Thirteenth USA/Europe Air Traffic Management Research and Development Seminar (ATM2019)

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

Lucas Ligny, Andréas Guitart and Daniel Delahaye
École Nationale de l'Aviation Civile
Toulouse, France
ligny.lucas9@gmail.com

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

***Abstract*—In this paper, an efficient algorithm to generate a short and safe trajectory for an aircraft in a situation of emergency is proposed. The algorithm is to be run on a Flight Management System, hence the computation performance, the size of the stored data and the quality of the solution are taken as primary stakes. The algorithm is based on a front propagation algorithm, the Fast Marching method. Fronts are propagated on a single glide slope, allowing the algorithm to provide a flyable trajectory in a very short time (a few seconds). The algorithm is tested in a mountainous environment, and both shortness and safeness are obtained in response to a critical emergency.**

***Keywords*—Emergency Trajectory Generation, Fast Marching, Glide Slope, Triangular Mesh, Quadtree.**

I. Introduction

THE framework of application of this paper is the optimization of descent trajectories when a situation of emergency occurs on an aircraft. Hence, the generation of the trajectory must be fast to allow pilots to react quickly and to maximize the chances of a successful landing.

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: the trajectory has to be flyable. Considering such constraints, there is a fundamental need for the algorithm to be efficient while providing a high quality solution. All computations needed to be done on a Flight Management System (FMS), then the balance between speed, accuracy and algorithmic cost is at the core of this study.

One can find examples of critical emergency situations in aviation history. One can remind 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 Figure 1).

These two events correspond to two separate types of emergencies: As Soon As Possible (ASAP) and At Nearest Suitable Airport (ANSA). In the case of ASAP emergencies,

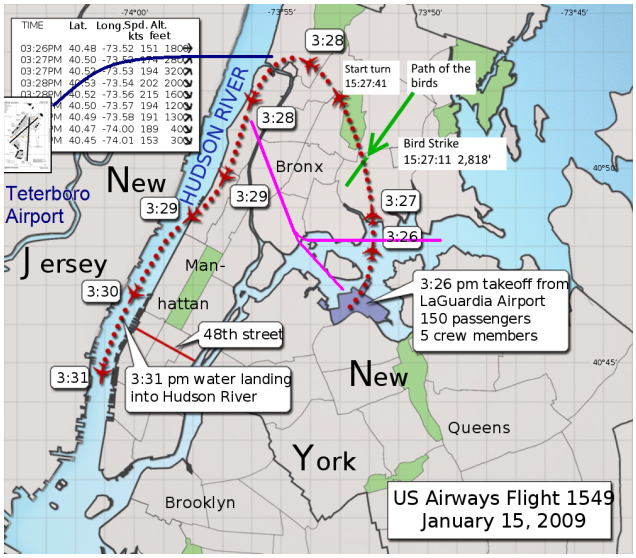


Figure 1: US Airways Flight 1549. 2 minutes after take-off from LaGuardia airport, the aircraft struck a flock of Canada geese at an altitude of 2,818 feet. 5 minutes after take-off, the aircraft made an unpowered ditching into the Hudson River.

for example when there is an on-board fire or an urgent medical issue, one has to find the fastest way to land. On the other hand, when considering ANSA emergencies such as a loss of engines, one has to find the safest path, to land in the best possible conditions.

The structure of this paper is the following. First, some previous related works regarding trajectory generation are introduced in Section II. Then, the mathematical modeling of the problem is described in Section III: considering the equations of Flight Dynamics, one can work in a two-dimensional space instead of a three-dimensional space, by restricting the aircraft to follow a single glide slope. Thereafter, the resolution algorithm based on a Fast Marching method is described in Section IV. The whole data structure is efficiently stored in a quadtree, which is first balanced and then meshed with triangles to propagate accurately the Fast Marching front. Finally, a validation of the algorithm is proposed in Section V for two key scenarios in a mountainous region, along with some performance indicators to check the efficiency of the method.

II. Previous related works

A. Aircraft emergency trajectory design

In [1], Atkins 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 [2]. This algorithm is applied to Flight 1549 in [3], and the algorithm returned a solution that could have enabled a safe return to LaGuardia airport, if such a technology was available in 2009. In [4], the Two Points Boundary Value Problem (TPBVP) is solved in to generate unpowered landing trajectories and improve the aircraft safety. These articles propose real-time solutions for an aircraft in a situation of emergency, to be run independently of the FMS. In our paper, we want to be able to use the current FMS to implement the trajectory generation algorithm.

Fallast and Messnarz [5] 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 connection is 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 [2], and limit these connections by considering the maximum climb and descent rates.

Guitart *et al.* in [6] 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 initialization 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 contraints.

B. Trajectory generation algorithms

1) Graph-based approaches: The Shortest Path Problem (SPP) has been explored by mathematicians during the second half of the twentieth century. One of the first well-known discrete algorithm to compute a trajectory between two points is the Bellman-Ford algorithm [7]–[9]. Bellman-Ford algorithm is very general, in the sense that it is able of handling graphs with negative edge weights.

Bellman-Ford runs in $\mathcal{O}(|\mathcal{V}||\mathcal{E}|)$, 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 [10], which has a faster convergence of $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}| \log |\mathcal{V}|)$. Knowing the destination vertex d, one can implement the A* search [11], an algorithm that remains similar to Dijkstra's but achieves convergence in $\mathcal{O}(|\mathcal{E}|)$.

Nevertheless, these algorithms compute the geodesic distance on the graph, hence they suffer from grid bias. For real-life applications such as the optimization 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 discretization. To avoid such problems, one can work with 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:

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

where T is the cost function to reach any point in space, and f is the "slowness" of the domain at any point. f characterizes 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. One can make the analogy with a forest fire, where some areas are moist (the fire propagation is slow) and others are dry (the fire propagation is fast).

The Fast Marching algorithm can be described in Algorithm 1. One can note the 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 Figure 2.

Algorithm 1: Fast Marching [Sethian, 1998]

```
1  Alive ← ∂Ω
2  Close ← N(Alive)
3  Far ← Ω \ {Alive ∪ Close}
4  while Far ≠ ∅ do
5  |   Trial ← ω ∈ Close with the smallest T value
6  |   Alive ← Alive + {Trial}
7  |   Close ← Close \ {Trial}
8  |   for neighbor ∈ N({Trial}) do
9  |   |   if neighbor ∉ Alive then
10 |   |   |   if neighbor ∈ Far then
11 |   |   |   |   Far ← Far \ {neighbor}
12 |   |   |   end
13 |   |   |   Close ← Close + {neighbor}
14 |   |   |   Update(neighbor)
15 |   |   end
16 |   end
17 end
```

These previous related works have shown that many approaches can be used to generate a trajectory. In the following section, the mathematical modeling is described. From the equations of Flight Dynamics, one can state the optimization problem to solve in order to obtain the optimal three-dimensional path. By limiting the aircraft descent to a single glide slope, the optimization problem is reformulated and simplified, fit to be solved by an efficient two-dimensional Fast Marching algorithm.

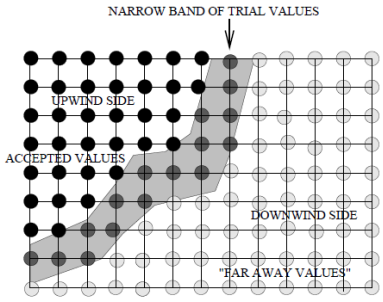
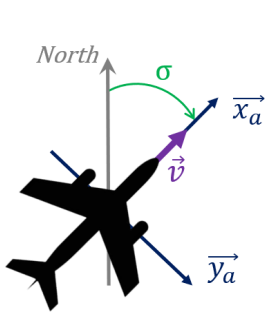


Figure 2: Upwind construction of accepted values [12, Fig. 14]. In the narrow band of trial values (*Close* in Alg. 1), the Eikonal equation is solved. The front propagates from accepted values (*Alive* in Alg. 1) towards "far away" values (*Far* in Alg. 1).

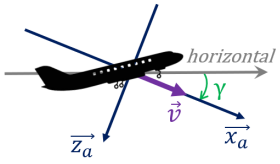
III. Mathematical Modeling

A. Flight Dynamics

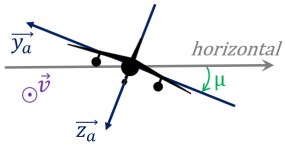
Let us consider an aircraft in the aerodynamic frame, with heading angle σ, flight path angle γ and bank angle μ (see Figure 3).



(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.

Figure 3: 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 $\vec{v}$ (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:

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max} \tag{III.1}$$

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

$$r = v_h \left(\frac{\partial \sigma}{\partial t} \right)^{-1} \tag{III.2}$$

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:

$$\left| \frac{\partial \sigma}{\partial x} \right| \leq \frac{1}{r_{\min}} \tag{III.3}$$

B. 3D general approach

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 optimization problem:

$$T(x,y,z) = \min_{\pi \in \Pi} \int_{P_{start}}^{P_{end}=(x,y,z)} f(\pi(\tau))d\tau \tag{III.4}$$

$$\text{subject to (III.1) and (III.3)}$$

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. $(\mathrm{L/D})_{\max}$ characterizes the ability of an aircraft to glide. An A320 has a maximum lift-to-drag ratio of 17, this means that when the aircraft has glided for 1 NM, it has lost a height of:

$$\frac{1}{17} \times \frac{1.852 \times 1000}{0.3048} = 357\,\text{ft}.$$

The constraint (III.1) can be reexpressed as follows:

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max}^{\text{gliding}} \tag{III.5}$$

with $\gamma_{\max}^{\text{gliding}} = -\arctan \dfrac{1}{(\mathrm{L/D})_{\max}}$.

This value is always negative (*e.g.* for an A320, $\gamma_{\max}^{\text{gliding}} = -3.37°$), which means that the aircraft is forced to go down.

C. Limitation of a single glide slope

In the following, as a first approximation, descent will be operated with a single slope. When a situation of emergency occurs, pilots have very little time to land, and performing complex procedures cannot be advised. Nevertheless, this limitation has consequences on the allowed behaviour of the aircraft and the optimization problem must be rewritten.

In two dimensions ($(\mathbf{x}, \mathbf{y})$ basis of $\mathbb{R}^2$), the objective can be written the same way as in the three dimensional case:

$$T(x,y) = \min_{\pi \in \Pi} \int_{P_{start}}^{P_{end}=(x,y)} f(\pi(\tau))d\tau \tag{III.6}$$

where $T(x,y)$ 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}.

On a single glide slope $\mathcal{GS}$, the constraint $\gamma = \gamma_0$ (a constant) cannot be satisfied, otherwise the aircraft would be forced to fly in straight line and would not be able to turn to avoid obstacles. Hence, the lighter constraint forcing the

aircraft trajectory to stay in the gliding slope is added. Considering expressions obtained in Section III-B, the constraints are as follows (in the general case):

$$\pi \subset \mathcal{GS}, \text{ (III.1) and (III.3)}$$

As mentioned before, in the special case where the engines are lost, (III.5) replaces (III.1).

D. Fast Marching algorithm

In Section III-C, a new formulation for the path optimization problem was established. Noting that the minimal cost paths are orthogonal to the level curves, one can notice that the equation left to solve is

$$|\nabla T| = f(x, y) \tag{III.7}$$

which is the two-dimensional Eikonal equation.

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

$$\begin{aligned} &\max\left(D_{ij}^{-x}T, -D_{ij}^{+x}T, 0\right)^2 \\ &\quad + \max\left(D_{ij}^{-y}T, -D_{ij}^{+y}T, 0\right)^2 = f_{i,j}^2 \end{aligned} \tag{III.8}$$

where the forward and backward operations are given by $D_{ij}^{-x}T = (T_{i,j} - T_{i-1,j})/\Delta x$, $D_{ij}^{+x}T = (T_{i+1,j} - T_{i,j})/\Delta x$, $D_{ij}^{-y}T = (T_{i,j} - T_{i,j-1})/\Delta y$ and $D_{ij}^{+y}T = (T_{i,j+1} - T_{i,j})/\Delta y$, with grid steps Δx and Δy. Finally, $T_{i,j}$ and $f_{i,j}$ are respectively the cost and the slowness at gridpoint (i, j).

This scheme is said "upwind", because it chooses gridpoints in terms of direction of the flow of information, or in other words from smaller values of T to larger values. Hence, the algorithm builds the solution outwards from the smallest T value.

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}, by solving

$$X(t) = -\nabla T \qquad \text{given } X(0) = P_{end} \tag{III.9}$$

A simulation example is given in Figure 4, with two obstacles and where the propagation on the left side of the domain is two times slower than the propagation on the right.

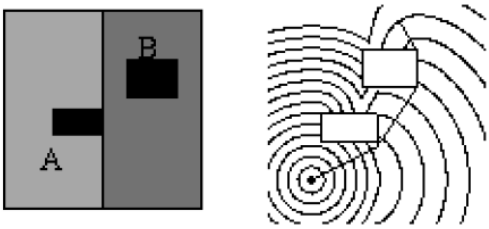


Figure 4: Two-dimensional navigation with constraints on variable domain [12, Fig. 21]. The domain is binary: the speed of propagation on the left is half the value as the one on the right. The generated trajectory is the optimal one considering the obstacles and the speed: it is orthogonal to the level curves.

The propagation depends on the chosen discretization of the domain Ω. In our paper, propagation is first presented over a grid, then over a triangular mesh. Note that in the back propagation phase, the optimal trajectory (geodesic) is built in a continuous maner without following the grid or mesh points.

1) Propagation over a grid: Let A, B and C be some of the grid vertices, and suppose that C is to be updated. If A is the only possible contributor, *i.e.* only A has already been updated, then the cost evaluated in C is $T(C) = hf_C + T(A)$ with h the grid step and f_C the slowness at point C.

If A and B are the only possible contributors with $T(B) \geq T(A)$, the *Update on a Gridpoint* Procedure must be used, which is illustrated in Figure 5.

Procedure Update on a Gridpoint

$T_1 \leftarrow$ solution of $(T - T(A))^2 + (T - T(B))^2 = h^2 f_C^2$ such that $T_1 > T(A)$ and $T_1 > T(B)$
if $T_1 \in \mathbb{R}$ **then**
  $T(C) = T_1$
else
  $T_2 \leftarrow$ solution of $(T - T(A))^2 = h^2 f_C^2$ such that $T_2 > T(A)$ and $T_2 \leq T(B)$
  $T(C) = T_2$

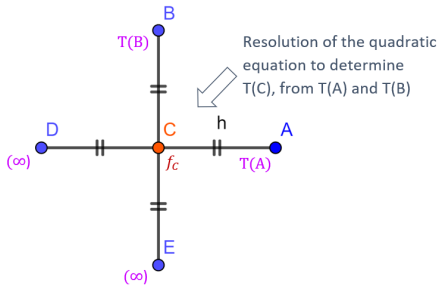


Figure 5: Update configuration on a gridpoint: A, B, C, D and E are some of the grid vertices, where C is the one to be updated. The magenta labels on nodes represent the current minimum costs to reach those nodes from the origin. A and B are the only possible contributors, hence C may be updated from both of these nodes.

2) Propagation over a triangular mesh: The same kind of algorithm can be applied on a triangular mesh. However, neighbors are no longer located on a regular grid but can be found on the vertices of irregular triangles.

Let us consider an acute triangle ABC with $T(A) \neq \infty$, *i.e.* A has already been updated. Such a configuration is represented in Figure 6. Once again the quadratic equation must be solved, to compute t such that $(t-u)^2 = h^2 f_C^2$ with $u = T(B) - T(A)$ and $t = T(C) - T(A)$.

Two cases can be studied:

- B and C are to be updated
- C is to be updated and $T(B) \geq T(A)$

The former case is trivial because the update comes from the edges: $T(B) = \min(T(B),\ cf_B + T(A))$ and $T(C) = \min(T(C),\ bf_C + T(A))$. The latter is more complex, please

refer to the work of Kimmel and Sethian [13] for detailed calculations. In this case, the quadratic equation can be written as follows:

$$(a^2 + b^2 - 2ab\cos\theta)t^2 + 2bu(a\cos(\theta) - b)t + b^2(u^2 - f_C^2 a^2 \sin^2(\theta)) = 0 \quad \text{(III.10)}$$

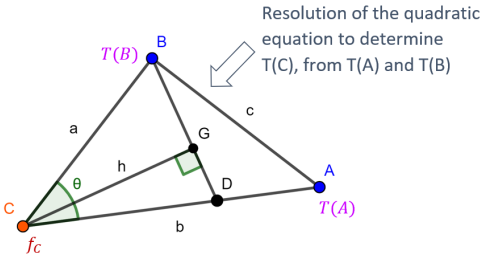


Figure 6: Update configuration on a meshpoint: A, B and C are some of the triangular mesh vertices. $\overrightarrow{GC}$ is the gradient associated with vertex C.

Since the update must be done within the triangle, the *Update on a Meshpoint* Procedure can be stated as follows:

Procedure Update on a Meshpoint

if $u < t \ \ \& \ \ a\cos\theta < b(t-u)/t < a/\cos\theta$ **then**
  $T(C) = \min\left(T(C),\ t + T(A)\right)$
else
  $T(C) = \min\left(T(C),\ bf_C + T(A),\ af_C + T(B)\right)$

This mathematical modeling allows us to develop an efficient algorithm based on a two-dimensional Fast Marching method over the glide slope of the aircraft.

IV. Resolution Algorithm

A. Generation of the 2D glide slope

The first step to build an efficient and complete structure is to collect terrain data. This operation can be done periodically (every 20s for instance) by using a forward looking radar, making terrain data available on the aircraft, or based on landscape data uploaded on the aircraft. Once the emergency is detected and declared, two points in space are considered to be known: the *EmergencyStartPoint i.e.* the position of the aircraft when emergency is declared, and the *EmergencyEndPoint i.e.* the Final Approach Fix (FAF) or the runway threshold of the best landing site. Thus, one can extrapolate the glide slope of the aircraft by setting one direction vector along the *StartCoordinate-EndCoordinate* axis, and another one orthogonal to this vector and parallel to the ground. The designation *Coordinate* is chosen to highlight their belonging to the glide slope. In Figure 7, an example of such glide slope coordinate system in the region of the Grenoble Alpes Isère airport (France) is presented.

One can easily see that the intersection between the terrain and the glide slope forms a grid composed of free-space cells and obstacles cells. Then, the objective is to navigate through the grid to create a trajectory from the *StartCoordinate* to the

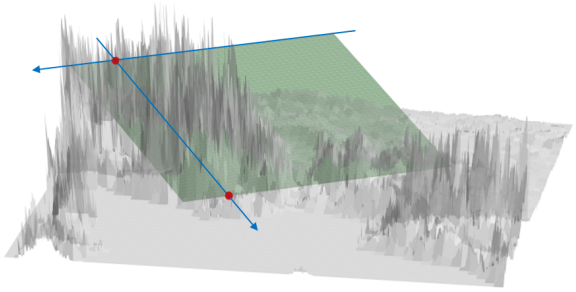


Figure 7: A glide slope in the mountainous region of Grenoble (France). *StartCoordinate* and *EndCoordinate* are both represented in red.

EndCoordinate (red points in Figure 7). An example of such plan is represented in Figure 8, where the obstacles are shown in black, and the free-space in white.

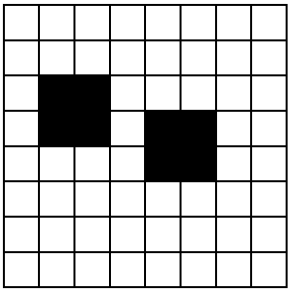


Figure 8: A 2D grid (of size 8x8). Two obstacles, in black, are restricting the free space.

B. Structural improvement

1) Quadtree generation: From Figure 8, a quadtree is generated. Originally formalized in [14], quadtrees are here presented in their linear version. A *linear quadtree* is a list containing free-space leaf nodes (or obstacles leaf nodes), characterized by their spatial address and their level. The encoding of spatial addresses is made using the Morton code [15]. Figure 9 shows the Morton codes, also called Z-values, for the two-dimensional case with integer coordinates.

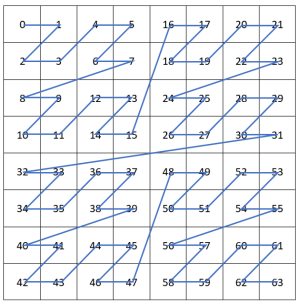


Figure 9: A Morton code on a 8x8 grid. A "Z" scheme is repeated successively throughout the grid to encode each cell.

In this paper, the convention is to work with the free-space leaf nodes and to set the convention for levels beginning at 0 for root node level, increasing with children. Result is shown in Figure 10.

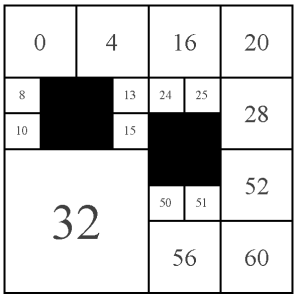


Figure 10: A quadtree computed from a 8x8 grid. It is composed of 4 levels ($8 = 2^{4-1}$). The free-space leaf nodes are assigned a spatial address from the Morton encoding. Leaves are denoted (*mortonCode*, *level*). For instance, the northwest leaf is denoted (0, 2): its Morton code is 0 because it is the minimum Morton code of cells which compose this bigger cell (here composed of 0, 1, 2 and 3, as in Figure 9), and its level is 2 because its size is 2x2. Note that level 3 corresponds to the maximum level a leaf can have (a 1x1 cell), and that level 0 corresponds to the entire grid.

One can see that leaf nodes have a higher level near the obstacles, and that huge zones of free-space are condensed into one leaf. This behaviour of quadtrees is interesting from the storage point of view, but when trying to generate a flyable trajectory it presents an issue. Indeed, the trajectory needs to be relatively smooth. To best meet this constraint and solve level discontinuities, the quadtree must be balanced.

2) Balancing operation: Balancing a quadtree consists in ensuring a level difference of at most 1 between two neighboring leaves. In Figure 10, the only leaf that needs to be divided is leaf (32, 1). The balancing operation (see Figure 11) will result in the removing of this leaf and the building of four new leaf nodes (32, 2), (36, 2), (40, 2) and (44, 2). The general algorithm to balance a quadtree is given in [16].

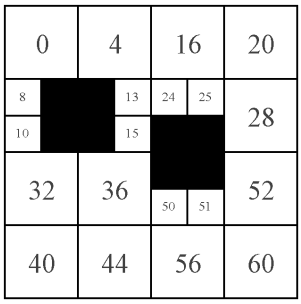


Figure 11: A balanced quadtree: each cell has a maximum of 1 difference level with its neighbors. In Figure 10, leaves (10, 3) and (32, 1) had a difference level of $3 - 1 = 2$. Here, the level difference between (10, 3) and (32, 2) is 1, thanks to the balancing operation.

3) Level differences computation: In order to quickly transform quadtree leaves from a grid into mesh cells (here, triangles), neighboring informations are added to those stored in the linear quadtree. The idea is to create a mesh vertex whenever there is a difference level of 1 between two leaves. Thus, one is able triangularize a leaf thanks to the knowledge of the level difference between the current leaf and the neighboring leaf. The linear quadtree becomes a *linear quadtree with level differences*, where each leaf is characterized by its Morton code m, its level l and its level differences δ. Considering domain sizes, one can easily assume that m is stored in an 4-byte signed integer and l in a byte.

One can show that δ can be stored in a byte. Indeed, after balancing the quadtree, the only possible values for a level difference are:

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

Therefore, each level difference is coded with 2 bits, or 8 bits for the four cardinal directions.

This is illustrated in Figure 12.

Figure 12: Level differences are coded with 1 byte (8 bits), with 4 duets of bits each corresponding to a cardinal direction in space.

4) Mesh generation: One has to define a scheme to build the mesh. The Delaunay triangulation is chosen, meshing that has proven adequate over balanced quadtrees in [16].

Thanks to the knowledge of level differences, the task of creating triangles is straightforward. Knowing the vertices that are on the edges of the square leaf on the grid (*e.g.* $\delta = 00000001 \implies$ 1 vertex in the middle of the West edge), the edges to add for building the mesh are directly given by the Delaunay triangulation scheme, in order to built two or more triangles inside this leaf. The building of the mesh is shown in Figure 13.

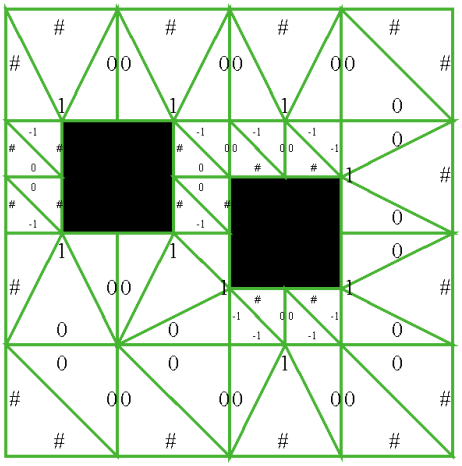


Figure 13: A triangular mesh adapted to the terrain. The triangulation is built over the balanced quadtree thanks to the level differences.

5) Aeronautical considerations: Depending on the aircraft position when the alert is raised, the following two situations may happen (see Figure 14): the *correct-energy* situation, when the aircraft is in the right range of altitude for which it can reach the landing site by keeping its flight path angle γ in the constraint limits (see Figure 14a), and the *over-energy* situation, when the aircraft is too high and cannot

reach the landing site without violating the γ constraint (see Figure 14b). The *under-energy* situation cannot happen, because the landing site is assumed to be reachable from *EmergencyStartingPoint*.

(a) The aircraft can reach directely the landing site.

(b) The aircraft has to lose altitude to be able to follow a single glide slope.

Figure 14: Case of (a) correct-energy, (b) over-energy.

For the *over-energy* problem, a solution would be to propose the pilots to make waiting racetracks from *EmergencyStartingPoint* to *StartCoordinate*. These racetracks can be modeled by Dubins curves, as shown in Figure 15. Such a procedure allow the pilots to easily lose altitude for connecting the glide slope plan in which the aircraft has to stay for approach (green plan on Figure 16).

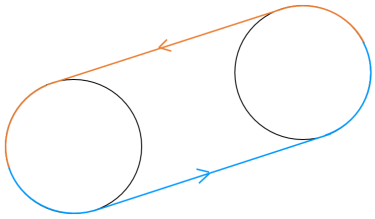


Figure 15: 360° turn with Dubins curves. Here, two Dubins curves of type Circle-Segment-Circle are necessary to make the turn.

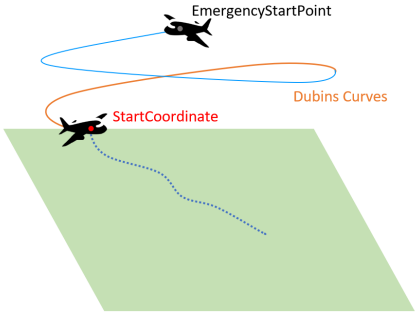


Figure 16: Initial procedure. From *EmergencyStartPoint* to *StartCoordinate*, if the generation of a single glide slope is impossible, the aircraft can make turns to lose altitude or get a better orientation.

Depending on the type of emergency the aircraft faces, some areas of the glide slope become unreachable (see Figure 17). Indeed, when engines are failing, the aircraft is limited by the minimum descent rate and cannot remain at the same flight level: it has to go down.

The heading of the aircraft in Final approach must be in accordance with the orientation of the landing site. Then, an approach procedure is implemented from *EndCoordinate* to *EmergencyEndPoint* to meet this requirement (see Figure 18).

A brief summary of the four critical points in space and their relations is presented in Figure 19.

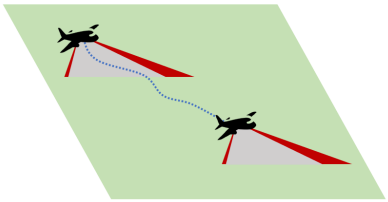


Figure 17: Heading constraints on the glide slope. The aircraft is forced to navigate inside a cone, limiting the local variations of heading angle.

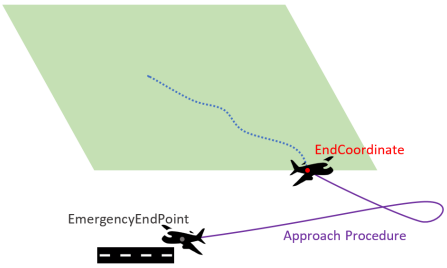


Figure 18: Final procedure. From *EndCoordinate* to *EmergencyEndPoint*, an additional procedure can be followed to get the aircraft aligned with the landing site.

C. Trajectory generation on the glide slope

The Fast Marching algorithm is based on the propagation over a mesh, but considering domain sizes, it is not optimal to generate the whole mesh at once. The mesh is built step by step, by dividing a quadtree leaf node into triangles only when the front enters inside it. Hence, the mesh is built only as needed, which allows to save both computation time and memory space.

The propagation of costs and gradients through the mesh occur between *StartCoordinate* and *EndCoordinate*. When visited, each mesh vertex is attributed a cost and a gradient according to Kimmel and Sethian's work [13] reminded in Section III-D2. The propagation stops when the quadtree leaf node containing the *EndCoordinate* is reached, as shown in Figure 20.

The final trajectory is found thanks to the back propagation of gradients, from *EndCoordinate* to *StartCoordinate*.

The choice of the mesh finds its importance with back propagation of gradients. Indeed, there must not be any too acute triangle, and there must not be too many triangles inside each node so that the quadtree structure is not altered. The Delaunay triangulation is a good candidate for these considerations: it ensures a minimum angle of 26.565° within each triangle [16], and it has the minimum number of triangles inside each node which connect all of the mesh vertices.

At each encounter with an edge, one needs to interpolate the gradient. It can be easily done by calculating the barycen-

Figure 19: Key Points and Coordinates.

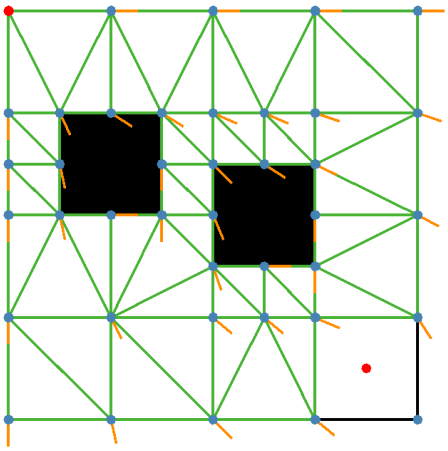


Figure 20: Propagation of costs and gradients. *StartCoordinate* (in the Northwest) and *EndCoordinate* (in the Southeast) are both represented in red. Gradients are represented in orange, mesh nodes in blue, and mesh triangles in green.

ter of the gradients linked with the two vertices of the edge considered (see Figure 21).

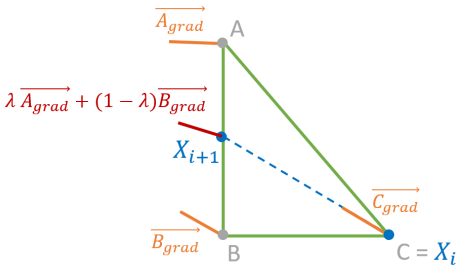


Figure 21: Interpolation of a gradient inside a triangle ABC. X_i is the last point visited by the back propagation. It coincides with the mesh vertex C, hence the gradient at X_i is $\overrightarrow{C_{grad}}$. The next point visited by the back propagation is X_{i+1}. It is at the intersection between $\overrightarrow{C_{grad}}$ and the edge AB, hence the gradient has to be interpolated. The interpolation parameter is $\lambda = \frac{||\overrightarrow{X_{i+1}B}||}{||\overrightarrow{AB}||}$. Thus, the gradient at X_{i+1} is $\lambda\overrightarrow{A_{grad}} + (1-\lambda)\overrightarrow{B_{grad}}$.

The algorithm stops on reaching *StartCoordinate*. Results for back propagation are shown in Figure 22.

The final trajectory between *StartCoordinate* and *EndCoordinate* can be found by linking the edge interception points together, as presented in Figure 23.

In the following section, validation of the resolution algorithm is made for the ASAP and the ANSA scenarios. A performance analysis is also established.

V. Results

Validation of the method is made in mountainous region of the Grenoble Alpes Isère airport. Two key scenarios are studied:

- ASAP emergency: The aircraft still has its fully operational engines. It is guided by a single glide slope, and is constrained by equations of Section III-C (general case).
- ANSA emergency: The aircraft has lost all of its thrust. It is guided by a single glide slope, and is constrained by equations of Section III-C (special case of lost engines).

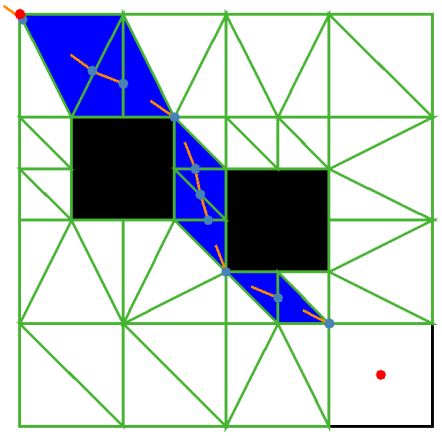


Figure 22: Back propagation from *EndCoordinate*. *StartCoordinate* (in the Northwest) and *EndCoordinate* (in the Southeast) are both represented in red, gradients in orange, mesh nodes in blue, and mesh triangles in green. Blue triangles are the triangles visited during back propagation.

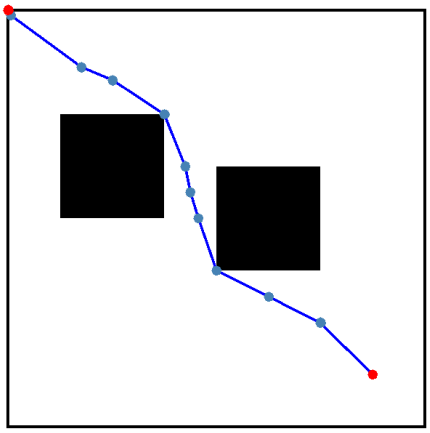


Figure 23: Trajectory generated with a Fast Marching algorithm on a 8x8 grid. The trajectory generated is very close to the optimal path, with weak errors due to the discretization.

The data is a 12000x12000 grid containing the heights of obstacles in a 50 NM radius around Grenoble airport. All simulations are run on a computer equipped with an i5-8250U processor and a RAM of 8 GB.

A. As Soon As Possible Emergencies (ASAP)

Results for the ASAP scenario are shown in Figure 24. In this figure, *StartCoordinate* is on the left, *EndCoordinate* on the right. The map corresponds to the glide slope of Figure 7.

This generated trajectory is really close to the shortest one. It has a mathematical interest, but it cannot be the one suggested to a pilot because it is a too difficult path to follow in such a situation, mainly because it uses some canyons in the mountainous area. One must not forget that the pilots are under a lot of stress, and asking to achieve very difficult maneuvers is too much. Therefore, instead of looking for the ideal shortest path, one can look for a sufficiently short and safe one, like in an ANSA scenario. This can be achieved by adding new constraints to the algorithm.

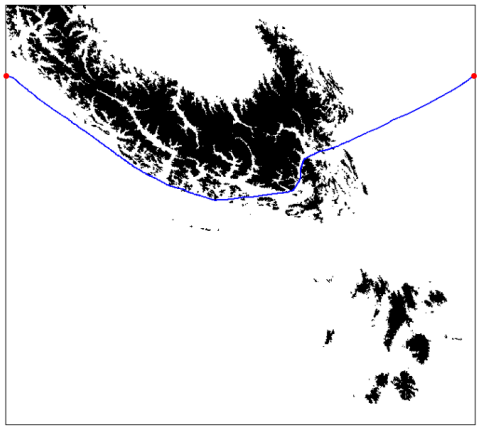


Figure 24: Trajectory generated in a ASAP scenario. The blue trajectory goes through the mountains, in order to minimize its length.

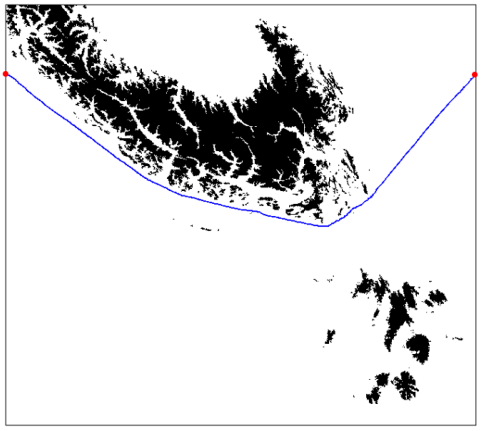


Figure 25: Trajectory generated in a ANSA scenario. The blue trajectory gets around the mountains, in order to minimize the length of the trajectory while keeping it safe and flyable.

B. At Nearest Suitable Airport Emergencies (ANSA)

Leaf nodes have a higher level near the obstacles. A first idea is to limit checking such nodes, because the aircraft cannot go too close to the obstacles and because it wastes time. Since the major part of discretization errors comes from the beginning and from the end of the propagation, a perimeter around *StartCoordinate* and *EndCoordinate* is created where the checking of high level nodes is allowed, and outside this perimeter their checking is forbidden. This technique let us go faster without damaging the trajectory too much, but also prevent the trajectory from getting very close to obstacles. This constraint makes the trajectory a little longer while making it safer.

Another update is the implementation of a metaheuristic on costs. Isotropic propagation is interesting but expensive in time, especially for the domain sizes considered. An idea is to slightly alter the *Update on a Meshpoint* Procedure detailed in Section III-D2 by focusing on nodes that are closer to the *EndCoordinate* during propagation. Thus, the cost of a point in space will not only be its distance from the origin but the sum of two distances, the one from the origin but also to the destination. Hence, fronts visit many fewer mesh nodes, and faster convergence of the algorithm is achieved. The revision of the procedure can be stated as follows:

Procedure Update on a Meshpoint, revised

if $u < t \ \ \& \ \ a\cos\theta < b(t-u)/t < a/\cos\theta$ **then**
  $T(C) = \min\left(T(C),\ t + T(A)\right)$
else
  $T(C) = \min\left(T(C),\ bf_C + T(A),\ af_C + T(B)\right)$
$T(C) = T(C) + distance(C,\ EndCoordinate)$

All these considerations are taken into account for the ANSA scenario, because the real shortest path is no longer the objective, which is now the shortest and the safest. Results for the ANSA scenario are shown in Figure 25.

This is the trajectory which intuitively connects the departure and arrival points in an optimal and safe way. This trajectory can be followed by a pilot quite easily, which guides the aircraft safely until Final approach.

It is interesting to evaluate the gain in computation time between the two scenarios. Table I presents the computation time of each major operation of the algorithm.

TABLE I. Comparison on Computation Times (in Milliseconds)

	ASAP	ANSA
Glide slope extraction	50	
Quadtree generation	100	
Balancing operation	100	
Level difference computation	50	
Fast Marching (*propagation*)	600	200
Fast Marching (*back propagation*)	30	20
Total	930	520

No matter the scenario, the algorithm manages to return a solution under a second. However, the ANSA scenario allow a much faster convergence of the Fast Marching algorithm (3 times faster) by visiting a small set of nodes.

In Table II are shown the different sizes of the stored data, and in Table III the number of nodes visited during the Fast Marching propagation.

TABLE II. Comparison on Stored Data Sizes (in Kilooctets)

	File Size
Terrain Data	491 649
Quadtree	225
Balanced Quadtree with level differences	302

TABLE III. Comparison on the Number of Nodes Visited During Propagation

	File Size
Fast Marching Nodes (ASAP)	14704
Fast Marching Nodes (ANSA)	3140

Low-sized structures are obtained, that can easily be embedded on FMS. As expected, thanks to the improvement brought for the ANSA scenario, the Fast Marching is able to divide the number of nodes visited by 5 while providing a trajectory of better quality than the ASAP scenario.

In Figure 26, the generated trajectory in its 3D environment is shown. The algorithm works in a highly mountainous environment, and produces a very satisfactory three-dimensional trajectory. Thus, the algorithm for the design of an aircraft emergency trajectory using a Fast Marching method on a triangular mesh is validated.

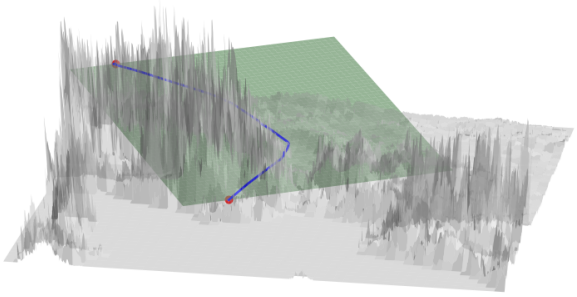


Figure 26: 3D trajectory generated with a 2D Fast Marching Algorithm over a single glide slope.

VI. Conclusion

The objectives of this paper were to develop an algorithm able to rapidly generate a safe trajectory in the event of an on-board emergency, in order to help pilots in such a situation. Constraints for the flight with or without power were established and were taken into account in a Fast Marching algorithm, able to provide efficiently a smooth and safe trajectory to be followed by the pilots. Results are good in terms of stored data size, computation time and quality of the solution. The algorithm is not able to take into account the initial and final heading of the aircraft, but procedures such as half turns or complete turns are implemented to provide a feasible solution in most scenarios. Moreover, the algorithm has the abilities to work on a FMS. If the emergency situation is complex, the algorithm can promptly run several times to evaluate different scenarios.

To go further, one could explore the effects of weather or traffic on the simulated trajectory. For instance, an Ordered Upwind method can be useful to allow taking into account the wind. Another extension would be to study more complex scenarios: to set an example, one could consider the particular scenario where the control wheel of the aircraft is broken, and the pilot only has the possibility to turn left. Finally, an implementation of a full 3D Fast Marching algorithm, along with a suitable meshing of the 3D domain, can be studied.

References

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

[2] L. E. Dubins, "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, pp. 497–516, 1957.

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

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

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

[6] A. Guitart, D. Delahaye, and E. Feron, "Automated Generation of Emergency Geometric Trajectory." Unpublished.

[7] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1958, pp. 87–90, 1958.

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

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

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

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

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

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

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

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

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

Author's Biographies

Lucas Ligny is a Master 2 student at École Nationale de l'Aviation Civile (ENAC) in Toulouse, France.

Andréas Guitart is a Ph.D candidate at École Nationale de l'Aviation Civile (ENAC) in Toulouse, France. He received his M.S.c in Aeronautic Engineering at ENAC in 2019. He is specialized in air traffic systems and optimization.

Daniel Delahaye is working as a Professor and Director in the OPTIM Laboratory of ENAC. He obtained his engineering degree from the ENAC school and a master of science in signal processing from the National Polytechnic Institute of Toulouse in 1991. He obtained his Ph.D in automatic control from the Aeronautic and Space National school in 1995 and did a post-doc at the Department of Aeronautics and Astronautics at MIT in 1996.

Banavar Sridhar is a Research Associate at the NASA Ames Research Center. Earlier he served as the NASA Senior Scientist for Air Transportation Systems. His research interests are in the application of modeling and optimization techniques to aerospace systems. Dr. Sridhar led the development of traffic flow management software, Future ATM Concepts Evaluation Tool (FACET), which received the AIAA Engineering Software Award in 2009, the NASA Invention of the Year Award in 2010 and the FAA Award for the Excellence in Aviation Research in 2010. He is a Fellow of the IEEE and the AIAA.