.98)	**74.50** **(9.00)**
50	211.77 (3.99)	210.15 (3.60)	209.37 (3.45)	208.94 (3.62)	**208.36** **(3.72)**
100	393.80 (13.59)	392.23 (13.95)	391.22 (13.11)	390.64 (14.53)	**390.02** **(13.76)**
200	794.43 (21.80)	792.91 (22.04)	790.32 (22.11)	789.73 (21.72)	**789.17** **(21.51)**

Number of Jobs	Generation Number		
	200	**300**	**400**
10	75.01 (8.88)	74.72 (8.97)	**74.57** **(8.93)**
50	211.77 (3.99)	210.00 (3.47)	**208.77** **(3.88)**
100	393.80 (13.59)	392.07 (13.61)	**391.12** **(14.09)**
200	794.43 (21.80)	790.97 (21.82)	**789.89** **(22.32)**

Number of Jobs	w_{max}			
	0.9		**1**	
	Completion Time	**Running Time (ms)**	**Completion Time**	**Running Time (ms)**
10	75.01 (8.88)	503.24 (63.85)	**74.62** **(9.02)**	757.76 (88.12)
50	211.77 (3.99)	1904.95 (294.13)	**208.93** **(3.67)**	9292.35 (2,820.57)
100	393.80 (13.59)	4096.88 (522.35)	**389.86** **(14.53)**	27259.44 (8,854.39)
200	794.43 (21.80)	10729.29 (2,074.46)	**788.64** **(21.30)**	86004.22 (27,411.77)

	w_{min}			
Number of Jobs	**0.5**	**0.4**	**0.3**	**0.2**
10	**74.91** **(8.90)**	75.01 (8.88)	75.41 (8.55)	75.24 (8.66)
50	**210.88** **(3.55)**	211.77 (3.99)	212.33 (3.75)	212.36 (3.88)
100	**393.30** **(14.01)**	393.80 (13.59)	395.61 (14.70)	395.56 (14.85)
200	**793.20** **(21.41)**	794.43 (21.80)	796.95 (21.68)	798.23 (22.94)

Curriculum Vitae

Name:	Gopi Gugan
Post-secondary Education and Degrees:	M.Sc. Candidate, Computer Science The University of Western Ontario 2018-2020
	B.M.Sc., Medical Science The University of Western Ontario 2013-2017
Honours and Awards:	Dean's Honor List The University of Western Ontario 2013-2018
Related Work Experience	Teaching Assistant The University of Western Ontario 2018-2020

Peer-Reviewed Publications:

1. N. Chow, G. Gugan, and A. Haque, "RADR: Routing for Autonomous Drones," in proceedings of the *15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, Tangier, Morocco, 2019, pp. 1445-1450. [Acceptance Rate 36%]
2. G. Gugan and A. Haque, "Towards the Development of a Robust Path Planner for Autonomous Drones", in proceedings of the *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring),* Antwerp, Belgium, 2020, pp. 1-6, doi: 10.1109/VTC2020-Spring48590.2020.9128387. [Acceptance Rate 46%]
3. F. Chen, G. Gugan, R. Solis-Oba, and A. Haque, "Simple and Efficient Algorithm for Drone Path Planning", accepted to the *55th 2021 IEEE International Conference on Communications* (ICC 2021).
4. G. Gugan and A. Haque, "Path Planning for Autonomous Drones: Challenges and Future Directions", manuscript under preparation for *IEEE Communications Magazine*
5. G. Gugan and A. Haque, "A Novel Cell-Based Design for Completing Long-Range Tasks with a Network of Autonomous Drones", manuscript under preparation for *IEEE Transactions on Systems, Man, and Cybernetics*

Industry Technical Report:

G. Gugan, and A. Haque, “Hyperspectral Imaging for Detecting VOC”, Industry Technical report submitted to *IBM Watson IoT Labs Canada*, March 2019

Patents/Report of Innovation:

1. Report of Innovation (ROI) Disclosure, World Discoveries - Western University, Filing Date: 2020/07/23 (patent filing in process), Inventors: A. Haque and G. Gugan, “A Novel Cell-Based Design for Completing Long-Range Tasks with a Network of Autonomous Drones”
2. Report of Innovation (ROI) Disclosure, World Discoveries - Western University, Filing Date: 2020/07/23 (patent filing in process), Inventors: R. Solis-Oba, Fu Chen, G. Gugan, and A. Haque, “ A Novel Algorithm for Drone Path Planning”