Aircraft Emergency Trajectory Design: A Fast Marching Method on a Tetrahedral Mesh

Lucas Ligny, Andréas Guitart, Daniel Delahaye, Banavar Sridhar

Aircraft Emergency Trajectory Design: A Fast Marching Method on a Tetrahedral Mesh

Lucas Ligny *, Andréas Guitart † and Daniel Delahaye ‡
École Nationale de l'Aviation Civile (ENAC), Toulouse, France

Banavar Sridhar §
USRA@ NASA Ames Research Center, Moffett Field, CA, USA

WHEN a situation of emergency is declared during a flight, pilots must perform an emergency procedure which leads them to revise the initial trajectory of the aircraft, in order to land as quickly and safely as possible. This paper proposes an algorithm that efficiently generates such a trajectory, which would be part of a complete emergency tool implemented in the Flight Management System. The aim of this algorithm is to relieve pilots in such critical situations by providing an additional decision support tool for pilots. The main issues for this algorithm are the computation performance, the size of the stored data, and the flyability of the returned trajectory. For these reasons, the proposed algorithm is based on the three-dimensional Fast Marching method. Indeed, this method computes trajectories within seconds while requiring little memory space during computations. Numerical tests have been run with data from the mountainous region of Grenoble in France. Results are promising, as the algorithm computes within seconds short, flyable and safe trajectories in response to a critical emergency.

Nomenclature

δ = Level differences of an octree node [-]

γ = Flight path angle [rad]

f = Slowness [s/m]

λ = Interpolation parameter [-]

r = Radius of turn [m]

σ = Heading angle [rad]

T = Cost [s]

v_h = Horizontal airspeed [m/s]

*Graduate of the ENAC, OPTIM Laboratory, Toulouse France, ligny.lucas9@gmail.com.
†Ph.D candidate at ENAC, OPTIM Laboratory, andreas.guitart@enac.fr.
‡Professor and Director of the OPTIM Laboratory at ENAC, Toulouse, France, daniel@recherche.enac.fr.
§Research Associate at the NASA Ames Research Center, bensridhar@gmail.com.

I. Introduction

New technologies guarantee that a maximum level of safety is achieved inside an aircraft, including automatic controls that significantly reduce the mental load on pilots. Thereby pilots must remain vigilant throughout the entire flight, especially at takeoff, climb and descent since they are the most eventful phases. The framework of this paper is the optimisation of descent trajectories when a situation of emergency does occur on board an aircraft. The generation of the trajectory must be fast to allow pilots to react quickly and maximise the chances of a successful landing. This problem is deeply linked with the CleanSky SafeNcy project [1], which aims to develop an advanced Flight Management System (FMS) onboard function to help pilots take efficient and effective decisions in emergency situations and adverse conditions.

Aeronautical constraints have to be taken into account in the development of the algorithm. Providing bounds for the aircraft flight, stall and range limitations are the most critical constraints. These constraints, linked with the aircraft fuel reserve and its maximum and minimum descent rate, must be satisfied. The smoothness of the computed trajectory is also a challenge: it has to be flyable. Considering such constraints, there is a fundamental need for the algorithm to be efficient in terms of computing time while providing a high quality solution. All computations needed to be done on a FMS, then the balance between speed, accuracy and algorithmic cost is at the core of this study. This study will not address computations of aircraft performance impacted by the potential failure. These data will therefore be considered as known and will be the inputs of the proposed algorithm. This paper mainly focuses on the computation time optimisation of the trajectory generation. The goal of this study is the development of an adaptable algorithm to take as input any type of data.

One can find examples of critical emergency situations in aviation history. One well-known accident is the Swissair Flight 111 (September 2, 1998), where an on-board cockpit-fire caused by arcing resulted in a loss of control of the aircraft instruments, and subsequently the crash of the aircraft. Another example is the US Airways Flight 1549 (January 15, 2009), where pilots successfully ditched their A320 in the Hudson River shortly after take-off, in response to a bird strike causing the loss of all engine power (see Fig. 1). This case is very interesting because, in the space of 30 seconds, the situation went from critical to impossible. The only solution was to land on the Hudson River. This example shows the importance of having a very fast algorithm to generate the emergency trajectory.

These two events correspond to two separate types of emergencies: As Soon As Possible (ASAP) and At Nearest Suitable Airport (ANSA). ASAP emergencies are the most critical because the pilot has to find the fastest way to land. For example, an on-board or an urgent medical issue are considered ASAP. On the other hand, when considering ANSA emergencies such as a loss of engines, the pilot has to find the safest path, to land in the best possible conditions.

The structure of this paper is the following. First, a state of the art regarding trajectory generation is presented in Section II. Then, the mathematical modeling of the problem is described in Section III. Thereafter, the resolution

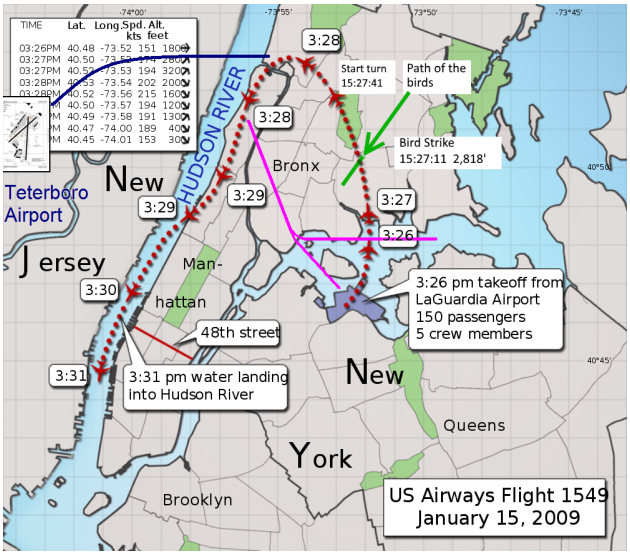


Fig. 1 US Airways Flight 1549. 2 minutes after take-off from LaGuardia airport, the aircraft struck a flock of Canada geese at an altitude of 2,818 feet. 5 minutes after take-off, the aircraft made an unpowered ditching into the Hudson River [2].

algorithm based on a Fast Marching method is described. In order to facilitate its understanding and to justify its use, the resolution algorithm is first presented in its two-dimensional version in Section IV and then in its three-dimensional version in Section V. Both versions are able to take into account wind fields and, to some extent, aeronautical constraints. Finally, a validation of the algorithm is proposed in Section VI for a key scenario in a mountainous region, along with some performance indicators to check the efficiency of the method.

II. State of the Art

A. Trajectory generation algorithms

1. Graph-based approaches

The Shortest Path Problem (SPP) has been studied during the second half of the twentieth century. One of the first well-known discrete algorithms to compute a path between two points is the Bellman-Ford algorithm [3–5]. Bellman-Ford algorithm is very general, in the sense that it is able of handling graphs with negative edge weights.

Bellman-Ford runs in $O(|\mathcal{V}||\mathcal{E}|)$, where $\mathcal{V}$ corresponds to the vertex set of the graph, $\mathcal{E}$ to the edge set. On a graph where all the weights are positive, a better algorithm is Dijkstra's [6], which has a faster convergence of $O(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$. Knowing the destination vertex d, one can implement the A* search [7], an algorithm that remains similar to Dijkstra's but achieves convergence in $O(|\mathcal{E}|)$. In order to apply A*, a lower bound of the distance to d must be available for all points in the search space . This bound reduce strongly the exploration of the search space. These methods use a given graph to compute the path.

Many methods propose to generate a graph in order to find the optimal path. These algorithms are called sampling

based path planning [8–10]. The main methods are Probabilistic RoadMaps (PRM), Rapidly Exploring Random Tree (RRT) and Fast Marching Tree (FMT) (presented in detail in Section V) . Research on this type of methods was first driven by robotics. The goal was initially to find the shortest possible path for robots to avoid obstacles. However, RRT* has been also used for aeronautical applications. In 2017, Pharpatara *et al.* [11] used RRT* to compute obstacle avoidance trajectories in 3D. However, the authors consider the wind to be negligible given the speed of the aircraft. The 3D wind track using RRT was however studied earlier, for glider-type drones by Chakrabarty and Langelaan [12]. In their paper, encouraging and interesting results are shown for maximising the glider's flight time or getting it to reach a given point by using air currents.

Nevertheless, these algorithms compute the geodesic distance on the graph, hence they suffer from grid bias. For real-life applications such as the optimisation of aircraft trajectories over tens or hundreds of nautical miles, a good idea is to break free from such error sources by working with a "continuous" algorithm.

2. Front propagation approaches

Graph-based methods, though efficient in computation time, are sensitive to the precision of the discretisation. To avoid such problems, one can work with semi-continuous methods such as Fast Marching.

These methods are simple, efficient and quite flexible. They are based on the physical propagation of a wavefront in a given domain Ω, containing obstacles. The front $\partial\Omega$ fixes the minimum cost to reach any point in space. To propagate the front, one has to solve the Eikonal equation:

$$|\nabla T| = f \tag{1}$$

where T is the cost function to reach any point in space, and f is the "slowness" of the domain at any point. f characterises some parts of the domain that are less accessible than others: in graph-based methods such areas are modeled with discrete edge weights, here they are defined by a continuous function over the domain. A good analogy is that of forest fires: the dryest the area, the fastest the propagation.

The Fast Marching algorithm is described in Algorithm 1.

One can note that line 14 of Algorithm 1 corresponds to the *Update Procedure*, in which the Eikonal equation is solved at the considered point, thus the T values of its neighbors are updated. This algorithm is illustrated in Fig. 2.

Ordered Upwind methods, developed by Sethian in [14], are very similar to Fast Marching methods. They consider an initial optimal problem, which allows to rewrite Eq. (1) as:

$$||\nabla u(X)||F\left(X, \frac{\nabla u(X)}{||\nabla u(X)||}\right) = 1 \tag{2}$$

Algorithm 1 Fast Marching [13]

```
Alive ← ∂Ω
Close ← N(Alive)
Far ← Ω \ {Alive ∪ Close}
while Far ≠ ∅ do
    Trial ← ω ∈ Close with the smallest T value.
    Alive ← Alive + {Trial}
    Close ← Close \ {Trial}
    for neighbor ∈ N({Trial}) do
        if neighbor ∉ Alive then
            if neighbor ∈ Far then
                Far ← Far \ {neighbor}
                Close ← Close + {neighbor}
            end if
            Update( neighbor )
        end if
    end for
end while
```

with X a point in space (usually $\mathbb{R}^2$ or $\mathbb{R}^3$), u is the cost function and F the propagation speed of the front defined by:

$$F\left(X, \frac{\nabla u(X)}{||\nabla u(X)||}\right) = \max_a \left(\frac{-\nabla u(X)}{||\nabla u(X)||} v(X, a)\right) \tag{3}$$

where v is the speed of the mobile, typically the ground speed of the aircraft and a is the unit vector determining the direction of motion.

The cost function u is computed on a discretised triangular grid in the increasing order of its value based on the causality principle. This principle is the following: "If $u(X)$ is computed from points X_j and X_k (from the triangle XX_jX_k), then $u(X) \geq \max(u(X_j), u(X_k))$". To maintain the causality principle, the cost $u(X)$ must therefore be computed from the triangle and thus the support line for the gradient of u must intersect the triangle (See Fig. 3).

This method has been used by Girardet [15] to solve the problem of aircraft trajectory design in the presence of wind.

B. Aircraft emergency trajectory design

Atkins et al. [16] provides an Adaptive 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 to generate geometric trajectories built with Dubins curves [17]. This algorithm has been applied to Flight 1549 in [18], and the algorithm returned a solution that could have enabled a safe return to LaGuardia airport. In [19], the Two Points Boundary Value Problem (TPBVP) is solved in to generate unpowered landing trajectories and improve aircraft safety. These articles propose real-time solutions for an aircraft in a situation of emergency, to be run independently of the FMS. In this paper, the current FMS is used to implement the trajectory generation algorithm.

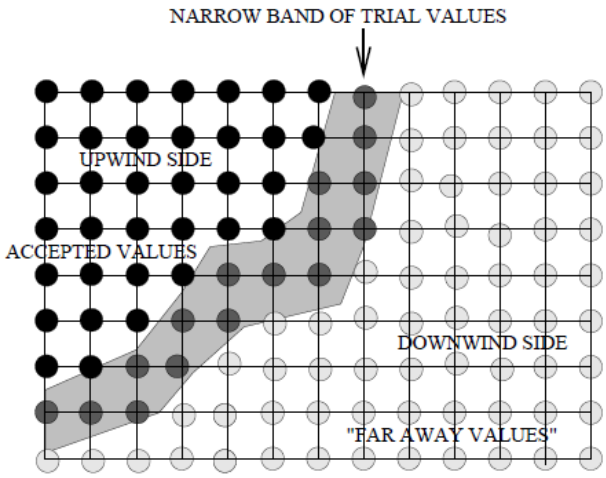
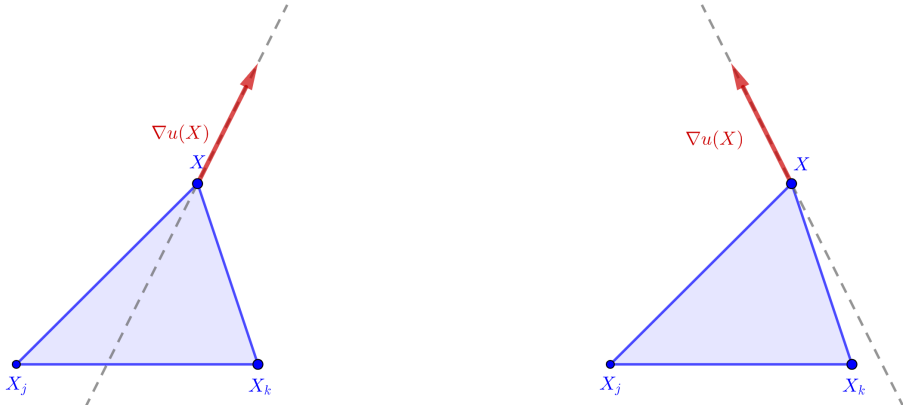


Fig. 2 Upwind construction of accepted values [13, Fig. 14]. In the narrow band of trial values (*Close* in Alg. 1), the Eikonal equation is solved. The front propagates from accepted values (*Alive* in Alg. 1) towards "far away" values (*Far* in Alg. 1).



(a) Causality principle respected **(b) Causality principle violated**

Fig. 3 Causality principle.

Fallast and Messnarz [20] propose a Rapidly-exploring Random Tree algorithm (RRT) to automatically select a landing site and generate an emergency trajectory. The RRT algorithm creates an entire graph from a single vertex and no edges. Vertices are sampled iteratively in the free space and if connections are possible, edges are added to the graph. Their algorithm is designed to converge towards an optimal solution, thus it is denoted RRT*. They are able to manage three different constraints: terrain avoidance, airspace restrictions and aircraft capabilities. To take them into account, they alter the connections between points by introducing Dubins curves [17], and limit these connections by considering the maximum climb and descent rates.

Guitart et al. [21] use Fast Marching Tree (FMT) approach to generate emergency trajectories. The FMT algorithm performs a forward dynamic programming recursion over several sampled points generated during the initialisation step

and generates a tree of paths. Again, Dubins curves are added to the process to make the trajectories flyable, and the maximum climb rate, the maximum descent rate but also the minimum radius of turn are taken as constraints.

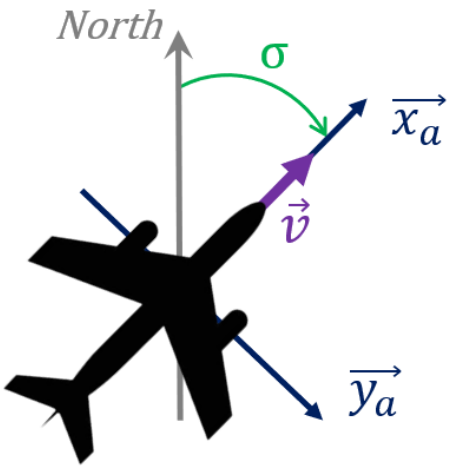
Sáez et al. [22] propose, from a given descent profile and a minimum curvature radius of the aircraft, a method based on RRT* to generate emergency trajectory. The data (profile and radius) depends on the type of emergency.

Haghighi et al. [23] present a post-failure performance analysis and develop an algorithm to generate emergency trajectories. Their method is based on Dubins curves and Apollonius results.

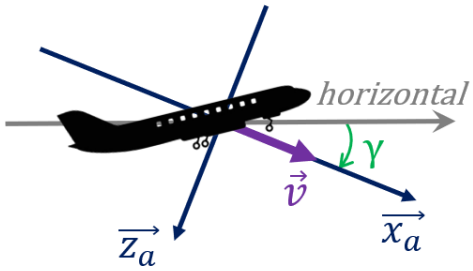
These previous related works have shown that many approaches can be used to generate a trajectory. Most of these works focus on the impact of the failure, sometimes to the detriment of the computing time. That's why this article proposes a very fast algorithm to generate emergency flyable trajectories which could be integrated into a complete system. This system would be composed of a landing sites selector, an aircraft performance generator and a path generator. This paper presents an algorithm that can be used as the path generator. In the following section, the mathematical modeling is described. From the equations of Flight Dynamics, one can state the optimisation problem to solve in order to obtain the optimal three-dimensional path.

III. Mathematical Modeling

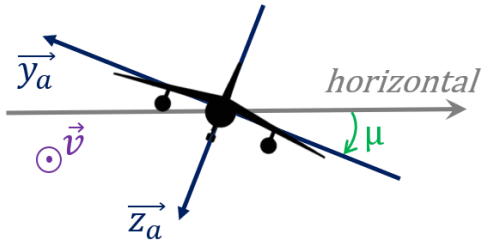
Let us consider an aircraft in the aerodynamic frame, with heading angle σ, flight path angle γ and bank angle μ (see Fig. 4).



(a) Top view of an aircraft with a heading angle σ.



(b) Side view of an aircraft with a flight path angle γ.



(c) Front view of an aircraft with a bank angle μ.

Fig. 4 The aerodynamic frame $(\overrightarrow{x_a}, \overrightarrow{y_a}, \overrightarrow{z_a})$. The frame origin is at the aircraft center of gravity, and the velocity vector $\overrightarrow{v}$ (the aircraft true airspeed) is along $\overrightarrow{x_a}$.

In the free space, the aircraft is constrained by a maximum rate of climb (RoC), a maximum rate of descent (RoD) and a minimum radius of turn $r_{\min}$.

A flight path angle constraint can be stated: one can define a maximum flight path angle $\gamma_{\max}$ from the RoC, and a minimum flight path angle $\gamma_{\min}$ from the RoD. This constraint is expressed as follows:

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max} \tag{4}$$

The aircraft is also bound to make turns during its descent. These gliding turns are constrained by the minimum radius of turn $r_{\min}$. By definition, the radius of turn is

$$r = v_h \left(\frac{\partial \sigma}{\partial t} \right)^{-1} \tag{5}$$

where v_h is the horizontal airspeed of the aircraft.

Thus, a second constraint can be stated, now limiting the variations of the heading angle σ:

$$\left| \frac{\partial \sigma}{\partial x} \right| \leq \frac{1}{r_{\min}} \tag{6}$$

In the context of path planning in three dimensions (here $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a basis of $\mathbb{R}^3$), given a starting point P_{start} and an ending point P_{end}, one can find the optimal path by solving the following optimisation problem:

$$T(x, y, z) = \min_{\pi \in \Pi} \int_{P_{start}}^{P_{end}=(x,y,z)} f(\pi(\tau)) d\tau \tag{7}$$

subject to constraints (4) and (6),

where $T(x, y, z)$ corresponds to the minimal cost required to travel from P_{start} to P_{end} considering the cost function f, and Π is the set of paths connecting P_{start} and P_{end}.

Considering a situation of emergency, it is possible that the aircraft loses all of its power. In such a situation, the maximum flight path angle is constrained by the maximum lift-to-drag (L/D) ratio of the aircraft. $(\mathrm{L/D})_{\max}$ characterises the ability of an aircraft to glide.

Defining by $\gamma_{\max}^{\text{gliding}} = -\arctan \dfrac{1}{(\mathrm{L/D})_{\max}}$, the constraint (4) can be reexpressed as follows:

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max}^{\text{gliding}} \tag{8}$$

The value of $\gamma_{\max}^{\text{gliding}}$ is always negative, which means that the aircraft is forced to go down.

In the next two sections, the resolution algorithm is presented first in two dimensions, then in three dimensions. The aim is to present the algorithm features with the simplest framework in Section IV, before tackling the extended

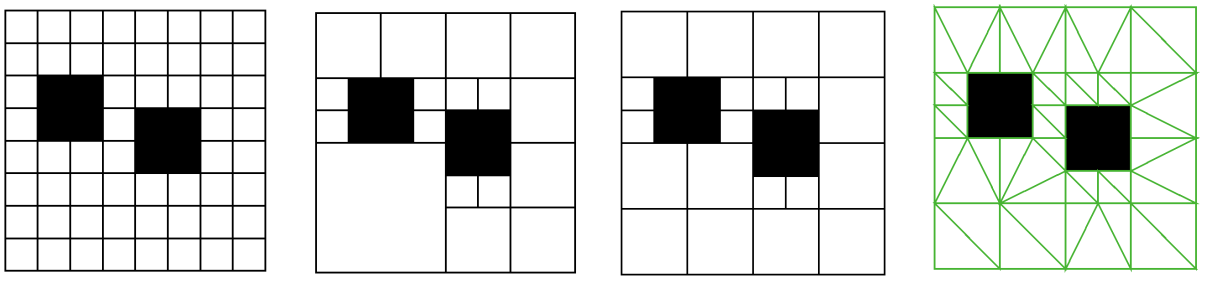
procedures in Section V.

IV. Resolution algorithm in two dimensions

The aim of this section is to introduce the algorithm in its two-dimensional version, first presented in [24]. This algorithm was proposed to generate an emergency trajectory by using a single descent plan, thus reducing the study to a two-dimensional problem. In order to obtain a smooth and flyable trajectory while being efficient with the computation time and the data storage, the space was discretised using triangular meshes built over quadtrees.

A. Data structure

A first approach to discretise a two-dimensional space could be to create a grid composed of free-space cells and obstacle cells. Free-space cells represent parts of the space where the trajectory is allowed to go through, and obstacles cells where it is not. An example of such a grid is represented in Fig. 5a, where the obstacles are shown in black, and the free-space in white. For real-life applications, grids must contain several thousand rows and columns in order to be accurate enough. To lighten the data, an optimised structure called *quadtree* is generated. Originally formalised in [25], a quadtree is a tree data structure in which each internal node has exactly four children. They are most often used to partition a two-dimensional space by recursively subdividing it into four quadrants or regions. A quadtree based on the grid example is shown in Fig. 5b. To avoid obtaining a bad approximation of the optimal trajectory, it is preferable to limit large size differences between two neighboring cells. This means balancing the quadtree by ensuring a level (depth of subdivision) difference of at most 1 between two neighboring cells in the four cardinal directions. The level differences can only be '-1', '0', '1' and '#', respectively representing a neighbor with lower, equal, higher level or an obstacle. The balanced quadtree for the ongoing example is shown in Fig. 5c. Finally, a triangular mesh can be obtained quite directly using a Delaunay triangulation over the balanced quadtree, as shown in Fig. 5d. This mesh has good properties; especially the generated triangles are all acute [26], which makes it quite suitable for the execution of a Fast Marching method.



(a) A 2D grid (of size 8x8). **(b) A quadtree computed from (a).** **(c) A balanced quadtree computed from (a).** **(d) A triangular mesh adapted to the terrain, computed from (a).**

Fig. 5 Refinement of the data structure modeling a two-dimensional space.

B. Fast Marching procedures

Fast Marching methods solve the optimisation problem by taking advantage of the fact that minimal cost paths are orthogonal to the level curves. It is expressed by the 2D Eikonal equation, which writes as follows:

$$|\nabla T| = f(x, y) \tag{9}$$

To solve this equation in a discretised environment, the Fast Marching method is used along with the following upwind scheme, close to finite difference approximation which is called *quadratic equation*:

$$\max\left(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0\right)^2 + \max\left(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0\right)^2 = f_{i,j}^2 \tag{10}$$

where the forward and backward operators are given by:

$$\begin{aligned} D_{ij}^{-x}T &= (T_{i,j} - T_{i-1,j})/\Delta x \qquad D_{ij}^{+x}T = (T_{i+1,j} - T_{i,j})/\Delta x \\ D_{ij}^{-y}T &= (T_{i,j} - T_{i,j-1})/\Delta y \qquad D_{ij}^{+y}T = (T_{i,j+1} - T_{i,j})/\Delta y \end{aligned} \tag{11}$$

with grid steps Δx and Δy. $T_{i,j}$ and $f_{i,j}$ are respectively the cost and the slowness at gridpoint $(i,\ j)$ (see Fig. 6).

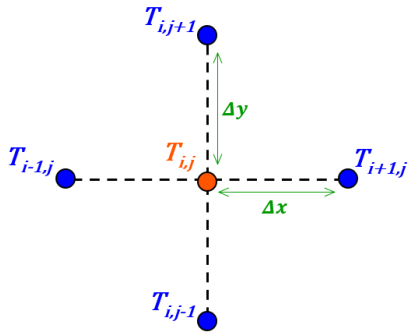


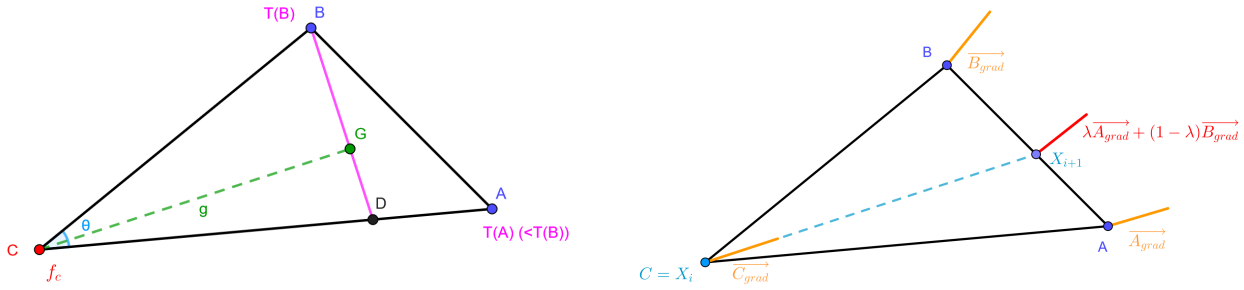
Fig. 6 Update directions on a 2D grid.

This scheme is said "upwind" because it chooses gridpoints with the direction of the flow of information. Hence, the algorithm builds the solution outwards from the smallest T value (initially at P_{start}). Once the propagation is performed and the destination is reached, one can extract the shortest path through back propagation of the gradient from P_{end} to P_{start}.

As presented before, the data structure is optimised in order to save storage space. The two-dimensional Fast Marching method has already been developed for triangular meshes in [27].

On an acute triangle ABC with $T(A) \neq \infty$, $T(B) \neq \infty$ and $T(A) < T(B)$, the quadratic equation must be solved, to

compute t such that $(t-u)^2 = g^2 f_C^2$ with $u = T(B) - T(A)$ and $t = T(C) - T(A)$.



(a) Update on a triangle: A, B and C are some of the vertices of the triangular mesh. $\overrightarrow{GC}$ is the gradient associated with vertex C.

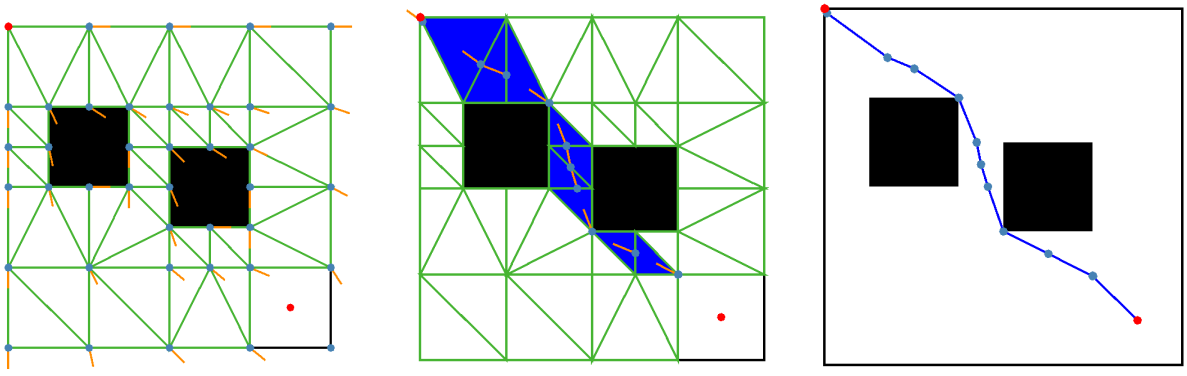
(b) Back propagation of gradients on a triangle.

Fig. 7 Propagation (a) and back propagation (b) procedures in 2D.

The propagation between P_{start} and P_{end} consists in solving the quadratic equation on each triangle (see Fig. 7a) and recording costs and gradients computed on each vertex of the mesh, as shown in Fig. 8a. In this figure, P_{start} (in the Northwest) and P_{end} (in the Southeast) are both represented in red. Gradients are represented in orange, mesh nodes in blue, and mesh triangles in green.

The back propagation phase mainly consists in reversing gradients computed in the propagation phase and computing the trajectory given by their direction (see Fig. 7b). If a gradient hits an edge instead of a vertex, it is interpolated by using a barycentric formulation. This phase is shown in Fig. 8b, where blue triangles are the triangles visited during back propagation.

Finally, the last step is to build the trajectory, which is achieved by linking the edge intersection points together. This is shown in Fig. 8c.



(a) Propagation phase.

(b) Back propagation phase using opposite direction of gradients.

(c) Computed trajectory.

Fig. 8 Optimal trajectory generation with the Fast Marching algorithm. Gradients are represented in orange, mesh nodes in blue, and mesh triangles in green.

As presented in [24], real-life applications of this algorithm are very promising, as trajectories are computed in about one second and memory management is very efficient. This algorithm works really well on many scenarios (See Fig. 9), but reaches its limits when the aircraft is too close from the landing site. Indeed, in this case, it is impossible to generate a single 2D descent plan.

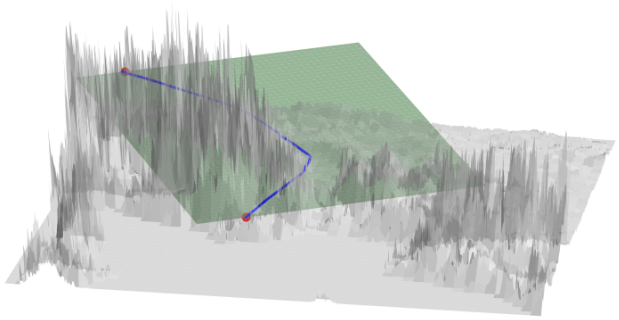


Fig. 9 3D trajectory generated with a 2D Fast Marching Algorithm over a single glide slope.

This paper presents the three-dimensional Fast Marching algorithm on a tetrahedral mesh, with the motivation to be able to generate trajectories in more complex scenarios.

V. Resolution algorithm in three dimensions

The aim of this section is to build a similar structure as the one presented in the previous section, but this time in three dimensions. An *octree*, which is the 3D-equivalent of a quadtree, is used to represent the obstacles and the free space. Again, the octree is balanced to limit large size differences between two neighboring cells. Finally, a mesh that consists of tetrahedrons, the 3D-equivalent of triangles, is built over the balanced octree structure.

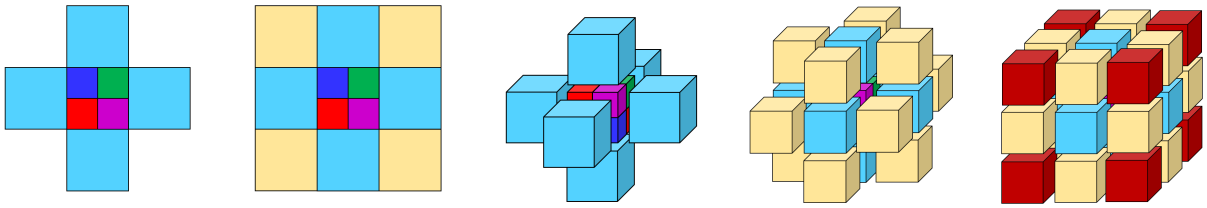
A. Octree balance

In two dimensions, there are only four cardinal directions to consider in order to balance a quadtree: *North, West, South, East*. Thus, there are $2^4 = 16$ different configurations for a quadtree node, that can easily be enumerated and studied. [24] showed that the Delaunay triangulation on these configurations always gives acute triangles that are bounded by a 26.565° minimum angle. Such properties do not exist in 3D.

To balance quadtrees, the technique called 1-*balance* was used, which corresponds to ensuring level differences between neighbors in the four cardinal directions. Another type of balance could also have been used, the 2-*balance*, which ensures level differences with neighbors in the four cardinal directions and the four inter-cardinal directions. They are illustrated in Fig. 10a and Fig. 10b.

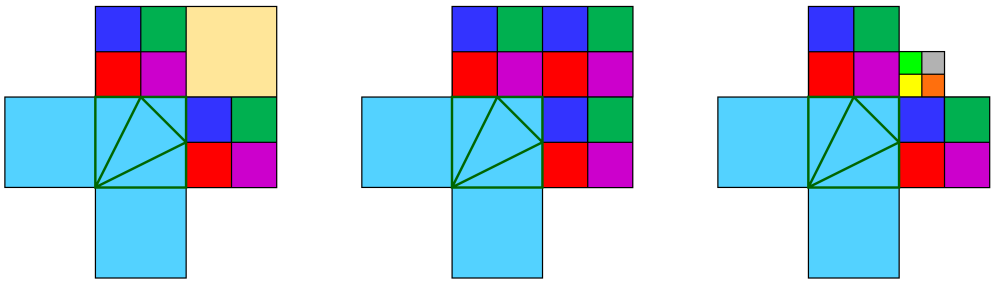
The aim is to have a structure that can be used to build directly the triangles over. It is sufficient to make a 1-balance in two dimensions, because inter-cardinal directions do not affect the possible quadtree node configurations (see Fig. 11).

This figure shows that no matter the level difference between the current node (at the center of each figure) and its



(a) 2D, 1-balance. **(b) 2D, 2-balance.** **(c) 3D, 1-balance.** **(d) 3D, 2-balance.** **(e) 3D, 3-balance.**

Fig. 10 Ways to balance a quadtree (a and b) or an octree (c, d and e). Adapted from [28, Fig. 5].



(a) $\delta_{\mathbf{NE}} = 0$**.** **(b)** $\delta_{\mathbf{NE}} = 1$**.** **(c)** $\delta_{\mathbf{NE}} = 2$**.**

Fig. 11 Configuration of a quadtree node with a NorthEast neighbor with variable level difference $\delta_{\mathbf{NE}}$**.**

NorthEast neighbor, the triangulation remains the same. Thus, considering 2-balance does not add any new configuration for the triangles. Surely, it would increase the precision of the algorithm, but it would also increase the number of triangles itself.

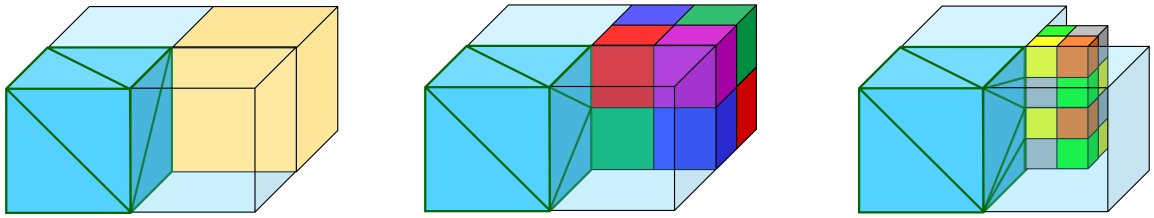
In three dimensions, one would also want to make a 1-balance (see Fig. 10c). It would amount in considering the six cardinal directions *North, West, Up, South, East, Down*, thus to study $2^6 = 64$ configurations for the octree nodes. Nevertheless, 1-balance in 3D is not enough to describe all the configurations and to build templates.

Indeed, in two dimensions, the junction between the current node and its NorthEast neighbor is a single vertex, which is always part of one triangle in the triangulation. In three dimensions, the junction between the current node and its NorthEast neighbor is an edge, and the tessellation (generalisation of triangulation in higher dimensions, here 3D) varies according to the level difference between these two nodes. This is highlighted in Fig. 12.

This figure shows that the current node configuration changes when the level difference between the two nodes increases. Then, it is insufficient to consider 1-balancing the octree.

For the same reasons as in two dimensions, it is sufficient to 2-balance the octrees (see Fig. 10d). A 3-balance, as in Fig. 10e, would amount in balancing according to the inter-cardinal directions of order 2 (*e.g.* NorthEastUp). However, since junctions between such nodes are single vertices, the 3-balance does not add any new configuration, then is useless to consider (except for precision considerations).

Hence, one needs to perform a 2-balance of the octree, which amounts to considering 18 directions (the 6 cardinals



(a) $\delta_{\mathbf{NE}} = 0$. **(b)** $\delta_{\mathbf{NE}} = 1$. **(c)** $\delta_{\mathbf{NE}} = 2$.

Fig. 12 Configuration of an octree node with a NorthEast neighbor with variable level difference $\delta_{\mathbf{NE}}$.

and the 12 inter-cardinals of order 1). This implies to study $2^{18} = 262144$ configurations.

B. Building of the tetrahedral mesh

In this paper, the 200000 configurations that emerge from 2-balancing the octree are not studied manually. Since Delaunay tessellations are not as trivial as for the 2D case, the function `Delaunay` from the Python module `scipy.spatial`, which provides methods for spatial algorithms and data structures, is used. The Delaunay tesselation is formed of tetrahedrons, polyhedrons composed of four triangular faces, six straight edges and four vertex corners, which are the 3D equivalent of triangles.

Thanks to this function, the 2^{18} configurations are evaluated in order to check whether they are all composed of acute tetrahedrons, the necessary condition that allows propagation in the Fast Marching algorithm. Computations use `scipy` version 1.7.1, and execute the function `Delaunay` without specifying any option. They show that about half of the configurations are not suited for propagation, *i.e.* at least one of the tetrahedrons of these configurations is obtuse (they present a face with at least one obtuse angle).

In order to get a clear view of the configurations that are problematic, the number of configurations is reduced by removing redundant schemes. It can be seen that all the possible configurations 2^{18} have not to be checked. On a cube, whenever there is a neighbor in a specific cardinal direction, the mesh contains a vertex at the center of a face and four vertices at the centers of the four edges that constitute the face. Thus, thanks to a combinatorial result obtained in Appendix A, only 6210 configurations remain to be studied. These 6210 configurations are evaluated with the Python function `Delaunay` and, as expected, about half of them still contain obtuse angles.

The second proposition is to remove all symmetrical schemes. Symmetries appear with reflection or rotation of configurations, leading from one to another. Such example is shown in Fig. 13.

In this figure, the study of the first configuration is sufficient to recover the Delaunay tessellations for all the other configurations. After removing all these symmetrical schemes, only 227 distinct configurations are left for checking. The Python function `Delaunay` gives 97 obtuse configurations and 130 acute configurations.

Finally, the last proposition is to use these 130 acute configurations as *templates*. The goal is to manage to rebuild the

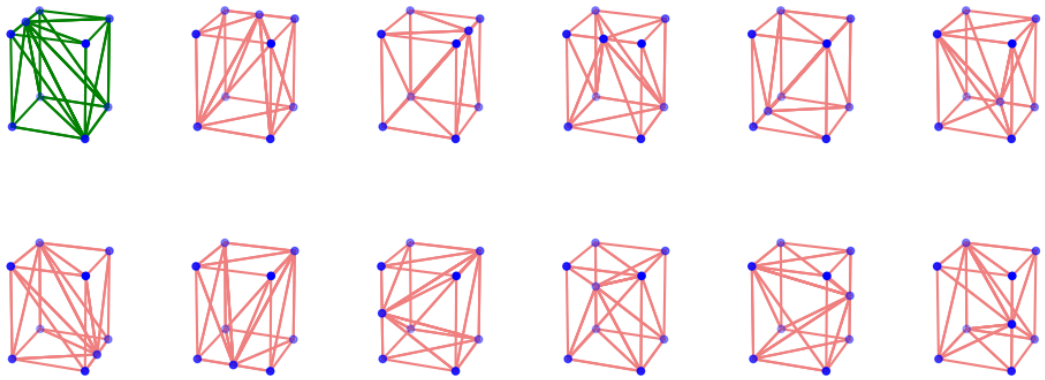


Fig. 13 Symmetrical schemes for the scenario of smaller neighbors in one inter-cardinal direction.

227 non-symmetrical configurations from these only 130 templates, and to be able to shape every octree node that can be encountered. To this end, good configurations are overlayed in order to rebuild the obtuse ones, as shown in Fig. 14.

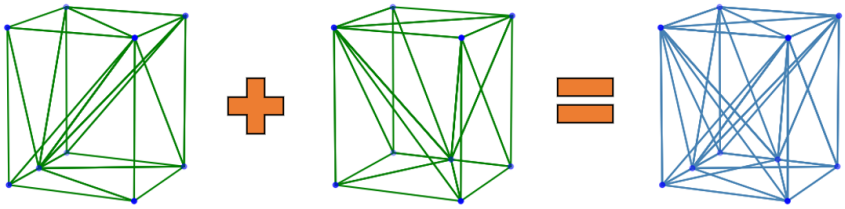


Fig. 14 A new configuration, previously invalid, built from two instances of a valid template.

This figure shows that configurations are overlayed in order to build new ones, which means that tetrahedrons are intertwined. This is not a problem, because the mesh is built for the sole purpose to perform the update procedure of the Fast Marching algorithm, which only affects vertices of the mesh. The more the tetrahedrons, the better the update. A tetrahedron in the mesh should not be seen as a solid made of four points, but rather as four triangle-vertex couples which serve the propagation.

It is possible to demonstrate that all 97 obtuse configurations can be built from the 130 templates. Since configurations are added up in order to build the acute configurations, one only has to check that the two basic templates of Fig. 15 are acute.

Templates of Fig. 15 represent respectively one neighbor in an inter-cardinal (of order 1) direction and one neighbor in a cardinal direction. It is clear that these two templates are the foundations to rebuild all possible configurations. Indeed, it is possible to add up as many versions of the first template as the number of inter-cardinal directions with smaller neighbors. The same reasoning can apply to the second template with the number of cardinal directions with smaller neighbors. Finally, since the two templates are composed of acute tetrahedrons, all obtuse configurations can effectively be rebuilt.

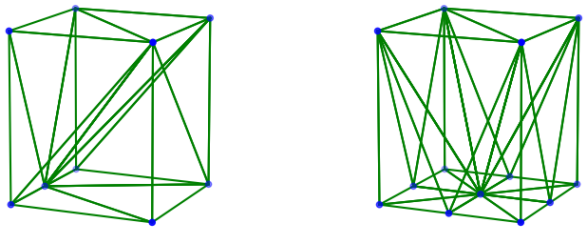


Fig. 15 Respectively, from left to right, templates with smaller neighbors in one inter-cardinal direction and in one cardinal direction.

This short demonstration shows that any obtuse configuration can be rebuilt from only two basic templates. However, rebuilding every obtuse configuration only with these two templates is not optimal for many reasons. Not only too many tetrahedrons would be built in nodes that have many neighbors, and thus slow down the Fast Marching algorithm, but other templates that can lead to better update procedures from smaller tetrahedrons would not be taken into account. Then, as there are various ways to rebuild the obtuse configurations, the objective would be to minimise the number of tetrahedrons in each node. This is achieved by considering the level differences quantity δ, as represented in Figure 16.

As in 2D, after 2-balancing the octree, the only possible values for a level difference are:

- '-1' representing a neighbor with a lower level;
- '0' representing a neighbor with equal level;
- '1' representing a neighbor with a higher level;
- '#' representing a neighboring obstacle or the 3D grid limits.

For instance, Fig. 16 represents an octree node with a smaller neighbor at North, a larger neighbor at EastUp and grid limits at South. Since a smaller neighbor is present in a cardinal direction (North), the level differences with NorthWest, NorthUp, NorthEast and NorthDown are fixed to '1', whether or not there are actually smaller neighbors in these directions. This number fully accounts for the configuration of the node, since its intersection points with neighboring cells are marked with a '1'.

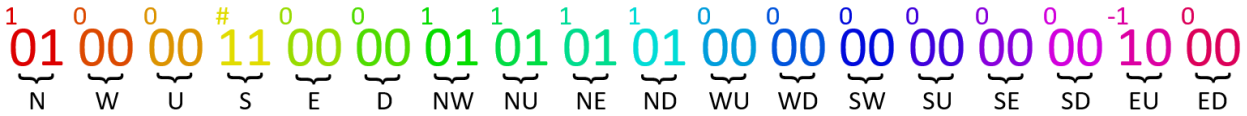


Fig. 16 Example of δ value for an octree node.

Considering this δ value, one can execute the following procedure in order to find the proper template decomposition for an obtuse configuration:

Algorithm 2 Repair of obtuse configurations

```
for i = δ to 0 do
    if Configuration i is acute and add useful tetrahedrons then
        Add configuration i to the decomposition.
    end if
end for
```

In this procedure, a *useful* tetrahedron is a tetrahedron that activates a point, or in other words that links a point to the mesh (see Appendix A for more details on *activated* points). Thus, by following such procedure, the smallest possible number of configurations is added in order to recreate an acute configuration. Moreover, these configurations are the closest possible to the considered obtuse configuration (in terms of δ).

This section presented how to build an octree able to directly tetrahedrise each of its nodes, allowing updates for any mesh points with the Fast Marching algorithm. Each leaf is characterised by its Morton code m (8 bytes, a spatial identifier for nodes) [29], its level l (2 bytes) and its level differences δ (8 bytes). The 130 templates are stored in a low-sized file (less than 100Ko), as well as the list of all possible configurations, each of which refers to one or more of these templates and the operations to be performed over these templates (rotations and/or symmetries).

The fact that every tetrahedron of the structure is acute implies that continuous updates are available throughout the whole free space, thus enabling the use of the Fast Marching algorithm. Since the structure is based on an octree, neighbors are obtained in a very short amount of time. Also, the structure does not require much memory space and can be embedded easily on an aircraft.

C. Procedures implementation

The property that the minimal cost paths are orthogonal to the level curves is valid in any dimension. In three dimensions, it is expressed by the 3D Eikonal equation, which writes as follows:

$$|\nabla T| = f(x, y, z) \tag{12}$$

Then, the *quadratic equation* is now expressed as:

$$\max\left(D_{ijk}^{-x}T, -D_{ijk}^{+x}T, 0\right)^2 + \max\left(D_{ijk}^{-y}T, -D_{ijk}^{+y}T, 0\right)^2 + \max\left(D_{ijk}^{-z}T, -D_{ijk}^{+z}T, 0\right)^2 = f_{i,j,k}^2 \tag{13}$$

where the forward and backward operators are given by

$$
\begin{aligned}
D_{ijk}^{-x}T &= (T_{i,j,k} - T_{i-1,j,k})/\Delta x \qquad & D_{ijk}^{+x}T &= (T_{i+1,j,k} - T_{i,j,k})/\Delta x \\
D_{ijk}^{-y}T &= (T_{i,j,k} - T_{i,j-1,k})/\Delta y \qquad & D_{ijk}^{+y}T &= (T_{i,j+1,k} - T_{i,j,k})/\Delta y \\
D_{ijk}^{-z}T &= (T_{i,j,k} - T_{i,j,k-1})/\Delta z \qquad & D_{ijk}^{+z}T &= (T_{i,j,k+1} - T_{i,j,k})/\Delta z
\end{aligned}
\tag{14}
$$

with grid steps Δx, Δy and Δz. $T_{i,j,k}$ and $f_{i,j,k}$ are respectively the cost and the slowness at gridpoint $(i,\ j,\ k)$ (see Fig. 17).

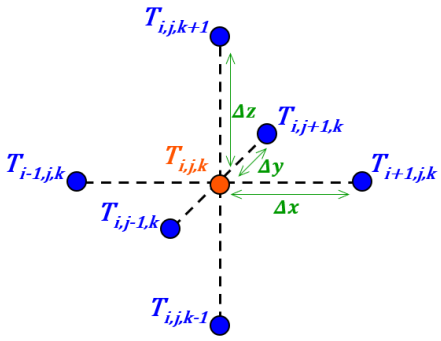


Fig. 17 Update directions on a 3D grid.

Propagation over a tetrahedral mesh Consider an acute tetrahedron ABCD with $T(A) \neq \infty$, *i.e.* A has already been updated. Such a configuration is represented in Fig. 18. The quadratic equation must be solved, to compute t such that $(t - u_{\max})^2 = h^2 f_D^2$ with $u_{\max} = T(C) - T(A)$ and $t = T(D) - T(A)$. Also, the quantity $u_{\min} = T(B) - T(A)$ is introduced.

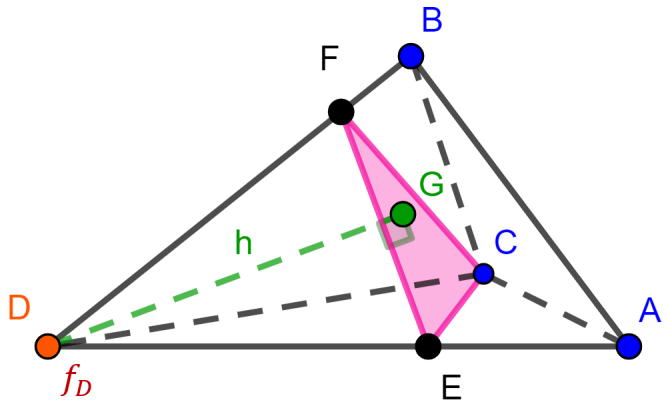


Fig. 18 Update on a tetrahedron: A, B, C **and** D **are some of the vertices of the tetrahedral mesh.** $\overrightarrow{GD}$ **is the gradient associated with vertex** D**.**

Three cases are studied:

- B, C and D are updated from A.
- C and D are updated from A and B with $T(B) \geq T(A)$.
- D is updated from A, B and C with $T(C) \geq T(B) \geq T(A)$

The first case is trivial because the update comes from the directions given by the edges: $T(B) = \min\left(T(B),\ \mathrm{AB}f_B + T(A)\right)$, $T(C) = \min\left(T(C),\ \mathrm{AC}f_C + T(A)\right)$ and $T(D) = \min\left(T(D),\ \mathrm{AD}f_D + T(A)\right)$. The second case corresponds to an update from the triangles ABC and ABD. Please refer to [27] for the detailed procedure. The third case corresponds to the update on a tetrahedron, for which all computations are made in Appendix B. In this case, the quadratic equation is written as follows:

$$\left(\mathcal{A}^2_{\mathrm{BCD}} + \mathcal{A}^2_{\mathrm{ACD}} + \mathcal{A}^2_{\mathrm{ABD}}\right) t^2 - 2\left(\mathcal{A}^2_{\mathrm{ACD}} u_{\min} + \mathcal{A}^2_{\mathrm{ABD}} u_{\max}\right) t + \mathcal{A}^2_{\mathrm{ACD}} u^2_{\min} + \mathcal{A}^2_{\mathrm{ABD}} u^2_{\max} - 9 f_D^2 \mathcal{V}^2_{\mathrm{ABCD}} = 0 \qquad (15)$$

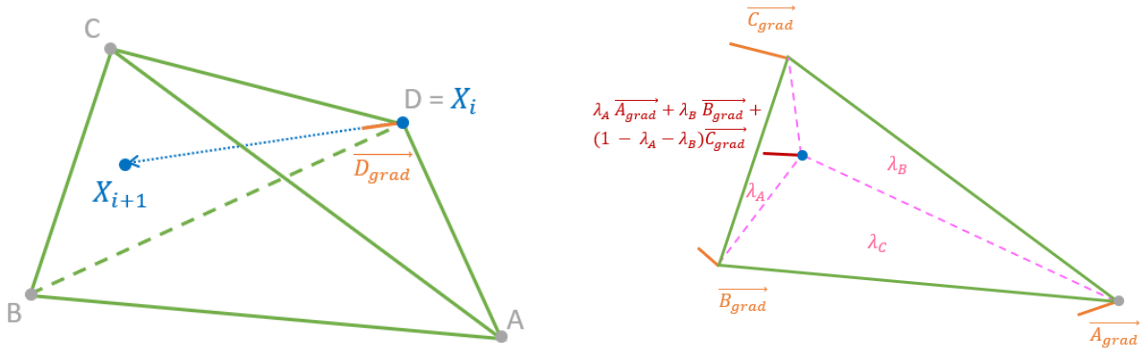
with $\mathcal{A}_{\mathrm{BCD}}$, $\mathcal{A}_{\mathrm{ACD}}$, $\mathcal{A}_{\mathrm{ABD}}$ the areas of the triangles BCD, ACD and ABD respectively. $\mathcal{V}_{\mathrm{ABCD}}$ is the volume of the tetrahedron ABCD.

If the direction of the gradient $\overrightarrow{GD}$ is effectively contained in the tetrahedron (*i.e.* G is inside the tetrahedron), then Eq. (15) can be solved. It will return a cost that can be set to the new cost for D, only if its value is lower than the current one.

Back propagation The final trajectory is found thanks to the back propagation of gradients, from P_{end} and P_{start}. On a vertex, the gradient to ascend is directly given: it is the opposite of the gradient vector computed at this particular vertex during the propagation phase. However, when the optimal direction strikes an edge or a triangle, no gradient is available because the intersection does not match with any vertex encountered during the propagation phase. Thus, at each encounter with an edge or a triangle, one needs to interpolate the gradient.

On an edge, it is easily done by computing the barycenter of the gradients linked with the two vertices of the considered edge. On a triangle, it is done by computing the barycenter of the gradients linked with the three vertices of the considered triangle (see Fig. 19).

Fig. 19 shows the interpolation of a gradient inside a tetrahedron ABCD. In Fig. 19a, X_i is the last point visited by the back propagation. It coincides with the mesh vertex D, hence the gradient at X_i is $\overrightarrow{D_{grad}}$ (the opposite gradient of the one computed during the propagation phase). The next point visited by the back propagation is X_{i+1}. It is at the intersection between $\overrightarrow{D_{grad}}$ and the triangle ABC, hence the gradient has to be interpolated. Fig. 19b presents how such interpolation is made. It is barycentric, with parameters $\lambda_A = \dfrac{\mathcal{A}_{X_{i+1}BC}}{\mathcal{A}_{\mathrm{ABC}}}$ and $\lambda_B = \dfrac{\mathcal{A}_{AX_{i+1}C}}{\mathcal{A}_{\mathrm{ABC}}}$, *i.e.* the ratios between the sub-triangle areas (formed with X_{i+1}) and the area of the triangle ABC. Thus, the gradient at X_{i+1} is $\lambda_A \overrightarrow{A_{grad}} + \lambda_B \overrightarrow{B_{grad}} + (1 - \lambda_A - \lambda_B)\overrightarrow{C_{grad}}$.



(a) Creation of the next trajectory point. **(b) Gradient interpolation.**

Fig. 19 Back propagation of gradients on a tetrahedron.

The algorithm stops on reaching P_{start}, and the final trajectory between P_{start} and P_{end} is found by linking the intersection points together, in the same way as for the 2D algorithm.

D. Improvement of the algorithm

This section presents two major improvements of the algorithm. The first one is a procedure that takes into account wind fields within the Fast Marching algorithm. The second improvement is a method to force the trajectory to respect the aeronautical constraints. Without loss of generality, both methods are presented in 2D.

Wind constraints Ordered Upwind methods [14] have been developed to take into account wind fields. They are adapted from Fast Marching methods, but are composed of more complex and costly procedures, thus greatly degrading the efficiency of the method. This section proposes another approach that preserves the "one-pass" procedures of the Fast Marching algorithm, as well as its computation time efficiency.

The gradient computed during the propagation phase (see Fig. 7a) is the direction of minimal cost. It means that it is the direction that the aircraft follows as soon as it enters the triangle. Thereby, the gradient is oriented in the direction of the aircraft's true airspeed $\overrightarrow{TAS}$ and, as no wind is yet considered, the aircraft ground speed $\overrightarrow{GS}$.

To account for a wind field $\overrightarrow{W}$ in the aircraft movement, one can refer to the equation known as Wind Triangle (see Fig. 20):

$$\overrightarrow{TAS} + \overrightarrow{W} = \overrightarrow{GS} \tag{16}$$

Hence, the new direction of minimal cost in a wind field is given by rotating this direction to match the direction of the ground speed $\overrightarrow{GS}$. This operation is shown in Fig. 21a.

The direction of minimal cost is oriented in a new direction, supported by the vector $\overrightarrow{HC}$. However, no information on costs is available along this vector, as there is no isocost that can be defined. It is useful to interpolate the cost at H,

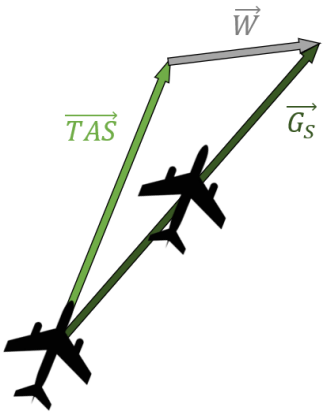


Fig. 20 The Wind Triangle graphically represents the relationships among velocity vectors used for air navigation. As the true airspeed $\overrightarrow{TAS}$ gives the heading of the aircraft, the ground speed $\overrightarrow{G_S}$ provides the trajectory of the aircraft (relatively to the ground).

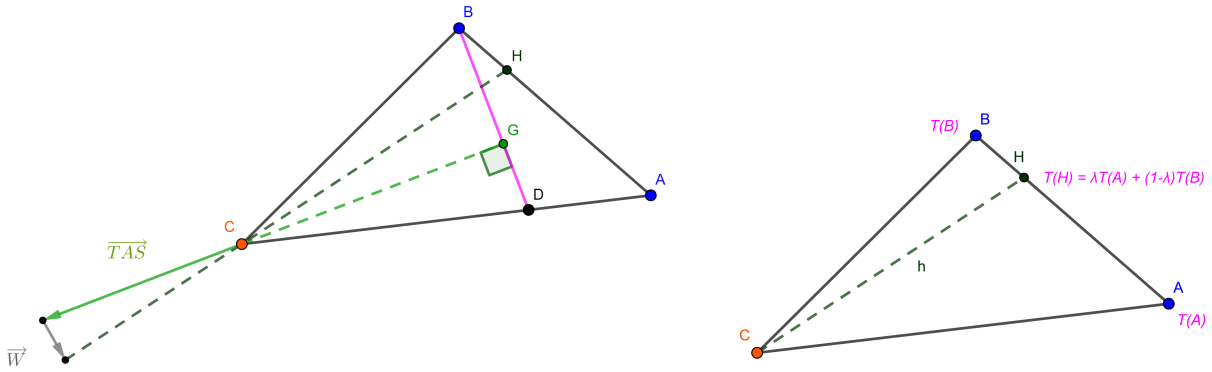
based on the costs of vertex A and vertex B. The interpolation (barycentric) is formalised as follows:

$$T(H) = \lambda T(A) + (1-\lambda)T(B)\,, \text{ with } \lambda = \frac{||\overrightarrow{HB}||}{||\overrightarrow{AB}||} \tag{17}$$

The interpolation procedure is shown in Fig. 21b. Finally, the cost at C is given by the following formula:

$$T(C) = T(H) + h/V \tag{18}$$

with $h = ||\overrightarrow{HC}||$ and V the norm of the aircraft ground speed.



(a) Wind Triangle from Aeronautics: $\overrightarrow{TAS} + \overrightarrow{W} = \overrightarrow{G_S}$.

(b) Gradient interpolation.

Fig. 21 Update on a triangle with wind.

To check the accuracy of this new procedure, the algorithm is tested on the Zermelo's navigation problem. Proposed in 1931 by Ernst Zermelo in [30], it is a classic optimal control problem that refers to a boat navigating on a flow of

water, originating from a point A to a destination point B. The boat is capable of a certain maximum speed, and the goal is to derive the best possible control to reach B in the least possible time. The exact solution to this problem is given by Zermelo's equation, which can be numerically solved.

When considering an aircraft surrounded by the air, Zermelo's problem remains unchanged. Sketches presenting solutions without wind (or with constant wind) and with a wind gradient are shown in Fig. 22.

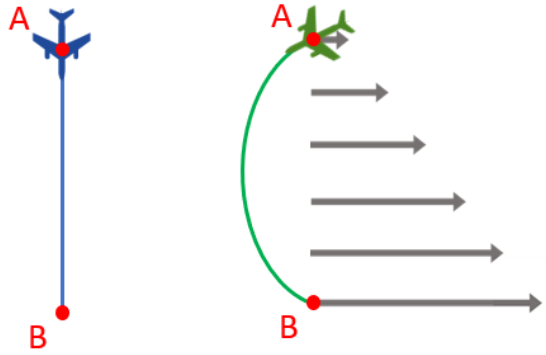
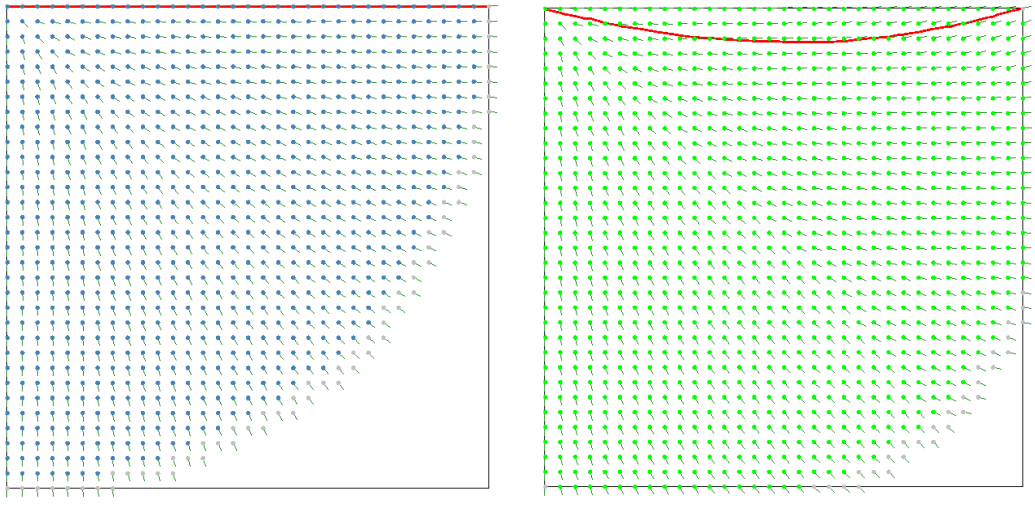


Fig. 22 Solutions to Zermelo's problem with constant wind (left) and with linearly variable wind (right).

Both solutions are very intuitive, as the straight line is the most efficient trajectory in the absence of wind, and it seems favorable to benefit from well-oriented wind gradients in the second scenario.

The Fast Marching algorithm is tested on Zermelo's problem with a wind gradient. Results are shown in Fig. 23 with a 64x64 grid, trying to link the up-left corner to the up-right corner in a wind strength that is 0 at the top of the figure, and 67% of the aircraft speed at the bottom of the figure.



(a) Without wind. **(b) With wind.**

Fig. 23 Optimal trajectory generation in a wind field with the Fast Marching algorithm.

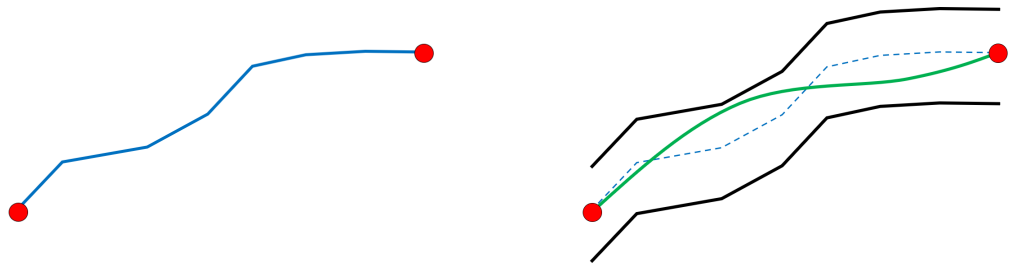
This figure shows the curved trajectory that is the solution to Zermelo's problem. The amplitude of the trajectory is consistent with the solution of Zermelo's equation for the 0 to 67% of the maximum speed gradient.

A close look at this figure shows that the rotation of all gradients is smooth, which indicates that the algorithm still gives the optimal trajectory if the start point or end point were moved.

In 3D, the method is the same as the optimal direction can be rotated in the 3D space. On a tetrahedron ABCD, where D is the point to update, the intersection between this direction and the triangle ABC becomes the point of reference for the update, and its cost is interpolated inside the triangle ABC (barycentric interpolation).

Aeronautical constraints The Fast Marching algorithm must be improved in order to take into account the Flight Dynamics constraints of the trajectory. Indeed, the Fast Marching algorithm returns the shortest path between two points, and it is rather unlikely that this path respects the radius of turn or the heading constraints expressed in Section III.

The idea is to create an approximate path to follow in order to reach the destination point optimally by respecting the constraints, then one would be able to build the true optimal trajectory near this path, which would also respect the constraints. This is the idea of generating a *guiding tube*, which is illustrated in Fig. 24.



(a) The approximate path, valid regarding the constraints.

(b) The optimal trajectory generated inside a tube, guiding the propagation.

Fig. 24 Building of a *guiding tube*.

Fig. 24a shows a path generated by some algorithm that easily takes into account the aeronautical constraints but returns a non-smooth solution far from the optimal. Fig. 24b presents how this path is used in order to create a guide for the Fast Marching algorithm. Inside this tube, the algorithm runs and returns a smooth trajectory and less costly.

To build this approximate path, the Fast Marching Tree algorithm is chosen for its ergonomics and its low computing time complexity. The FMT* algorithm was briefly introduced in the State of the Art as a graph generation algorithm. Fig. 25 illustrates the growth of the tree in a 2D environment with 2,500 samples.

Hence, the FMT* algorithm generates a tree by moving steadily outward in cost-to-arrive space. This growth reaches the limits of the free space in a short amount of time, which makes it a very good candidate to generate a first "approximate" trajectory. Indeed, in order to obtain a fast solution (thus of poor quality), one only has to limit the number of samples considered.

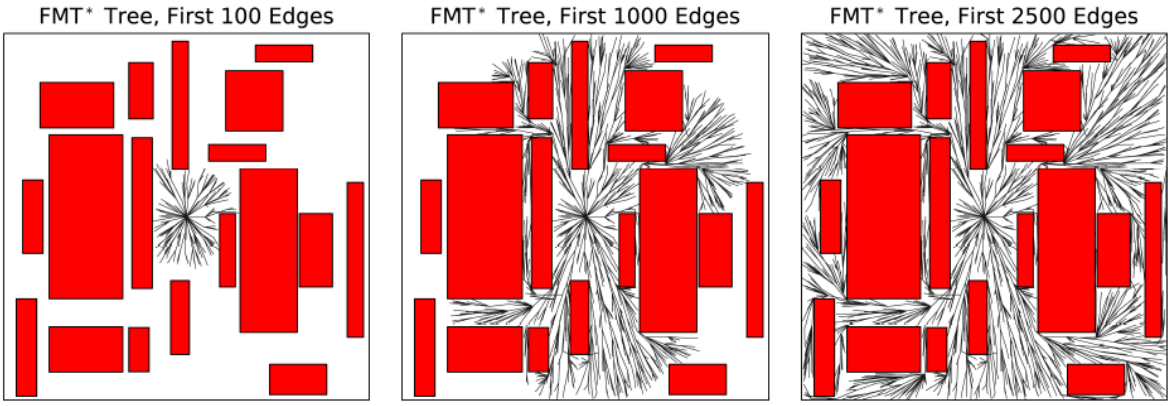
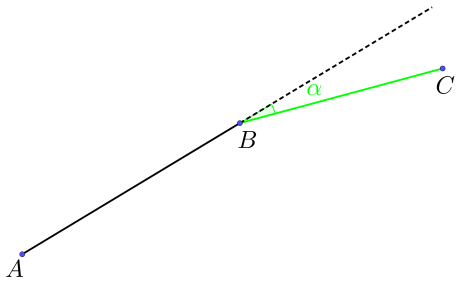
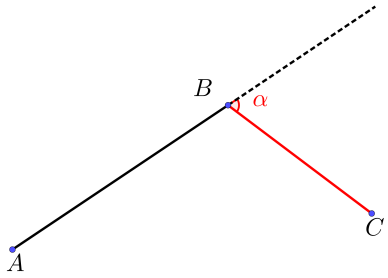


Fig. 25 The FMT* algorithm [31, Fig. 1].

This algorithm is able to take into account aeronautical constraints (See Fig. 26). Indeed, the algorithm has a *best neighbor* selection phase, with the possibility to discard neighbors that do not satisfy the constraints. It also adapts really well to a three-dimensional space (See Fig. 27). The two main constraints are turning and descent constraints. The algorithm does not consider connections that create too high discontinuities of heading. Indeed, if the angle α (See Fig. 26) is too high, the aircraft will not be able to turn from A to C. The algorithm also does not create a connection if the new point is not in the descent cone (See Fig. 27).



(a) The angle α is small, the connection BC is possible.



(b) The angle α is too big, the connection BC is ignored.

Fig. 26 Taking into account turning constraints.

To summarise, the method consists first of all in generating a first approximate path by using FMT* with a small number of samples (<3000). This path takes into the aeronautical constraints. Then a guiding tube is generated around it. Finally, the Fast Marching algorithm is then applied in this tube to obtain the final trajectory.

The three procedures presented above (building a first approximate solution, propagation and back propagation) form the foundations of the algorithm developed for this study. The results obtained with this algorithm are presented in the next section.

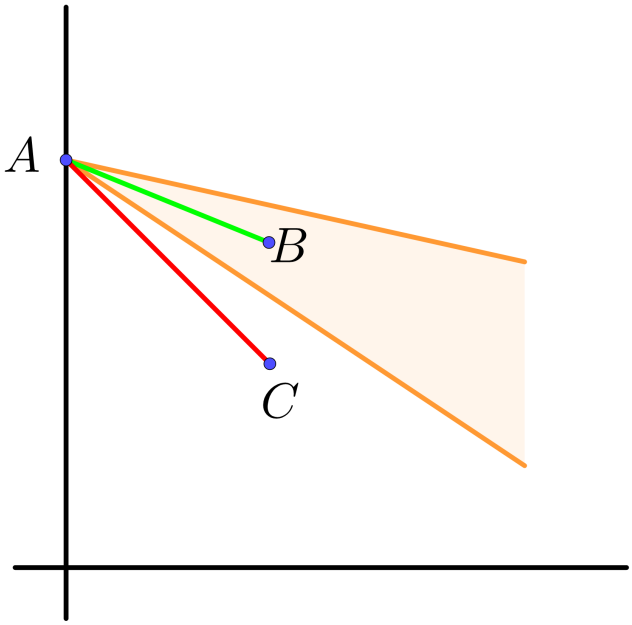


Fig. 27 Taking into account descent constraints: the connection between the points A and B is possible because the segment is in the descent cone, but the connection AC is impossible.

VI. Results and Findings

This section presents a solution given by the second algorithm for an emergency in the mountainous region of Grenoble, France. It is still crucial to make sure that the optimal trajectory is computed and that the stakes are satisfied: the rapid convergence, the low data sizes and the flyability of the trajectory.

A. Numerical tests

Validation of the method is made in a mountainous region of the Grenoble Alpes Isère airport. This time, only one key scenario is studied:

- ANSA emergency: The aircraft has lost all of its thrust and cannot reach the landing site with a straight line, because of flight path constraints.

This scenario is complementary to the two scenarios studied in 2D in [24], because it can only be tackled by the 3D algorithm (one single descent plan cannot guide the aircraft towards the nearest airport).

Again, the whole mesh is not generated at once. The mesh is built step by step, by dividing an octree leaf node into tetrahedrons only when the front enters inside it. Also, as simulations run in 3D and all computation costs become higher, the step sizes have been increased by a factor of 10 compared to the 2D results. This enables the algorithm to run in an appropriate amount of time (real-time).

1. Unconstrained Fast Marching Algorithm

First, to provide validation for the Update procedure on a tetrahedron as well as the building of the data structure, the simulation is run without taking into account aeronautical constraints.

Results for the unconstrained ANSA scenario are shown in Fig. 28.

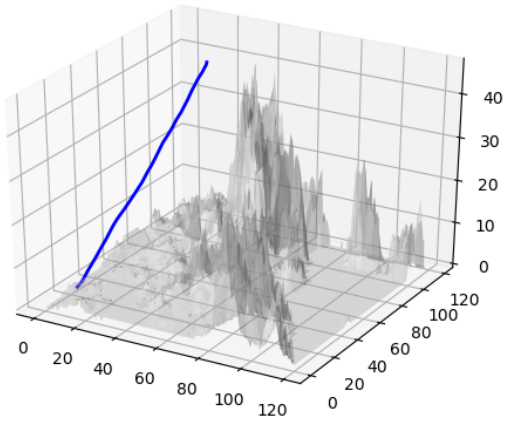


Fig. 28 3D trajectory generated in an ANSA scenario (arb. unit).

As there are no mountains encountered and no constraints to satisfy, the optimal path is the straight line (blue trajectory). It is illegible to represent the mesh in 3D, but keep in mind the whole free space is composed of tetrahedrons, that the number of update procedures is big and that the back propagation of gradients led to computing this straight line.

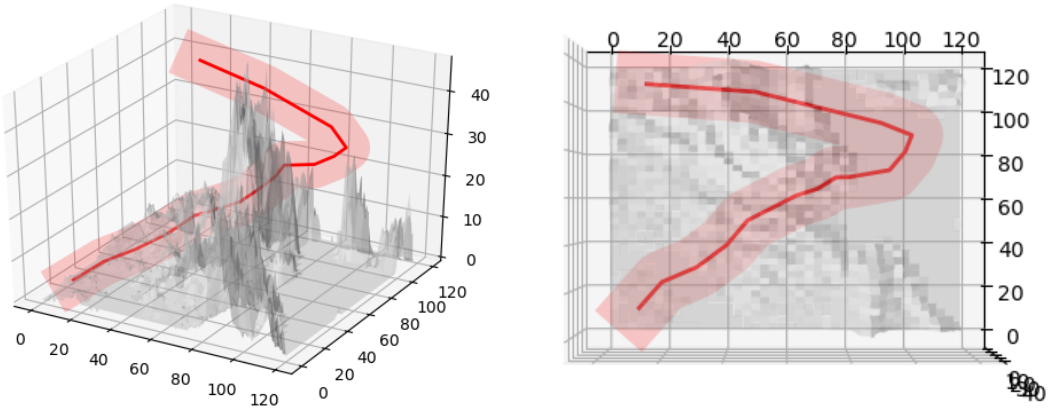
In order to satisfy the constraints, the trajectory would have to move in the mountain direction, lose altitude and land safely at the airport. This is shown in the next section.

2. Constrained Fast Marching Algorithm

Now, flight path and radius of turn constraints are taken into account. The idea is to use the FMT* algorithm to generate a first approximate trajectory. An example of such trajectory can be found in Fig. 29.

The guiding tube, surrounding the approximate trajectory, is also represented in this figure. The propagation is made inside this tube, considering that every node outside this tube is an obstacle. The size of the tube is an external parameter to define, that can be a function of the domain sizes. Its size needs to provide a balance between two opposite objectives:

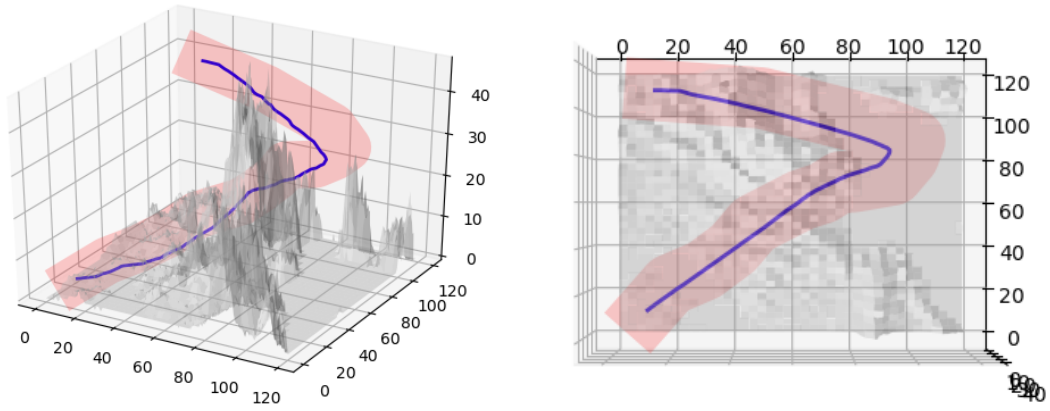
1) The tube must not be too narrow because the optimal trajectory is supposed to be different from the one returned by the FMT* algorithm.
2) The tube must not be too large because the constraints need to be satisfied as much as possible by the optimal trajectory.



(a) Side view. **(b) Top view.**

Fig. 29 Approximate trajectory generated in an ANSA scenario with the FMT* algorithm (arb. unit).

The trajectory generated by the constrained Fast Marching algorithm is shown in Fig. 30. This figure shows the optimal path within the guiding tube. It highlights the advantages and drawbacks of the considered method. This trajectory is smooth, short, avoids obstacles and effectively stays within the guiding tube. However, it is clear that the trajectory does not fully respect the constraints. Indeed, it has a steeper descent at the beginning of the procedure, which can potentially be out of the constraints. Indeed, if the new trajectory is significantly shorter than the approximate trajectory, the descent angle will be too high to respect the constraints. This can be easily corrected by reducing the size of the guiding tube, but again the generated trajectory would look a lot like the approximate trajectory generated with the FMT*, thus questioning the utility of the Fast Marching algorithm. Another solution is to severely restrict FMT* in order to give more freedom to the Fast Marching algorithm. The connection constraints are therefore more limiting. For example, the descent cone is reduced.

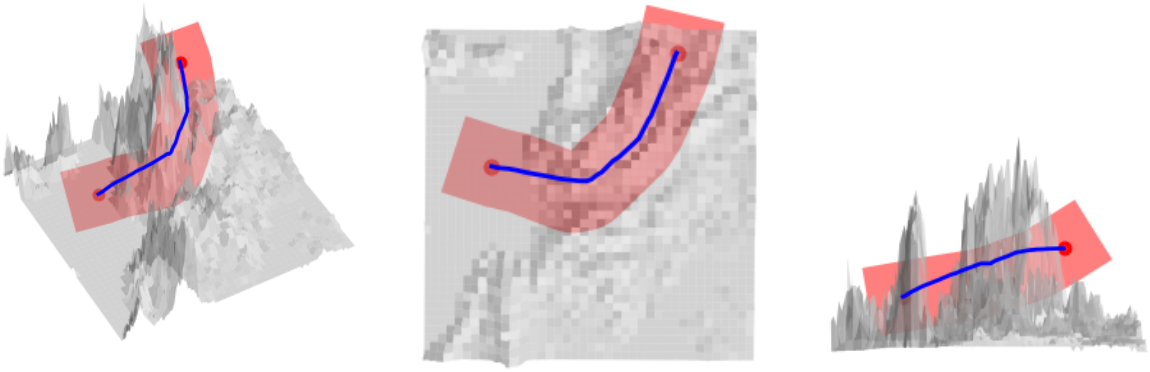


(a) Side view. **(b) Top view.**

Fig. 30 Trajectory generated in an ANSA scenario with the Fast Marching algorithm (arb. unit).

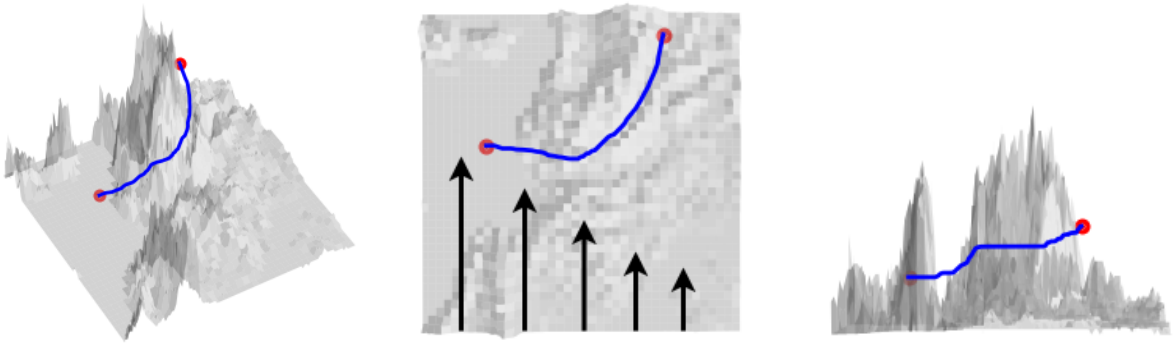
3. Wind Fast Marching Algorithm

Finally, the algorithm has been tested with wind. The scenario is different from the previous one. The aircraft is not constrained by a minimum descent angle. Fig. 31 shows the result obtained by the algorithm without considering the wind. Fig. 32 presents, for the same scenario considering the wind, the trajectory computed by the proposed method. In this scenario, the wind comes from the south. This intensity decreases from west to east and is constant verticaly (See Fig. 32b). The wind being unfavorable, the solution computed with the wind is 10% more costly.



(a) Side/Top view. **(b) Top view.** **(c) Side view.**

Fig. 31 Trajectory generated without wind.



(a) Side/Top view. **(b) Top view.** **(c) Side view.**

Fig. 32 Trajectory generated with wind from the south (top view)

The algorithm still needs improvements in order to return a better and more realistic trajectory. Though these first results are encouraging, there are still some validations to be made in order to make sure that the optimal trajectory is computed and that the constraints are fully respected.

B. Computation analysis

In this section, the computation times and the stored data sizes for the proposed algorithm are presented. Such results are displayed in Tables 1a and 1b. They summarise the computation time and the stored data size of each part

of the algorithm. The phases considered in this analysis are the FMT* approximate trajectory generation, the octree generation, the balancing operation, the level difference computation, the propagation and the back propagation of the 3D Fast Marching algorithm.

(a) Computation Times (in Milliseconds) of the complete algorithm (FMT* + Fast Marching)

	FMT* + FM
FMT* trajectory generation	200
Octree generation	700
Balancing operation	100
Level difference computation	50
Fast Marching (*propagation*)	2000
Fast Marching (*back propagation*)	500
Total	3550

(b) Stored Data Sizes (in Kilo bytes)

	File Size
Terrain Data	491 649
FMT* trajectory	1
Octree	67
Balanced Octree with δ	75

Though the step sizes are different, the Fast Marching algorithm in three dimensions takes more time to converge than the Fast Marching algorithm in two dimensions. This is mainly due to the more complex procedures that have to be evaluated throughout the propagation. However, the algorithm computes a solution in less than 4 seconds, which is acceptable for an emergency trajectory generation.

Again, some improvements have to be made in order to make the algorithm workable on an aircraft. The implementation of functions (creation of octrees, balancing, Fast Marching...) must be optimised in order to obtain a stronger method. The study of "critical" scenarios can also help to improve the algorithm, as it is here tested on a simple case.

Nevertheless, we have great expectations for the algorithm, as it checks almost every stake for the problem of generating optimal emergency trajectories.

VII. Conclusion

The objectives were to develop an algorithm able to rapidly generate a safe trajectory in the event of an on-board emergency, in order to provide help to pilots in such a situation. The constrained optimisation problem has been established: the final trajectory must respect flight path and heading angle constraints. The problem has been proposed to be solved by an algorithm based on a three-dimensional Fast Marching method. These methods are known to be efficient due to their low complexity and to provide smooth trajectory as they are not limited by the graph structure. In order to build an optimised meshing of the three-dimensional space, an octree has been built and balanced in order to strongly reduce the number of cells to consider. An acute tetrahedrisation of this structure, obtained thanks to the study of combinatorics and symmetries, has been also proposed as tetrahedrons better propagate information than cubes.

The 3D update procedure has been developed on a tetrahedron. It has a similar formulation to the one in two-dimensions, only a surface isocost is now used to propagate the costs. Procedures to account for the wind and

aeronautical constraints have also been developed for both 2D and 3D frameworks.

Results for the three-dimensional algorithm show that it is able to provide efficiently a smooth and safe trajectory to be followed by the pilots. This algorithm takes into account the initial and final heading of the aircraft, the aeronautical constraints from flight dynamics and the wind. Finally, the small memory space used and the reduced computation time make the algorithm promising for use in an FMS.

An extension of this work would be to study more complex scenarios and real cases (US Airways 1549, Swissair 111...) in order to fully validate the method, as well as to compare its performance with other trajectory generation algorithms. Improvements are yet to be found with the octree tetrahedrisation. The idea of using Delaunay tessellation was used in order to highlight similarities with the work in 2D, but other tessellations might be more convenient for these applications.

Finally, one can also propose other applications for this algorithm, as it gives a solution to a more general problem of finding the shortest paths for a vehicle that must satisfy certain dynamic constraints (robots, soaring flights, helicopters,...).

Appendices

A. Enumeration of the octree configurations

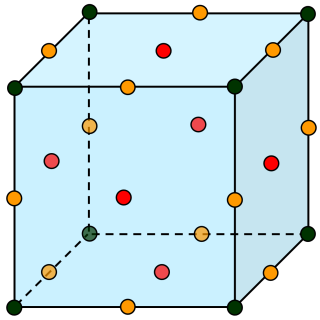
The objective of this appendix is to enumerate all the possible configurations for a node in a 2-balanced octree. An octree node is represented as a cube, with $F = 6$ faces, $E = 12$ edges and $V = 8$ vertices.

Two configurations are said to be *identical* if they possess the same *activated points*. A point is said to be *activated* if it is part of the final mesh. There are $6 + 12 + 8$ points that can be activated in a cube: the centers of the 6 faces, the centers of 12 edges and the 8 vertices, as shown in Fig. 33a.

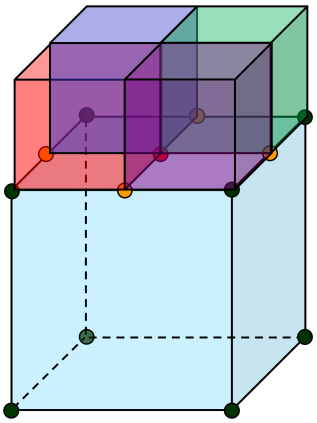
At first glance, there are $2^{F+E} = 2^{18}$ configurations possible for a node, whether each of its points are activated or not (its $N = 8$ vertices are always part of the mesh, thus are always activated).

However, if a center of one face is activated, the four centers of the edges composing this face are also activated. Indeed, since the octree is 2-balanced, neighbors in this direction have a level difference of 1 with the current node, thus there are 4 of these neighbors, which match the edges of the octree node and activate the centers of these 4 edges. This is illustrated in Fig. 33b.

This observation reduces strongly the number of configurations to study. Below are presented the only 10 cases to consider in order to enumerate all the possible configurations exactly once. For each case, the number and relative positions of neighbors in cardinal directions of considered node is set. The number of *free edges*, *i.e.* the number of



(a) Points of an octree node that can be part of the final mesh.

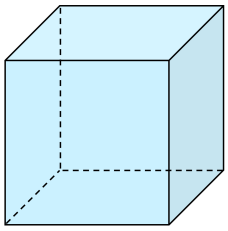


(b) Points activated by smaller neighbors in the Up direction.

Fig. 33 Notion of *activated points*.

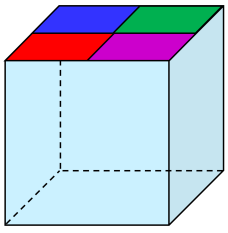
centers of edges that still can be activated (by inter-cardinal neighbors) among the $E = 12$ possibles points, are counted. If f is the number of free edges, and s the number of symmetrical configurations for the considered case, the total number of configurations for this particular case is $s \times 2^f$.

Case 1: No neighbors in any cardinal direction



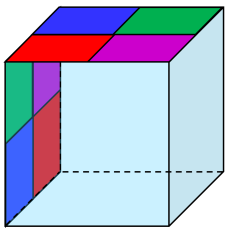
- Number of free edges: 12.
- Number of symmetrical configurations: 1.
- Total number of configurations : $1 \times 2^{12} = 4096$.

Case 2: Neighbors in 1 cardinal direction



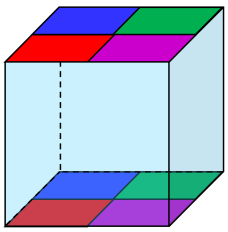
- Number of free edges: 8.
- Number of symmetrical configurations: $F = 6$.
- Total number of configurations : $6 \times 2^8 = 1536$.

Case 3: Neighbors in 2 adjacent cardinal directions



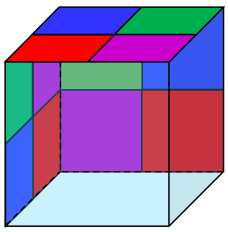
- Number of free edges: 5.
- Number of symmetrical configurations: $E = 12$.
- Total number of configurations : $12 \times 2^5 = 384$.

Case 4: Neighbors in 2 opposite cardinal directions



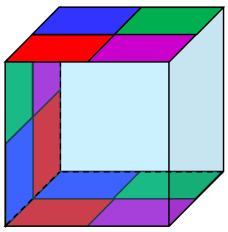
- Number of free edges: 4.
- Number of symmetrical configurations: $F/2 = 3$.
- Total number of configurations : $3 \times 2^4 = 48$.

Case 5: Neighbors in 3 adjacent cardinal directions



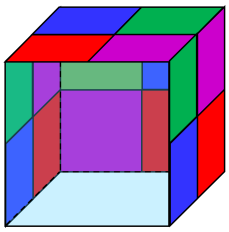
- Number of free edges: 3.
- Number of symmetrical configurations: $V = 8$.
- Total number of configurations : $8 \times 2^3 = 64$.

Case 6: Neighbors in 2 opposite cardinal directions and 1 other



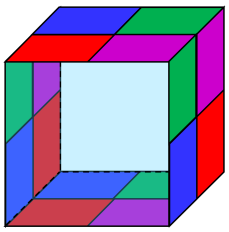
- Number of free edges: 2.
- Number of symmetrical configurations: $2 \times F = 12$.
- Total number of configurations : $12 \times 2^2 = 48$.

Case 7: Neighbors in 3 adjacent cardinal directions and 1 other



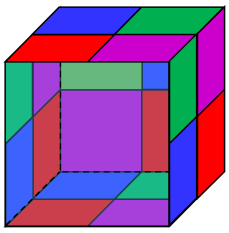
- Number of free edges: 1.
- Number of symmetrical configurations: $E = 12$.
- Total number of configurations : $12 \times 2^1 = 24$.

Case 8: Neighbors in 2 opposite cardinal directions and 2 other opposite cardinal directions



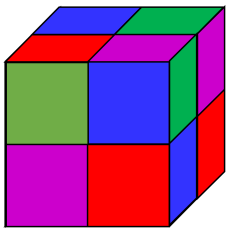
- Number of free edges: 0.
- Number of symmetrical configurations: $F/2 = 3$.
- Total number of configurations : $3 \times 2^0 = 3$.

Case 9: Neighbors in 5 cardinal directions



- Number of free edges: 0.
- Number of symmetrical configurations: $F = 6$.
- Total number of configurations : $6 \times 2^0 = 6$.

Case 10: Neighbors in 6 cardinal directions



- Number of free edges: 0.
- Number of symmetrical configurations: 1.
- Total number of configurations : $1 \times 2^0 = 1$.

Thereby, the overall number of configurations to consider in now : 4096+1536+384+48+64+48+24+3+6+1 = 6210 configurations.

B. Update Procedure on a Tetrahedron

The objective of this appendix is to extend the update procedure established by Kimmel and Sethian in [27] to the three-dimensional case.

We want to build a simple update procedure for the tetrahedron ABCD in which the point to update is D, with $T(A) \geq T(B) \geq T(C)$. All 4 triangles are supposed to be acute, as shown in Fig. 34.

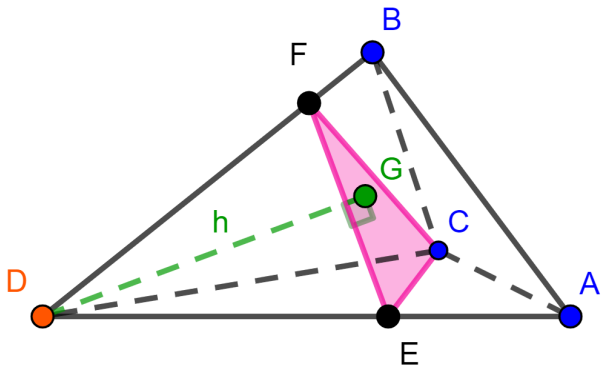


Fig. 34 Given the tetrahedron ABCD, such that $u_{\min} = T(B) - T(A)$ and $u_{\max} = T(C) - T(A)$, find $T(D) = T(A) + t$ such that $(t - u_{\max})/h = F$.

This means that we search for t such that

$$\frac{t - u_{\max}}{h} = F. \tag{19}$$

By similarity:

$$\frac{T(D) - T(A)}{DA} = \frac{T(E) - T(A)}{EA} \implies \frac{t}{DA} = \frac{u_{\max}}{EA} \implies \boxed{DE = DA\,\frac{t - u_{\max}}{t}} \tag{20}$$

$$\frac{T(D) - T(B)}{DB} = \frac{T(F) - T(B)}{FB} \implies \frac{t - u_{\min}}{DB} = \frac{u_{\max} - u_{\min}}{FB} \implies \boxed{DF = DB\,\frac{t - u_{\max}}{t - u_{\min}}} \tag{21}$$

Considering that h is the distance between the point D and the plane $(\overrightarrow{CE}, \overrightarrow{CF})$, we have:

$$h = \frac{|\overrightarrow{n} \cdot \overrightarrow{CD}|}{||\overrightarrow{n}||} \tag{22}$$

with $\overrightarrow{n}$ the normal vector of the plane $(\overrightarrow{CE}, \overrightarrow{CF})$, which is defined as:

$$\overrightarrow{n} = \overrightarrow{CE} \times \overrightarrow{CF} \tag{23}$$

From equations Eqs. (20) and (21), $\overrightarrow{CE}$ and $\overrightarrow{CF}$ are expressed as functions of t:

$$\boxed{\overrightarrow{CE} = \overrightarrow{CD} + \frac{t - u_{\max}}{t}\,\overrightarrow{DA}} \tag{24a}$$

$$\boxed{\overrightarrow{CF} = \overrightarrow{CD} + \frac{t - u_{\max}}{t - u_{\min}}\,\overrightarrow{DB}} \tag{24b}$$

Eq. (23) is developed as:

$$\boxed{\overrightarrow{n} = \frac{t - u_{\max}}{t - u_{\min}}(\overrightarrow{CD} \times \overrightarrow{DB}) + \frac{t - u_{\max}}{t}(\overrightarrow{DA} \times \overrightarrow{CD}) + \frac{(t - u_{\max})^2}{t(t - u_{\min})}(\overrightarrow{DA} \times \overrightarrow{DB})} \tag{25}$$

Hence, the numerator of Eq. (22) is rewritten:

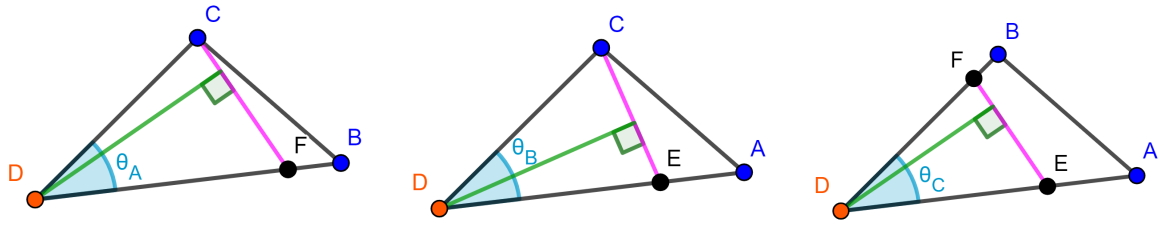
$$\left|\overrightarrow{n} \cdot \overrightarrow{CD}\right| = \frac{6(t - u_{\max})^2}{t(t - u_{\min})}\mathcal{V}_{\text{ABCD}} \tag{26}$$

with $\mathcal{V}_{\text{ABCD}}$ the volume of the tetrahedron ABCD.

The denominator of equation Eq. (22) can also be developed:

$$||\overrightarrow{n}|| = 2(t - u_{\max})\sqrt{\frac{\mathcal{A}^2_{\text{BCD}}}{(t - u_{\min})^2} + \frac{\mathcal{A}^2_{\text{ACD}}}{t^2} + \frac{(t - u_{\max})^2}{t^2(t - u_{\min})^2}\mathcal{A}^2_{\text{ABD}}} \tag{27}$$

with $\mathcal{A}_{\text{BCD}} = \frac{1}{2}||\overrightarrow{DB}||\,||\overrightarrow{DC}||\,|\sin\theta_A|$, $\mathcal{A}_{\text{ACD}} = \frac{1}{2}||\overrightarrow{DA}||\,||\overrightarrow{DC}||\,|\sin\theta_B|$ and $\mathcal{A}_{\text{ABD}} = \frac{1}{2}||\overrightarrow{DA}||\,||\overrightarrow{DB}||\,|\sin\theta_C|$ the areas of the triangles BCD, ACD and ABD respectively (see Figure 35).



(a) Triangle BCD **(b) Triangle ACD** **(c) Triangle ABD**

Fig. 35 Triangles in the tetrahedron.

Replacing (26) and (27) in (22), then in (19):

$$\mathcal{A}^2_{\text{BCD}}t^2 + \mathcal{A}^2_{\text{ACD}}(t - u_{\min})^2 + \mathcal{A}^2_{\text{ABD}}(t - u_{\max})^2 = 9F^2\mathcal{V}^2_{\text{ABCD}} \tag{28}$$

We end up with the quadratic equation for t:

$$\boxed{\left(\mathcal{A}_{\text{BCD}}^2 + \mathcal{A}_{\text{ACD}}^2 + \mathcal{A}_{\text{ABD}}^2\right) t^2 - 2\left(\mathcal{A}_{\text{ACD}}^2 u_{\min} + \mathcal{A}_{\text{ABD}}^2 u_{\max}\right) t + \mathcal{A}_{\text{ACD}}^2 u_{\min}^2 + \mathcal{A}_{\text{ABD}}^2 u_{\max}^2 - 9F^2\mathcal{V}_{\text{ABCD}}^2 = 0} \tag{29}$$

The solution t must satisfy $u_{\max} < t$, and should be updated from within the tetrahedron, namely:

$$DC\cos\theta_A < DF < \frac{DC}{\cos\theta_A} \tag{30a}$$

$$DC\cos\theta_B < DE < \frac{DC}{\cos\theta_B} \tag{30b}$$

$$DF\cos\theta_C < DE < \frac{DF}{\cos\theta_C} \tag{30c}$$

Now in terms of t:

$$\boxed{DC\cos\theta_A < DB\,\frac{t-u_{\max}}{t-u_{\min}} < \frac{DC}{\cos\theta_A}} \tag{31a}$$

$$\boxed{DC\cos\theta_B < DA\,\frac{t-u_{\max}}{t} < \frac{DC}{\cos\theta_B}} \tag{31b}$$

$$\boxed{DB\cos\theta_C < DA\,\frac{t-u_{\min}}{t} < \frac{DB}{\cos\theta_C}} \tag{31c}$$

The update procedure is given as follows:

Algorithm 3 Update on a 3D Meshpoint Procedure

Solve Eq. (29). Note t the solution.
if $t > u_{\max}$ and all 3 conditions of Eq. (31) are checked **then**
  $T(D) = \min\left(T(D), t + T(A)\right)$
else
  Solve quadratic equation in the triangle ABD. Note t' the solution.
  if $t' > u_{\min}$ and condition of Eq. (31c) is checked **then**
    $T(D) = \min\left(T(D), t' + T(A)\right)$
  else
    $T(D) = \min\left(T(D), AD \times F + T(A), BD \times F + T(B), CD \times F + T(C)\right)$
  end if
end if

References

[1] European Comission, "SafeNcy - the safe emergency trajectory generator," , 2019. URL `https://cordis.europa.eu/project/id/864771`.

[2] ChrisnHouston, "Trajet du vol US Airways 1549 le 15 janvier 2009," , Feb 2019.

[3] Bellman, R., "On a Routing Problem," *Quarterly of Applied Mathematics*, Vol. 16, No. 1958, 1958, pp. 87–90.

[4] Ford Jr., L. R., "Paper P-923," *Network Flow Theory*, 1956.

[5] Moore, E. F., "The Shortest Path Through a Maze," *Proc. Internat. Sympos. on Theory of Switching*, 1957.

[6] Dijkstra, E. W., "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, Vol. 1, No. 1, 1959, pp. 269–271.

[7] Hart, P. E., Nilsson, N. J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100–107.

[8] Karaman, E., S.and Frazzoli, "Incremental sampling-based algorithms," *Robotics Science and Systems VI*, 2010.

[9] Karaman, S., and Frazzoli, E., "Sampling-based Algorithms for Optimal Motion Planning," *arXiv:1105.1186 [cs]*, 2011. URL `http://arxiv.org/abs/1105.1186`, arXiv: 1105.1186.

[10] Gammell, S., J.and Srinivasa, and Barfoot, T., "Informed RRT* : Optimal Sampling-based Path Planning Focused via Direct Sampling of an admissible Ellipsoidal Heuristic," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[11] Pharpatara, P., Herisse, B., and Bestaoui, Y., "3-D Trajectory Planning of Aerial Vehicles Using RRT*," *IEEE Transactions on Control Systems Technology*, Vol. 25, No. 3, 2017, pp. 1116–1123. https://doi.org/10.1109/TCST.2016.2582144, URL http://ieeexplore.ieee.org/document/7505628/.

[12] Chakrabarty, A., and Langelaan, J., "UAV flight path planning in time varying complex wind-fields," *2013 American Control Conference*, IEEE, Washington, DC, 2013, pp. 2568–2574. https://doi.org/10.1109/ACC.2013.6580221, URL http://ieeexplore.ieee.org/document/6580221/.

[13] Sethian, J. A., "Fast Marching Methods and Level Set Methods for Propagating Interfaces," *von Karman Institute Lecture Series, Computational Fluid Mechanics*, 1998.

[14] Sethian, J. A., and Vladimirsky, A., "Ordered Upwind Methods for Static Hamilton–Jacobi Equations: Theory and Algorithms," *SIAM Journal on Numerical Analysis*, Vol. 41, No. 1, 2003, pp. 325–363. https://doi.org/10.1137/S0036142901392742, URL https://epubs.siam.org/doi/10.1137/S0036142901392742.

[15] Girardet, B., Lapasset, L., Delahaye, D., and Rabut, C., "Wind-optimal path planning: Application to aircraft trajectories," *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, IEEE, Singapore, 2014, pp. 1403–1408. https://doi.org/10.1109/ICARCV.2014.7064521, URL http://ieeexplore.ieee.org/document/7064521/.

[16] Atkins, E. M., Portillo, I. A., and Strube, M. J., "Emergency Flight Planning Applied to Total Loss of Thrust," *Journal of Aircraft*, Vol. 43, No. 4, 2006, pp. 1205–1216.

[17] Dubins, L. E., "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *Am. J. Math*, Vol. 79, 1957, pp. 497–516.

[18] Atkins, E. M., "Emergency Landing Automation Aids: An Evaluation Inspired by US Airways Flight 1549," *AIAA Infotech@ Aerospace 2010*, 2010.

[19] Tang, P., Zhang, S., and Li, J., "Final Approach and Landing Trajectory Generation for Civil Airplane in Total Loss of Thrust," *Procedia Engineering*, Vol. 80, 2014, pp. 522–528.

[20] Fallast, A., and Messnarz, B., "Automated trajectory generation and airport selection for an emergency landing procedure of a CS23 aircraft," *CEAS Aeronautical Journal*, 2017.

[21] Guitart, A., Delahaye, D., and Feron, E., "An Accelerated Dual Fast Marching Tree Applied to Emergency Geometric Trajectory Generation," *Aerospace*, Vol. 9, No. 4, 2022. https://doi.org/10.3390/aerospace9040180, URL https://www.mdpi.com/2226-4310/9/4/180.

[22] Sáez, R., Khaledian, H., Prats, X., Guitart, A., Delahaye, D., and Feron, E., "A Fast and Flexible Emergency Trajectory Generator Enhancing Emergency Geometric Planning with Aircraft Dynamics," *Fourteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2021)*, New Orleans (virtual), United States, 2021. URL https://hal-enac.archives-ouvertes.fr/hal-03351220.

[23] Haghighi, H., Delahaye, D., and Asadi, D., "Performance-based emergency landing trajectory planning applying meta-heuristic and Dubins paths," *Applied Soft Computing*, Vol. 117, 2022, p. 108453. https://doi.org/https://doi.org/10.1016/j.asoc.2022.108453, URL https://www.sciencedirect.com/science/article/pii/S1568494622000242.

[24] Ligny, L., Guitart, A., Delahaye, D., and Sridhar, B., "Aircraft Emergency Trajectory Design: A Fast Marching Method on a Triangular Mesh," *Fourteenth USA/Europe Air Traffic Management Research and Development Seminar*, New-Orleans, United States, 2021.

[25] Finkel, R. A., and Bentley, J. L., "Quad Trees: A Data Structure for Retrieval on Composite Keys," *Acta Informatica*, Vol. 4, No. 1, 1974, pp. 1–9.

[26] Reddy GVS, P., Montas, H. J., Samet, H., and Shirmohammadi, A., "Quadtree-Based Triangular Mesh Generation for Finite Element Analysis of Heterogeneous Spatial Data," *ASAE*, 2001.

[27] Kimmel, R., and Sethian, J. A., "Computing geodesic paths on manifolds," *Proc. Natl. Acad. Sci. USA*, Vol. 95, 1998, pp. 8431–8435.

[28] Isaac, T., Burstedde, C., and Ghattas, O., "Low-Cost Parallel Algorithms for 2:1 Octree Balance," *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 426–437.

[29] Morton, G. M., "A Computer Oriented Geodetic Data Base; And a New Technique in File Sequencing," Tech. rep., IBM Ltd, Ottawa, Canada, 1966.

[30] Zermelo, E., "Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung," *Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 11, No. 2, 1931, pp. 114–124.

[31] Janson, L., Schmerling, E., Clark, A., and Pavone, M., “Fast Marching Tree : a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions,” , 2015.