Article

An Accelerated Dual Fast Marching Tree Applied to Emergency Geometric Trajectory Generation

Andréas Guitart [1,*], Daniel Delahaye [1] and Eric Feron [2]

[1] OPTIM Laboratory, ENAC—National School of Civil Aviation, 7 Avenue Edouard Belin CS 54005, CEDEX 4, 31055 Toulouse, France; delahaye@recherche.enac.fr
[2] Computer, Electrical and Mathmatical Science and Engineering, KAUST—King Abdullah University Science and Technology, Thuwal 23955-6900, Saudi Arabia; eric.feron@kaust.edu.sa
* Correspondence: andreas.guitart@enac.fr

Abstract: This paper addresses the generation of aircraft emergency trajectories with obstacle avoidance. After presenting in detail the fast marching tree algorithm, in this paper we propose an improvement of its performance. First, the free space checking function is sped up. Then, the algorithm is used twice, firstly with the sampling of a few points to generate an approximate trajectory, and secondly with a sampling of points close to the first computed trajectory to refine it. The proposed method significantly reduces the computing time of the emergency geometric trajectory generation.

Keywords: emergency trajectory; quadtree; octree; sampling-based path planning algorithm; Dubins curve

Citation: Guitart, A.; Delahaye, D.; Feron, E. An Accelerated Dual Fast Marching Tree Applied to Emergency Geometric Trajectory Generation. *Aerospace* **2022**, *9*, 180. https://doi.org/10.3390/aerospace9040180

Academic Editors: Joost Ellerbroek

Received: 1 February 2022
Accepted: 23 March 2022
Published: 25 March 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In the event of an emergency, pilots do not have a tool to help them in this kind of extremely critical situation. Their decisions, sometimes disturbed by the stress of the situation, can cause an accident. Moreover, in some very critical cases, it is almost impossible for the pilot to make the best decision. This observation highlights the importance of developing a tool to help pilots land safely. This tool would predict a safe and optimal trajectory that pilots would have had difficulty finding alone in a moment of intense stress. It would therefore significantly increase the chance of saving the aircraft. Indeed, in the case of an emergency due to a dual engine failure in cruise, a good prediction makes it possible to glide longer. Therefore, the aircraft can reach more airports and perhaps have a safer outcome.For example, in 2001, Air Transat Flight 236 lost all engine power while flying over the Atlantic Ocean. The Airbus A330 ran out of fuel due to a fuel leak. After 21 min of gliding, the plane managed to land on a military base in the Azores. The pilot made the approach by hand and the aircraft arrived on the runway too fast. It was badly damaged and the landing could have resulted in a fatal fall for the passengers, as the runway ended with a cliff.This flight is historic because it was the longest passenger aircraft glide without engines. In 1959, under similar conditions (similar aircraft and weather), the Caravelle "Lorraine" glided for 46 min from Paris to Dijon. The fact that the Caravelle's trajectory was predicted before the flight explains the difference in glide time. These two examples show that the good prediction of a trajectory can significantly increase the gliding distance and therefore increase the chance of a safe landing.

The problem of safe emergency trajectory generation raises two main issues. The first one is the limited computing time. One well-known case is the landing on the Hudson river of US Airways Flight 1549 after bird strikes caused dual engine failure (See Figure 1). This case is very interesting because, in the space of 30 s, the situation went from critical to unmanageable, and the only solution was to land on the Hudson River. This accident highlights the importance of proposing an efficient algorithm in terms of computing time.

The paper deals primarily with this issue. The goal is to propose a fast method to generate an emergency trajectory, taking obstacles into account.

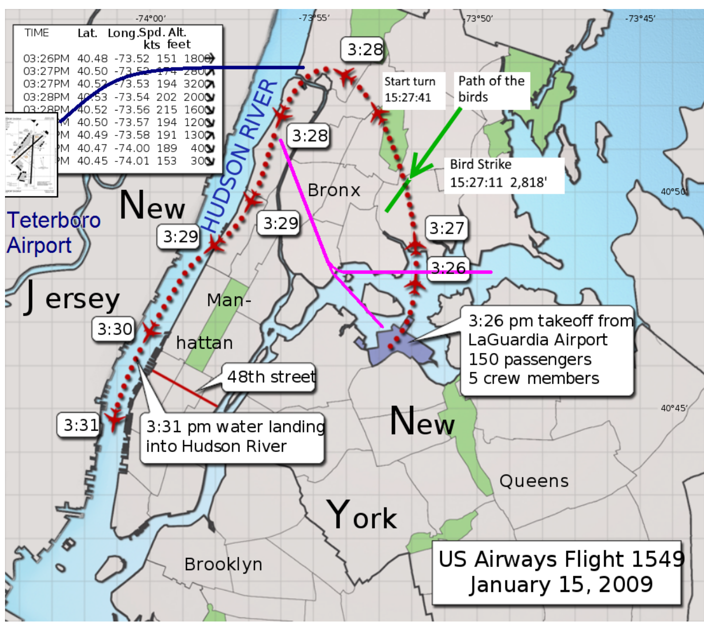


Figure 1. US Airways Flight 1549: Two minutes after takeoff from LaGuardia, the aircraft struck a flock of birds at an altitude of 2818 ft, which caused the loss of engines and forced the pilot to make a sea landing on the Hudson river [1].

The second issue is the diversity of types of emergency (loss of engine power, cabin fire, depressurization, etc.). They can be grouped into two types. The first type is referred to as ASAP (As Soon As Possible). For example, a cabin fire and medical emergency are considered ASAP emergencies. The other type is referred to as ANSA (At Nearest Suitable Airport). In this case, the selected landing site is the safest among those reachable. A failure of all engines is an ANSA emergency. This paper does not address the selection of the landing site. In the following, the landing site will be assumed to be known before the trajectory computation takes place. Indeed, the landing site could be computed by another algorithm that considers the altitude data around the aircraft's position. Moreover, the impact of an eventual failure on aircraft performance will not be considered. However, the proposed methods take into account the minimum curvature radius and the descent rate (presented in detail below). These values could be obtained by a previous algorithm that takes into account aircraft features. The method adapts to all input values and therefore to all types of situations. Moreover, this algorithm could take into account the wind. The curvature radius and descent angle would therefore depend on the heading.

The objective of this research was to develop an algorithm that rapidly generates an emergency trajectory from the failure position to a landing site. For this study, the landing site is considered known, determined by an algorithm that takes into account the altitude data around the emergency aircraft position. The heading constraints on departure and arrival have been added. The aircraft has a given turning curvature radius r, a maximum climb rate, and a maximum descent rate. This paper proposes to use a sampling-based path planning algorithm. Moreover, one of its main functions is modified, in order to improve the algorithm performance. Finally, Dubins curves have been added to the process to make the trajectories flyable. The proposed initial solution does not take into account either the weather or degraded dynamics of the aircraft resulting from some failures. This paper focuses on the speed of the generation of a trajectory.

The paper is organized as follows: Section 2 presents the state of the art in relation to trajectory generation. Then, Section 3 describes the mathematical modeling process. In Section 4, the approach used to address the problem is presented. Finally, in Section 5,

several cases on which the algorithm has been tested are given, some designed specifically to compare algorithms from the literature and the proposed algorithm, and another one designed based on real data.

2. Prior State of the Art

Several approaches have been attempted to find a solution to the path planning problem. However, the emergency trajectory generation problem has been less studied. This problem is very complex because the trajectory has to be quickly generated. Nevertheless, the computed trajectory is not necessarily optimal but it must be flyable. Therefore, the proposed algorithm can make approximations in order to reduce the computing time.

2.1. Design of Emergency Landing Trajectories

In 2006, Atkins et al. [2] provided an adaptative flight planning (AFP) algorithm in order to select a landing site and generate a safe emergency trajectory in real time. The trajectory planner takes into account the initial state of the aircraft, as well as flight dynamics and wind constraints. This algorithm was applied to Flight 1549 in [3], and the algorithm generated a safe trajectory to return to LaGuardia Airport. Tang et al. [4] proposed to solve the two-point boundary value problem (TPBVP) to generate unpowered landing trajectories and improve aircraft safety. Fallast and Messnarz [5] proposed a solution to automatically select an airport and generate an emergency trajectory to this landing site which avoids obstacles including a safety margin. Their algorithm is an adapted version of the rapidly exploring random tree algorithm. It generates a search tree within the free space, starting from an initial position and trying to connect final positions with this tree. Its main difference from the original RRT* algorithm is that the search tree points are connected with a Dubins curve (explained in detail in Section 2.4). This difference allows it to take into account the start and end heading constraints and also the aircraft's performance capabilities. Moreover, their proposed algorithm reduces the point connections by introducing another constraint linked to the maximum climb and descent rates. Another work addressed the problem of emergency trajectory design. In his thesis, Zhao [6] introduced a landing path primitive generation method based on the suboptimal solution of a three-dimensional variation of the classical Markov–Dubins problem. It considers the generation of geometric paths. The problem is defined as an optimal control problem. In his study, first, the minimal length curve problem in the horizontal plane is addressed and then the three dimensional landing path is obtained by generating a vertical profile. The proposed algorithm was tested in two different scenarios (US Airways 1549 and Swissair 111). The results showed that the method was efficient. However, it did not consider obstacles; therefore, it cannot be integrated into a FMS. Sáez et al. [7] proposed a method based on the RRT* to generate an emergency trajectory. The trajectory follows a given profile and considers the minimum curvature radius of the aircraft. They take into account the impact of the eventual failure on the aircraft performance. However, the computing time of their algorithm can be high. Haghighi et al. [8] presented a post-failure performance analysis and an optimization method to generate a fast and safe landing trajectory that avoids obstacles. Their method was based on Dubins curves and Apollonius results [9].

The works on emergency landing propose very interesting methods but they mainly concern the impact of the failure on the aircraft. The computing time required for the generation of a trajectory is not mentioned or is above one minute. Ligny et al. [10] propose a very efficient algorithm to solve this problem. Their method, based on the fast marching method, generates a trajectory in less than 1 s. However, the usability of the algorithm remains limited. This method is compared to the proposed method in Section 5. Several recent papers have proposed fast trajectory generation methods that are potentially usable for the studied problem. They are presented below.

2.2. Trajectory Generation Algorithm

Depending on the path planning problem, the objective may be very different. It may consist in simply finding a safe path or a path minimizing or maximizing a given criterion (distance, consumption, time, etc.). Moreover, depending on the type of problem, the criteria for success are not the same. In this study, the goal is to have a very fast algorithm but one that is adaptable to any type of emergency situation. Hong et al. [11] propose a computationally efficient method to generate a smooth level change in trajectory. Their proposed algorithm consists of a line search method in combination with a fixed-horizon sequential convex optimization method. This method seems very efficient (the computing time is around 0.4 s), but the size of the search is very small (100 m vertically and 1500 m horizontally). Therefore, the computing time could be high with bigger space. Moreover, in their problem, the obstacles are not considered but the free space test can be very expensive in terms of computing time. Another work [12] proposes an efficient method to generate a collision-free trajectory. This algorithm is based on P-RRT* [13] combined with a line-of-sight path optimizer. Other works [10,14,15] have proposed generating trajectories using methods based on fast marching. This is a front-propagation method that functions in the manner of a forest fire.This method is very efficient but it does not seem to be very adaptable to diverse types of problem. For example, it seems complicated to take into account the aeronautical constraints, as explained by Ligny et al. [10] in their paper. Some papers [16–18] have proposed using a very efficient graph-based path planning algorithm. These methods seem to be very efficient in terms of computing time. Moreover, they are very adaptable because they are used in different fields (robots, helicopters, aircrafts...). This paper presents these methods in detail and proposes to improve one of them in order to obtain an even more efficient algorithm.

2.3. Graph-Based Path Planning Algorithm

There are many methods to determine the shortest path in a graph. This paper presents three well known algorithms (Dijkstra, Bellman and A*). Dijkstra's algorithm [19] is an exact shortest-pathfinding algorithm in a weighted graph that does not contain any absorbing circuit, which is a closed path with negative weight. The Bellman–Ford algorithm is equivalent to Dijkstra's algorithm for the single-source multiple-shortest-paths problem. It was proposed by Richard Bellman and Lester Randolph Ford Jr., who published the algorithm in 1956 [20] and in 1958 [21]. A* is a path search algorithm [22], which is often used in computer science due to its completeness, optimality, and optimal efficiency. This algorithm has to define a priority queue, similarly to the Dijkstra algorithm. In the case of A*, the priority is defined by:

$$f(n) = g(n) + h(n) \tag{1}$$

where n is a node of the graph, $g(n)$ is the cost of the path from the start node to n, and h is a heuristic function that estimates the cost from a node n to the goal. The heuristic function has to underestimate the real cost to reach the goal to ensure that the solution obtained is optimal. These classical algorithms assume the existence of a graph. A simple way to construct this graph would be to use a grid. However, the trajectory obtained would be greatly dependent on the accuracy of the grid. More recently, new and more efficient graph-based algorithms have been proposed. These algorithms generate a graph to find a path between two points. Moreover, some versions of these methods are asymptotically optimal.

Over the past decades, many methods have been proposed to generate a graph in order to find the optimal path between a pair of nodes. The most fashionable methods are sampling-based path planning algorithms. These methods are based on free space sampling [23–27]. Before discussing the algorithms, the problem must be formulated and some primitive functions used by such algorithms must be introduced.

Let $\chi = (0,1)^d$ be the configuration space, where $d \in \mathbf{N}$ is the space dimension , $(d \geq 2)$. Let χ_{obs} be the obstacle region, such that $\chi \backslash \chi_{obs}$ is an open set, and denote the

obstacle-free space as $\chi_{free} = cl(\chi \backslash \chi_{obs})$, where $cl(\chi)$ denotes the closure of a set χ. The initial condition is denoted by $x_{init} \in \chi_{free}$, and the goal region χ_{goal} is an open of χ_{free}.

Sampling-based path planning algorithms mainly use four functions:

Sampling: The Sample function generates a sequence of points in χ. The distribution of points can be uniform or randomly generated. It should be noted that random sampling makes the solutions of algorithms non-reproducible. It is better when the algorithm is able to sample points directly in χ_{free}.

Nearest Neighbor: This function returns a vertex that is the closest to a point $x \in \chi$ in terms of a given distance.

Near Vertices: This function returns the vertices that are contained in a ball of radius r centered at a point $x \in \chi$.

Collision Test: This function returns True if the straight line between two points $x, x' \in \chi$ lies in χ_{free} and False otherwise.

There are three main sampling-based path planning algorithms, probabilistic roadmaps (PRM), rapidly-exploring random tree (RRT), and fast marching tree (FMT) algorithms [28].

The PRM algorithm is composed of a pre-processing phase which constructs a road map using n randomly-sampled points in χ_{free} and a second phase to find the shortest path between the initial point and the final point [28].

The RRT algorithm works differently. In the beginning, the graph is composed of the initial vertex and no edges. At each iteration, the algorithm tries to connect a new sampled point $x_{rand} \in \chi_{free}$ to the nearest vertex of the tree. If such a connection is possible (i.e., there is no obstacle between them), x_{rand} is added to the vertex set V, and (v, x_{rand}) is added to the edge set. The graph construction and the path search can be performed concurrently.

An extension of such an algorithm is RRT*. The RRT* algorithm creates connections similarly toRRT. Moreover, at each iteration, it also tries to improve the graph by connecting $x_{new} \in \chi_{free}$ to vertices that are within distance r. A connection to x_{new} is created, if the cost to reach it is minimal. In the event of a cost improvement, the edge linking the vertex and its parent is deleted and is replaced by the new connection [28] .

The FMT algorithm [29] (see Algorithm A1) performs a forward-propagation over several sampled points generated during the initialization step and generates a tree of paths. Three key features characterize the algorithm:

- Two samples are considered neighbors if their distance is below a given radius;
- The graph construction and path search are performed concurrently;
- If the locally-optimal connection to a new sample intersects an obstacle, the algorithm skips this sample. It does not consider the other connections to the neighborhood.

Before presenting the details of the algorithm, some sets must be introduced. $V_{unvisited}$ is a set composed of nodes that do not yet have an assigned cost. V_{closed} is composed of nodes that are visited and have a cost that can no longer be modified because it is optimal. V_{open} is a set that contains visited nodes but their cost is temporally assigned. The following four drawings (see Figure 2) represent the local optimization phase of Algorithm A1, which is repeated as long as the destination x_{goal} is not reached. The algorithm begins with a graph composed of only one node, which is the starting point x_{start}.

According to the literature, FMT is the fastest algorithm (See Table 1). Due to the fact that the algorithm checks only one connection, the computing complexity of the collision test is reduced to $O(n)$, whereas for PRM* and RRT*, it is $O(n \log n)$ where n is the number of sampling points.

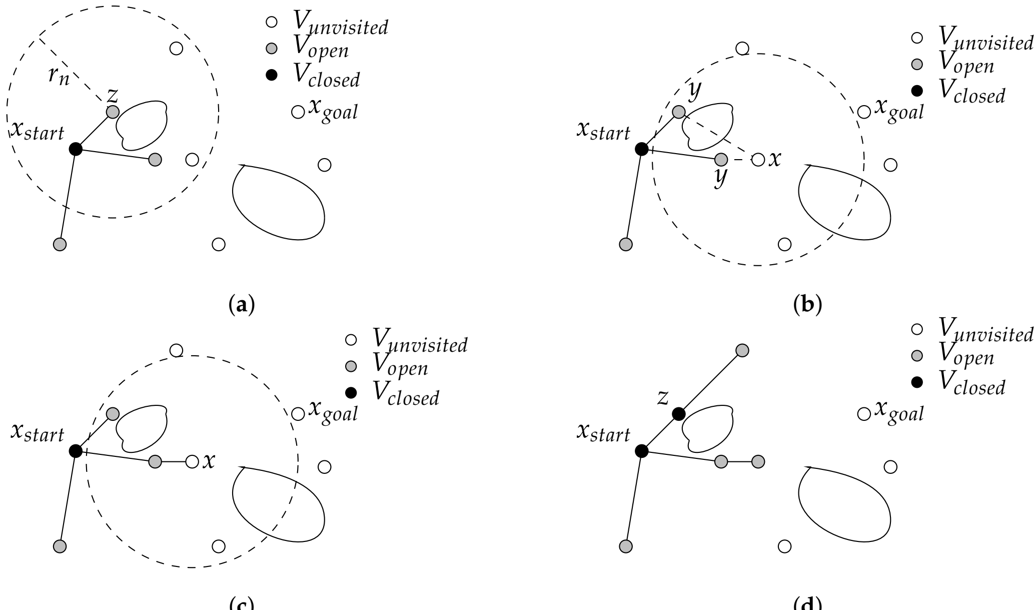


Figure 2. Local optimization phase [29]. (**a**) Lines 3–9. In this first step, the lowest-cost node z from set V_{open} is selected and the nodes within $V_{unvisited}$ which are near to z are found. (**b**) Lines 10–13. For each unvisited node x near to z, its neighbors within V_{open} are found, and x is connected to have an optimal cost without going through an obstacle. (**c**) Lines 14–18. A connection is created between x and the neighbor of the locally-optimal step connection. (**d**) Lines 20–25. All neighbors of z are visited and are added to V_{open}. z is added to V_{closed}. FMT* moves to the next iteration with the node which has the lowest cost.

Table 1. Algorithm complexity depending on the number of samples n [28].

Graph Generation Algorithm	Time Complexity		Space Complexity
	Processing	Query	
PRM	$O(n \log n)$	$O(n \log n)$	$O(n)$
PRM*	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
RRT	$O(n \log n)$	$O(n)$	$O(n)$
RRT*	$O(n \log n)$	$O(n)$	$O(n)$
FMT	$O(n \log n)$	$O(n)$	$O(n)$

One of the main success criteria of the proposed solution is the computing time. For this reason, the algorithm proposed herein is based on the fast marching tree algorithm. Therefore, PRM and RRT are not tested in this study.

The main issue of this algorithm is that the computed path is not flyable because all the points building the path are connected with straight lines, which can create heading discontinuity at each point. Moreover, heading constraints on arrival and departure are not satisfied.

2.4. Dubins Curve

One of the main constraints taken into account in this study is the curvature constraint. The Dubins curve is a solution in respect to this constraint [30,31]. The Dubins path typically refers to the shortest curve that connects two points with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path [32,33]. There are six types of Dubins curve; four are of the Circle-Segment-Circle type and two are of the Circle-Circle-Circle type. If the distance between two connected points is smaller than the sum of the two radii and if the initial and final heading are in opposite directions,

no straight line segment can be fitted between the circle segments [5]. In this case, the shortest route is a path of the Circle-Circle-Circle type. Figure 3 shows an example of a Dubins Curve.

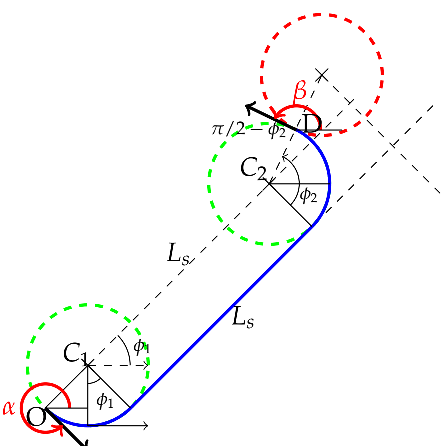


Figure 3. Left-Segment-Left Dubins Curve: The curve connects a start point with an orientation angle α and an end point with an orientation angle β. It is composed of a turn to the left around the first green circle, a segment of length L_s, and a second turn to the left around the second green circle [7].

These previous related works have shown the efficiency of sampling-based path algorithms in terms of computing time. However, these algorithms cannot be used directly; indeed, the paths generated by this type of method are not flyable. The introduction of Dubins curve enables to solve this problem. Moreover, some main functions of these algorithms, such as the free space checker, can be improved to reduce the computing time required for path generation. In the following section, the space search, the objective, and the constraints are presented.

3. Mathematical Modeling

This section presents the search space and the aeronautical constraints.

3.1. Search Space

3.1.1. Terrain Data

Based on altitude data around the aircraft's position, a cube is created to model the space search. In this cube, each point p generated during the sampling phase is a state vector containing the coordinates x and y and the altitude alt, giving the vector:

$$p_{new} = \begin{pmatrix} x \\ y \\ alt \end{pmatrix} \tag{2}$$

where:

$$terrain(x, y) < alt < alt_{max} \tag{3}$$

3.1.2. Route Representation

In this study, the route is represented by a set of n points $(p_0,\ p_1, \ldots,\ p_{n-1})$ such as $\forall i, 0 \leq i < n-1$, and p_i is connected to p_{i+1}. In the first attempt, the points are connected with straight lines. Then, to make the path flyable, such straight lines are replaced by

Dubins curves. The method is presented in detail in Section 4. For each point of the path, the heading (h_i, in radians; see Figure 4) is computed as follows:

$$\begin{cases} h_i = \mod\left(\frac{\pi}{2} - \arctan\frac{y_{i+1}-y_i}{x_{i+1}-x_i}, 2\pi\right), 0 < i < n-1 \\ h_0 = h_{start} \\ h_{n-1} = h_{landingSite} \end{cases} \quad (4)$$

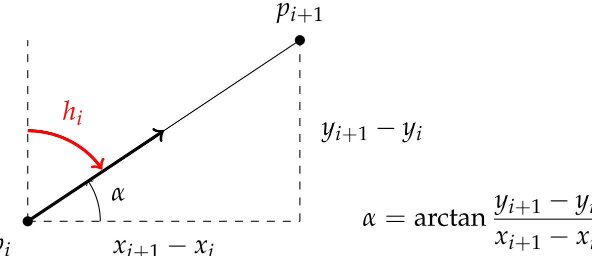


Figure 4. Heading computation: From the position of the points p_i and p_{i+1}, the orientation angle α is computed, followed by the heading h_i.

The heading enables the algorithm to connect the points with a Dubins curve and therefore take into account the curvature radius of the aircraft.

3.1.3. Objective Function

The goal of this study is to generate a flyable trajectory as quickly as possible. We chose to determine the shortest possible route. The selected objective function is the Euclidean distance squared. For a path $(p_0, p_1, \ldots, p_{n-1})$, the cost is computed as follows:

$$cost(p_0, p_1, \ldots, p_{n-1}) = \sum_{i=0}^{n-2} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2 \quad (5)$$

This choice was made to reduce the computing time required for a distance between two points as much as possible. Indeed, this computation is cheaper than the computation of a Dubins curve. This choice introduces error, as the Dubins curve distance can be very different from the Euclidean distance. However, it is important to remember that the algorithm does not aim to find the optimum path but rather a safe path.

3.2. Aeronautical Considerations

Depending on the situation, the constraints can be extremely different. For example, if the aircraft loses its engines, the dynamics of the aircraft are drastically modified (curvature radius, minimum descent rate. . .), whereas a rudder issue can just prevent turning left. The proposed algorithm takes into account the descent and the heading constraints.

3.2.1. Descent Constraints

Each type of emergency affects the descent in a different way. For example, in the case of an emergency due to a cabin fire, the aircraft's performance is not affected. Therefore, the aircraft can climb and descend as usual. However, in the case of an emergency due to a dual engine failure, the aircraft has to descend to maintain sufficient speed to avoid stalling. In this case, the pilots decide most often to choose the lowest possible descent rate so that they can cover the longest possible distance and have enough time to choose a safe landing site (see Figure 5). It is also constrained by a maximum descent rate. Indeed, if it is higher than the maximum value, it can lead to an overspeed. However, to land safely, the speed of the aircraft should not be too high.

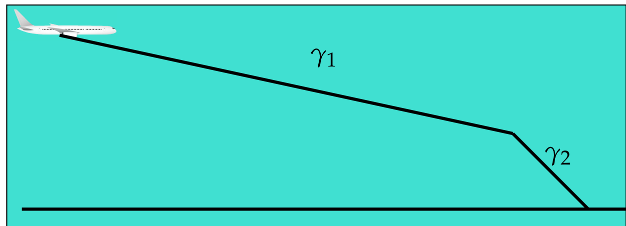


Figure 5. Example of descent profile: γ_1 corresponds to the minimal descent angle (Maximal lift-to-drag ratio) and γ_2 is the descent angle during the approach phase.

In this study, the aircraft is constrained by a maximal and a minimal descent angle (see Figure 6). During the building tree phase, to connect two points in the free space, the descent angle has to be checked between these two values.

The descent constraints can be written as follows:

$$\forall i < n \begin{cases} \arctan \dfrac{dist_{horizontal}(p_i, p_{i+1})}{|z_{i+1} - z_i|} \geq \gamma_{min} \\ \arctan \dfrac{dist_{horizontal}(p_i, p_{i+1})}{|z_{i+1} - z_i|} \leq \gamma_{max} \end{cases} \quad (6)$$

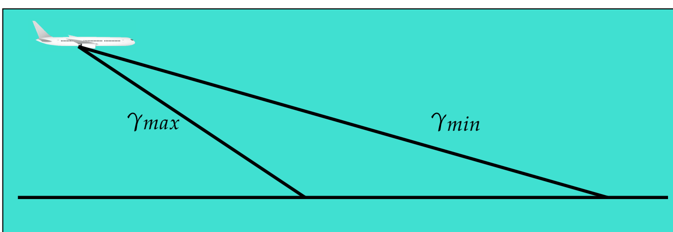


Figure 6. Descent constraint (minimal descent angle = γ_{min} and maximal descent angle = γ_{max}).

3.2.2. Heading Constraints

When obstacles are present (mountains, buildings, etc.), the aircraft has to change its heading to avoid them. The pilot, therefore, turns the aircraft to the right or to the left. The pilot chooses a bank angle (see Figure 7) between 0° and the maximum bank angle (θ_{max}), which depends on the features of the aircraft. This angle is maintained during the turn and finally, the pilot straightens the aircraft.

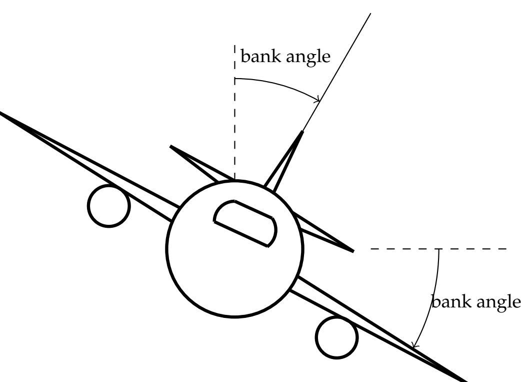


Figure 7. Bank angle.

The maximal bank angle is associated with another value, which is the minimum curvature radius r_{min} (see Figure 8). This value depends on the true airspeed v and the maximum bank angle θ_{max} as follows:

$$r_{min} = \frac{v^2}{g \tan \theta_{max}} \quad (7)$$

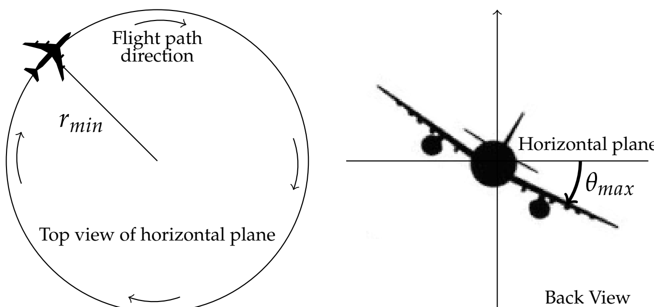


Figure 8. Link between the maximal bank angle and the minimal curvature radius.

The heading constraints taken into account are the following:

- At the time of the declaration of the emergency, the aircraft has a given orientation. The proposed trajectory has to start with this orientation.
- Throughout the flight, the aircraft has to avoid obstacles. For this, it makes turns which are constrained by the maximal bank angle, which depends on aircraft features. This value is associated with the minimal curvature radius. In this study, this radius is considered to be constant. The turns are modeled by means of Dubins curves.
- The final heading of the aircraft has to be the same as the runway or the landing site orientation.

All data allowing the consideration of these constraints are considered as known. They could be computed using an algorithm that takes into account the weather and the impact of a possible failure.

After a detailed description of the mathematical modeling, the next section presents the proposed algorithm to automatically generate a flyable trajectory and an improvement of the free space checker.

4. Flyable Trajectory Generation

4.1. Algorithm Description

The proposed method is based on the FMT algorithm. Each node N generated during the sampling phase is defined by its coordinates in the free space. The nodes contain two additional data to construct the structure of the tree. The first piece of information is the cost of the node N. During the sample phase, the cost value is initialized to infinity. The second piece of information is the parent node N_{parent}. This represents the previous node on the shortest path which connects node N. The sampling can be uniform or random. With a random sampling, the results are usually better than with a uniform deterministic sampling approach. However, for the same problem, two random samplings generate two different solutions. The proposed algorithm generates the trajectories from a random sampling in the free space. To obtain good solutions, the number of samples must be sufficiently high, but if this number is too high, the computing time is too long. To obtain a good solution while having a low computing time, the proposed algorithm generates a first sampling with a small number of points in order to quickly generate a first initial trajectory with the FMT algorithm.This solution is very far from the shortest path. It is therefore refined by means of a second sampling step, generated around it (see Figure 9).It can be noted that in the worst case, the second path is the same as the first.

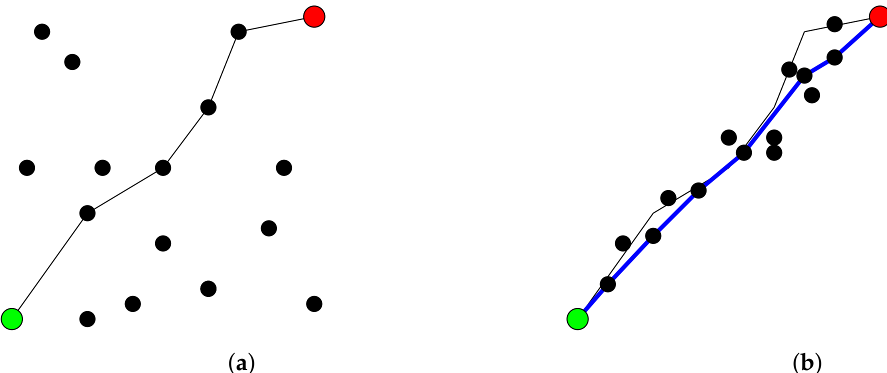


Figure 9. Sampling: The first path (**left**) is computed by the FMT algorithm from a sampling on the whole space. The second path (**right** in blue) is computed from a sampling around the first approximate path (start point in green and end point in red). (**a**) First sampling; (**b**) second sampling.

Finally, to make the trajectory flyable, all points that compose the previously computed path are connected by replacing straight lines with Dubins curves. The heading constraints are now considered. The connection of points with Dubins curves could be accomplished during the FMT phase, but as Dubins curve generation is too slow, the points are first connected by straight lines and then adjusted with Dubins curves (see Figure 10).

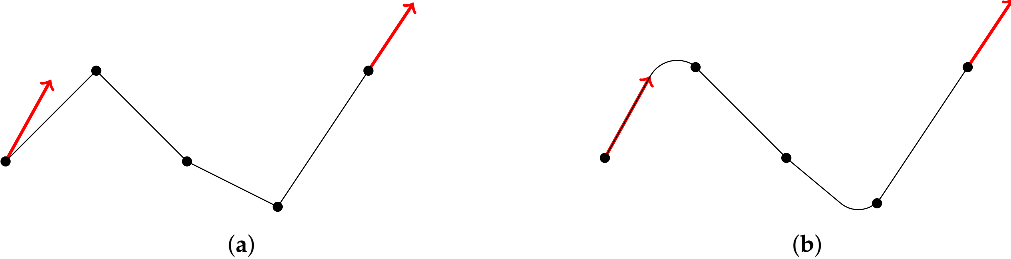


Figure 10. Path modification: The first path is computed by the FMT algorithm. The second path is the path after the addition of Dubins curves to respect the heading constraints (in red). (**a**) Path without Dubins curve; (**b**) Path with Dubins curve.

This process requires the enlargement of the obstacles in order to be sure that the Dubins curves will be in the free space. Indeed, if the straight line path passes near an obstacle, the Dubins path could pass through this obstacle due to the turn constraints (see Figure 11).

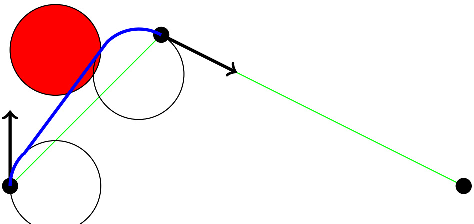
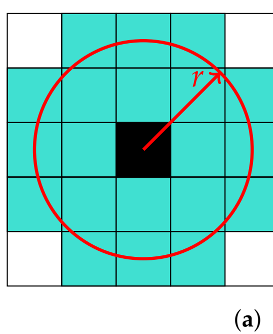
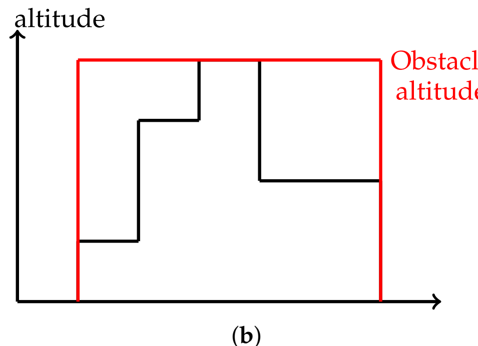


Figure 11. Example of collision with an obstacle after the addition of Dubins curves: the straight line path is drawn in green and is in the free space. The Dubins path is shown in blue and passes though the obstacle in red [7].

The obstacles have therefore been enlarged horizontally by a distance corresponding to the curvature radius of the aircraft trajectory. The cells located at a horizontal distance lower than the curvature radius of an obstacle cell become obstacles (See Figure 12a). Moreover, all cells under these cells are also considered obstacles. This implies that the altitude of this circular area is equal to the altitude of the obstacle cell (see Figure 12b).



(**a**)



(**b**)

Figure 12. Enlargement of obstacles [7]. (**a**) Horizontal enlargement (obstacle cell: black, new obstacle cells: blue); (**b**) vertical enlargement.

4.2. Improvement of the Free Space Checker

One of the main functions of the previously mentioned algorithm is the free space test function. It consists in verifying if there is an obstacle between the two points in order to validate the connection in the graph. FMT* spends half its time on this function. It is therefore critical to have a very efficient free space test function to reduce the computing time required for trajectory generation. One way to check that a segment between two points is in the free space is to discretize the segment and to check if all the points of the segment are in the free space. This method is very slow and therefore makes the algorithms less efficient. To reduce the computing time of this function, Quadtree (2D) and Octree (3D) have been used.

Without the loss of generality, this paper presents the free space checker in 2D to facilitate reader comprehension. However, the method is used in 3D to be applied to the studied problem. A quadtree (Figure 13) is a tree data structure in which each internal node has exactly four children. A quadtree is generated from a grid composed of free space cells and obstacle cells [34].

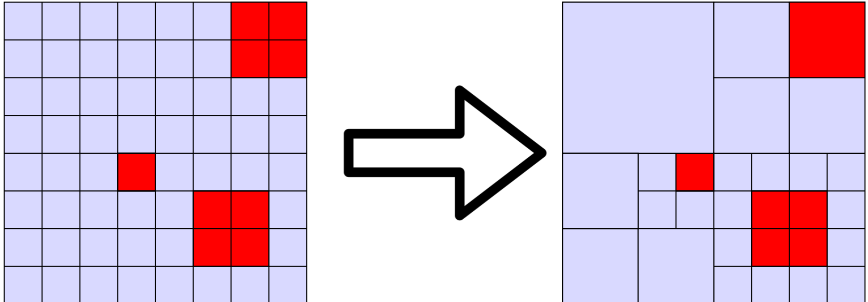


Figure 13. Example of a quadtree.

A naive method would be to code the Quadtree using pointers. The root node would point to four children nodes, each child would point to four children nodes, etc. However, if the grid is very large, memory usage is very high. To reduce memory usage, it is better to use a linear Quadtree [35]. This enables us to store for all leaves of the tree only two pieces of information for each of them. The first piece of information is called the Morton code [36–38]. This is an integer that uniquely defines a cell in a grid. The computation of its value consists in converting the row number and the column number of a cell into a binary string. Then, the Morton code is created by alternating a column digit and a row digit (see Figure 14a). The second piece of information is the depth level in the tree. This defines the position of the leaf in the tree and therefore also defines the cell size.

Linear Quadtree generation from a grid is composed of several steps. Firstly, the number of levels is computed. This value corresponds to the smallest integer nl such as $NbRows \leq 2^{nl-1}$ and $NbColumns \leq 2^{nl-1}$ (3^{nl-1} for octrees). Then, an empty list is created, which contains the Quadtree free nodes. This list is ordered in ascending order of Morton

Code. Then, for each Morton code $MC < 4^{nl-1}$ (8^{nl-1} for octrees), a new node is added to the list. Its Morton code is MC and its level is equal to $nl - 1$. At each step, as long as it is possible, the last four elements of the list are deleted and a new node is created. The level is decremented by one and its Morton code is the minimum Morton code of the previous nodes. If the grid is very large, the generation of this Quadtree is slightly long. However, once calculated, it is very easy to store it in a text file. This computation can be performed in a pre-processing phase. For example, during the flight the octree can be computed every 10 min around the aircraft position and its future positions (see Figure 15).

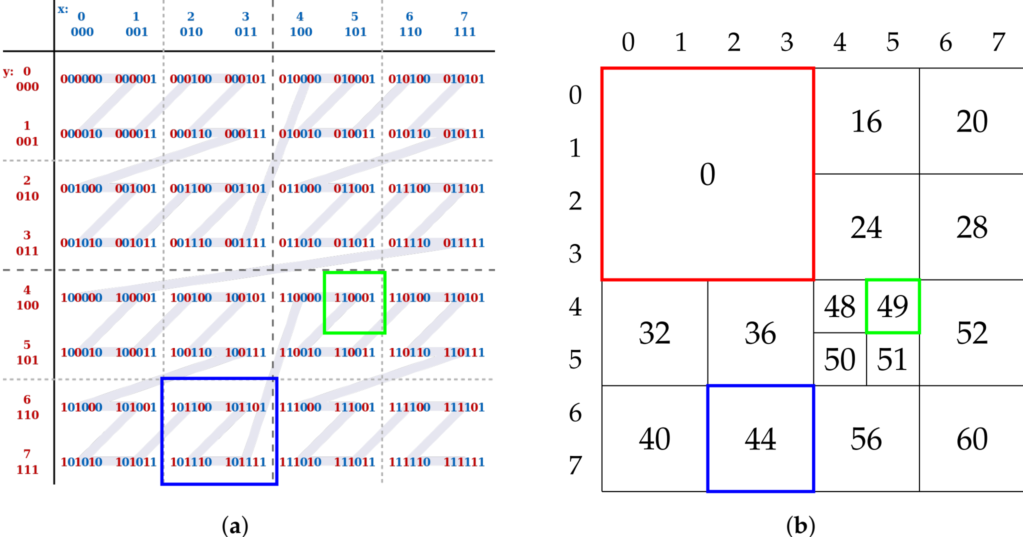


Figure 14. Linear quadtree generation from a grid. (**a**) Morton code computation: Its value is computed by alternating a column digit (blue) and a row digit (red) [39]. (**b**) Linear Quadtree example: This quadtree is computed from an 8×8 grid; it is therefore composed of 4 levels ($8 = 2^{4-1}$). The level of the red cell is 1 because its size is 4×4 and its Morton code is 0 because it is the minimum Morton code of cells which compose this big cell. The blue cell level is 2 (size = 2×2) and the green cell level is 3 (size = 1×1). Note that level 0 corresponds to the entire grid.

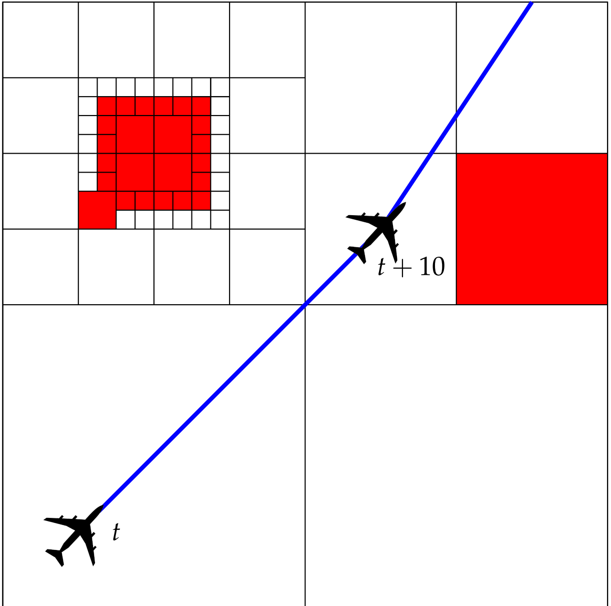


Figure 15. QuadTree around the aircraft position.

Figure 14b shows an example of a linear Quadtree. This Quadtree is built from an 8×8 grid. This implies that the level of the Quadtree is 4 ($8 = 2^{4-1}$). The level of the red cell is one because its size is 4×4 and its Morton is zero. The blue cell has a level equal to two (size = 2×2). This cell is composed of four grid cells ([6,2], [6,3], [7,2], [7,3]). In the table shown in Figure 14, the four Morton codes are 44, 45, 46, and 47. The Morton code of the cell is the minimum of these; therefore, it is 44. The green cell is composed of only one cell ([5,4]), its level is three, and its Morton code is 49.

The main function of the previously presented algorithms is to verify if a straight line segment is in the free space. The Quadtree offers a strong improvement as a free space checker.The first step of this function is to determine the Quadtree cells corresponding to the start and the end position of the straight line. The Morton code associated with a coordinate is computed with a table similar to that shown in Figure 14 (lines 1 and 2 in Algorithm 1). The associated cell is searched in the table of nodes (lines 3 and 4). If one node does not exist in the table—that is to say, it is an obstacle cell—the method returns False (line 5). Then, if these two cells are the same (line ABin Figure 16), the function returns True (Line 6). However, if the two cells are different, a dichotomy is carried out and the function is called recursively (line CD in Figure 16, lines 9 and 10).

If, in the initial grid, there are few obstacles, this method is very efficient and divides the computing time by ten but if the obstacle space is very large, the benefit is strongly reduced. On average, the computing time is divided by three.

After this detailed presentation of the proposed algorithm to generate an emergency path, it will now be compared to algorithms from the literature and then tested with a real case. The results are presented in the next section.

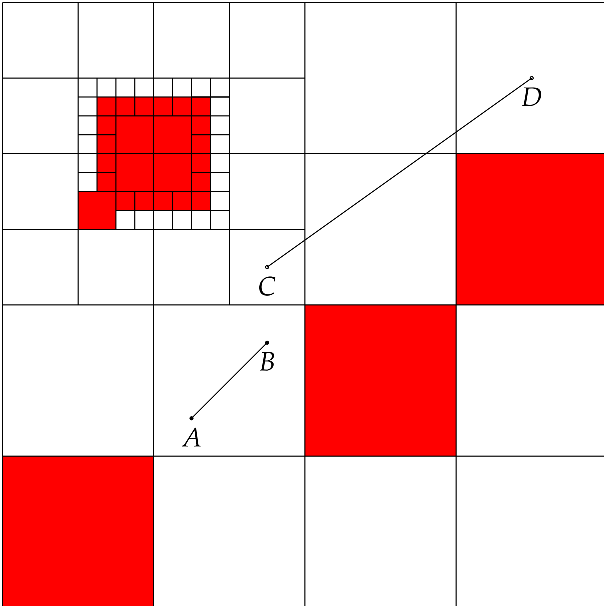


Figure 16. Free space checker example: in the case of the segment [AB], the octree cell is the same for A and B. However, for the segment [CD], the start cell is different from the end cell; therefore, a dichotomy is performed.

Algorithm 1 Free space checker: from the start position (*start*) and the end position (*end*), the two associated Morton codes are computed and then a dichotomy is performed to check that the segment is in the free space.

```
1:  StartMortonCode = CoordinateToMC(start)
2:  EndMortonCode = CoordinateToMC(end)
3:  StartNode = MCTab.get(startMortonCode)
4:  EndNode = MCTab.get(endMortonCode)
5:  if StartNode ≠ NULL and EndNode ≠ NULL then
6:    if StartNode = EndNode then
7:      return True
8:    else
9:      middle = middleCoordinate(start, end)
10:     return FreeSpaceCheck(start, middle) and FreeSpaceCheck(middle, end)
11:   end if
12: else
13:   return False
14: end if
```

5. Results

5.1. Description of the Test Method

Preliminary tests were conducted to demonstrate the efficiency of the proposed algorithm. The goal was to show that the proposed improvements decrease the computing time required for trajectory generation. The methodology proposed to compare FMT and the algorithm is illustrated by the cases of one flight crossing a 3D cube of $1000 \times 1000 \times 1000$. For these tests, the orientations, the start point, and the final point are given. The obstacles are very simple in order to easily observe the cost of the shortest path. In these early simulation tests, the Dubins curves are not computed and the descent constraints are not considered. The experiments were carried out with a computer equipped with an i7-8550 processor and a RAM of 8 GB. The algorithm was tested on a dozen simple scenarios with obstacles of several sizes to evaluate its computing-time efficiency. These scenarios were chosen because the optimal cost was known; therefore, the error was easily computed as: $error = \frac{c_{FMT} - c_{real}}{c_{real}}$, where c_{FMT} represents the cost of the path computed by the algorithm and c_{real} is the cost of the optimal path. The first presented case corresponds to two points (the origin and destination) separated by a large obstacle. This example shows the difference between the FMT tree and the tree in our algorithm. It explains the reduction in computation time. Then, we present a more complex scenario to show the avoidance of the obstacles. This scenario raises the limits of the algorithm if the sampling size is too small. After testing the algorithm on simple cases, it is tested on real cases to show that the proposed algorithm can take into account aeronautical constraints.

5.2. Simple Case Test

Figure 17 present the results obtained with the FMT algorithm. These two figures show that the FMT algorithm explored many unnecessary points. This implies that the number of points should be increased in order to compute a path that has a cost near the optimal cost. In this test case, with 3000 samples, the computing time was 589.6 ms and the error was 2%. Figure 18 presents the tree and the optimal path computed by the proposed algorithm (improved FMT). These figures show that the tree is less extended than the tree computed by the original FMT. This enables us to reduce significantly the number of samples and therefore the path-computing time. Indeed, for the same error rate (2%) the computing time is 208.6 ms.

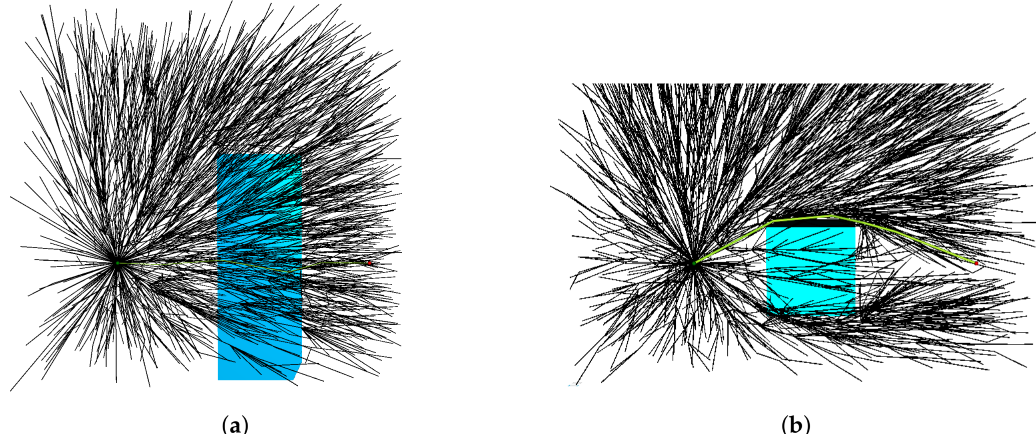


Figure 17. Example of the result of the FMT algorithm with one obstacle: number of samples = 3000, computing time = 589.6 ms, error = 2%. The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green. (**a**) Side view. The computed trajectory passes behind the obstacle; (**b**) top view.

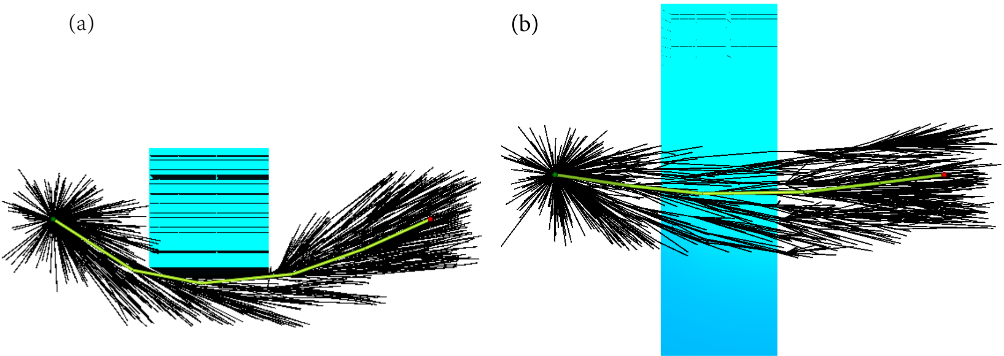


Figure 18. Example result of the improved FMT algorithm with large obstacle: first step number of samples = 1500, second step number of samples = 1000, computing time = 208.6 ms, error = 2%.The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green. (**a**) Top view; (**b**) side view.

5.3. Multi-Obstacle Case Test

Figure 19 shows another result obtained using the proposed algorithm in the presence of four different obstacles. This test was performed with a first sample composed of 3000 samples and then a second sample with 1500 points. This figure shows that the path horizontally and vertically avoided the obstacles.

The tests performed during this study showed that the proposed algorithm computed a good solution very quickly. However, in some specific cases, if the number of samples of the first sampling is too small, the first path could be different from the optimal path and therefore the second sampling could not correct the error.

Figure 20 shows an example of a result obtained using the algorithm with a small first sampling step (500 samples). The path is very far from the path in Figure 19. Indeed, it is 8% longer. To avoid this problem, the first sampling should have a sampling number that is higher than the second sampling step. Indeed, the second sampling step is just used to refine the first solution. In the worst case, the new solution is the same as the first.

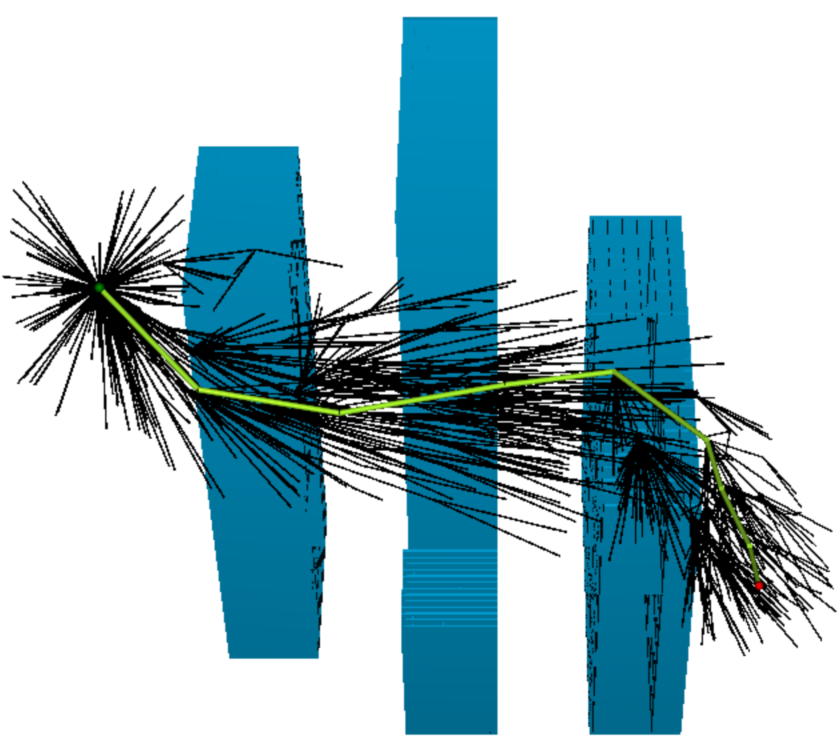


Figure 19. Example of the result of the improved FMT algorithm with four different obstacles: first step number of samples = 3000, second step number of samples = 1500, computing time = 1123.6 ms. The generated graph is drawn in black, the obstacles in blue and the path (start point in green and end point in red) in green.

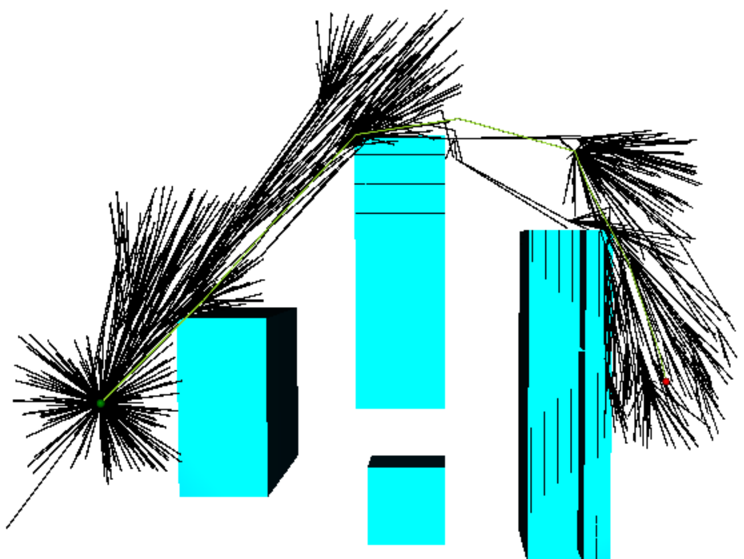


Figure 20. Example result of the improved FMT algorithm with four different obstacles: first step number of samples = 500, second step number of samples = 1500. The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green.

5.4. Computing Time Results

Other tests were performed to demonstrate the speed of the proposed algorithm. The test conditions were presented in Section 5.1. Table 2 summarizes the results of the simulations. It shows the computing time, depending on the error. These tests showed

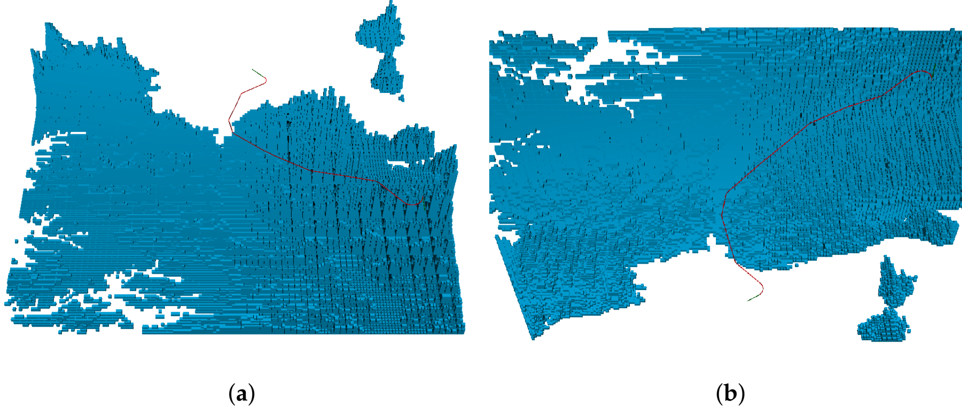
first that the proposed FMT was clearly faster than the original FMT. Indeed, the proposed algorithm reduced the computing time by three.

Table 2. Average computing time (ms) for scenarios between two points separated by a distance of 500 cells.

Error (%)	FMT	Proposed FMT
5	98.7	51.3
2	493.8	203.4
1	6896.5	2057.3

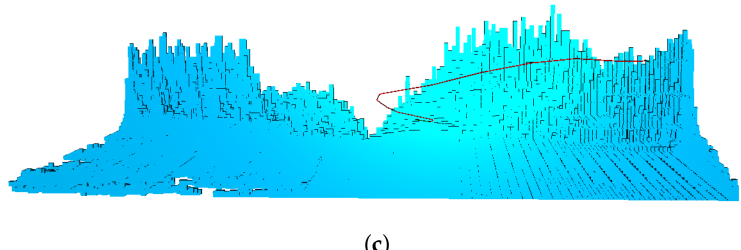
5.5. Real Case Test

After testing the algorithm on simple cases, it was tested on real cases. The data used for these tests were obtained from an altitude data map around Grenoble airport in France with a radius of 50 Nm. The data were represented by a 3D cube of $1200 \times 1200 \times 500$. Altitude data are given at each 0.083 Nm $(\frac{2 \times 50}{1200})$. The precision of the altitude was 30 ft. After a data processing step to enlarge the obstacles, the computed octree had about 6 million leaves and the file size was equal to 80 MB. The number of cells was reduced by 120 via the octree trick. Indeed, the initial cube contained 720 million cells ($1200 \times 1200 \times 500$). The absence of obstacles at high altitudes explains this significant reduction in the number of cells. They were therefore easily grouped to create large free-space cells. As explained in the last part, this speeds up the free space checker algorithm, and consequently, the trajectory generation algorithm. As previously stated, the landing site, the curvature radius and descent rates were considered known. Figure 21 represents one studied scenario. The initial altitude was 10,000 ft, the initial heading was 180°, the final altitude was the ground altitude and the final heading was 315°. These figures show that the trajectory avoids the mountains near to the emergency position and seems smooth with the addition of Dubins curves. However, this smoothing significantly increased the computing time required for the generation of a trajectory but it was still reasonable. The average computing time was less than 10 s. Ligny et al. [10] presented different computational performance tests on similar scenarios. Their algorithm generated a trajectory very quickly (about 1 s). However, their trajectories were not 100% flyable. Moreover, the algorithm was constrained to a fixed descent plane, which prevented U-turns. This can be problematic if the landing site is behind the aircraft. The proposed method is certainly slower but it computes a trajectory regardless of the position of the landing site and takes into account aeronautical constraints.



(**a**) (**b**)

Figure 21. *Cont.*



(**c**)

Figure 21. Trajectory generation example: The computed path is represented in red, the obstacles in blue, and the start heading and the final heading in green. (**a**) Example of emergency trajectory around Grenoble in France; (**b**) the same example seen from the other side; (**c**) other view of the trajectory to show the avoidance of the obstacle.

6. Conclusions

This paper addresses emergency geometric path generation. The study was focused on the efficiency of the trajectory generation in terms of computing time. In the first part of this paper, we presented some methods to generate a path. Sampling-based path planning algorithms seemed to be more appropriate for the studied problem due to their computation performance. However, the paths computed by this type of algorithm are not flyable. To solve this problem, the proposed algorithm uses Dubins curves. In the second part, we explained the mathematical modeling approach used for the problem and particularly the aeronautics constraints. In the next part we described the approach used to address the problem. The proposed algorithm is an adapted version of the fast marching tree, which used an octree for the free space checking function, Dubins curves to make the trajectory flyable, and two different sampling steps. The goal of these improvements was to reduce the computing time of the algorithm as much as possible. Finally, early simulation results were presented to illustrate the proposed solution. According to the numerical results, this approach looks promising. The proposed algorithm generated a trajectory from the emergency to the landing site in less than 10 s. Moreover, the proposed method is adaptable to any type of emergency. Indeed, the algorithm can take as its input any value of descent angle and radius of curvature. However, it will be necessary to develop a complete tool composed of three different modules, the selection of the landing site, the computation of the performance data (curvature radius and descent angle), and the generation of the trajectory. The first two modules will depend on the emergency and the weather. This paper focused on the last module. The proposed method seems promising for integration into a complete system because it is independent, fast, and adaptable. The final steps will consist in describing the computed path by means of a list of waypoints to be integrated into the FMS. This study opens the way to the integration of an emergency autolanding system into the FMS with in-flight automatic resolution of potential traffic conflicts, obstacle avoidance, and weather hazard avoidance. This remains a subject for future research. This type of method could be used throughout the flight. A landing site selection algorithm would compute a set of airports sorted by distance from the aircraft's position. The algorithm would then compute a trajectory from the future positions of the airplane by iterating over each airport until a solution is found. Since the algorithm provides a result in about 10 s, it could be considered to run the process every minute. With this method, the pilot would receive immediate help because the trajectory would already be determined. However, the large number of emergency types would require the repetition of this process for each type of failure. This would reduce the frequency of repeating the process. To limit this, the performance of the trajectory generation algorithm would have to be further improved.

Author Contributions: The contributions of the authors are the following: conceptualization A.G., D.D., and E.F.; methodology A.G.; software A.G.; writing A.G., D.D., and E.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Acknowledgments: The authors would like to thank Clean Sky for its support of the project SafeNcy: the safe emergency trajectory generator. The authors would also like to thank all the members of the project for their advice and their expertise.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Fast Marching Tree

Algorithm A1 Fast marching tree: from a sampling composed of n samples, the algorithm creates the graph. At each step, the minimum cost node is selected and the algorithm tries to connect these neighbours (nodes at a distance lower than a radius depending on the number of samples) to the graph [29].

1: $V \leftarrow \{x_{start}\} \cup SampleFree(n); E \leftarrow \emptyset$
2: $V_{unvisited} \leftarrow V \backslash \{x_{start}\}; V_{open} \leftarrow \{x_{init}\}, V_{closed} \leftarrow \emptyset$
3: $z \leftarrow x_{start}$
4: $N_z \leftarrow Near(V \backslash \{z\}, z, r_n)$
5: $Save(N_z, z)$
6: **while** $z \notin \chi_{goal}$ **do**
7: $V_{open,new} \leftarrow \emptyset$
8: $X_{near} = N_z \cap V_{unvisited}$
9: **for each** $x \in X_{near}$ **do**
10: $N_x \leftarrow Near(V \backslash \{x\}, x, r_n)$
11: $Save(N_x, x)$
12: $Y_{near} \leftarrow N_x \cap V_{open}$
13: $y_{min} \leftarrow \arg\min_{y \in Y_{near}} \{c(y) + Cost(y, x)\}$
14: **if** $CollisionFree(y_{min}, x)$ **then**
15: $E \leftarrow E \cup \{(y_{min}, x)\}$
16: $V_{open,new} \leftarrow V_{open,new} \cup \{x\}$
17: $c(x) = c(y_{min}) + Cost(y_{min}, x)$
18: **end if**
19: **end for**
20: $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{z\}$
21: $V_{closed} \leftarrow V_{closed} \cup \{z\}$
22: **if** $V_{open} = \emptyset$ **then**
23: **return** Failure
24: **end if**
25: $z \leftarrow \arg\min_{y \in V_{open}} \{c(y)\}$
26: **end while**
27: **return** $Path(z, T = (V_{open} \cup V_{closed}, E))$

References

1. ChrisnHouston. Trajet du vol US Airways 1549 le 15 janvier 2009. 2019. Available online: https://commons.wikimedia.org/wiki/File:Flight_1549-OptionsNotTaken.PNG (accessed on 5 February 2022).
2. Atkins, E.M.; Portillo, I.A.; Strube, M.J. Emergency Flight Planning Applied to Total Loss of Thrust. *J. Aircr.* **2006**, *43*, 1205–1216. [CrossRef]
3. Atkins, E.M. Emergency Landing Automation Aids: An Evaluation Inspired by US Airways Flight 1549. *AIAA Infotech Aerosp. 2010* **2010**, *2010*, 3381.
4. Tang, P.; Zhang, S.; Li, J. Final Approach and Landing Trajectory Generation for Civil Airplane in Total Loss of Thrust. *Procedia Eng.* **2014**, *80*, 522–528. [CrossRef]
5. Fallast, A.; Messnarz, B. Automated trajectory generation and airport selection for an emergency landing procedure of CS23 aircraft. *DEAS Aeornautical J.* **2017**, *8*, 481–492. [CrossRef]

6. Zhao, Y. Efficient and Robust Aircraft Landing Trajectory Optimization. Ph.D. Thesis, Georgia Institute of Technology, Atlanta, Georgia, 2012.
7. Sáez, R.; Khaledian, H.; Prats, X.; Guitart, A.; Delahaye, D.; Feron, E. A Fast and Flexible Emergency Trajectory Generator Enhancing Emergency Geometric Planning with Aircraft Dynamics. In Proceedings of the Fourteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2021), New Orleans, LA, USA, 20–23 September 2021.
8. Haghighi, H.; Delahaye, D.; Asadi, D. Performance-based emergency landing trajectory planning applying meta-heuristic and Dubins paths. *Appl. Soft Comput.* **2022**, *117*, 108453. [CrossRef]
9. Coxeter, H.S.M. The Problem of Apollonius. *Am. Math. Mon.* **1968**, *75*, 5–15. [CrossRef]
10. Ligny, L.; Guitart, A.; Delahaye, D.; Sridhar, B. Aircraft Emergency Trajectory Design: A Fast Marching Method on a Triangular Mesh. In Proceedings of the Fourteenth USA/Europe Air Traffic Management Research and Development Seminar, New Orleans, LA, USA, 20–23 September 2021.
11. Hong, H.; Piprek, P.; Gerdts, M.; Holzapfel, F. Computationally Efficient Trajectory Generation for Smooth Aircraft Flight Level Changes. *J. Guid. Control. Dyn.* **2021**, *44*, 1532–1540. [CrossRef]
12. Woo, J.W.; An, J.Y.; Cho, M.G.; Kim, C.J. Integration of path planning, trajectory generation and trajectory tracking control for aircraft mission autonomy. *Aerosp. Sci. Technol.* **2021**, *118*, 107014. [CrossRef]
13. Qureshi, A.; Ayaz, Y. Potential Functions based Sampling Heuristic For Optimal Path Planning. *Auton. Robot.* **2016**, *40*, 1079–1093. [CrossRef]
14. Girardet, B.; Lapasset, L.; Delahaye, D.; Rabut, C. Wind-optimal path planning: Application to aircraft trajectories. In Proceedings of the 2014 13th International Conference on Control Automation Robotics & Vision (ICARCV), Singapore, 10–12 December 2014; pp. 1403–1408. [CrossRef]
15. González, V.; Monje, C.A.; Moreno, L.; Balaguer, C. Fast Marching Square Method for UAVs Mission Planning with consideration of Dubins Model Constraints. *IFAC-PapersOnLine* **2016**, *49*, 164–169. [CrossRef]
16. Yu, Y.H.; Zhang, Y.T. Collision avoidance and path planning for industrial manipulator using slice-based heuristic fast marching tree. *Robot. -Comput.-Integr. Manuf.* **2022**, *75*, 102289. [CrossRef]
17. Tehrani, N.D.; Cherepinsky, I.; Carlson, S. Closed-loop Fast Marching Tree (CL-FMT*) with Application to Helicopter Landing Trajectory Planning. In Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 27 September–1 October 2021; pp. 346–351. [CrossRef]
18. Kang, J.G.; Choi, Y.S.; Jung, J.W. A Method of Enhancing Rapidly-Exploring Random Tree Robot Path Planning Using Midpoint Interpolation. *Appl. Sci.* **2021**, *11*, 8483. [CrossRef]
19. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
20. Ford, L.R., Jr. *Network Flow Theory*; RAND Corporation: Santa Monica, CA, USA, 1956.
21. Bellman, R. On a routing problem. *Q. Appl. Math.* **1958**, *16*, 87–90. [CrossRef]
22. Hart, P.; Nilsson, N.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
23. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT* : Optimal Sampling-based Path Planning Focused via Direct Sampling of an admissible Ellipsoidal Heuristic. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 14–18 September 2014.
24. Gammell, J.; Srinivasa, S.; Barfoot, T. Bit* : Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation, Seattle, WA, USA, 26–30 May 2015.
25. Pharpatara, P.; Hérissé, B.; Bestaoui, Y. 3-D Trajectory Planning of Aerial Vehicles Using RRT*. *IEEE Trans. Control. Syst. Technol.* **2017**, *25*, 1116–1123. [CrossRef]
26. Karaman, S.; Frazzoli, E. Incremental sampling-based algorithms. In *Robotics Science and Systems VI*; MIT Press: Cambridge, MA, USA, 2010.
27. Gammell, J.D.; Strub, M.P. Asymptotically Optimal Sampling-Based Motion Planning Methods. *Annu. Rev. Control. Robot. Auton. Syst.* **2021**, *4*, 295–318. [CrossRef]
28. Karaman, S.; Frazzoli, E. Sampling-based Algorithms for Optimal Motion Planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [CrossRef]
29. Janson, L.; Schmerling, E.; Clark, A.; Pavone, M. Fast Marching Tree : A Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimension. *Int. J. Robot. Res.* **2015**, *34*, 883–921. [CrossRef]
30. Huifang, W.; Lucia, P.; Antonio, B. Motion planning for Formations of Dubins Vehicles. In Proceedings of the 49th IEEE Conference on Decision and Control, Atlanta, GA, USA, 15–17 December 2010.
31. Gianfranco, P. Shortest paths for Dubins vehicles in presence of via points. *IFAC-PapersOnLine* **2019**, *52*, 295–300.
32. Satyanarayana, G.; Manyam, D.C.; Von Moll, A.L.; Fuchs, Z. Shortest Dubins Path to a circle. In Proceedings of the AIAA Scitech 2019 Forum, San Diego, CA, USA, 7–11 January 2019.
33. Le Ny, J.; Feron, E.; Frazzoli, E. On the Dubins Traveling Salesman Problem. *IEEE Trans. Autom. Control.* **2012**, *57*, 265–270. [CrossRef]
34. Baklouti, Z. Système de Planification de Chemins Aériens en 3D: Préparation de Missions et Replanification en cas d'Urgence. Ph.D. Thesis, Université Polytechnique Hauts de France, Valenciennes, France, 2018.

35. Kunio, A.; Koyo, M.; Shintaro, K.; Ryosuke, K.; Jia, F. Constant time neighbor finding in quadtrees : An experimental result. In Proceedings of the 2008 3rd International Symposium on Communications, Control and Signal Processing, Saint Julian's, Malta, 12–14 March 2008.
36. Morton, G.M. *A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing*; Technical Report; IBM: Ottawa, ON, Canada, 1966.
37. Gargantini, I. An effective way to represent quadtrees. *Commun. ACM* **1982**, *25*, 905–910. [CrossRef]
38. Chang, H.K.C.; Liu, J.L. A linear quadtree compression scheme for image encryption. *Signal Process. Image Commun.* **1997**, *10*, 279–290. [CrossRef]
39. Eppstein, D. Z-Order Curve. 2010. Available online: https://commons.wikimedia.org/wiki/File:Z-curve.svg (accessed on 5 February 2022).