Development of trajectory generation algorithms to improve air transport safety

Andréas Guitart

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Andréas GUITART

Le 15 décembre 2023

Développement d'algorithmes de génération de trajectoire pour l'amélioration de la sécurité du transport aérien.

Ecole doctorale : **AA - Aéronautique, Astronautique**

Spécialité : **Mathématiques et Applications**

Unité de recherche :
ENAC-LAB - Laboratoire de Recherche ENAC

Thèse dirigée par
Daniel DELAHAYE et Eric FERON

Jury

Mme Eri ITOH, Rapporteure
M. Valentin POLISHCHUK, Rapporteur
M. Franck IUTZELER, Examinateur
M. John HAUSER, Examinateur
M. Daniel DELAHAYE, Directeur de thèse
M. Eric FERON, Co-directeur de thèse

Résumé

Cette thèse porte sur les algorithmes de génération de routes et de trajectoires permettant d'améliorer la sécurité du transport aérien. Cette thèse traite trois problèmes majeurs pouvant causer des accidents : les atterrissages d'urgence, l'évitement d'obstacles météorologiques et la résolution de conflits entre des avions. Durant cette thèse, nous nous sommes posé la question sur la manière de résoudre ces problèmes. La première idée a été de les traiter au niveau tactique. Tout d'abord, cette thèse porte sur le développement d'un outil d'aide à la décision pour les pilotes en cas de situation d'urgence. Ce travail a été mené dans le cadre du projet SafeNcy visant à créer un système composé d'un sélecteur de lieux d'atterrissage et d'un module générant la trajectoire. L'enjeu principal de ce problème est de développer un algorithme générant des trajectoires d'atterrissage en quelques secondes. L'autre enjeu majeur est qu'il existe énormément de situations d'urgence qui ont chacune leurs spécificités. Afin de répondre à ces enjeux, nous avons comparé trois méthodes: le Fast Marching Tree, le Fast Marching et le Rapidly-exploring Random Tree. Après plusieurs tests, le Rapidly-exploring Random Tree a été choisi pour le projet SafeNcy. Les résultats sont très satisfaisants, car pour l'intégralité des scénarios définis, le système génère plusieurs trajectoires en quelques secondes. Après s'être intéressés aux situations les plus critiques, nous avons étudié l'évitement d'obstacles météorologiques en temps réel. L'étude portait sur le développement d'un système collaboratif permettant à chaque avion de contourner l'orage tout en évitant des conflits. Une méthode basée sur le Fast Marching Tree et le principe du premier arrivé premier servi a été développée. Elle a été ensuite modifiée afin de prendre en compte l'équité entre les avions. Les simulations montrent que la méthode est équitable et génère des trajectoires d'évitement sans conflit en quelques secondes. Après l'étude des problèmes du point de vue tactique, nous nous sommes intéressés à des résolutions stratégiques. Tout d'abord, nous avons étudié le design de procédures sécurisés dans les TMA. L'approche est basée sur le recuit simulé. La TMA est discrétisée par un ensemble de cercles définissant la position des points de connexion entre les routes. Le profil horizontal de chaque route est choisi parmi un ensemble d'options prédéfinies. Le profil vertical est défini à partir des pentes minimales et maximales, auxquelles peut s'ajouter des paliers. En plus de la distance des routes, l'impact sonore a été considéré. La méthode a été comparée à la littérature sur l'aéroport Paris Charles-De-Gaulle. Ensuite, une étude de l'impact sonore des procédures a été effectuée. Enfin, l'algorithme a été testé sur la TMA de Paris. La conclusion de ces travaux est satisfaisante car la méthode est meilleure que celles de la littérature et fonctionne sur des scénarios complexes. Pour conclure cette thèse, une ouverture est proposée sur le développement d'un système pour prévenir les événements imprévus en générant en avance des trajectoires alternatives. Les enjeux de cette étude sont : les trajectoires alternatives ne doivent pas être trop éloignées du plan de vol initial; les trajectoires doivent être suffisamment différentes entre elles et au moins une trajectoire doit éviter un événement imprévu. Pour y répondre, la méthode est basée sur le Rapidly-exploring Random Graph, une clusterisation et une métrique de similarité. L'algorithme a été testé sur l'évitement de trainées de condensation considérées comme des obstacles durs. Les résultats semblent satisfaisants, car l'algorithme permet d'éviter des obstacles

de différentes tailles. Ce travail reste préliminaire, car de nombreuses améliorations restent à ajouter. Cette thèse ouvre la voie au développement d'outils d'aide à la décision permettant d'assister les acteurs du transport aérien dans des situations plus ou moins critiques.

Abstract

This thesis addresses route and trajectory generation algorithms to improve air transport safety. Despite being one of the safest means of transport, incidents and accidents occur every year. This thesis deals with three major problems which, when poorly managed, can cause accidents: emergency landings, avoiding meteorological obstacles and resolving conflicts between aircraft. During this thesis, we asked ourselves how to solve these problems. The first idea was to deal with them at the tactical level. First, this thesis deals with the development of a decision support tool for pilots in emergency situations. This work was conducted in the context of the project SafeNcy. Its aim is to develop a system comprising a landing site selector and a trajectory generation module. During this project, we mainly worked on the second module. The main challenge of this problem is to develop an algorithm that generates landing trajectories in a few seconds, to quickly propose solutions to pilots and thus maximize the chances of a safe landing. The other major challenge is linked to the big number of emergency situations, each with its own specific features. To address these issues, we compared three methods: the Fast Marching Tree algorithm, the Fast Marching method and the Rapidly-exploring Random Tree algorithm. After several tests, the Rapidly-exploring Random Tree algorithm was chosen to be part of the SafeNcy system. The results of the project are highly satisfactory, since for all the scenarios defined, the system generates several trajectories in a matter of seconds. After studying the most critical situations, we studied the avoidance of meteorological obstacles in real time. The aim of this research was to develop a collaborative system, enabling each aircraft to bypass the storm while avoiding conflicts. First, a method based on the Fast Marching Tree algorithm and first-come, first-served principle was developed. It was then modified to take into account the fairness constraint between aircraft. Simulations show that the method is fair and generates conflict-free avoidance paths in a few seconds. After dealing with problems from a tactical point of view, we turned our attention to strategic resolutions. First, we studied the design of safe procedures in TMAs. The approach is based on the Simulated Annealing. The TMA is discretized by a set of circles defining the position of connection points between routes. The horizontal profile of each route is defined by a set of predefined options. The vertical profile is defined by minimum and maximum slopes, to which level flights can be added. In addition to route distance, noise impact was taken into account. The proposed method was compared with previous works on procedures at Paris Charles-De-Gaulle airport. A noise impact study was then carried out. Finally, the algorithm was tested on the Paris TMA. The conclusion of this work is satisfactory, as the proposed method outperforms those in the literature and works on complex scenarios. To conclude this thesis, an approach is proposed to develop a system to prevent unforeseen events by generating alternative trajectories in advance. The main challenges of this study are: the alternative trajectories generated must not be too far from the initial flight plan; the trajectories must be sufficiently different from each other; and at least one trajectory must avoid an unforeseen event. To meet these challenges, the method is based on the Rapidly-exploring Random Graph algorithm, a clusterization and a similarity metric. The proposed algorithm was tested on the avoidance of contrails, which were considered as hard obstacles.

Initial results appear satisfactory, with the algorithm able to avoid obstacles of various sizes. This work remains preliminary, as many improvements still need to be added. This thesis opens the way to the development of decision support tools to assist air transport operators in more or less critical situations.

Acknowledgement

I would like to thank all the people who have helped me, directly or not, in this work. These years have been an opportunity for me to work on a subject and in a context that I love, alongside great people.

First of all, my thoughts go to my supervisors, Daniel Delahaye and Eric Feron, for their continuous support, expert guidance and encouragement throughout the challenging process of completing my thesis. They have always been helpful and caring, and it has been a real pleasure to share these years of my life with them. I am very fortunate to be able to continue working with Daniel for the next 3 years.

I would like to express my sincere gratitude to the members of my jury: the reviewers Eri Itoh and Valentin Polishchuk, and the examiners John Hauser, Franck Iutzeler and Bertrand Masson. They took the time to review my thesis and showed interest in it, which is a great honour coming from such experts in their field. The valuable comments and insights of the reviewers on my manuscript will help me to improve my work and I am very grateful for their thorough review.

I would also like to thank the other PhD students. It has been a pleasure to share this unique experience, during which we have had fascinating discussions, fruitful collaborations and moments of joy over a few beers. My special thanks go to all my colleagues in office Z102/103, Adrien, Clara, Geoffrey, Jean-Claude and Julien, and the coffee-loving office visitors, Bastien and Remi, for their very pleasant company. I would also like to thank Céline, with whom I had the pleasure of working on several projects, resulting in two articles and a survey. I am also thinking of Lucas Bonin, Lucas Ligny and Alexis Brun, whose projects I had the privilege of supervising. My thanks also go to the whole OPTIM lab, full of people who are as nice as they are competent. It will be a pleasure to continue working with this team over the next 3 years during my post-doc.

I thank my friends who supported me during my PhD and helped me to distract myself from the difficult times. I have a very special thought for Thomas and Julien who shared with me this special experience that is the PhD.

Finally, I would like to thank my parents for always being there for me in the good times and the bad. They have always helped me both mentally and financially. Without them, this thesis would not have been possible.

Contents

List of Figures

List of Tables

Introduction

Air transportation is one of today's quickest ways to travel. The first commercial flight took place on January 1^{st}, 1914. Since then, air transport has continued to grow until it reached 4.5 billion passengers and 46.8 million commercial flights in 2019 generating $3.5 trillion (4.1% of the world's global economic activity) and supporting 87 million jobs worldwide [10]. The COVID crisis recently slowed this growth in 2020 and 2021 before returning to normal in recent months. Despite this crisis, aviation will reach impressive figures in the coming decades. According to Eurocontrol [63], in 2040, the number of flights in Europe will have increased by more than 50%. This increase will certainly be even more important in China and in South and South East Asia according to Boeing [28]. Such growth raises many issues such as safety, economy, human factor, etc. Moreover, air transport is faced with a new major challenge linked to the reduction of its impact on the environment. Indeed, aviation is responsible for about 3-5% of total global warming [131]. In 2019, 914 million tonnes of CO_2 were emitted by airlines, 2% of the global man-made emissions [10]. Airlines and governments have already taken action on the matter, and CO_2 emissions per passenger per kilometer have been halved since 1990 [10]. This opens the way for new research to address this major issue. All the life cycle of the aircraft can be optimized and all the stakeholders can have an impact. The phases of a flight are quite complex and each requires optimization at different scales. In Europe, well before a flight, the airline submits a flight plan to the network manager who validates it or requests modifications to respect the capacities of the airspace overflown. On the day of the flight, the flight plan is modified according to the weather conditions. This flight plan is optimized depending on the airline's strategy represented by the cost index. It is the ratio of the time-related cost of an aircraft operation and the cost of fuel. Moreover, the flight plan takes into account hazardous areas such as thunderstorms and turbulence areas. Once the aircraft is ready for departure, pilots request to leave the parking and take off. Once taken off, they contact with approach control in *Terminal Maneuvering Area* (TMA) airspace and follow the *Standard Instrument Departure* (SID) to climb and reach an enroute sector. The enroute control is in charge of its safety, guaranteeing sufficient separations between aircraft. According to *International Civil Aviation Organization* (ICAO) [101], in enroute airspace, two aircraft have to be separated laterally by more than 5 Nautical Miles (NM) or vertically by more than 1000 feet (ft). The aircraft then follows its flight plan composed of navigation points called waypoints. The cruise phase is the longest part of a flight and is totally managed by the automatic pilot. During this phase, the pilot may have to perform short maneuvers to avoid conflicts or stormy areas. In the case of long-haul flights, the aircraft changes its altitude. Indeed, the aircraft consumes fuel and is lighter, so it can fly higher to increase its performance. Very exceptionally, the aircraft can also face emergency situations involving a forced landing. If all goes well, once near its destination, the aircraft begins its descent to the arrival TMA. It reaches the *Initial Approach Fix* (IAF) and follows the *Standard Terminal Arrival Route* (STAR). It is finally authorized to land by the tower controllers. Figure 1 summarizes all the steps of a standard flight.

All these flight phases presented above require optimization in order to guarantee efficient op-

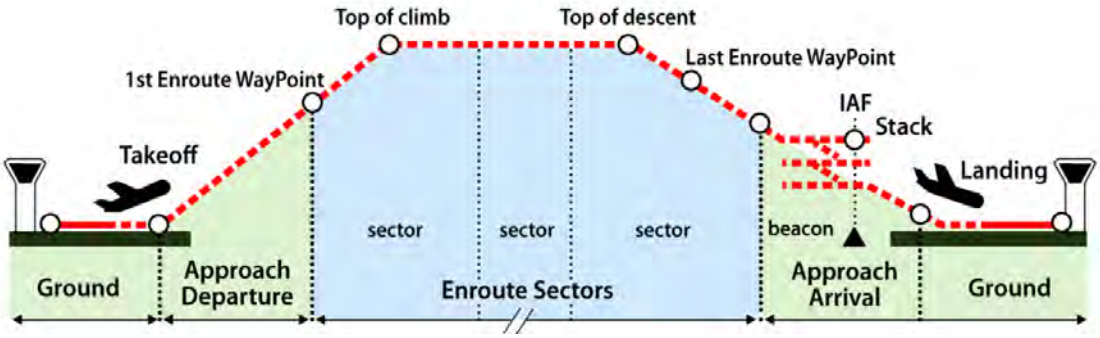


Figure 1: The different steps of a flight (taken from [132])

erations that meets the challenges of its growth. These optimizations can be done at different levels: strategic and tactical. The first level corresponds to planning level optimization at network scales. For instance, SID/STAR design and flight planning optimization can be done at the strategic level. The tactical level optimization corresponds to the development of real-time algorithms. For example, conflict resolution, hazards weather avoidance, or emergencies are problems that are tactically solved. However, for these types of problems, would it be possible to develop strategic optimization tools to prevent these problems? This question leads to another one: Is an ounce of prevention better than a pound of cure? Behind this expression is a simple and logical idea: it is better to foresee, prevent, and prepare rather than react at the last minute. Initially used in medicine to mean that it is better to prevent a disease than to cure it when it occurs. This expression was then diverted in a more global sense. Benjamin Franklin famously advised fire-threatened Philadelphians in 1736. Clearly, preventing fires is better than fighting them but to what extent can we protect ourselves from natural disasters? Hazards such as earthquakes, tsunamis, floods, hurricanes, and volcanic eruptions are not in themselves preventable, but some of their devastating effects could be reduced through forward planning. Can we also ask the question about the design of aircraft trajectories and more particularly in the case where the aircraft is in a critical situation? During a flight, an aircraft can be confronted with unforeseen events. These events are of varying criticality. For example, the occurrence of a hazardous weather area or a conflict with another aircraft is low critical. Indeed, in this type of situation, the pilots just have to perform a short maneuver to quickly solve the problem. These types of events are very frequent and are resolved without major problems. However, if this situation is not resolved or bad decisions are taken, it can create accidents. This is notably the case of the accident of the ADC Airlines Flight 053 killing 96 out of 105 people on board. Flying from Nnamdi Azikiwe International Airport to Sadiq Abubakar III International Airport, the flight crashed onto a corn field shortly after take-off. The investigation of the crash, conducted by Nigeria's Accident Investigation Bureau, blamed the pilot's decision to take off in unsuitable weather as the primary cause of the crash [155]. The aircraft can also be confronted with extremely critical situations with the obligation to land as soon as possible. For example, a fire may break out in the aircraft and require an emergency landing. For example, on 2 September 1998, the flight crew of Swissair Flight 111 detected 52 minutes after takeoff an odor in the cockpit and determined it to be smoke from the air conditioning system. Four minutes later, the odor returned and smoke became visible, prompting the pilots to make a "pan-pan" (emergency due to smoke) radio call to Moncton air traffic control (ATC), the area control center (ACC) station in charge of air traffic over the Canadian province of Nova Scotia. Moncton ATC's proposes to pilots a landing at the closer Halifax International Airport in Enfield. After a maneuver, for fuel shedding, in accordance with the Swissair checklist "In case of the smoke of unknown origin", the crew shut off power to the cabin, which also turned off the recirculating

fans in the cabin's ceiling. This allowed the fire to spread to the cockpit, eventually shutting off power to the aircraft's autopilot. The pilots, therefore, declared themselves in an absolute emergency but it was too late, they lost radio contact and the aircraft crashed a few minutes later. These two accidents raise many issues. Could they have been avoided? Could they have been predicted? Can everything be predicted? How to solve this type of problem? Feron *et al.* [71] introduce *Ariadne*, a thread dedicated to facilitating productivity gains for air traffic service providers around the world. It defines as an engineered extension of the common sense practice of always keeping a "Plan B", and possibly "plans C, D, E, F, etc" against unexpected events when any decision is made by pilots, air traffic controllers, dispatchers, and any other safety-critical actor of the air transportation system. Figure 2 illustrates the concept of *Ariadne* on the case of a transatlantic flight.

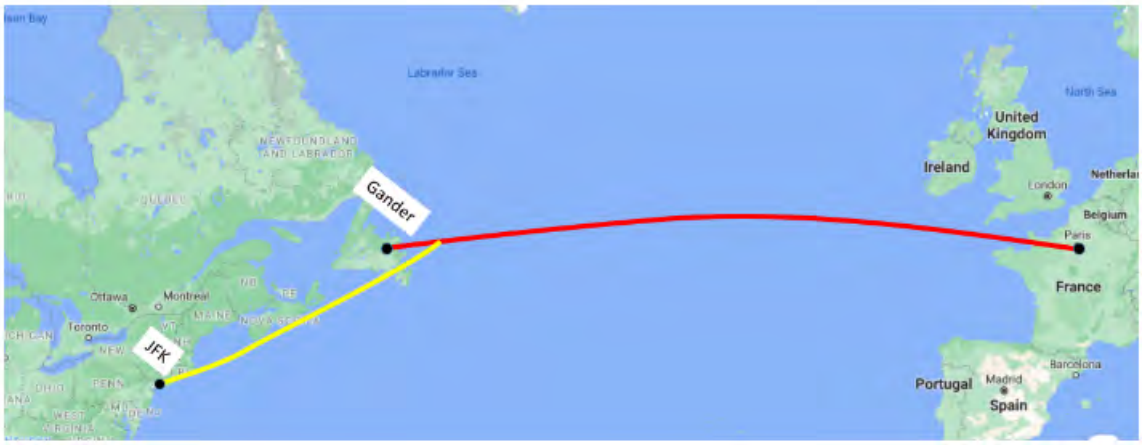


Figure 2: Typical transatlantic flight from CDG to JFK. Initial flight plan lands the aircraft in YQX (Gander). Flight plan is amended close to YQX to continue onto JFK, its final destination [71].

Although, prevention can significantly reduce the risk of unexpected events or solve them when they occur, in some very complex situations a cure is necessary. Moreover, some events are unpredictable in advance. Due to the great uncertainty about their positions, thunderstorms are difficult to predict and the associated avoidance trajectories cannot be computed in advance. Moreover, emergency situations are not easily forecasted. For example, it was impossible to predict the dual engine failure after bird strikes on US Airways Flight 1549. This accident is one of the most famous accidents in aviation history; it has been shooted in the movie Sully with Tom Hanks. This example shows that it seems essential to develop curative tools to help safety-critical actors of the air transportation system.

The goal of this thesis is to study the two concepts: "cure" and "prevention", by addressing four different problems. This manuscript is divided into six chapters. Chapter 1 presents a literature review of methods related to path planning algorithms and aircraft trajectory generation methods. Chapter 2 deals with the problem of aircraft emergency trajectory design which is the main topic addressed in this thesis. This work was guided by the European project SafeNcy [43]. Chapter 3 focuses on another "cure" problem: local conflict free trajectories with weather hazards avoidance. The two last chapters of this manuscript cover the strategic design of trajectories to improve safety and the prevention of unforeseen events. Chapter 4 presents the work on the design of departure and arrival procedures. Chapter 5 addresses preliminary works on alternative cruise trajectories generation. Finally, Chapter 6 concludes this manuscript with a review of the work and a discussion on the perspectives of this thesis.

Chapter 1

Literature Review

In this chapter, we present the previous related works on aircraft trajectory design with five aviation applications covered in this thesis: aircraft emergency trajectory design, departure and arrival procedures, hazards weather avoidance, conflict resolution and alternative trajectories generation.

1.1 Aircraft Trajectory

This section presents the representation of an aircraft trajectory and its features.

1.1.1 Trajectory modeling

Berger and Gostiaux [17] proposed models of aircraft trajectories. They are described as mappings from a time interval $[a, b]$ to state space E with $E = \mathbb{R}^3$ representing the three coordinates (x, y, z). The space state can be completed by the speed vector (v_x, v_y, v_z). In this case, E is $\mathbb{R}^6$.

1.1.1.1 Notations and Terminology

Let $\gamma[a, b] \rightarrow E$ be a trajectory. $\gamma(a)$ and $\gamma(b)$ are respectively the origin and the destination of the trajectory (See Figure 1.1).

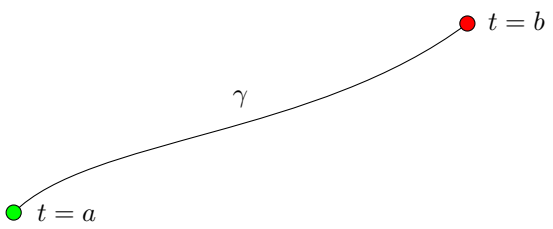


Figure 1.1: Trajectory representation

The trajectories are assumed to be at least continuously differential (class C^1). Therefore, the length of a trajectory $\gamma[a, b] \rightarrow E$ can be defined as follows:

$$l(\gamma) = \int_a^b ||\gamma^{'}(t)||dt. \tag{1.1}$$

In the following, the case $E = \mathbb{R}^3$ is considered.

Any C^1 curve can be parameterized by arc length. The mapping $s(a, b) \to (0, l(\gamma))$ is defined as:

$$s(t) = \int_a^t ||\gamma^{'}(x)||dx. \tag{1.2}$$

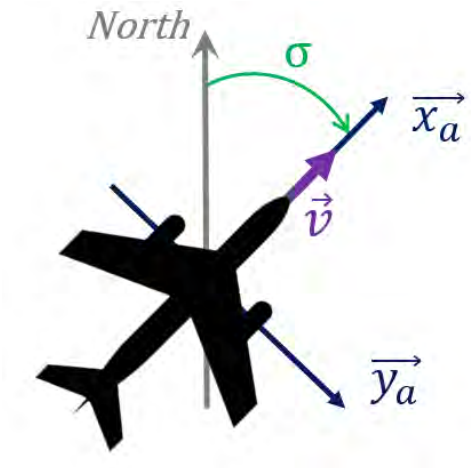
Taking into account the curvature is essential in the aircraft trajectory generation problems. If γ is C^2, the curvature can be defined as follows :

$$K(t) = ||\gamma^{''}(t)|| = \frac{||\gamma^{'}(t) \wedge \gamma^{''}(t)||}{||\gamma^{'}(t)||^3}. \tag{1.3}$$

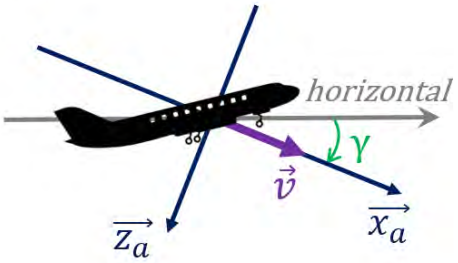
The case for which $E = \mathbb{R}^6$ is more complex. Indeed, the definition of curvature is not obvious. The extra degrees of freedom impose using higher order derivatives in order to build up an equivalent description.

1.1.2 Aeronautical features

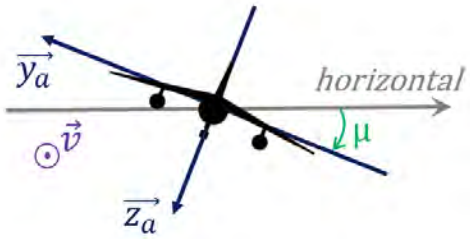
Let's consider an aircraft in the aerodynamic frame, with heading angle σ, flight path angle γ, and bank angle μ (see Figure 1.2).



(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 1.2: 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 with a maximum flight path angle $\gamma_{\max}$ defined from the RoC, and a minimum flight path angle $\gamma_{\min}$ defined from the RoD. This constraint is expressed as follows:

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max}. \tag{1.4}$$

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

$$r = v_h \left(\frac{\partial \sigma}{\partial t} \right)^{-1}, \tag{1.5}$$

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:

$$\left| \frac{\partial \sigma}{\partial x} \right| \leq \frac{1}{r_{\min}}. \tag{1.6}$$

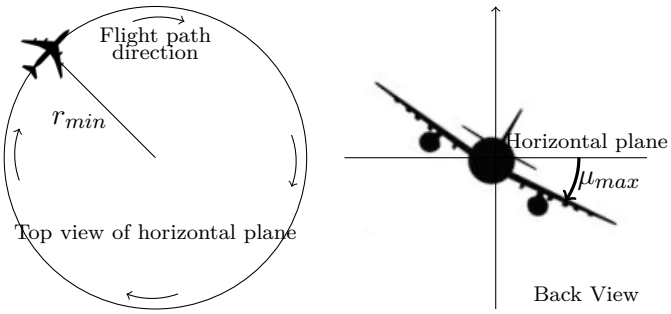


Figure 1.3: Link between the maximal bank angle and the minimal curvature radius.

These constraints apply in the general case of trajectory generation. However, depending on the type of path planning problem, they can be modified. For example, in case of the aircraft loses all of its power, the maximum flight path angle is constrained by the maximum lift-to-drag (L/D) ratio of the aircraft.

After presenting the features of aircraft trajectory modeling, the next section focuses on a state of the art of trajectory generation.

1.2 Trajectory Generation Methods

This section presents a set of methods related to path planning algorithms.

1.2.1 Methods Based on Dimension Reduction

This section presents some dimension reduction tricks in order to reduce the dimension of the state space for which an optimization process is searching for an optimal vector of parameters. This approach is summarized in Figure 1.4.

The optimization process controls the parameter vector which is then used to build the trajectory γ for evaluation. Each coordinate can be considered separately in order to build a given

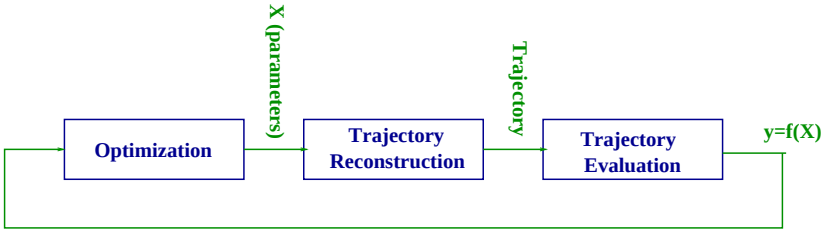


Figure 1.4: The optimization process control the X vector in order to build a trajectory γ for evaluation.

trajectory: $\vec{\gamma}(t) = [x(t), y(t), z(t)]^T$. In this section, several trajectory models are presented and compared. Simple models are first introduced.

1.2.1.1 Straight line segments

One of the easiest ways to design a trajectory is to use waypoints connected by straight lines (see Figure 1.5). This easy principle ensures continuity for the trajectory but not for its derivatives. If one wants to approximate a trajectory with many shape turns, one has to increase the number of waypoints in order to reduce the error between the model of the real trajectory.

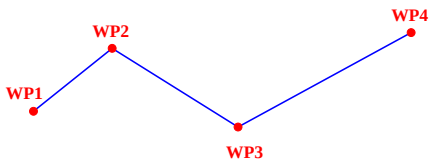


Figure 1.5: Trajectory defined by four way points connected by straight lines.

In order to improve the concept, Lagrange interpolation process adjusts a polynomial function to a given set of waypoints.

1.2.2 Lagrange interpolation

Given $n+1$ real numbers y_i,$0 \leq i \leq n$, and $n+1$ distinct real numbers $x_0 < x_1 < ... < x_n$, *Lagrange polynomial [112] of degree* n ($L_n(x)$) associated with $\{x_i\}$ and $\{y_i\}$ is a polynomial of degree n solving the interpolation problem :

$$L_n(x_i) = y_i, \;\; 0 \leq i \leq n \quad L_n(x) = \sum_{i=0}^{n} y_i.l_i(x) \quad \text{where } l_i(x) = \prod_{j \neq i} \frac{(x-x_j)}{(x_i-x_j)} \tag{1.7}$$

An example of Lagrange interpolation is given in Figure 1.6 for which four points are interpolated by the black curve which represents $L_4(x)$. The four polynomial functions $\{l_0(x), l_1(x), l_2(x), l_3(x)\}$ are also given by the red, blue, green, and yellow curves.

When derivatives have also to be interpolated, Hermite interpolation has to be used [13].

1.2.2.1 Hermite interpolation

Hermite interpolation [13] generalizes Lagrange interpolation by fitting a polynomial ($H(x)$) to a function f that not only interpolates f at each knot but also interpolates a given number of

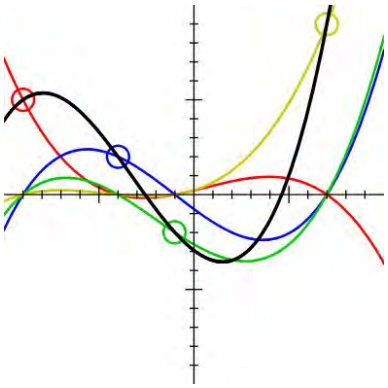


Figure 1.6: $L_n(x)$ is represented by the black curve. The others curves are the polynomials $l_i(x)$.

consecutive derivatives of f at each knot. This means that the first derivatives of the polynomial $H(x)$ has to fit the first derivatives of the function $f(x)$:

$$\left[\frac{\partial^j H(x)}{\partial x^j}\right]_{x=x_i} = \left[\frac{\partial^j f(x)}{\partial x^j}\right]_{x=x_i} \quad \forall j \in \{0,1,\ldots,m\}, \forall i \in \{1,2,\ldots,k\}. \tag{1.8}$$

This means that $n(m+1)$ values

$$\begin{array}{llll} (x_0,y_0), & (x_1,y_1), & \ldots, & (x_{n-1},y_{n-1}), \\ (x_0,y_0'), & (x_1,y_1'), & \ldots, & (x_{n-1},y_{n-1}'), \\ \vdots & \vdots & & \vdots \\ (x_0,y_0^{(m)}), & (x_1,y_1^{(m)}), & \ldots, & (x_{n-1},y_{n-1}^{(m)}) \end{array} \tag{1.9}$$

must be known, rather than just the first n values required for Lagrange interpolation. The resulting polynomial may have a degree at most $n(m+1)-1$, whereas the Lagrange polynomial has a maximum degree $n-1$.

These interpolation polynomials seem attractive but they both induce oscillations between interpolation points (*Runge's phenomenon*). Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree (which is the case for Lagrange and Hermite interpolation). An example of such Runge's phenomenon is given in Figure 1.7 for which Lagrange interpolation has been used.

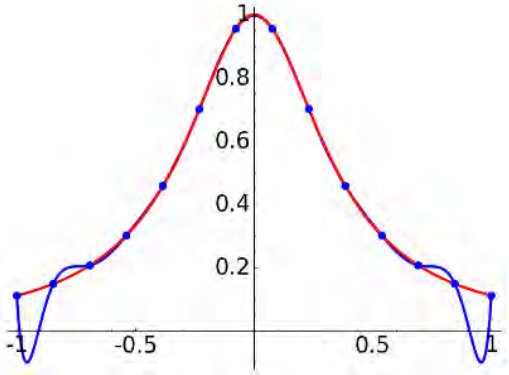


Figure 1.7: Lagrange interpolation result for a set on a given curve (red line).

We can conclude that interpolation with a high degree polynomial is risky. In order to avoid this drawback of high degree polynomial interpolation, one must use piecewise interpolation.

1.2.2.2 Piecewise Linear Interpolation

This is the simplest piecewise interpolation method.

Given $n+1$ real numbers y_i, $0 \leq i \leq n$, and $n+1$ distinct real numbers $x_0 < x_1 < ... < x_n$, we consider the n linear curves $lin_i(x) = a_i x + b_i$ on the intervals $[x_i, x_{i+1}]$ for $i = 0, ...n-1$ ($lin_i(x)$ represent linear functions for which a_i is the slope and b_i a constant).

Each $l_i(x)$ has to connect two points $((x_i, y_i),(x_{i+1}, y_{i+1})$

$$y_i = a_i x_i + b_i \text{ and } y_{i+1} = a_i x_{i+1} + b_i \tag{1.10}$$

In order to associate a piecewise formulation of this interpolation method, the following "tent" functions are defined:

$$\psi_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if} x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if} x \in [x_i, x_{i+1}] \\ 0 & \text{elsewhere} \end{cases} \tag{1.11}$$

Then,

$$f(x) = \sum_{i=0}^{i=n} y_i . \psi_i(x) \tag{1.12}$$

An example of such a linear piecewise interpolation is given on Figure 1.8.

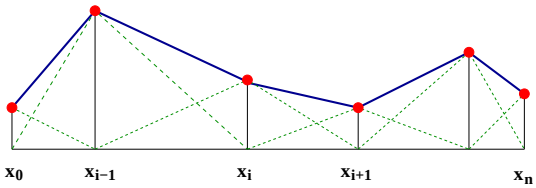


Figure 1.8: Piecewise linear interpolation.

The derivative of the resulting curve is not continuous. In order to fix this drawback, one can use piecewise quadratic interpolation.

1.2.2.3 Piecewise Quadratic Interpolation

We consider the n quadratic curves $\psi_i(x) = q_i(x) = a_i x^2 + b_i x + c_i$ on the intervals $[x_i, x_{i+1}]$ for $i = 0, ...n-1$. Each $q_i(x)$ has to connect two points $((x_i, y_i),(x_{i+1}, y_{i+1}) \Rightarrow y_i = a_i x_i^2 + b_i x_i + c_i$ and $y_{i+1} = a_i x_{i+1}^2 + b_i x_{i+1} + c_i)$. Furthermore, on each point, the derivative of the previous quadratic has to be equal to the derivative of the next one; $\Rightarrow 2a_i + b_i = 2a_{i-1} + b_{i-1}$. For the first segment, the term $2a_{i-1} + b_{i-1}$ is arbitrarily chosen (this will affect the rest of the curve). An example of piecewise quadratic interpolation is given in Figure 1.9. The main drawback of piecewise quadratic interpolation is linked to the effect induced on the curve by moving one point. As a matter of fact, moving one point may totally change the shape of the interpolating curve. The piecewise cubic interpolation avoids this drawback.

1.2.2.4 Piecewise cubic interpolation

This interpolation is also called Hermite cubic interpolation [91]. For this interpolation:

$$\psi_i(x) = C_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \tag{1.13}$$

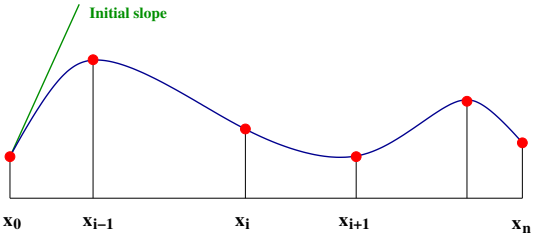


Figure 1.9: Piecewise quadratic interpolation. The shape of the entire curve depends on the choice of the initial slope. Between two points, a quadratic polynomial is fitted.

and we have the following constraints :

$$C_i(x_i) = y_i \qquad C_i(x_{i+1}) = y_{i+1} \tag{1.14}$$

$$C_i'(x_i) = y_i' = \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} \quad C_i'(x_{i+1}) = y_{i+1}' = \frac{y_{i+2} - y_i}{x_{i+2} - x_i}. \tag{1.15}$$

An example of piecewise cubic interpolation is given on Figure 1.10.

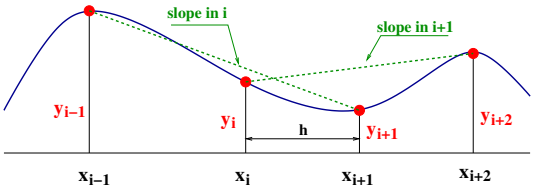


Figure 1.10: Piecewise cubic interpolation. The derivative at point x_i is given by line joining the point (x_{i-1}, y_{i-1}) and (x_{i+1}, y_{i+1}).Between two points, a cubic polynomial is fitted. The term h represents the distance between two consecutive points.

Moving a point does not affect all the curve which is the main advantage of this interpolation. The resulting curve is C^1 but not C^2 (the second derivative is not continuous). The curvature radius of a curve may be expressed by the following expression:

$$R = \frac{1 + \left(\frac{df(x)}{dx}\right)^{\frac{3}{2}}}{\left|\left(\frac{d^2 f(x)}{dx^2}\right)\right|}. \tag{1.16}$$

The piecewise cubic interpolation does not insure that trajectory curvature is continuous which is not adapted for aircraft trajectory mainly in TMAs and cubic spline interpolation has to be used.

1.2.2.5 Cubic Spline Interpolation

This method has been developed by General Motors in 1964 [26]. For this piecewise interpolation $\psi_i(x) = S_i(x)$ with the following constraints:

$$\begin{array}{ll} S_i(x_i) = y_i & S_i(x_{i+1}) = y_{i+1} \\ S_i'(x_i) = S_{i-1}'(x_{i+1}) & S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \\ S_i''(x_i) = S_{i-1}''(x_{i+1}) & S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \end{array}. \tag{1.17}$$

One can show that $S_i(x)$ for $x \in [x_i, x_{i+1}]$ is given by:

$$\begin{aligned} S_i(x) &= \frac{\sigma_i}{6}.\frac{(x_{i+1}-x)^3}{x_{i+1}-x_i} + \frac{\sigma_{i+1}}{6}.\frac{(x-x_i)^3}{x_{i+1}-x_i} \\ &+ y_i.\frac{x_{i+1}-x}{x_{i+1}-x_i} - \frac{\sigma_i}{6}.(x_{i+1}-x_i)(x_{i+1}-x) \\ &+ y_{i+1}.\frac{x-x_i}{x_{i+1}-x_i} - \frac{\sigma_{i+1}}{6}.(x_{i+1}-x_i)(x-x_i) \end{aligned} \quad , \tag{1.18}$$

where

$$\sigma_i = \frac{d^2 S_i(x)}{dx^2}. \tag{1.19}$$

An example of such interpolation is given in Figure 1.11.

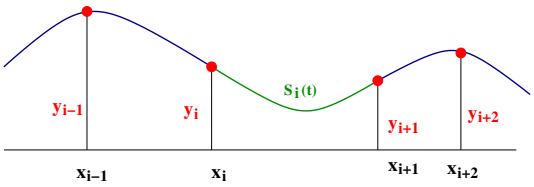


Figure 1.11: Cubic Spline Interpolation.

Such a spline is also called a natural spline because it represents the curve of a metal spline constrained to interpolate some given points.

When interpolation is not a hard constraint, one can use some control points that change the shape of a given trajectory without forcing this trajectory to go through such control point; such an approach is called approximation for which one of the famous methods is the Bézier curve.

1.2.2.6 Bézier Approximation Curve

Bézier curves [69] were widely publicized in 1962 by the French engineer Pierre Bézier, who used them to design automobile bodies. But the study of these curves was first developed in 1959 by the mathematician Paul de Casteljau using de Casteljau's algorithm [70], a numerically stable method to evaluate Bézier curves. A Bézier curve is defined by a set of control points $\vec{P}_0$ through $\vec{P}_n$, where n is called its order ($n = 1$ for linear, 2 for quadratic, etc.). The first and last control points are always the end points of the curve; however, the intermediate control points (if any) generally do not lie on the curve. Given points $\vec{P}_0$ and $\vec{P}_1$, a linear Bézier curve $\vec{B}(t)$ is simply a straight line between those two points. The curve is given by:

$$\vec{B}(t) = \vec{P}_0 + t(\vec{P}_1 - \vec{P}_0) = (1-t)\vec{P}_0 + t\vec{P}_1 \ , \ t \in [0,1]. \tag{1.20}$$

With four points ($\vec{P}_0$, $\vec{P}_1$, $\vec{P}_2$, $\vec{P}_3$), a Bézier curve of degree three can be built. The curve starts at $\vec{P}_0$ going towards $\vec{P}_1$ and arrives at $\vec{P}_3$ coming from the direction of $\vec{P}_2$. Usually, it will not pass through $\vec{P}_1$ or $\vec{P}_2$; these points are only there to provide directional information (see Figure 1.12).

Properties

- The polygon formed by connecting the Bézier points with lines, starting with $\vec{P}_0$ and finishing with $\vec{P}_n$, is called the Bézier polygon (or control polygon).
- The convex hull [1] of the Bézier polygon contains the Bézier curve.

[1] The convex hull or convex envelope of a set X of points in the Euclidean plane or Euclidean space is the smallest convex set that contains X.

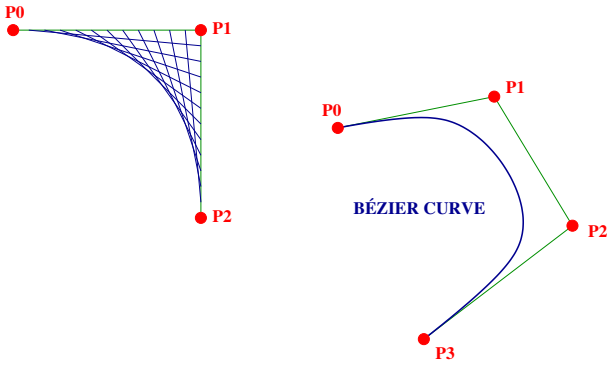


Figure 1.12: Cubic Bézier curve.

- The start (end) of the curve is tangent to the first (last) section of the Bézier polygon.

The explicit form of the curve is given by:

$$\vec{B}(t) = (1-t)^3\vec{P}_0 + 3(1-t)^2 t\vec{P}_1 + 3(1-t)t^2\vec{P}_2 + t^3\vec{P}_3 \ , \ t \in [0,1]. \tag{1.21}$$

$$\vec{B}(t) = \sum_{i=0}^{n} b_{i,n}(t)\vec{P}_i, \quad t \in [0,1], \tag{1.22}$$

where the polynomials

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, \dots n \tag{1.23}$$

are known as Bernstein basis polynomials of degree n. So, if there are many points, one has to manipulate polynomials with a high degree. In order to circumvent this weak point one must use Basis-Splines.

1.2.2.7 Basis Splines

A B-spline [44] is a spline function that has minimal support with respect to a given degree, smoothness, and domain partition. B-splines were investigated as early as the nineteenth century by Nikolai Lobachevsky. A fundamental theorem states that every spline function of a given degree, smoothness, and domain partition, can be uniquely represented as a linear combination of B-splines of that same degree and smoothness, and over that same partition. It is a powerful tool for generating curves with many control points, B stands for basis. A single B-spline can specify a long complicated curve and B-splines can be designed with sharp bends and even "corners". B-Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. Furthermore, spline interpolation avoids the problem of Runge's phenomenon which occurs when interpolating between equidistant points with high degree polynomials.

1.2.2.8 Uniform B-Splines of Degree Zero

We consider a node vector $\vec{T} = \{t_0, t_1, ..., t_n\}$ with $t_0 \leq t_1 \leq, ..., \leq t_n$ and n points $\vec{P}_i$. One wants to build a curve $\vec{X}_0(t)$ such that:

$$\vec{X}_0(t_i) = \vec{P}_i, \tag{1.24}$$

$\Rightarrow \vec{X}_0(t) = \vec{P}_i \ \forall t \in [t_i, t_{i+1}]$,

$$\vec{X}_0(t) = \sum_i B_{i,0}(t).\vec{P}_i, \tag{1.25}$$

where

$$B_{i,0}(t) = \begin{cases} 1 & \text{if } t \in [t_i, t_{i+1}] \\ 0 & \text{elsewhere} \end{cases} . \tag{1.26}$$

The shape of the $\vec{X}_0(t)$ function in one dimension is given on Figure 1.13.

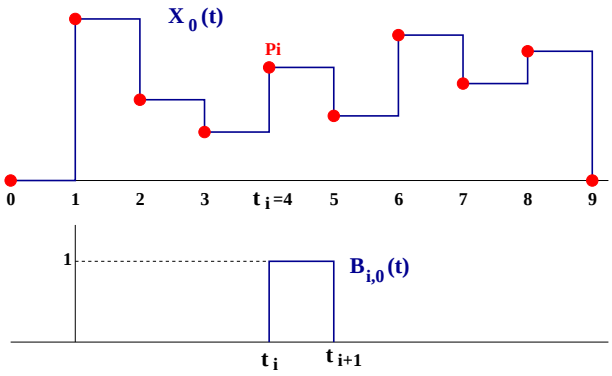


Figure 1.13: Uniform B-Splines of Degree Zero

1.2.2.9 Uniform B-Splines of Degree One

We are searching for a piecewise linear approximation $\vec{X}_1(t)$ for which:

$$\vec{X}_1(t) = \left(1 - \frac{t - t_i}{t_{i+1} - t_i}\right)\vec{P}_{i-1} + \left(1 - \frac{t - t_i}{t_{i+1} - t_i}\right)\vec{P}_i \quad \forall t \in [t_i, t_{i+1}]. \tag{1.27}$$

One can write $\vec{X}_1(t)$:

$$\vec{X}_1(t) = \sum_i B_{i,1}(t).\vec{P}_i, \tag{1.28}$$

where:

$$B_{i,1}(t) = \begin{cases} \frac{t - t_{i-1}}{t_i - t_{i-1}} & \text{if } t \in [t_{i-1}, t_i] \\ \frac{t_{i+1} - t}{t_{i+1} - t_i} & \text{if } t \in [t_i, t_{i+1}] \\ 0 & \text{elsewhere} \end{cases} . \tag{1.29}$$

The shape of the $\vec{X}_1(t)$ function in one dimension is given in Figure 1.14.

1.2.2.10 Uniform B-Splines of Degree Three

Uniform B-Splines of degree three have been developed at Boeing in the 70s and represent one of the simplest and most useful cases of B-splines. Degree three B-Spline with $n + 1$ control points is given by:

$$\vec{X}_3(t) = \sum_{i=0}^{n} B_{i,3}(t).\vec{P}_i \quad 3 \leq t \leq n + 1, \tag{1.30}$$

where $B_{i,3}(t) = 0$ if $t \leq t_i$ or $t \geq t_{i+4}$.

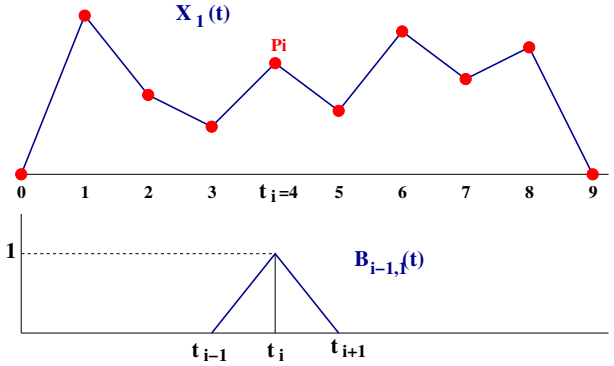


Figure 1.14: Uniform B-Splines of Degree One

$$\vec{X}_3(t) = \sum_{i=j-3}^{j} \vec{P}_i . B_{i,3}(t) \;\; t \in [j, j+1], \;\; 3 \leq j \leq n. \tag{1.31}$$

When a single control point $\vec{P}_i$ is moved, only the portion of the curve $\vec{X}_3(t)$ is changed (with $t_i < t < t_{i+4}$) ensuring local control property.

The basis functions have the following properties:

- They are translates of each other i.e $B_{i,3}(t) = B_{0,3}(t-i)$;
- They are piecewise degree three polynomial;
- Partition of unity $\sum_i B_{i,3}(t) = 1$ for $3 \leq t \leq n+1$;
- The functions $\vec{X}_i(t)$ are of degree 3 for any set of control points.

$$B_{i-2,3}(t) = \frac{1}{h} \begin{cases} (t - t_{i-2})^3 \;\; \text{if } t \in [t_{i-2}, t_{i-1}] \\ h^3 + 3h^2(t - t_{i-1}) + 3h(t - t_{i-1})^2 - 3(t - t_{i-1})^3 \\ \;\; \text{if } t \in [t_{i-1}, t_i] \\ h^3 + 3h^2(t_{i+1} - t) + 3h(t_{i+1} - t)^2 - 3(t_{i+1} - t)^3 \\ \;\; \text{if } t \in [t_i, t_{i+1}] \\ (t_{i+2} - t)^3 \text{ if } t \in [t_{i+1}, t_{i+2}] \\ 0 \text{ otherwise} \end{cases}, \tag{1.32}$$

where h is the distance between two consecutive points.

Those basis functions are shown on Figure 1.15.

1.2.2.11 Principal Component Analysis

When trajectories samples are available (from radar for instance), one can build a dedicated base which will minimize the number of coefficients for trajectory reconstruction. Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables.

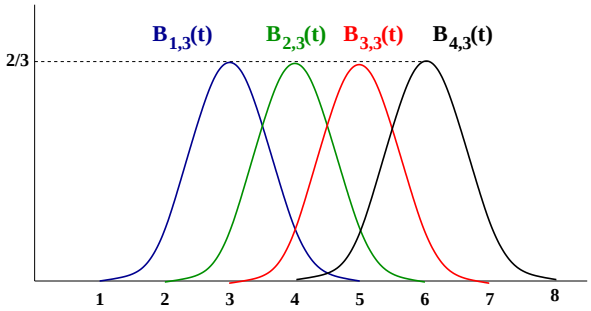


Figure 1.15: Order 3 basis functions

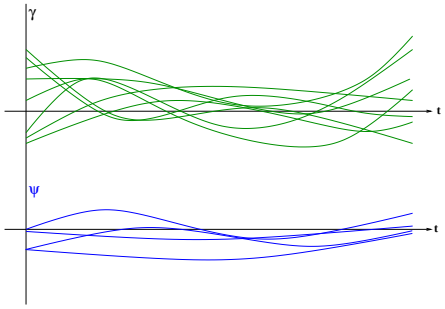


Figure 1.16: The green trajectories represent registered samples for which 4 principal components are extracted (in this artificial example) for minimum error reconstruction process.

In the example presented in Figure 1.16 a set of trajectories $\gamma_i(t), i = 1 \dots n$ are used to build $K = 4$ principal components $(\psi_k(t))$ which can be used to reconstruct the initial trajectories.

$$\gamma_i(t) = \sum_{k=1}^{k=K} a_{ik}\psi_k(t). \tag{1.33}$$

When probability density functions of the coefficient a_{ik} can be identified, one can use this trick to plug a stochastic optimization process that generates random coefficients in order to produce a relevant random trajectory. More information about Functional PCA can be found in [169].

1.2.2.12 Homotopy trajectory design

An easy way to build a trajectory is to use reference trajectories (regular trajectories used by aircraft) and to compute a weighted sum of such reference trajectories to build a new one. If we consider two (or more) reference trajectories joining the same origin-destination pair based for instance on past flown trajectories (see Figure 1.17):

$$\gamma_1, \gamma_2 \tag{1.34}$$

a new trajectory γ_α can be created by using an homotopy:

$$\gamma_\alpha = (1 - \alpha)\gamma_1 + \alpha\gamma_2. \tag{1.35}$$

In this example, only one coefficient α has been used but one can extend this principle to several parameters.

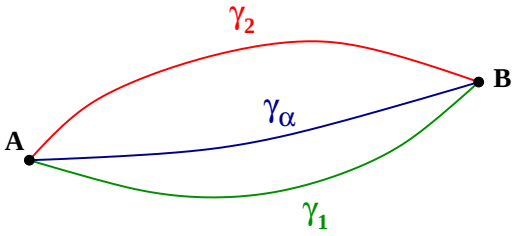


Figure 1.17: One new trajectory is built by using a weighted sum of two reference trajectories.

All the models described in this section may be used in an optimization process for which dimension reduction is needed. Depending on the targeted properties of the designed trajectories one must select the most adapted representations for the underlying application.

After having reviewed algorithms based on optimization, the next section presents methods using graphs.

1.2.3 Discrete methods: methods using graphs

One of the approaches frequently used for trajectory generation is to go through a discretization of the search space. This discretization leads to an easier resolution for two main reasons:

- The search space is restricted and finite (or at least countable);
- Path finding algorithms on graph are well-known algorithms.

A first category of methods consists in discretizing the space in a fixed and generally regular way. These methods always give the same solution on the same instance, they are then called deterministic. On the other hand, some methods involve randomness in the construction of the graph on which the solution will be searched. They are then called non-deterministic methods or sampling methods. They will be presented in the sequel.

Computation of the grid/graph Deterministic methods compute the shortest path or the path with the lowest cost on a graph. Sometimes such a graph is represented by a regular grid, i.e. a grid with a fixed step. Deterministic methods find the shortest path on the graph, so the grid must be dense enough for the solution found to be close to the optimum in the underlying space. Many algorithms have been developed for computing shortest paths in such a discrete framework. The most popular methods are now introduced.

1.2.3.1 Dijkstra's algorithm

Dijkstra's method [48] finds the lowest-cost path from a source node to all the other nodes (or just one if it is stopped) on a graph where the costs of edges are positive.

It follows the following process:

1. The cost of all the nodes is fixed to $+\infty$, the cost of the source is fixed to 0;
2. Find the node s with the lowest cost;
3. Update the costs of the neighbors of s;

4. Return to step 2 until the goal is reached (or until all nodes have been reached).

By using an efficient sort algorithm, the complexity of the algorithm is in $O(NlogN)$ where N is the number of nodes of the graph.

1.2.3.2 Bellman's algorithm

Unlike the Dijkstra algorithm, the Bellman algorithm, developed in [16], allows the presence of some arcs with negative weight and allows to detect the presence of absorbing circuits, i.e. of strictly negative total weight, accessible from the source vertex.

The complexity of the algorithm is in $O(|V||A|)$ where $|V|$ is the number of vertices and $|A|$ is the number of links.

Let s be the origin vertex and t the target vertex. The algorithm uses the principle of dynamic programming. It computes $c(t,k)$, the cost from source vertex s to t with a path that contains at most k arcs, by increasing the value of k. The cost from s to t is $c(t,|V|-1)$. To compute the different labels, it uses the following rules:

- $c(t,0) = +\infty$ if $t \neq s$ and $d[s,0] = 0$;
- $c(t,k) = \min\left(c(t,k-1), \min_{(u,t)\in A}\left(c(u,k-1) + cost(u,t)\right)\right)$ where $cost(u,t)$ is the cost of the arc (u,t).

1.2.3.3 A* algorithm

The A* algorithm, introduced by Hart, Nilsson, and Raphael in 1968 [90], is based on a heuristic evaluation at each node to assess the best path through it. The graph is explored according to this evaluation and not only according to the computed cost as it is done in Dijkstra's algorithm. This technique allows to guide the search in the graph toward the final node and thus limits the number of nodes that have to be evaluated.

This heuristic evaluation estimates the distance from the current node to the destination and this is done according to equation (1.36) where $g(n)$ is the cost from the initial point to the current point n and $h(n)$ is the heuristic function which gives a lower bound of the cost from the current point n to the destination. The choice of the heuristic is very important and determines the efficiency of the algorithm. A heuristic function is said to be admissible if it never overestimates the cost to reach the goal (it underestimates it). The choice of an admissible heuristic guarantees to always find the shortest path.

$$f(n) = g(n) + h(n) \tag{1.36}$$

Thanks to the heuristic, the number of operations is strongly reduced.

1.2.3.4 D* algorithm

D* designates one of the three related incremental search algorithms *i.e.* three algorithms that combine both incremental and heuristic search to speed up searches of sequences of similar search problem:

- The original D* developed by Stentz [187] which is an informed incremental search algorithm;

- Focused D* algorithm [188] is a combination of the A* algorithm and the original D*.
- D* Lite, developed by Koenig and Likhachev [122], is an incremental heuristic search algorithm. It is based on LPA* [124], another incremental heuristic search algorithm that combines A* and Dynamic SWSF-FP [168].

All three search algorithms address assumption-based path planning problems, specifically involving planning with the freespace assumption [123]. A robot navigates to a set of predetermined goal coordinates in an unknown terrain. To achieve this, the robot makes certain assumptions about the unknown parts of the terrain, such as assuming they contain no obstacles. It then proceeds to find the shortest path from its current coordinates to the goal coordinates, based on these assumptions. The robot follows this path and continually updates its map as it observes new information, such as the presence of previously unknown obstacles. If necessary, the robot replans a new shortest path from its current location to the designated goal coordinates, incorporating the newly acquired data. This process is repeated until the robot successfully reaches the goal coordinates or determines that reaching them is not possible.

As the robot traverses through unknown terrain, it frequently encounters new obstacles, requiring rapid replanning. To expedite the search process for sequences of similar search problems, incremental (heuristic) search algorithms come into play. These algorithms leverage the robot's experience from previous problems to accelerate the search for the current one. When the goal coordinates remain constant, all three search algorithms demonstrate greater efficiency compared to conducting repeated A* searches.

Some other methods build the graph by taking into account the underlying obstacles like the Visibility Graph approach.

1.2.3.5 Visibility Graph

The Visibility Graph method is one of the first path planning methods which have been developed [90]. It allows you to find the shortest path connecting a starting point and an ending point in a 2D space filled with polygonal obstacles. The mobile is represented by a point. An illustration of a visibility graph is given in Figure 1.18.

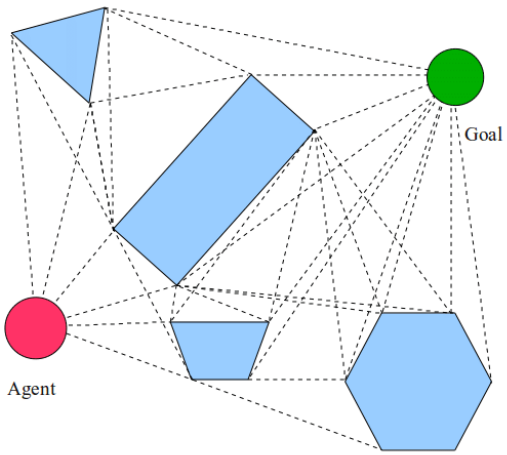


Figure 1.18: Example of visibility graph.

The method can be broken down into two successive steps: the first step constructs a graph connecting all the vertices of the obstacles with straight lines, as well as the starting and ending

point. Only straight lines not crossing obstacles are kept when creating the graph. The second step consists in finding in this graph the shortest path leading from the starting point to the ending point (thanks to a Dijkstra algorithm for example).

This method was quickly adapted to take into account the space occupied by the robot in the avoidance of fixed obstacles [139]. Another adaptation makes it possible to take into account mobile obstacles by controlling the speed at which the path is followed [127].

- Advantages: Ensure an obstacle free path connecting the starting point to the destination.
- Drawback: the calculated path most often touches the border of the obstacles, but without crossing it. The visibility graph method does not allow the avoidance of moving obstacles only by changing speed.

This method can not be extended to 3D spaces, because the shortest path does not necessarily pass through the vertices of the parallelepiped representing the obstacles. However, the shortest path is likely to pass by one of the edges.

1.2.4 Decomposition into Cells

Cell decomposition consists of breaking down the vehicle's free configuration space into a collection of non-overlapping regions (cells), the union of which exactly represents the free configuration space. A connectivity graph is then constructed to represent the adjacency relationships between cells (see Figure 1.19). A graph traversal method can then be used to find a succession of cells connecting the initial cell (containing the starting point) to the final cell (containing the ending point).

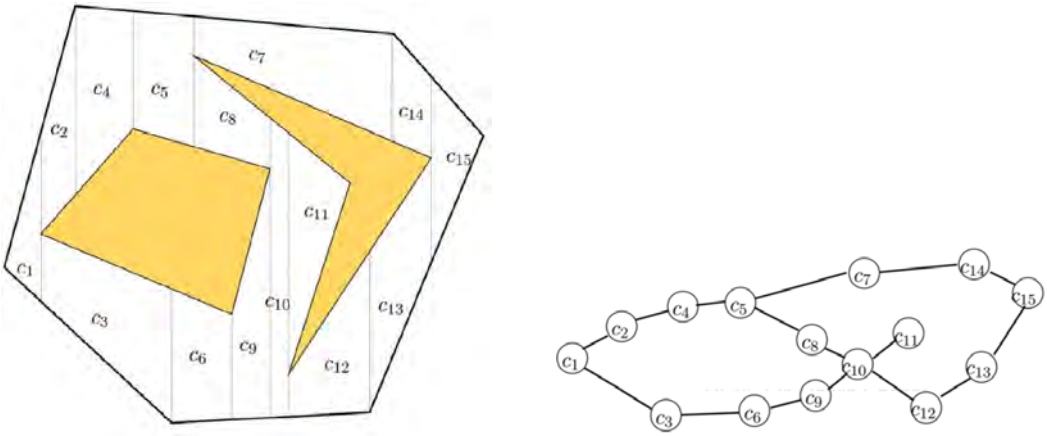


Figure 1.19: Cell decompisition of a space with two obstacle.

A path can then be simply calculated inside these cells (succession of straight lines, for example) without fear of encountering an obstacle (in a static environment).

Decomposing the space into cells and then constructing the cell connectivity graph are operations that can be costly in terms of both time and resources. The most frequently used decomposition is of exponential complexity in n, with n the number of dimensions of the configuration space [135]. But decomposition into cells has the advantage of allowing the planning of several paths (with different starting and destination points) without having to discretize the free space again.

Originally implemented for a manipulator robot moving in static space [33], the cell decomposition method has recently been adapted to the avoidance of moving obstacles in the video game Pacman [73]. The method applies well to the game of Pacman since the free space to be cut up is limited (labyrinth), the moving obstacles are few (enemies) and their behaviors can be anticipated.

1.2.5 Voronoï Diagram

Many path planning methods use Voronoi diagrams [66]. In a space filled with obstacles represented by dots, a Voronoi diagram is obtained by connecting the nearby obstacles with line segments, then drawing the perpendiculars passing through the middle of these segments. These perpendiculars are drawn until they meet other perpendiculars, thus forming a graph demarcating cells at the center of which are the obstacles. An example is given by Figure 1.20.

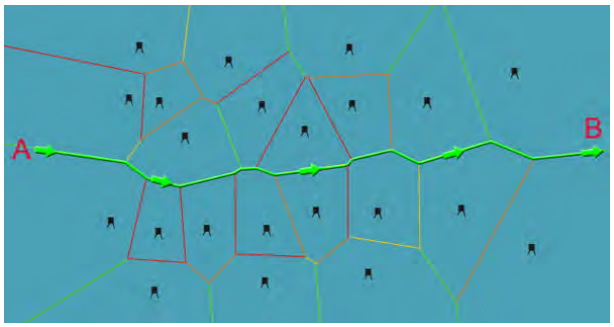


Figure 1.20: Voronoï diagram. Each cell contain an obstacle that has to be avoided.

A simple technique of trajectory planning consists in adding the starting and ending point to the graph thus applying a search algorithm for the shortest path on this graph. To improve the shape of the path obtained, McLain et al. use the analogy with a chain [144]. A force is applied to the path found to pull it towards the straight line in order to reduce the turns. A repulsive force is exerted by each obstacle on the chain, to make sure that the path does not come too close to the obstacles.

1.2.6 Sampling based path planning methods

Many methods propose to generate a graph in order to find the optimal path between two points. The most fashionable methods are sampling-based path planning algorithms [75, 76, 115, 116]. Research on this type of methods was first driven by robotics. The goal was initially to find a path, as short as possible, for a robot, avoiding obstacles. If all the information about the environment is taken into account, the time complexity of the algorithms to solve this problem is also very large.

Before discussing the algorithms, it is important to formulate the problem and define the primitive functions that they use.

Let $\chi = (0,1)^d$ be the configuration space, where $d \in \mathbb{N}$ is the space dimension , $(d \geq 2)$. Let χ_{obs} be the obstacle region, such that $\chi \backslash \chi_{obs}$ is an open set, and denotes the obstacle-free space as $\chi_{free} = cl(\chi \backslash \chi_{obs})$, where $cl(\chi)$ denotes the closure of a set χ. The initial condition is denoted by $x_{init} \in \chi_{free}$, and the goal region χ_{goal} is an open of χ_{free}.

Sampling-based path planning algorithms mainly use four functions.

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

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

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

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

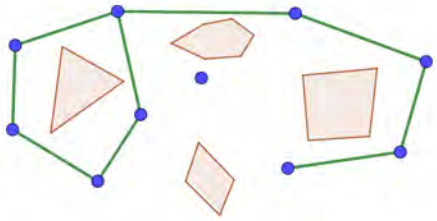
This state of the art presents the Probabilistic RoadMaps (PRM). The Fast Marching Tree and the Rapidly-exploring Random Tree algorithms will be presented in Chapter 2.

1.2.6.1 Probabilistic RoadMaps (PRM)

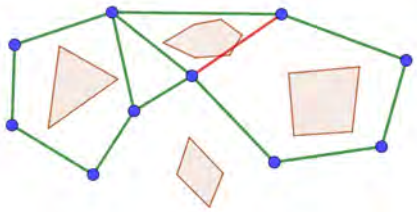
These methods are based on the following two steps:

1. Building of a graph from the sampling of the environment;
2. Finding the shortest path from one point to another.

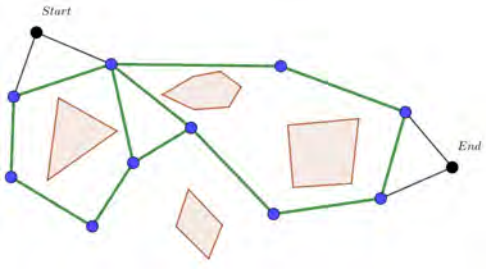
Figure 1.21 shows the process used by PRMs methods.



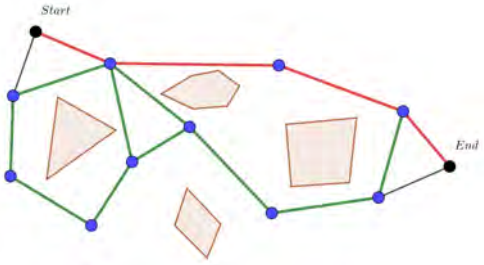
(a) Graph built at a given iteration. The graph is in green, the sample points in blue and the obstacles in red.



(b) Adding a node to the graph: edges are added with its neighbors when they do not meet an obstacle (green) and are not added if they do (red).



(c) Adding the start and end points of the trajectory to the graph.



(d) Search for the shortest path (in red) between the starting point and the end point in the graph by using classical methods.

Figure 1.21: Probabilistic Roadmap principle.

PRM is based on random sampling which determines the quality of the solution. To avoid relying too much on random sampling, other sampling methods have been developed, especially Rapidly-

exploring Random Tree (RRT) [128]. The goal of this type of algorithm is to reduce the number of sample points in the environment and to create a graph in an efficient way, by exploiting useful information from the environment in order to find a path close to the optimal one, while saving a lot of computation time. This method will be presented in detail in Section 2.5. The last developed sampling-based path planning method is the Fast Marching Tree (FMT*). It has been introduced by Janson *et al.* in [108]. This method will also be presented later in Section 2.3. These methods are known for their low computing time complexity. Table 1.1 summarizes the complexity of each algorithm.

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

Table 1.1: Algorithms Complexity depending on the number of samples n [117].

1.2.7 Propagation methods

From the analogy of light propagation in an open environment and the Eikonal equation governing this propagation or other Hamilton-Jacobi-type equations, very efficient methods have been developed, called propagation methods. The principle of propagation methods is, therefore, *to propagate a "wave"* (or equivalent front) on a grid and then *back propagate the gradients obtained on this grid to draw a continuous trajectory* as shown by Figure 1.22.

(a) Propagation of the front. (b) Gradient back propagation.

Figure 1.22: Principle of propagation methods.

The fast marching [179] (presented in detail in Section 2.4.1) and ordered upwind [181] methods were developed to solve stationary Hamilton-Jacobi equations, *i.e.* equations of the type :

$$||\nabla T(X)|| = f(X, \frac{\nabla T(X)}{||\nabla T(X)||}) \tag{1.37}$$

where X is the state vector, T is the optimal cost representing the minimum arrival time, and f is the "slowness" of the domain at any point, for example the inverse of the propagation speed of the front.

These methods can be used in this context since the optimal control problem can be written in the form of a particular Hamilton-Jacobi equation since it becomes stationary.

In the case of an isotropic medium, *i.e.* when the characteristics depend only on the position and not on the directions of space, this equation can be rewritten in another form to obtain an Eikonal-type equation :

$$||\nabla T(X)|| = f(X) \tag{1.38}$$

The fast marching approach is particularly adapted for this type of equation. However, the studied problem does not fit in such an isotropic context since the wind speed does depend on the direction. It would have been possible to implement fast marching methods in an anisotropic environment, citing the work of Mirebeau *et al.* [148]. Another method, simpler and already proven to be efficient, is the *ordered upwind* method which has been used in [83].

The purpose of the ordered upwind algorithm is to compute the cost function u on each point of a mesh and to keep in memory, for all these points, the gradients that allowed the algorithm to compute such a cost. These gradients are used in a second step to compute the optimal path from the initial point X_0 (from which the propagation starts) to any other point in the mesh.

1.2.8 Potentiel Field Methods

The Artificial potential fields method appears for the first time in the work of Khatib [119]. A potential field can be seen as a force field in which the mobile is immersed. The destination exerts an attractive force, while the obstacles generate repulsive forces. By following this force field, the mobile thus reaches its destination while avoiding obstacles. Built on the configuration space of the mobile, a potential function is the sum of an attraction function attracting the robot to its destination and repulsion functions pushing the robot away from obstacles [118]. The trajectory of the mobile is then obtained by following the associated potential field: opposite direction of the gradient of the potential function. This amounts to following the decreasing values of the potential function. One of the major drawbacks of potential fields is the presence of other local minima than the destination. The moving part can, by following the decreasing values of the potential function, reach one of these local minima and remain blocked there.

- Advantages: fast computation and is able to optimize several trajectories as long as the space does not evolve.
- Drawbacks: this method is an incomplete method: the mobile may remain blocked in a local minimum of the map built on the configuration space. This method requires a discretization of the space which can be an expensive operation.
- Application to the planning of the aircraft trajectory: this method can be used if the problem of local minima is solved.

1.2.9 Vector Field Histogram (VFH)

The VFH method [29], was developed to improve the management of mobile obstacles by potential fields, as well as the uncertainty about the position of these obstacles. The novelty of this method comes from the use of histogram grids to statistically represent the position of obstacles. Thus, each cell of the grid contains the probability of the presence of an obstacle. The grid is constantly updated according to the information available on the obstacles. This grid is then used to build a potential field that guides the mobile from the starting point to its destination. Considered a local planning method, VFH has the disadvantage that it does not guarantee that an optimal path is obtained to reach the destination. This drawback was subsequently resolved using an A* algorithm and the method was renamed VFH* [193].

- Advantages: allows the uncertainties about the position of obstacles to be taken into account.
- Drawbacks: no information on the completeness of the method.
- Application to aircraft trajectory planning: without information on the completeness of the method, arrival at the destination cannot be guaranteed.

1.2.10 Navigation Function

A navigation function is a particular potential field, built in order to guarantee the absence of local minima in the configuration space [173]. This guarantee depends on a free parameter that must be adjusted for each case. A navigation function is analytical. Its gradient can therefore be obtained without having to go through discretization of the configuration space, which greatly reduces the computation time. In addition, obstacles are directly taken into account in the navigation function, which allows it to work in both static and dynamic spaces.

- Advantages: solves the problem of local minima of potential fields; no discretization of the configuration space; well suited to take into account moving obstacles.
- Drawback: unlimited mobile speed;free parameters have to be adjusted manually to guarantee the completeness of the method.
- Application to aircraft trajectory planning: the method of navigation functions appears to be one of the most promising for aircraft path planning. It adapts very well to the coordinated movements. The two points that could be constraining concern the speed of the mobile and the completeness linked to the resolution which depends on some free parameters.

1.2.11 Potential Fields and Harmonic Functions

In physics, a potential field is an irrotational continuous field. It derives from a potential function U and is obtained from the gradient of $U = -\nabla U$. The principle is the same in the context of trajectory planning. A potential function U is built on the free space of the mobile so as to be maximal on the boundaries of the free space and minimal at the destination (see Figure 1.23). A trajectory for the mobile is then obtained by calculating $-\nabla U$, i.e. by following the decreasing values of the potential function.

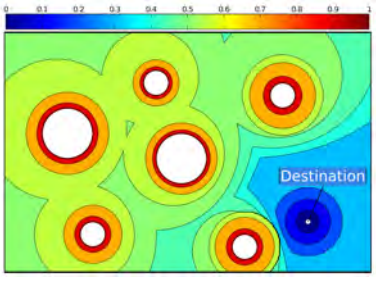


Figure 1.23: Example of a potention function. White circles = obstacles; white point = destination.

In his Ph.D. thesis [159] (in french), Khatib proposes several different potential fields for the navigation of a manipulator robot in a space containing an obstacle. We present here one of these fields, detailed in [118].

For a mobile wishing to navigate in a 2D space containing a fixed obstacle, the potential field is constructed from a potential function:

$$U_K(x,y) = U_{dest}(x,y) + U_O(x,y), \tag{1.39}$$

where $U_{dest}(x,y)$ represents the contribution of the destination and $U_O(x,y)$ the contribution of the obstacle to the construction of the potential field. The potential field is then obtained by calculating:

$$-\nabla U_K(x,y) = -\nabla U_{dest}(x,y) - \nabla U_O(x,y), \tag{1.40}$$

where $-\nabla U_{dest}(x,y)$ is the attractive field and $-\nabla U_O(x,y)$ the repulsive field.

By noting $q = (x,y)$ the position of the mobile, $q_d = (x_d, y_d)$ the position of the destination and k a positive gain, the function U_{dest} proposed by Khatib is written:

$$U_{dest}(q) = \frac{k}{2}\|q - q_d\|^2. \tag{1.41}$$

The repulsion part generated by the obstacle can, for its part, be expressed as a function of the distance between the mobile and the point closest to the obstacle $\rho(q)$, the area of influence of the obstacle ρ_0 and a positive gain η:

$$U_O(q) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{\rho(d)} - \frac{1}{\rho_0}\right)^2 & \text{if} \rho \leq \rho_0 \\ 0 & \text{otherwise} \end{cases}. \tag{1.42}$$

The function $\rho(q)$ can be complicated to calculate depending on the geometry of the obstacle present in the workspace. But in the case where the obstacle to be avoided is a disc with center q_O and radius r, it is written as:

$$\rho(q) = \|q - q_O\| - r. \tag{1.43}$$

Consider now a space filled with m obstacles, all considered as disks with center $q_{O_j} = (x_{O_j}, y_{O_j})$ and radius r_j, the potential function is written as:

$$U_K(q) = U_{dest}(q) + \sum_{j=1}^{j=m} U_{O_j}(q), \tag{1.44}$$

with

$$U_{O_j}(q) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{\|q-q_{O_j}\|-r_j} - \frac{1}{\rho_0}\right)^2 & if \rho \leq \rho_0 \\ 0 & if \rho > \rho_0 \end{cases}. \tag{1.45}$$

By applying this method to a space containing six circular obstacles and choosing $k = 1$, $\eta = 1$, and $\rho_0 = 1$, we obtain the potential function presented in Figure 1.24. This potential function decreases well as one approaches the destination and takes on high values around obstacles.

The corresponding potential field is obtained by differentiating $U_K(q)$:

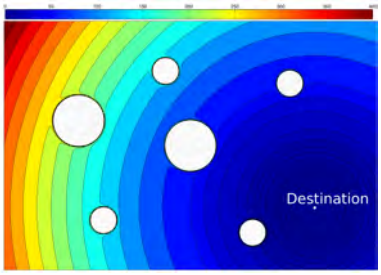


Figure 1.24: Potential function from Khatib [159] with $k = 1$, $\eta = 1$ and $\rho_0 = 1$. White circles = obstacles; white point = destination.

$$-\nabla U_K(q) = -\nabla U_{dest}(q) - \sum_{j=1}^{j=m} \nabla U_{O_j}(q), \tag{1.46}$$

with

$$-\nabla U_{dest}(q) = \begin{bmatrix} -(x - x_d) \\ -(y - y_d) \end{bmatrix}, \tag{1.47}$$

and

$$-\nabla U_{O_j}(q) = \left(\frac{1}{\rho_j(q)} - \frac{1}{\rho_0} \right) \frac{1}{\|q - q_{O_j}\| \rho_j(q)^2} \begin{bmatrix} -(x - x_{O_j}) \\ -(y - y_{O_j}) \end{bmatrix}. \tag{1.48}$$

The control law associated with the potential field is given by:

$$\dot{q} = -\nabla U_K(q). \tag{1.49}$$

The potential field norm controls the speed of the mobile. In order to obtain trajectories at a constant speed, we use the normalized vector field:

$$\dot{q} = -\frac{\nabla U_K(q)}{\|\nabla U_K(q)\|}. \tag{1.50}$$

In the case of the normalized potential field Khatib shown in Figure 1.25, the repulsive fields appear clearly around the obstacles, while the rest of the free space is covered with a field pointing to the destination.

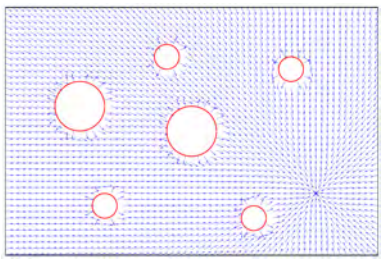


Figure 1.25: Normalized Khatib potential field. Red circles = obstacles.

The trajectories are then calculated as follows:

- **Initialization** : $q_{t=0} = q_s$
- **While**: $\|q_t - q_d\| > \epsilon$
  - $q_{t+1} = q_t - dt * \nabla U_K(q_t)$
  - $t = t + 1$

with q_s the starting point of the trajectory, dt the time discretization step and ϵ the desired precision at destination.

1.2.12 Path-Speed Decomposition

The Path-Speed Decomposition method [114] proposes a heuristic approach to the problem of trajectory planning for a mobile robot evolving in a dynamic environment. It consists in breaking down the trajectory planning into two subproblems:

- planning a path avoiding static obstacles;
- planning the speed of the vehicle to avoid dynamic obstacles.

Kant and Zucker show in [114] that this decomposition is valid, provided that the displacements of the obstacles are independent of the displacements of the mobile. If the obstacles follow the mobile, come in front of it or even stop in its path, the path-speed decomposition does not make it possible to find solutions to the problem of trajectory planning.

The advantage of this method is to reduce a complex problem (trajectory planning) to two simpler subproblems (planning the path, then the speed along this path). By the time the authors presented this method (1986), many path planning methods had already been studied (visibility graph, or decomposition into cells presented later), while path planning methods were still rare. The path-speed decomposition then made it possible to carry out trajectory planning based on the path planning methods already developed.

In order to solve the problem of mobile obstacles that can stop on the planned path, Fraichard and Laugier improved the principle of the path-speed decomposition by planning not one, but several parallel paths [74]. Depending on the movement of the moving obstacles, the vehicle moves from one path to another (such as a change of lane on the freeway) in order to keep moving while avoiding the moving obstacles. The path-speed decomposition method has been reused more recently in order to make a team of mobile robots crosses a space filled with fixed obstacles by imposing visual contact between the robots [134]. The authors define eye contact as the absence of an obstacle on the straight line connecting two robots. Obviously, not all robots can stay in constant eye contact, but the goal of Lindhe *et al.* is to ensure that all robots stay, directly or indirectly (using other robots as intermediaries), in contact with all other robots. The solution proposed in [134] consists in planning a path for each robot (more or less parallel) and avoiding the fixed obstacles present in space. Eye contact is then maintained at all times by controlling the speed of the robots.

This method is simple and efficient but does not guarantee to find a conflict free trajectory.

1.2.13 Optimal Control for Trajectory Generation

1.2.13.1 Optimal Trajectory Generation

In the physical space, a trajectory is occasionally represented as a four-dimensional flight path, following the tradition of air traffic control [37], with time as the fourth dimension, in addition to the normally used three-dimensional representation of a path. Generating *time-parameterized* paths necessitate the incorporation of the aircraft dynamics and/or kinematics, which makes the problem much more difficult than simply finding a path that avoids obstacles in the physical three-dimensional space. Path-planning is a term commonly used in the robotics and artificial intelligence communities to refer to the problem of generating an obstacle-free path to be followed by a vehicle (robot, aircraft, vehicle, etc) in a two or three dimensional space containing obstacles [129].

Because the vehicle dynamics are not taken into account in these path-planning methods (the solution of which only considers the geometric constraints of the problem) it is often the case that the resulting path is infeasible, that is, it cannot be followed exactly or even close by the vehicle. One way to ensure that the resulting paths correspond to feasible trajectories satisfying the vehicle dynamics, is to use optimal control theory. The objective of optimal control theory is to determine the control input(s) that will cause a process (i.e., the response of a dynamical system) to satisfy the physical constraints, while, at the same time, minimizing (or maximizing) some performance criteria. The feasibility of the trajectories is automatically ensured using this approach. The typical optimal control problem (OCP) can be stated as follows:

Given initial conditions x_0, final conditions $x_f \in \mathcal{X}$, and an initial time $t_0 \geq 0$, determine the final time $t_f > t_0$, the control input $u(t) \in \mathcal{U}$ and the corresponding state history $x(t)$ for $t \in [t_0, t_f]$ which minimize the cost function

$$J(x,u) = \int_{t_0}^{t_f} L(x(t), u(t))\,\mathrm{d}t, \tag{1.51}$$

where $x(t)$ and $u(t)$ satisfy, for all $t \in [t_0, t_f]$ the differential and algebraic constraints:

$$\dot{x}(t) - f(x(t), u(t)) = 0, \tag{1.52}$$

$$C(x(t), u(t)) \leq 0. \tag{1.53}$$

Optimal control has its roots in the theory of calculus of variations, which originated in the 17th century by Fermat, Newton, Leibniz, and the Bernoullis, and was subsequently further developed by Lagrange, Weirstrass, Legendre, Clebsch and Jacobi and others in the 18th and 19th centuries [64]. Calculus of variations deals with the problem of minimizing (1.51) subject to the simple differential equality constraint of the form $\dot{x}(t) - u(t) = 0$, and is not able to handle more complicated differential equality constraints such as (1.52) or algebraic constraints such as (1.53). It was not until the middle of the 20th century when the Soviet mathematician L. S. Pontryagin developed a complete theory that could handle constraints such as (1.52) and (1.53). Simply put, Pontryagin's celebrated Maximum Principle [166] states that the optimal control for the solution of the problem (1.51)-(1.53) is given as the pointwise minimum of the so-called Hamiltonian function, that is,

$$u_{\text{opt}} = \text{argmin}_{u \in U} H(t, x, \lambda, u), \tag{1.54}$$

where $H(t, x, \lambda, u) = L(x, u) + \lambda^T f(x, u)$ is the Hamiltonian, and λ are the co-states, computed from

$$\dot{\lambda}(t) = -\frac{\partial H}{\partial x}(x(t), \lambda(t), u(t)), \tag{1.55}$$

subject to certain boundary (transversality) conditions on $\lambda(t_f)$. Unfortunately, an analytic solution to the previous problem is difficult. The optimal control formulation of a trajectory optimization problem using Pontryagin's Maximum Principle (PMP) leads to a Two-point Boundary Value Problem (TBVP), or a Multi-point Boundary Value Problem (MBVP) when the optimal trajectory is composed of multiple phases.

1.2.14 Resolution Algorithms

There are three different methods of solving, more or less adapted according to the case: the direct methods, the indirect methods, and the Hamilton-Jacobi-Bellman equation.

1.2.14.1 Direct method

The direct method consists in discretizing the control or the state of the system, in order to transform the problem of optimal control into a nonlinear programming problem with constraints in finite dimension. The resulting problem can then be solved using classical optimization methods. The main advantage of direct methods is their relative insensitivity to the initial condition imposed on the system. If the direct methods offer good solutions, they do not however guarantee finding the best solution to the problem posed. The direct method has been used in aircraft trajectory planning problems [22], but also in UAVs [30].

- Advantages: does not require a priori knowledge of the solution; easily takes into account the constraints on the state of the system; there is a large set of methods for solving nonlinear problems with finite dimensional constraints.
- Drawbacks: significant cost in computation time and in memory: the direct method allows to deal only with problems of small dimension; the existence of local minima.

1.2.14.2 Indirect method

The indirect method is based on the principle of the Pontryagin maximum which gives the necessary conditions of optimality for the problem considered. The principle of Pontryagin's maximum makes it possible to reduce the problem to a differential system with two equations and two conditions: one initial and the other terminal. The optimal control problem is thus reduced to a problem under constraints.

Writing the Pontryagin maximum principle is unfortunately difficult for optimal control problems with constraints on the state of the system. And its resolution can also pose a problem, forcing most of the time to turn to methods of firing, single or multiple. It nevertheless has the advantage of not requiring a discretization of the problem and therefore of making it possible to deal with large problems. The indirect method assumes that a solution to the problem is already known and that an optimal solution is sought around this solution. Originally developed to optimize rocket trajectories or re-entry into the atmosphere of space shuttles, the indirect method has since been studied to optimize aircraft trajectories: [154, 186].

- Advantages: allows you to solve large-scale problems; very high numerical precision on the solution obtained.
- Drawbacks: requires a priori knowledge of the structure of the desired trajectory; sensitivity to the initial conditions; theoretical difficulties in taking into account constraints on the state

of the system due to the principle of the Pontryagin maximum; only allows local optimization of the problem.

1.2.14.3 Hamilton-Jacobi-Bellman equation

Instead of directly seeking to solve an optimal control problem, it can be rewritten as an equation: the Hamilton-Jacobi-Bellman equation. The solution of this equation then gives the optimal cost for reaching a given point in space from a starting point.

Rewriting the optimal control problem as a Hamilton-Jacobi-Bellman equation makes it possible to work with both discrete and continuous problems. In the discrete case, we speak of the Bellman equation, which can be solved by dynamic programming. In the continuous case, we speak of the Hamilton-Jacobi equation which is a partial differential equation.

Whether in discrete time or in continuous time, the passage through the Hamilton-Jacobi-Bellman equation makes it possible to seek a global optimum over all space. If this search over the entire space offers the advantage of finding the global optimum, it nevertheless makes the method very dependent on the size of the space studied.

- Advantages: global optimum
- Drawbacks: performances linked to the dimension of the studied space.

1.2.14.4 Numerical features

Numerical techniques such as shooting and multiple shooting methods can be applied to solve accurately TBVP and MBVP problems, but their convergence is very sensitive to the choice of an initial guess for the solution. Software that solves the optimal control problem using this approach is BNDSCO [160]. BNDSCO is an example of a class of numerical optimization methods that are often referred to as indirect methods. The term “indirect” reflects the fact that in these methods a solution is sought not by maximizing (or minimizing) the cost (1.51) but, rather, by computing potential optimizers by solving the corresponding necessary optimality conditions (1.54)-(1.55).

In recent years, direct methods have become increasingly popular for solving trajectory optimization problems, the major reason being that direct methods do not require an analytic expression for the necessary conditions, which for complicated nonlinear dynamics can be intimidating. Moreover, direct methods do not need an initial guess for the co-states whose time histories are difficult to predict a priori. As mentioned earlier, direct methods, do not try to satisfy the necessary conditions of optimality from PMP, instead, they minimize directly (1.51) subject to (1.52)-(1.53).

The main idea behind direct methods is to discretize the states and controls of the original continuous-time optimal control problem in order to obtain a finite-dimensional nonlinear programming problem (NLP). The solution of this NLP, which consists of discrete variables, is used to approximate the continuous control and state time histories. Typical direct methods are collocation methods, which discretize the ordinary differential equations (ODEs) of the problem using collocation or interpolation schemes [55, 89, 175, 196]. They introduce the collocation conditions as NLP constraints together with the initial and terminal conditions. The so-called “pseudospectral” methods use orthogonal polynomials to choose the collocation points and are very efficient, exhibiting superlinear convergence when the solution is smooth. Numerical optimal control software packages that implement direct methods for the solution of OCPs include

SOCS [19], RIOTS [177], DIDO [174], PSOPT [14], GPOPS [170], MTOA [106] and DENMRA [207] among many others. A recent survey of numerical optimal control techniques for trajectory optimization can be found in [18].

In all these direct methods, the convergence rate and the quality of solution depend on the grid used to discretize the equations, the cost, and the problem constraints. Uniform or fixed grid methods tend to perform poorly, especially when the problem has several discontinuities or irregularities. Not surprisingly, adaptive grid methods have been developed to accurately capture any discontinuities or switchings in the state or control variables. The main idea behind all these adaptive grid methods is to use a high resolution (dense) grid only in the vicinity of control switches, constraint boundaries etc, and a coarse grid elsewhere. Examples of such adaptive gridding techniques for the solution of optimal control problems are [20, 21, 25, 84, 107, 177].

A major issue with almost all current trajectory optimization solvers (direct or indirect) is the fact that their computational complexity is high and their convergence depends strongly on the initial conditions, unless certain rather stringent convexity conditions hold. As a result, the solution to the trajectory optimization problem in *real-time* is still elusive. A common line of attack for solving trajectory optimization problems in real time (or near real time) is to divide the problem into two phases: an offline phase and an online phase [109, 140, 182, 200]. The offline phase consists of solving the optimal control problem for various reference trajectories and storing these reference trajectories onboard for later online use. These reference trajectories are used to compute the actual trajectory online via a neighboring optimal feedback control strategy typically based on the linearized dynamics. Another strategy for computing near-optimal trajectories in real-time is to use a receding horizon (RH) approach [15, 161, 197]. In a receding horizon approach a trajectory that optimizes the cost function over a period of time, called the *planning horizon*, is designed first. The trajectory is implemented over the shorter *execution time* and the optimization is performed again starting from the state that is reached at the end of the execution time. A third approach is to use a two-layer architecture, where first an acceptable (in terms of length, safety, etc) path is computed using common path-planning techniques, and then an *optimal* time-parameterization is imposed on this path to yield a feasible trajectory. As mentioned earlier such an approach needs to be carefully designed to ensure compatibility of the resulting path with the vehicle dynamics. However, when successful, such an approach is numerically very efficient and can be implemented in real-time with current computer hardware. Even if the resulting trajectory is not exactly feasible, it is often close to a feasible trajectory, or it can be made as such using smoothing techniques [208]. As a result, alternatively, the final trajectory can be used as a good initial guess for a follow-up optimal trajectory generator. The next section summarizes this approach for applications related to aircraft maneuvering under strict time and fuel constraints. For more details, the interested reader is referred to [12, 205, 206].

1.3 Applications

To conclude this literature review, this section presents some prior works on different aircraft trajectory applications which have been addressed in this thesis.

1.3.1 Procedures design

The TMA design problem involves generating multiple routes. The main difficulty of this type of problem is ensuring that different routes meet separation constraints. To design multiple routes satisfying the separation criterion, some authors propose to generate the routes one after

the other according to a priority order defined by the user. Gianazza *et al.* [78, 79, 80] generate the routes according to traffic load and take into account the separation constraints by using A* algorithm. The order can also be determined by a metaheuristic as the simulated annealing method illustrated by Zhou *et al.* [210]. In this paper, the authors propose to design the departure and arrival routes using the Fast Marching method. Simulated Annealing was also used by Liang et al. [133] to design final descent trajectories. In [211], a Branch and Bound method is used to generate 3D routes in TMA which avoid obstacles. Grandberg *et al.*[86] proposed an algorithm to optimize the design of STAR in the vicinity of Stockholm Arlanda airport. Their method is based on Integer Programming (IP). The constraints taken into account are the following :

- a merging point only connects two routes;
- two merging points must be far enough from each other;
- no sharp turns;
- terrain avoidance;
- vertical separation between the SIDs and STARs, with their crossing, only authorized at a minimum distance from the runway.

This work is continued in [165], which takes into account the possibilities of *Required Navigation Performance* (RNP), and more particularly the Radius-to-Fix possibility. Chevalier [39] proposed in his Ph.D. thesis two different approaches to design SIDs and STARs. The first approach is a heuristic based on the dynamic programming principle. This method is composed of two different steps. The first one generates a graph to take into account the altitude constraints. The second step computes SID/STAR routes one by one. The use of a heuristic that assumes an order for the design of the routes, deteriorates the quality of the solution. The second approach is based on a metaheuristic: the Simulated Annealing. As for the previous method, the routes are computed one by one and optimized with a given criteria. However, the order of route design is not an issue due to the addition of a random component. Some of the constraints, such as obstacle avoidance, are relaxed and integrated into the objective function. These two approaches were tested in the cases of Stockholm, Paris Charles-de-Gaulle, and Zurich airports.

1.3.2 Tactical Conflict Detection Resolution Methods

Although most conflicts can be resolved by strategic optimization, new conflicts may arise due to uncertainties in aircraft trajectories. Currently, these conflicts are resolved "by hand" by air traffic controllers. The maneuvers they propose are rarely optimal and the aircraft can be significantly deviated from its initial flight plan. It is therefore important to assist air traffic controllers in this type of situation so that they can order pilots to perform shorter but still safe maneuvers. In the literature, this problem has been treated in an exact or approximate way. Bicchi and Pallottino [24] propose an analytical method based on kinematic models of aircraft flying in a horizontal plane with constant velocity to find suboptimal solutions. Mixed-integer linear and mixed-integer non-linear optimization models have also been proposed to solve this problem in [7, 36, 162]. All these methods are based on continuous maneuvers. Other works propose discrete approaches. The maneuvers are based on speed changes, vectoring (heading changes), Flight Level (FL) changes, or any possible combination of them. This is the case of Durand [52] and Durand and Alliot [53] who propose metaheuristics based on genetic and ant colony algorithms. Uncertainties are essential in this type of problem to ensure that the

solutions are actually safe. Probabilistic approaches have been also explored to solve the CDR problem at the tactical level [93, 137, 138, 143, 167, 194] to take into account wind uncertainties.

1.3.3 Weather Hazards avoidance

Different methods have been developed to solve the trajectory planning problem for multiple aircraft in the presence of stormy areas.

Erzberger *et al.*[57] [58] propose geometrical approaches to solve conflicts in TMA taking into account convective weather. Kamgarpour *et al.* [113] address the problem of hazardous weather avoidance in the en-route phase of the flight. The regions to avoid are considered to be time-varying and deterministic. In [189], a dynamic programming based approach to a stochastic reach-avoid problem for a controlled discrete-time stochastic hybrid system is proposed. In [92], the authors propose a thunderstorm stochastic model which is integrated in a tool based on the stochastic reach-avoid methodology presented in [189]. Due to the use of 4D space Dynamic Programming, the high computation times of the proposed method are not compatible with online implementation. In [192], a decision support tool composed of three parts is proposed. The first part is a rerouting algorithm which is based on the Dijkstra's shortest-path algorithm. The second part, based on a genetic algorithm, characterizes the operational acceptability of the generated trajectories. The last part, based on principal component analysis and spectral clustering, identifies representative routes inside each group. In [164], an algorithm based on Nonlinear Model Predictive Control (NMPC) is presented to solve aircraft motion planning problem in converging flows of aircraft in the presence of convective storms in the en-route airspace. The optimal solution is computed using dynamic programming under aircraft separation constraints, aircraft operational bounds, and stormy areas avoidance constraints. In [85], a numerical optimal control method is presented to generate en-route avoidance trajectories under convective weather evolution with uncertainties while using a realistic model of aircraft dynamics. In [178], the trajectory planning problem for multiple aircraft in converging arrival routes with multi-cell storm in development is addressed. A non linear model predictive control scheme has been used to represent the evolution of the stormy areas, allowing to re-compute the trajectories depending on the new state of the storms.

1.3.4 Emergency Trajectory Design

Although the emergency trajectory design problem is very complicated, many methods have already been developed to try to answer it. In 2006, Atkins et al. [9] propose an adaptative flight planning (AFP) algorithm to select a landing site and compute a safe emergency trajectory. The trajectory generator takes into account the flight dynamics and the wind. This algorithm was tested on the US Airways flight 1549 in [8]. Tang et al. [191] proposed a method to generate unpowered landing trajectories. The algorithm consists in solving the two-point boundary value problem (TPBVP). Fallast *et al.* in [67] proposes a method to automatically select an airport and then generate the associated emergency trajectory. The proposed algorithm is an adapted version of the RRT*. The main difference with the original RRT is that the search tree points are connected by Dubins curve (explained in detail in Section 2.2.4). It takes into account the aircraft specifications, especially the minimum curvature radius. Moreover, the proposed algorithm limits feasible points in the tree by introducing another constraint linked to the maximum climb rate and the maximum descent rate. Zhao [204] proposed a method based on the suboptimal solution of a three-dimensional variation of the classical Markov–Dubins problem. In his study, first, the minimal length curve problem in the horizontal plane is addressed and then the three dimensional landing path is obtained by generating a vertical profile. The proposed

algorithm was tested in two different scenarios (US Airways 1549 and Swissair 111). Haghighi et al. [88] presented a post-failure performance analysis and an optimization method to generate a fast and safe landing trajectory that avoids obstacles.

1.3.5 Alternative trajectories generation

The alternative path problem is a well known graph problem, often referred to as the k-shortest path problem. This problem has been addressed mainly in two ways: graph structure change [201] and ripple spreading [94]. The former approach consists in first computing a shortest path between an Origin-Destination (OD) pair thanks to a shortest path algorithm such as the Dijkstra Algorithm [16]. Then, each link of this path is removed from the graph one at a time. At each time, a shortest path is computed. Once all links have been disconnected, the computed shortest paths are sorted and the k-shortest are selected. The second approach generates k ripples from the origin node. When a ripple meets a node, it triggers a new ripple at this node. The process keeps going on until k ripples have reached the destination.

The main problem of these methods is the similarity between the generated k shortest paths. The number of overlapped links is often important. This observation leads to the study on the k dissimilar shortest path problem [6]. Several similarity metrics were developed to compare paths. In [40], the overlap ratio is used to measure such a similarity. Considering two paths, this ratio is the sum of the weights of the shared links over the length of one path. Liu et al. [136] define a lower bound length for two paths and a set of heuristics to avoid exploring some unnecessary paths to compute the k shortest paths with diversity. They use the similarity metrics presented in Table 1.2. $\ell(p_i)$ is the length of the path p_i and $p_i \cap p_j$ represents the overlap of paths p_i and p_j. Talarico, Sörensen, and Springael [190] define a similarity metric to compare two solutions to the Vehicle Routing Problem (VRP). Their metric is similar to $sim_2(p_i, p_j)$, the difference is that the similarity of two VRP solutions is the maximum similarity between their routes.

Notation	Definition
$sim_1(p_i, p_j)$	$\frac{\ell(p_i \cap p_j)}{\ell(p_i \cup p_j)}$
$sim_2(p_i, p_j)$	$\frac{\ell(p_i \cap p_j)}{2\ell(p_i)} + \frac{\ell(p_i \cap p_j)}{2\ell(p_j)}$
$sim_3(p_i, p_j)$	$\sqrt{\frac{\ell(p_i \cap p_j)^2}{\ell(p_i)\ell(p_j)}}$
$sim_4(p_i, p_j)$	$\frac{\ell(p_i \cap p_j)}{\max(\ell(p_i), \ell(p_j))}$
$sim_5(p_i, p_j)$	$\frac{\ell(p_i \cap p_j)}{\min(\ell(p_i), \ell(p_j))}$

Table 1.2: Path similarity metrics from the literature.

1.4 Conclusion

This literature review presented many methods of aircraft trajectory design. Each of them has its characteristics, advantages, and drawbacks. Therefore, they cannot be used in all trajectory optimization problems, i.e. the different phases of a flight. Their characteristics, constraints, and success criteria guides the choice of which method to use for a given problem. The main criteria of choice are the following:

- **Optimality:** Does the method compute the optimal solution?

- **Computing time:** How long does it take to get a solution? This criterion can be essential if one wishes to use the algorithm in real time.
- **Adaptability:** Can the method be adapted to the constraints of the problem?
- **Memory usage:** How much memory space does the method need? This criterion is critical for use in the aircraft's on-board computer, which has limited memory space.
- **Multi trajectories:** Is the method able to compute several trajectories at the same time? It may be useful to compute several trajectories from point A to point B.

These criteria guided the choice of methods for the problems studied in this thesis: Emergency trajectory design, weather hazards avoidance trajectory, tactical conflict detection resolution problem, procedures design, and alternative trajectories generation. The following chapters present these works.

Chapter 2

Aircraft Emergency Trajectory Design

This chapter deals with the aircraft emergency trajectory design. This work was guided by the European project SafeNcy during which the author of this manuscript participated in the design of the path generator algorithm. This chapter is divided into 5 sections. First, we present a short review of the history of aviation accidents. Then, the issues of the problem are detailed. In the last three sections, we present 3 different methods to solve this problem. These methods have been compared to select the one that will be used in the project. They are based on Fast Marching Tree, Fast Marching method and Rapidly exploring Random Tree respectively. The last method is integrated into the last section called SafeNcy that presents the project.

2.1 History of aviation accidents

Although many people are afraid of flying, the plane is the safest mode of transportation in the world. Between 2007 and 2020, the American air passenger deaths rate per 100,000,000 passenger miles is less than 1% (See Figure 2.1) while that of people in road vehicles is over 50%. This fear is explained by the media coverage of aircraft crashes. Indeed, as soon as an accident occurs, it makes the front page of all the newspapers. These accidents even inspire films such as Sully, which tells the story of the ditching of US Airways Flight 1549 on the Hudson [153]. More recently, the TV series "The Crash" relates the story of the El Al Flight 1862 [27].

Despite the significant increase in the number of flights, Figure 2.2 shows a decrease in the number of crashes by year from 1982 to 2019. This can be explained by the evolution of aviation technologies as well as by the introduction of new safety procedures. Unfortunately, these changes are often initiated after accidents. For instance, following the Swissair 111 accident, the Transportation Safety Board of Canada has made 23 recommendations for changes to aircraft materials (tougher testing, certification, inspections, and maintenance procedures), electrical systems, and flight data recording.

Statistics show that up to 80 percents of all aviation accidents can be associated to human errors [163]. Accidents often occur near airports during take-off or landing (Ethiopian Airlines Flight 302 [158] or Asiana Airlines Flight 214 [156] for instance). Pilot error is thought to account for 53% of aircraft accidents, with mechanical failure (21%) and weather conditions (11%) following behind.

Although aviation is becoming safer, many fatal accidents have occurred. Table 2.1 shows the 10

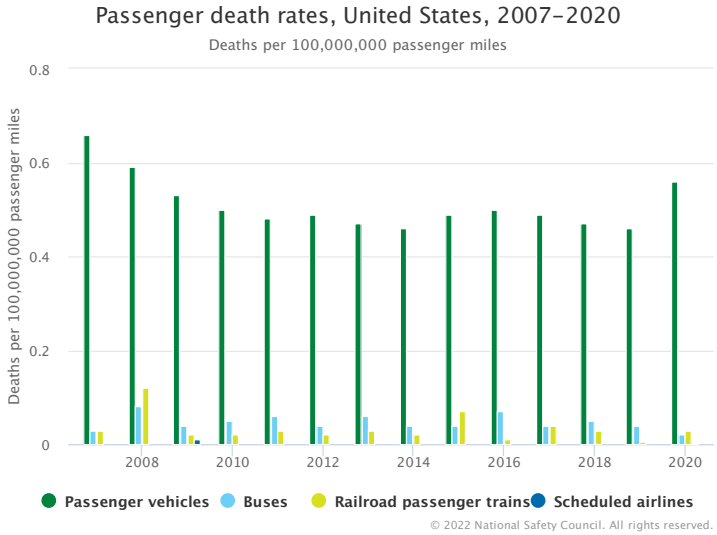


Figure 2.1: Passenger death rates per 100,000,000 passenger miles, United States, 2007-2020 from [151].

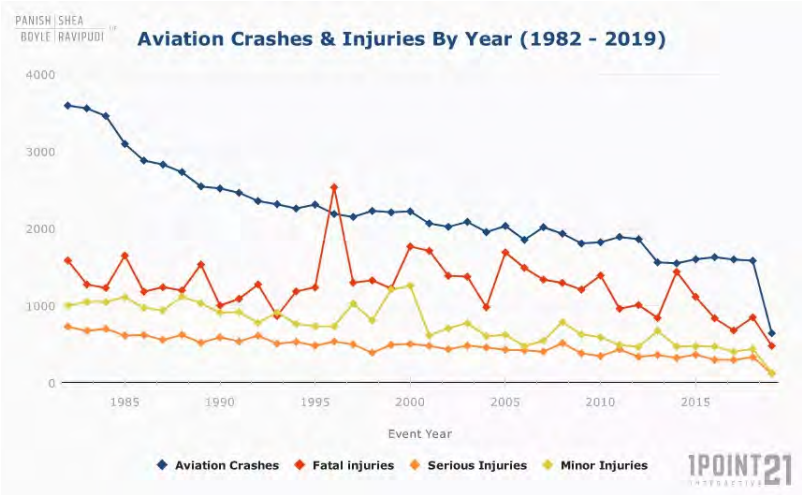


Figure 2.2: Aviation Crashes and Injuries by year (1982-2019) [163].

deadliest crashes in aviation history. These terrible numbers can make travelers anxious during their flights. According to the study presented in [45], more than 30% of people surveyed feel some kind of anxiety when it comes to flying. Moreover, between 2.5 and 5% of the population suffer from a real fear of flying, which can be described as a clinical phobia.

Following this historical study of aviation accidents, it seems necessary to develop tools to help the pilot in case of an emergency situation in order to limit human errors. In addition, these systems would reduce the number of accidents and thus reduce the fear of some people. The next section presents the issues of the development of this type of system.

2.2 Issues

“I realized that flying a plane meant not making mistakes. You had to maintain control of everything. You had to look out for the wires, the birds, the trees, the fog, while monitoring

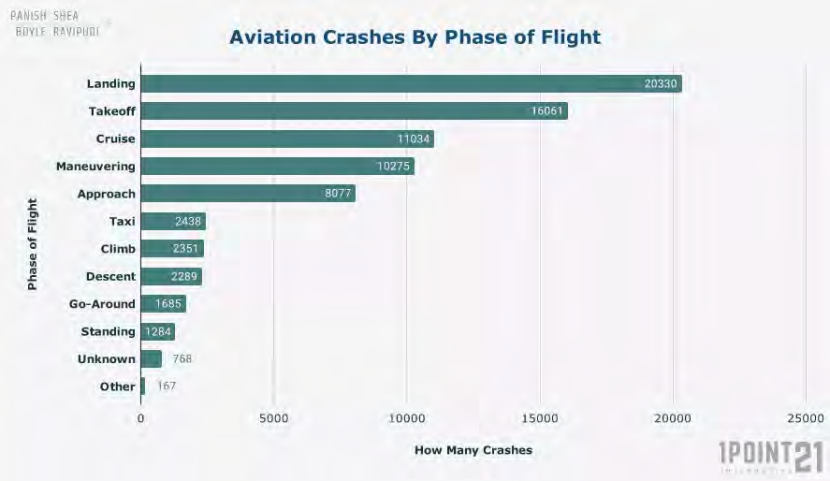


Figure 2.3: Aviation Crashes By Phase of Flight from 1982 to 2019 [163].

Incident	Date	Deaths	Aicraft	Location	Reason
Pan Am Flight 1736 and KLM Flight 4805	1977-03-27	583	Boeing 747-121 and Boeing 747-206B	Tenerife, Spain	Collision
Japan Air Lines Flight 123	1985-08-12	520	Boeing 747SR-46	Ueno, Japan	Depressurization
Saudi Arabian Airlines Flight 763 and Kazakhstan Airlines Flight 1907	1996-11-12	349	Boeing 747-168B and Ilyushin Il-76TD	Charkhi Dadri, India	Collision
Turkish Airlines Flight 981	1974-03-03	346	McDonnell Douglas DC-10-10	Fontaine-Chaalis, France	Depressurization
Air India Flight 182	1985-06-23	329	Boeing 747-237B	Atlantic Ocean, area of Cork, Ireland	Terrorism
Saudia Flight 163	1980-08-19	301	Lockheed L-1011-200 TriStar	Riyadh, Saudi Arabia	Fire
Malaysia Airlines Flight 17	2014-07-17	298	Boeing 777-2H6ER	near Hrabove, Donetsk Oblast, Ukraine	Shot down
Iran Air Flight 655	1988-07-03	290	Airbus A300B2-203	Strait of Hormuz, off Shib Deraz, Iran	Shot down
Iranian Revolutionary Guard Corps Aerospace Force	2003-02-19	275	Ilyushin Il-76MD	near Kerman, Iran	Controlled Flight Into Terrain (CFIT)
American Airlines Flight 191	1979-05-25	273	McDonnell Douglas DC-10-10	Des Plaines, Illinois, U.S.	Loss of an engine

Table 2.1: The 10 deadliest crashes in aviation history.

everything in the cockpit. You had to be vigilant and alert. It was equally important to know what was possible and what was not. One simple mistake could mean death." These words from Captain Chesley B. Sullenberger highlight the permanent need for pilots to focus whenever they fly an aircraft. Hence, in an emergency situation, pilots are effectively prepared but under a huge amount of stress. Moreover, in some very critical cases, it is almost impossible for the pilot to make the best decision. These observations highlight the importance of developing a tool to help pilots land safely in case of emergency. This tool would predict a safe and optimal trajectory that pilots would have had difficulty finding alone in a situation of intense stress. It would therefore significantly increase the chance of saving the aircraft.

To better understand the issues at stake, it is essential to look at the history of aviation. Indeed, accidents have often led to changes in aircraft systems, structures, and safety procedures. In addition, these accidents have prompted the industry to develop new safety tools. In this section, two accidents are presented. The first one is one of the most famous accidents: the sea landing on the Hudson River of the US Airways Flight 1549 after bird strikes caused dual engine failure (see Figure 2.4). This case is very interesting because in just 30 seconds the situation went from critical to impossible. The only solution was to land on the Hudson River. This observation

highlights the fact that an emergency trajectory generation algorithm has to be very efficient in terms of computation time.

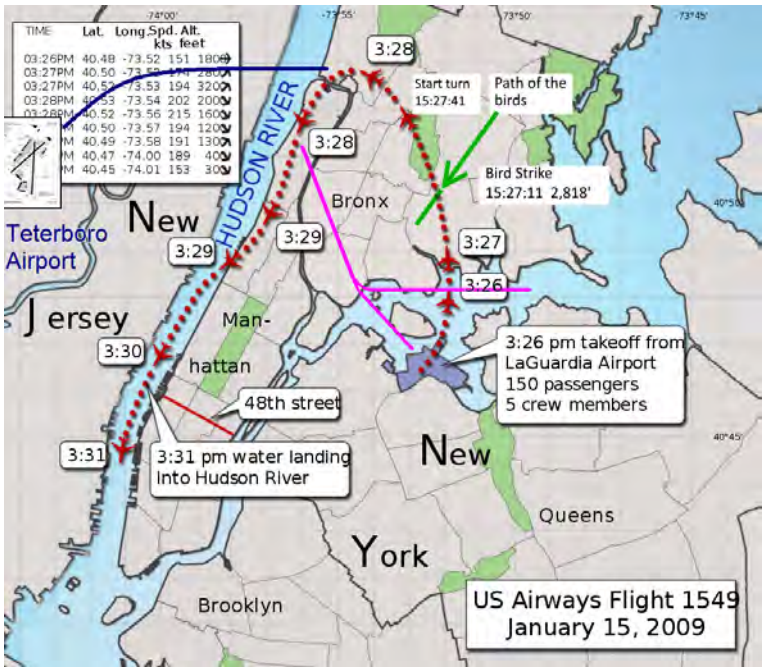


Figure 2.4: US Airways Flight 1549: Two minutes after takeoff from LaGuardia, the aircraft struck a flock of birds at an altitude of 2818 ft, which caused the loss of engines and forced the pilot to make a sea landing on the Hudson river [41].

The second accident is Air Transat Flight 236 in 2001 [183]. It lost all engine power while flying over the Atlantic Ocean. The Airbus A330 ran out of fuel due to a fuel leak. After 21 minutes of gliding, the pilot managed to land on a military base in the Azores. The pilot made the approach by hand and the aircraft arrived on the runway too fast. It was badly damaged and the landing could have resulted in a fatal fall for the passengers, as the runway ended with a cliff. This flight is historic because it was the longest commercial aircraft glide without engines. It is interesting to compare this flight with another one in 1959. In similar conditions (light wind and lift-to-drag ratios almost equal), the Caravelle "Lorraine" succeeded to glide during 46 minutes [147]. More than twice that of Air Transat flight 236. The main difference between the two flights is that the trajectory of the Caravelle was predicted. This prediction, based on weather forecasts and the aircraft's characteristics, helped pilots glide longer. These two examples show that a good prediction can significantly increase the gliding distance and therefore increase the chance of a safe landing.

The last very important point is that there is an almost infinite number of emergency cases, which makes this problem very complicated. In the literature, emergency situations are classified into three categories: *ASAP (As Soon as Possible)*, *ANSA (At Nearest Suitable Airport)* and *TEFO (Total Engine Flame Out)*. ASAP emergencies are the most critical situations, the aircraft has to land as soon as possible. For example, a cabin fire and medical emergency are considered ASAP. An ANSA emergency is less critical in terms of time. In this case, the selected landing site is the safest among those reachable. Finally, the TEFO case is the most specific, it corresponds to the case of the total loss of the power of the engines following a breakdown or a lack of fuel. Depending on the situation, the constraints are extremely different. Indeed, if the aircraft loses its engines, the dynamics of the aircraft are drastically modified. Whereas, in the case of a cabin fire, the aircraft performance are not degraded. The adaptability of a proposed

algorithm is therefore essential. The definition of the objective function is not obvious. Indeed, in this type of problem, the goal is to generate a feasible trajectory, not the optimal trajectory. Therefore, the choice of the objective function is more flexible. The aim can be, for example, to optimize the time, the distance, or the curvature.

To summarize, a decision support tool for emergency situations must be fast, compute a solution close to the optimum and be adaptable to all situations. During these last few years, industry has been interested in the development of this type of tool. In 2016, Cirrus Aircraft released a new single-pilot aircraft called Vision Jet equipped with an autolanding system [42]. This system named Safe Return was developed in partnership with Garmin. This system is equipped with a button in the cabin, allowing passengers to activate it in case the pilot has a medical problem. Once activated, it automatically detects a landing field and generates a trajectory for the autopilot to follow. In 2014, Austin Meyer developed an iOS application that connects to the autopilot of a small private plane and computes an emergency trajectory in the event of engine failure or fuel shortage [146]. Commercial aircraft manufacturers are also interested in this type of tool. Airbus begins testing autonomous flight technology on an A350-1000 aircraft. The aircraft manufacturer is innovating to improve the safety and efficiency of specific flight operations [4]. This problem also leads to the creation of a European project called SafeNcy [43] which will be presented in detail in the last section.

Following this study of the problem, we propose three different methods to solve it. These methods are based on fast algorithms to meet the constraint of using a real-time system. The first method is based on an adapted and accelerated version of the Fast Marching Tree algorithm. Then, algorithms based on 2D and 3D Fast Marching methods are proposed. Finally, the SafeNcy project is presented and more particularly the path generator based on the Rapidly Exploring Random Tree algorithm.

2.3 Fast Marching Tree Star Approach

The first proposed method is based on the Fast Marching Tree algorithm. The goal of this approach is to develop a fast trajectory generator algorithm that can be adapted to the problem of emergency trajectory design. For this adaptation to the emergency case, the landing site is considered known, determined by an algorithm that takes into account the altitude data around the emergency aircraft position. The heading constraints on departure and arrival have been added. The aircraft has a given turning curvature radius, a maximum climb rate, and a maximum descent rate. This section is divided into 5 parts. The first part presents the Fast Marching Tree algorithm. Then, the mathematical modeling is detailed. Then, operational constraints taken into account are described. The fourth part presents the proposed improvement to speed up the algorithm. Finally, several cases on which the algorithm has been tested are given, some designed specially to compare algorithms from the literature and the proposed algorithm, and another one designed based on real data.

2.3.1 Fast Marching Tree

The Fast Marching Tree (FMT*) algorithm has been introduced by Janson et al. [108]. It performs a forward-propagation over several sampled points generated during the initialization step and generates a tree of paths. Three key features characterize this algorithm:

- two samples are considered neighbors if their distance is below a given radius;

- graph construction and path search are performed concurrently;
- if the locally-optimal connection to a new sample intersects an obstacle, the algorithm skips this sample. It does not consider the other connections in the neighborhood.

Before presenting the details of the algorithm, some sets must be introduced. $V_{unvisited}$ is a set composed of nodes that do not yet have an assigned cost. V_{closed} is composed of nodes that are visited and have a cost that can no longer be modified because it is optimal. V_{open} is a set that contains visited nodes but their cost is temporally assigned. Figure 2.5, from [108], represents the local optimization phase of Algorithm 1, which is repeated as long as the destination x_{goal} is not reached. The algorithm begins with a graph composed of only one node which is the starting point x_{start}.

All nodes at a distance r of a node are considered as its neighbors. r is the neighborhood radius and is a function of the number N of samplings. It is defined as follows:

$$r = (1 + \eta)\gamma_{FMT^*}, \tag{2.1}$$

where $\eta > 0$ is a parameter which has to be defined and γ_{FMT^*} is the minimal radius computed as follows:

$$\gamma_{FMT^*} = 2\left(\frac{1}{d}\right)^{1/d}\left(\frac{\mu(\chi_{free})}{\zeta_d}\right)^{1/d}\left(\frac{\log N}{N}\right)^{1/d}, \tag{2.2}$$

where d is the space dimension, $\mu(\chi_{free})$ its Lebesgue measure [1], ζ_d the volume of the d-dimensional unitary ball [2].

According to the literature, FMT* is the fastest algorithm (See Table 1.1). Indeed the fact that the algorithm checks only one connection, then the computing complexity of the collision test is reduced to $O(N)$ whereas for PRM* and RRT*, it is $O(N \log N)$ where N is the number of sampling points.

The FMT* algorithm has been mainly used in robotics [195, 198, 199, 202] but it can be adapted to aircraft trajectory design. The main issue of this algorithm is that the computed path is not flyable because all the points building the path are connected with straight lines, which can create heading discontinuity at each point. Moreover, heading constraints on arrival and departure are not satisfied. A solution to solve this problem will be presented in 2.3.3. The next section details the mathematical modeling.

2.3.2 Mathematical modeling

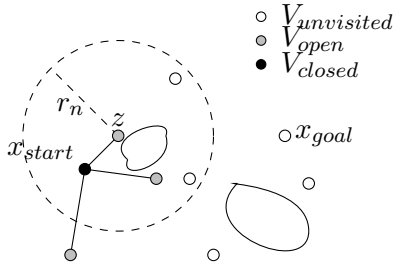
This part presents the search space, the path representation, the objective function, and the constraints.

2.3.2.1 Search Space

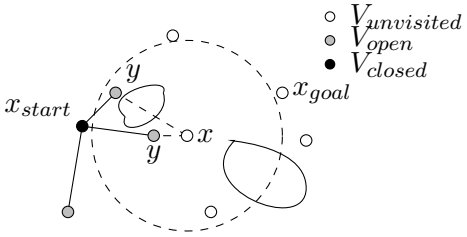
Based on altitude data around the aircraft's position, a cube is created to model the search space. In this cube, each point p generated during the sampling phase is a state vector containing the

[1] $\mu([a_1, b_1] \times [a_2, b_2]) = (b_1 - a_1)(b_2 - a_2)$ where $b_1 > a_1$ and $b_2 > a_2$.
$\mu([a_1, b_1] \times [a_2, b_2] \times [a_3, b_3]) = (b_1 - a_1)(b_2 - a_2)(b_3 - a_3)$ where $b_1 > a_1$,$b_2 > a_2$ and $b_3 > a_3$.

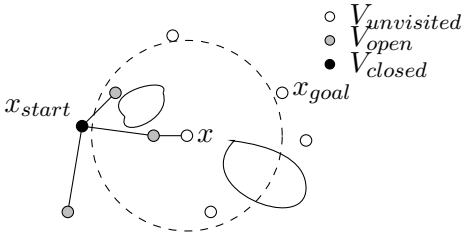
[2] In 2D space, ζ_2 is the surface of a disk of radius 1 ($\zeta_2 = \pi$).
In 3D space, ζ_3 is the volume of a ball of radius 1 ($\zeta_3 = \frac{4\pi}{3}$)



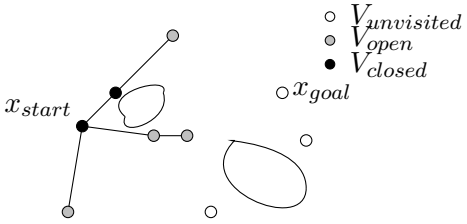
(a) In this first step, the lowest-cost node z from set V_{open} is selected and the nodes within $V_{unvisited}$ which are near to z are found.



(b) In the second step, the neighbors of x within V_{open} are found, and x is connected to have an optimal cost without going through an obstacle.



(c) A connection is created between x and the neighbor of the locally-optimal step connection.



(d) All neighbors of z are visited and are added to V_{open}. z is added to V_{closed}. FMT* moves to the next iteration with the node which has the lowest cost.

Figure 2.5: Steps of an iteration of the FMT* Algorithm, from [108].

coordinates x and y and the altitude h, giving the vector:

$$p = \begin{pmatrix} x \\ y \\ h \end{pmatrix}, \tag{2.3}$$

where:

$$terrain(x, y) < h < h_{max}. \tag{2.4}$$

In this study, the route is represented by a set of n points $\{p_0,\ p_1,\ \ldots,\ p_{n-1}\}$ such as $\forall i, 0 \leq i < n-1$, p_i is connected to p_{i+1}. In the first attempt, the points are connected with straight lines. Then, to make the path flyable, such straight lines are replaced by Dubins curves. The method is presented in detail in Section 2.3.3. To do it, it is necessary to define the heading σ_i of a point p_i. For each point of the path, the heading (σ_i, in radians; see Figure 2.6) is computed as follows:

$$\begin{cases} \sigma_i = \mod\left(\frac{\pi}{2} - \arctan\frac{y_{i+1} - y_i}{x_{i+1} - x_i}, 2\pi\right), 0 < i < n-1 \\ \sigma_0 = \sigma_{start} \\ \sigma_{n-1} = \sigma_{landingSite} \end{cases}. \tag{2.5}$$

The heading enables the algorithm to connect the points with a Dubins curve and therefore take into account the curvature radius of the aircraft (presented in detail in the next section).

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

Algorithm 1 Fast Marching Tree: from a sampling composed of n samples, the algorithm creates the graph. At each step, the minimum cost node is selected and the algorithm tries to connect these neighbours (nodes at a distance lower than a radius depending on the number of samples) to the graph.[108]

$V \leftarrow \{x_{start}\} \cup \texttt{SampleFree}(n); E \leftarrow \emptyset$
$V_{unvisited} \leftarrow V \backslash \{x_{start}\}; V_{open} \leftarrow \{x_{init}\}, V_{closed} \leftarrow \emptyset$
$z \leftarrow x_{start}$
$N_z \leftarrow \texttt{Near}(V \backslash \{z\}, z, r_n)$
$Save(N_z, z)$
while $z \notin \chi_{goal}$ **do**
  $V_{open,new} \leftarrow \emptyset$
  $X_{near} = N_z \cap V_{unvisited}$
  for all $x \in X_{near}$ **do**
    $N_x \leftarrow \texttt{Near}(V \backslash \{x\}, x, r_n)$
    $Save(N_x, x)$
    $Y_{near} \leftarrow N_x \cap V_{open}$
    $y_{min} \leftarrow \arg\min_{y \in Y_{near}} \{c(y) + Cost(y, x)\}$
    if $\texttt{CollisionFree}(y_{min}, x)$ **then**
      $E \leftarrow E \cup \{(y_{min}, x)\}$
      $V_{open,new} \leftarrow V_{open,new} \cup \{x\}$
      $c(x) = c(y_{min}) + \texttt{Cost}(y_{min}, x)$
    end if
  end for
  $V_{open} \leftarrow (V_{open} \cup V_{open,new}) \backslash \{z\}$
  $V_{closed} \leftarrow V_{closed} \cup \{z\}$
  if $V_{open} = \emptyset$ **then**
    return Failure
  end if
  $z \leftarrow \arg\min_{y \in Vopen} \{c(y)\}$
end while
return $Path(z, T = (V_{open} \cup V_{closed}, E))$

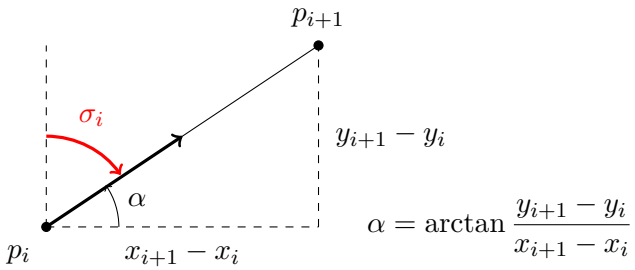


Figure 2.6: Heading computation: From the position of the points p_i and p_{i+1}, the orientation angle α is computed, from the heading h_i.

$$cost(p_0, p_1, \ldots, p_{n-1}) = \sum_{i=0}^{n-2} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2. \tag{2.6}$$

This choice was made to reduce the computing time required for a distance between two points as much as possible. Indeed, this computation is cheaper than the computation of a Dubins

curve. This choice introduces error, as the Dubins curve distance can be very different from the Euclidean distance. However, it is important to remember that the algorithm does not aim finding the optimum path but rather a safe path.

2.3.3 Operational Constraints

One of the main constraints taken into account in this study is the curvature constraint. The Dubins curve is a solution to take into account this constraint [81, 95].

2.3.3.1 Dubins Curve

The Dubins path typically refers to the shortest curve that connects two points with a constraint on the curvature of the path and with prescribed initial and terminal tangents to the path [130, 176]. There are six types of Dubins curves; four are of the Circle-Segment-Circle type and two are of the Circle-Circle-Circle type. If the distance between two connected points is smaller than the sum of the two radii and if the initial and final heading are in opposite directions, no straight line segment can be fitted between the circle segments [68]. In this case, the shortest route is a path of the Circle-Circle-Circle type.

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be the start position and the final position. Let θ_1 and θ_2 be their heading direction. Let α and β be the start and final orientation angle. α and β are therefore defined by $\alpha = \mod(\frac{\pi}{2} - \theta_1, 2\pi)$ and $\beta = \mod(\frac{\pi}{2} - \theta_2, 2\pi)$. The following paragraphs present the 2 types of Dubins curves. A Circle-Segment-Circle Dubins (LSL, RSR, LSR, RSL) curve is composed of a first turn to the right or the left, then a straight line, and finally a second turn to the right or the left. A Circle-Circle-Circle Dubins curve (LSL and RSR) is composed of 3 different alternating turns to the right or the left.

To determine feasible paths, it is necessary to introduce the three elementary motions: turning to the left, turning to the right (both along a circle C of radius r), and straight line motion S of length l. The three corresponding operators, L_ϕ (left turn), R_ϕ (right turn) and S_l (straight line) transforms a given 2D point (x, y, α) with its orientation angle α as follows:

$$\begin{cases} L_\phi(x, y, \alpha) = (x + \sin(\alpha + \phi) - \sin\alpha, y - \cos(\alpha + \phi) + \cos\alpha, \alpha + \phi) \\ R_\phi(x, y, \alpha) = (x - \sin(\alpha - \phi) + \sin\alpha, y + \cos(\alpha - \phi) - \cos\alpha, \alpha - \phi) \\ S_l(x, y, \alpha) = (x + l\cos\alpha, y + l\sin\alpha, \alpha) \end{cases} . \tag{2.7}$$

Circle-Segment-Circle Dubins Curve Let ϕ_1 and ϕ_2 be the angles defining the rotation around the first and second circle respectively. Let r_1 and r_2 be the radius of the first and second circle respectively. Let l be the length of the straight line. In the case of Left-Segment-Left (LSL) Dubins curve, the following constraint is defined:

$$L_{\phi_1}(S_l(L_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.8}$$

This constraint leads to the following system of equations:

$$\begin{cases} \alpha + \phi_1 + \phi_2 = \beta \\ x_2 = x_1 + r_1\sin(\alpha + \phi_1) - r_1\sin\alpha + l\cos(\alpha + \phi_1) + r_2\sin\beta + r_2\sin(\alpha + t) \\ y_2 = y_1 - r_1\cos(\alpha + \phi_1) + r_1\cos\alpha + l\sin(\alpha + \phi_1) - r_2\cos\beta + r_2\cos(\alpha + \phi_1) \end{cases} . \tag{2.9}$$

The solution of this system is the following:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X l + \Delta Y(r_2 - r_1)}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \\ \phi_2 = \beta - \arctan\left(\dfrac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X l + \Delta Y(r_2 - r_1)}\right) \end{cases}, \tag{2.10}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1 \sin\alpha - r_2 \sin\beta \\ \Delta Y = y_2 - y_1 - r_1 \cos\alpha + r_2 \cos\beta \end{cases}. \tag{2.11}$$

Figure 2.7 shows an example of an LSL Dubins curve.

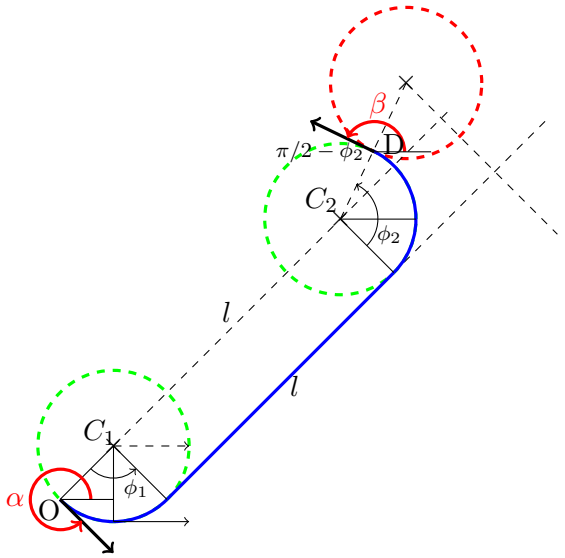


Figure 2.7: Left-Segment-Left Dubins Curve: The curve connects a start point with an orientation angle α and an end point with an orientation angle β. It is composed of a turn to the left around the first green circle, a segment of length l and a second turn to the left around the second green circle.

In the case of a Right-Segment-Right (RSR) Dubins curve, the following constraint is defined:

$$R_{\phi_1}(S_l(R_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.12}$$

This constraint leads to the following system of equations:

$$\begin{cases} \alpha - \phi_1 - \phi_2 = \beta \\ x_2 = x_1 - r_1 \sin(\alpha - \phi_1) + r_1 \sin\alpha + l\cos(\alpha - \phi_1) - r_2 \sin\beta + r_2 \sin(\alpha - \phi_1) \\ y_2 = y_1 + r_1 \cos(\alpha - \phi_1) - r_1 \cos\alpha + l\sin(\alpha - \phi_1) + r_2 \cos\beta - r_2 \cos(\alpha - \phi_1) \end{cases}. \tag{2.13}$$

The solution of this system is the following:

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(r_2 - r_1) + \Delta Y l}{\Delta X l + \Delta Y(r_1 - r_2)}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \\ \phi_2 = -\beta + \arctan\left(\dfrac{\Delta X(r_2 - r_1) + \Delta Y l}{\Delta X l + \Delta Y(r_1 - r_2)}\right) \end{cases}, \tag{2.14}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1 \sin\alpha + r_2 \sin\beta \\ \Delta Y = y_2 - y_1 + r_1 \cos\alpha - r_2 \cos\beta \end{cases} . \tag{2.15}$$

Figure 2.8 shows an example of an RSR Dubins curve.

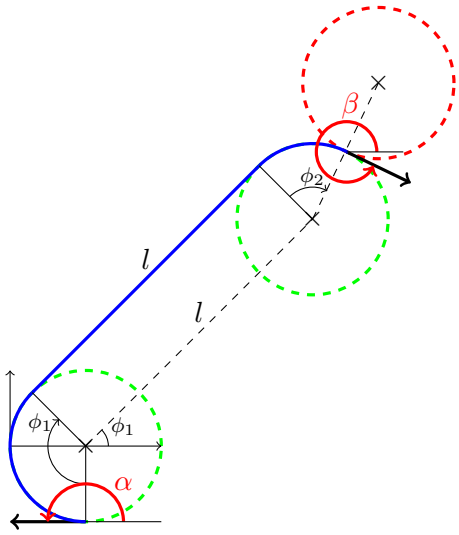


Figure 2.8: Right-Segment-Right Dubins curve: The curve connects a start point with an orientation angle α and an end point with an orienation angle β. It is composed of a turn to the right around the first green circle, a segment of length l, and a second turn to the right around the second green circle.

In the case of a Right-Segment-Left (RSL) Dubins curve, the following constraint is defined:

$$R_{\phi_1}(S_l(L_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.16}$$

This constraint leads to the following system of equations:

$$\begin{cases} \alpha - \phi_1 + \phi_2 = \beta \\ x_2 = x_1 - r_1 \sin\alpha - \phi_1 + r_1 \sin\alpha + r_2 \sin\beta - r_2 \sin\alpha - \phi_1 \\ y_2 = y_1 + r_1 \cos\alpha - \phi_1 - r_1 \cos\alpha + r_2 \cos\beta - r_2 \cos\alpha - \phi_1 \end{cases} . \tag{2.17}$$

The solution of this system is the following:

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 + r_1)^2} \\ \phi_2 = \beta - \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \end{cases} , \tag{2.18}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1 \sin\alpha - r_2 \sin\beta \\ \Delta Y = y_2 - y_1 + r_1 \cos\alpha + r_2 \cos\beta \end{cases} . \tag{2.19}$$

Figure 2.9 shows an example of a RSL Dubins Curve.

In the case of a Left-Segment-Right (LSR) Dubins curve, the following constraint is defined:

$$L_{\phi_1}(S_l(R_{phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.20}$$

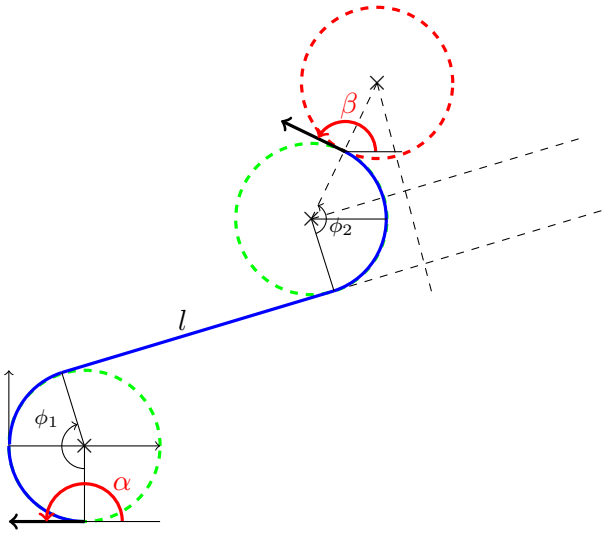


Figure 2.9: Right-Segment-Left Dubins curve: The curve connects a start point with an orientation angle α and an end point with an orienation angle β. It is composed of a turn to the right around the first green circle, a segment of length l, and a second turn to the left around the second green circle.

This constraint leads to the following system of equations:

$$\begin{cases} \alpha + \phi_1 - \phi_2 = \beta \\ x_2 = x_1 + r_1 \sin\alpha + \phi_1 - r_1 \sin\alpha - r_2 \sin\beta + r_2 \sin\alpha + \phi_1 \\ y_2 = y_1 - r_1 \cos\alpha + \phi_1 + r_1 \cos\alpha - r_2 \cos\beta + r_2 \cos\alpha + \phi_1 \end{cases} . \tag{2.21}$$

The solution of this system is the following:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X(r_1+r_2) + \Delta Y l}{\Delta X l - (r_1+r_2)\Delta Y}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2+r_1)^2} \\ \phi_2 = -\beta + \arctan\left(\dfrac{\Delta X(r_1+r_2) + \Delta Y l}{\Delta X l - (r_1+r_2)\Delta Y}\right) \end{cases} , \tag{2.22}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1 \sin\alpha + r_2 \sin\beta \\ \Delta Y = y_2 - y_1 - r_1 \cos\alpha - r_2 \cos\beta \end{cases} . \tag{2.23}$$

Figure 2.10 shows an example of an LSR Dubins curve.

Circle-Circle-Circle Dubins Curve Let ϕ_1, ϕ_2, and ϕ_3 be the angles defining the rotation around the first, the second, and the third circle respectively. Let r_1, r_2 and r_3 be the radius of the first, the second, and the third circle respectively.

In the case of a Left-Right-Left (LRL) Dubins curve, the following constraint is defined:

$$L_{\phi_1}(R_{\phi_2}(L_{\phi_3}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.24}$$

This constraint leads to the following system of equations:

$$\begin{cases} \alpha + \phi_1 - \phi_2 + \phi_3 = \beta \\ x_2 = x_1 + r_1 \sin(\alpha + \phi_1) - r_1 \sin\alpha - r_2 \sin(\alpha + \phi_1 - \phi_2) \\ \qquad + r_2 \sin(\alpha + \phi_1) + r_3 \sin\beta - r_3 \sin(\alpha + \phi_1 - \phi_2) \\ y_2 = y_1 - r_1 \cos(\alpha + \phi_1) + r_1 \cos\alpha + r_2 \cos(\alpha + \phi_1 - \phi_2) \\ \qquad - r_2 \cos(\alpha + \phi_1) - r_3 \cos\beta + r_3 \cos(\alpha + \phi_1 - \phi_2) \end{cases} . \tag{2.25}$$

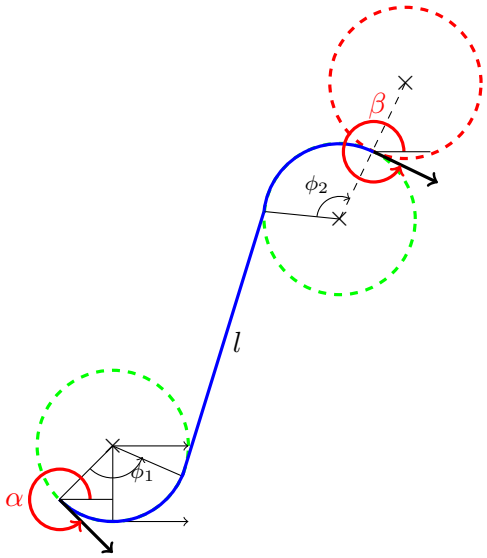


Figure 2.10: Left-Segment-Right Dubins curve: The curve connects a start point with an orientation angle α and an end point with an orienation angle β. It is composed of a turn to the left around the first green circle, a segment of length l, and a second turn to the right around the second green circle.

The solution of this system is the following:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X((r_1+r_2)-(r_2+r_3)\cos\phi_2)+\Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2+\Delta Y(-(r_1+r_2)+(r_2+r_3)\cos\phi_2)}\right) \\ \phi_2 = \arccos\left(\dfrac{(r_1+r_2)^2+(r_2+r_3)^2-\Delta X^2-\Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \\ \phi_3 = \beta - \arctan\left(\dfrac{\Delta X((r_1+r_2)-(r_2+r_3)\cos\phi_2)+\Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2+\Delta Y(-(r_1+r_2)+(r_2+r_3)\cos p)}\right) \\ \qquad + \arccos\left(\dfrac{(r_1+r_2)^2+(r_2+r_3)^2-\Delta X^2-\Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \end{cases}, \tag{2.26}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1\sin\alpha - r_3\sin\beta \\ \Delta Y = y_2 - y_1 - r_1\cos\alpha + r_3\cos\beta \end{cases}. \tag{2.27}$$

Figure 2.11 shows an example of an LRL Dubins curve.

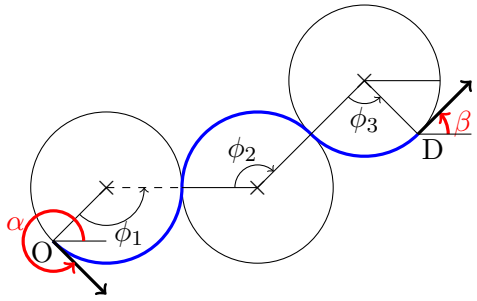


Figure 2.11: Left-Right-Left Dubins curve: The curve connects a start point with an orientation angle α and an end point with an orientation angle β. It is composed of 3 different turns.

In the case of Right-Left-Right (RLR) Dubins curve, the following constraint is defined:

$$L_{\phi_1}(R_{\phi_2}(L_{\phi_3}(x_1, y_1, \alpha))) = (x_2, y_2, \beta). \tag{2.28}$$

This constraint leads to the following system of equations:

$$\begin{cases} \alpha - \phi_1 + \phi_2 - \phi_3 = \beta \\ x_2 = x_1 - r_1 \sin(\alpha - \phi_1) + r_1 \sin\alpha + r_2 \sin(\alpha - \phi_1 + \phi_2) \\ \qquad - r_2 \sin(\alpha - \phi_1) - r_3 \sin\beta + r_3 \sin(\alpha - \phi_1 + \phi_2) \\ y_2 = y_1 + r_1 \cos(\alpha - \phi_1) - r_1 \cos\alpha + r_2 \cos(\alpha - \phi_1 + \phi_2) \\ \qquad - r_2 \cos(\alpha - \phi_1) - r_3 \cos\beta + r_3 \cos(\alpha - \phi_1 + \phi_2) \end{cases}. \tag{2.29}$$

The solution of this system is the following:

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(-(r_1+r_2) + (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin p}{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y((r_1+r_2) - (r_2+r_3)\cos\phi_2)}\right) \\ \phi_2 = \arccos\left(\dfrac{(r_1+r_2)^2 + (r_2+r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \\ \phi_3 = -\beta + \arctan\left(\dfrac{\Delta X((r_1+r_2) - (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y(-(r_1+r_2) + (r_2+r_3)\cos\phi_2)}\right) \\ \qquad + \arccos\left(\dfrac{(r_1+r_2)^2 + (r_2+r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \end{cases}, \tag{2.30}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1 \sin\alpha + r_3 \sin\beta \\ \Delta Y = y_2 - y_1 + r_1 \cos\alpha - r_3 \cos\beta \end{cases}. \tag{2.31}$$

Figure 2.12 shows an example of an RLR Dubins curve.

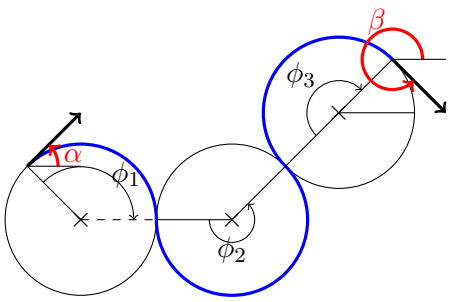


Figure 2.12: Right-Left-Right Dubins curve: The curve connects a start point with an orientation angle α and an end point with an orientation angle β. It is composed of 3 different turns.

All demonstrations are given in Appendix A.

To make the trajectory flyable and take into account the curvature constraints, it is proposed to smooth the path generated by the FMT algorithm by replacing straight lines with Dubins curves. The connection of points with Dubins curves could be accomplished during the FMT phase, but as Dubins curve generation is too slow, the points are first connected by straight lines and then adjusted with Dubins curves (see Figure 2.13).

This process requires the enlargement of the obstacles in order to be sure that the Dubins curves will be in the free space. Indeed, if the straight line path passes near an obstacle, the Dubins path could pass through this obstacle due to the turn constraints (see Figure 2.14).

(a) Path without Dubins curve

(b) Path with Dubins curve

Figure 2.13: Path modification: The first path is computed by the FMT algorithm. The second path is the path after the addition of Dubins curves to respect the heading constraints (in red).

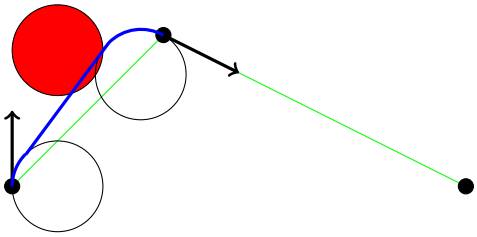
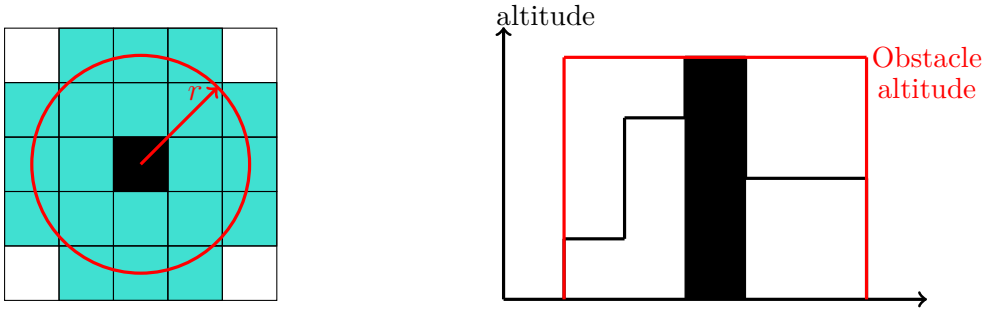


Figure 2.14: Example of collision with an obstacle after the addition of Dubins curves: the straight line path is drawn in green and is in the free space. The Dubins path is shown in blue and passes though the obstacle in red.

The obstacles have therefore been enlarged horizontally by a distance corresponding to the curvature radius of the aircraft. The cells located at a horizontal distance lower than the curvature radius of an obstacle cell become obstacles (see Figure 2.15a). Moreover, all cells under these cells are also considered as obstacles. This implies that the altitude of this circular area is equal to the altitude of the obstacle cell (see Figure 2.15b).



(a) Horizontal enlargement (obstacle cell: black, new obstacle cells: blue).

(b) Vertical enlargement.

Figure 2.15: Enlargement of obstacles.

2.3.3.2 Descent Constraints

The second constraint that has to be taken into account is the descent constraint. Each type of emergency affects the descent in a different way. For example, in the case of an emergency due to a cabin fire, the aircraft's performance is not affected. Therefore, the aircraft can climb and descend as usual. However, in the case of an emergency due to a dual engine failure, the aircraft has to descend to maintain sufficient speed to avoid stalling. In this case, the pilots decide most often to choose the lowest possible descent rate so that they can cover the longest

possible distance and have enough time to choose a safe landing site (see Figure 2.16). It is also constrained by a maximum descent rate. Indeed, if it is higher than the maximum value, it can lead to an overspeed for landing. However, to land safely, the speed of the aircraft should not be too high.

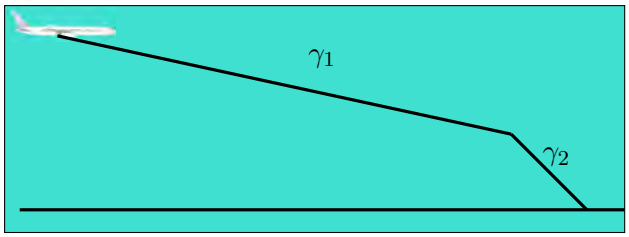


Figure 2.16: Example of descent profile: γ_1 corresponds to the minimal descent angle (Maximal lift-to-drag ratio) and γ_2 is the descent angle during the approach phase.

In this study, the aircraft is constrained by a maximal and a minimal descent angle (see Figure 2.17). During the building tree phase, to connect two points in the free space, the descent angle has to be checked between these two values.

The descent constraints can be written as follows:

$$\forall i < n \begin{cases} \arctan \dfrac{d_h(p_i, p_{i+1})}{|z_{i+1} - z_i|} \geq \gamma_{min} \\ \arctan \dfrac{d_h(p_i, p_{i+1})}{|z_{i+1} - z_i|} \leq \gamma_{max} \end{cases}, \tag{2.32}$$

where d_h is the euclidean horizontal distance function.

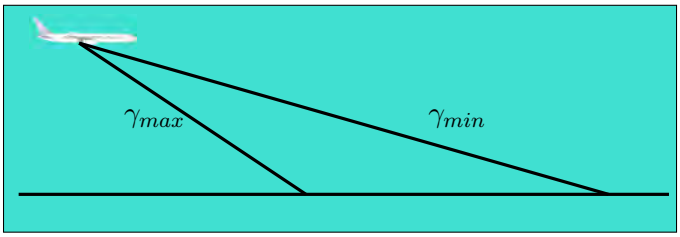


Figure 2.17: Descent constraint (minimal descent angle = γ_{min} and maximal descent angle = γ_{max}).

This subsection has presented the adaptation of the Fast Marching Tree algorithm to emergency trajectory generation. One of the most important points in this problem is to generate the trajectory as fast as possible. The next subsection presents some improvements to speed up the algorithm.

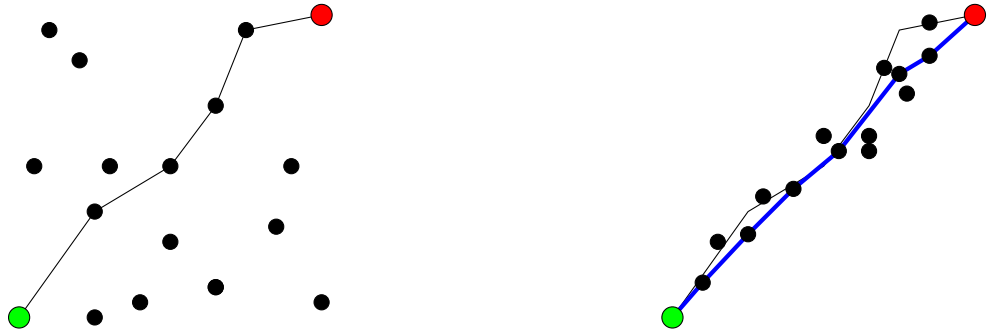
2.3.4 Improvements

The two main functions of the FMT* algorithm are the sampling and the free space checker. This subsection presents improvements of these functions.

2.3.4.1 Smart Sampling

The FMT* algorithm is asymptotically optimal, however it is necessary to generate a large number of samples to obtain a solution close to the optimum. To get a good solution while

having a low computing time, we propose to generate two sampling mechanisms. The first sampling is composed of a small number of samples (See Figure 2.18a). Thanks to it, the FMT* algorithm generates a first approximate solution. This solution can be far from the optimal path. To refine this first solution, a new sampling is generated around this initial solution (See Figure 2.18b). It can be noted that in the worst case, the second path is the same as the first one.



(a) First sampling (b) Second sampling

Figure 2.18: Sampling: The first path (left) is computed by the FMT algorithm from a sampling on the whole space. The second path (right in blue) is computed from a sampling around the first approximate path (start point in green and end point in red).

2.3.4.2 Free Space Checker

The other main component of the FMT* algorithm is the free space test function. It consists in verifying if there is an obstacle between the two points in order to validate the connection in the graph. FMT* spends half its time on this process. It is therefore critical to have a very efficient free space test checking to reduce the computing time required for trajectory generation. One way to check that a segment between two points is in the free space is to discretize the segment and to check if all the points of the segment are in the free space. This discretization can be done by Bresenham's algorithms. There are two Bresenham's algorithms: Bresenham's line and Bresenham's circle. The first is a line drawing algorithm that determines the points of an n-dimensional raster that should be selected to form a close approximation to a straight line between two points [32] (see Figure 2.19). Bresehnam's circle algorithm is derived from the

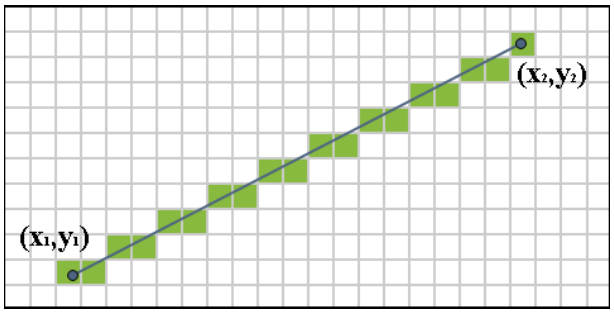


Figure 2.19: Illustration of the Bresenham straight line algorithm.

midpoint circle algorithm [31] [141] . The midpoint circle algorithm is an algorithm used to determine the points needed for rasterizing a circle. This algorithm consists in determining all eight octants simultaneously from each cardinal direction (0°, 90°, 180°,270°) extended in both ways to reach the nearest point of direction multiple of 45° (see Figure 2.20).

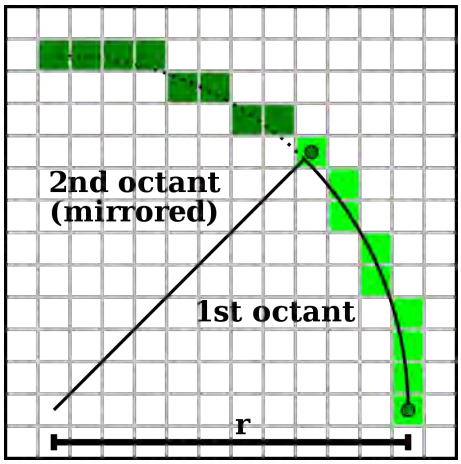


Figure 2.20: Illustration of the Bresenham's circle algorithm.

These algorithms are efficient but can be slow if the precision of the grid is very high. To reduce the computing time of this function, Quadtree (2D) and Octree (3D) have been used. Without loss of generality, this manuscript presents the free space checker in 2D to increase the reader comprehension. However, the method is used in 3D in our algorithm implementation. A quadtree is a tree data structure in which each internal node has exactly four children (eight for octrees). A quadtree is generated from a grid composed of free space cells and obstacle cells [11] (see Figure 2.21).

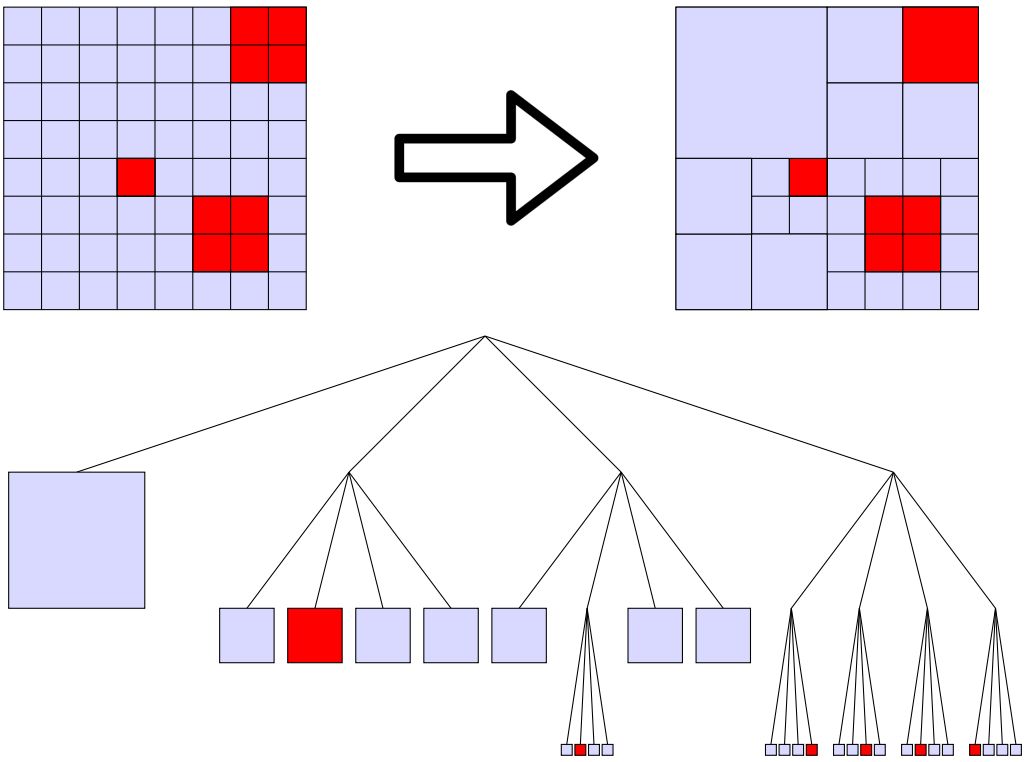


Figure 2.21: Example of a quadtree. From the grid containing obstacles in red, a data structure in the form of a tree in which each father has 4 sons. Cells in groups of four are grouped together into a larger cell if they are of the same type (free or obstacle), and the leaves of the tree are cut off. The level in the tree defines the size of the quadtree cell.

A naive method would be to code the Quadtree using pointers. The root node would point to four children nodes, each child would point to four children nodes, etc. However, if the grid is very large, memory usage is very high. To reduce memory usage, it is better to use a linear Quadtree [125]. This enables us to store for all leaves of the tree only two pieces of information for each of them. The first piece of information is called the Morton code [38, 77, 150]. This is an integer that uniquely defines a cell in a grid. The computation of its value consists in converting the row number and the column number of a cell into a binary string. Then, the Morton code is created by alternating a column digit and a row digit (see Figure 2.22a). The second piece of information is the depth level in the tree. This defines the position of the leaf in the tree and therefore also defines the cell size.

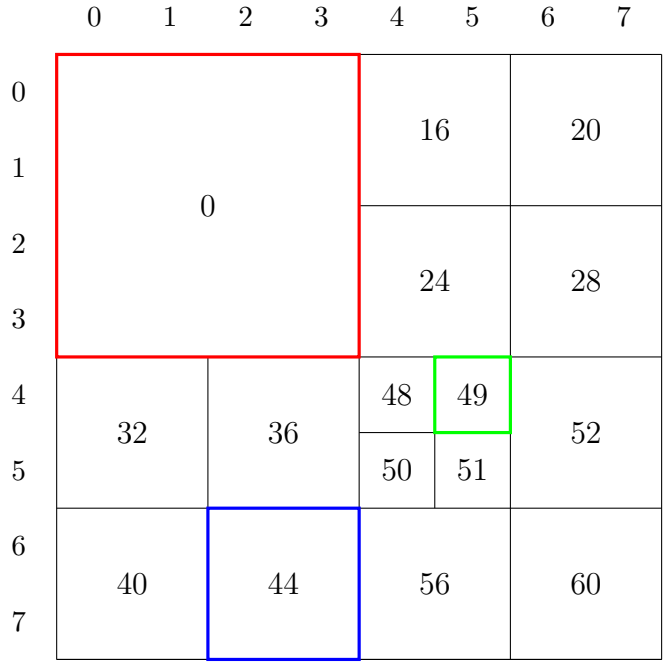
Linear Quadtree generation from a grid is composed of several steps. Firstly, the number of levels is computed. This value corresponds to the smallest integer nl such as grid.rows $\leq 2^{nl-1}$ and grid.columns $\leq 2^{nl-1}$ (3^{nl-1} for octrees). Then, an empty list is created, which contains the Quadtree free nodes. This list is ordered in ascending order of Morton Code. Then, for each Morton code MC $< 4^{nl-1}$ (8^{nl-1} for octrees), a new node is added to the list. Its Morton code is MC and its level is equal to $nl - 1$. At each step, as long as it is possible, the last four elements of the list are deleted and a new node is created. The level is decremented by one and its Morton code is the minimum Morton code of the previous nodes. If the grid is very large, the generation of this Quadtree is slightly long. However, once calculated, it is very easy to store it in a text file. This computation can be performed in a pre-processing phase. For example, during the flight the octree can be computed every 10 min around the aircraft position and its future positions (see Figure 2.23).

Figure 2.22b shows an example of a linear Quadtree. This Quadtree is built from an 8×8 grid. This implies that the level of the Quadtree is 4 ($8 = 2^{4-1}$). The level of the red cell is one because its size is 4×4 and its Morton is zero. The blue cell has a level equal to two (size $= 2 \times 2$). This cell is composed of four grid cells ([6,2], [6,3], [7,2], [7,3]). In the table shown in Figure 2.22, the four Morton codes are 44, 45, 46, and 47. The Morton code of the cell is the minimum of these; therefore, it is 44. The green cell is composed of only one cell ([5,4]), its level is three, and its Morton code is 49. Back to the free space checker function which verifies if a straight line segment is in the free space, the Quadtree offers a strong improvement as a free space checker. The first step of this function is to determine the Quadtree cells corresponding to the start and the end position of the straight line. The Morton code associated with a coordinate is computed with a table similar to that shown in Figure 2.22 (lines 1 and 2 in Algorithm 3). The associated cell is searched in the table of nodes (lines 3 and 4). If it is not an obstacle cell—the method returns False (line 5). Else (line AB in Figure 2.24), the function returns True (Line 6). However, if the two cells are different, a dichotomy is carried out and the function is called recursively (line CD in Figure 2.24, lines 9 and 10).

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

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

(a) Morton code computation: Its value is computed by alternating a column digit (blue) and a row digit (red) [56]



(b) Linear Quadtree example: This quadtree is computed from an 8×8 grid; it is therefore composed of 4 levels ($8 = 2^{4-1}$). The level of the red cell is 1 because its size is 4×4 and its Morton code is 0 because it is the minimum Morton code of cells that compose this big cell. The blue cell level is 2 (size $= 2 \times 2$) and the green cell level is 3 (size $= 1 \times 1$). Note that level 0 corresponds to the entire grid.

Figure 2.22: Linear quadtree generation from a grid.

Algorithm 2 Linear QuadTree Generation

MCTab $\leftarrow \emptyset$
levels $= 1$
while grid.columns $\geq 2^{\text{nl}-1}$ **and** grid.rows $\geq 2^{\text{nl}-1}$ **do**
  nl $=$ nl $+ 1$
end while
for `MC` $= 0$; `MC++`; `MC` $< 4^{nl-1}$ **do**
  coordinate $=$ `MortonCodeToCoordinate`(MC)
  if grid.`IsAvailable`(coordinate) **then**
    MCTab $\leftarrow$ (MC, nl $- 1$)
    while MCTab.`size` ≥ 4 **do**
      $i =$ MCTab.`size`
      $n_i =$ MCTab.`get`(i)
      $n_{i-1} =$ MCTab.`get`$(i-1)$
      $n_{i-2} =$ MCTab.`get`$(i-2)$
      $n_{i-3} =$ MCTab.`get`$(i-3)$
      if n_i.level $= n_{i-1}$.level **and** n_i.level $= n_{i-2}$.level **and** n_i.level $= n_{i-3}$.level **then**
        if n_i.MC $= n_{i-1}$.MC **and** n_i.MC $= n_{i-2}$.MC **and** n_i.MC $= n_{i-3}$.MC **then**
          MCTab $\leftarrow \{n_i, n_{i-1}, n_{i-2}, n_{i-3}\}$
          MCTab $\leftarrow$ `Node`$(n_{i-3}$.MC, n_{i-3}.level $- 1)$
        else
          break
        end if
      else
        break
      end if
    end while
  end if
end for
return `QuadTree`(MCTab, levels)

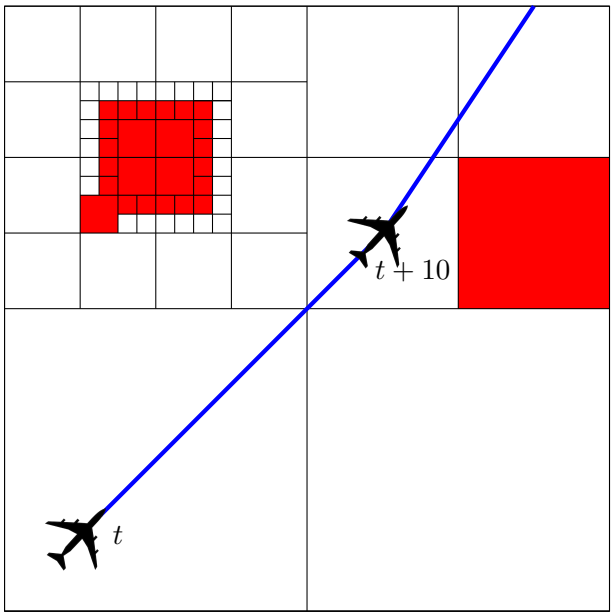


Figure 2.23: QuadTree around the aircraft position.

2.3.5 Results

2.3.5.1 Description of the test method

Preliminary tests were conducted to demonstrate the efficiency of the proposed algorithm. The goal was to show that the proposed improvements decrease the computing time required for

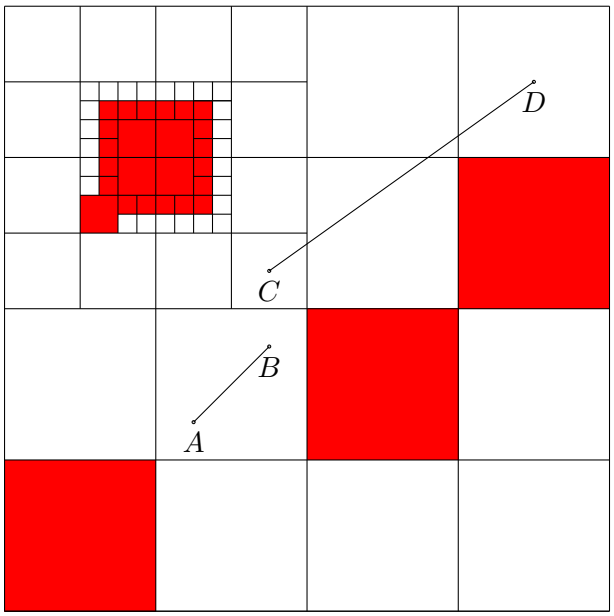


Figure 2.24: Free space checker example: in the case of the segment [AB], the octree cell is the same for A and B. However, for the segment [CD], the start cell is different from the end cell; therefore, a dichotomy is performed.

Algorithm 3 Free space checker: from the start position (*start*) and the end position (*end*), the two associated Morton codes are computed and then a dichotomy is performed to check that the segment is in the free space.

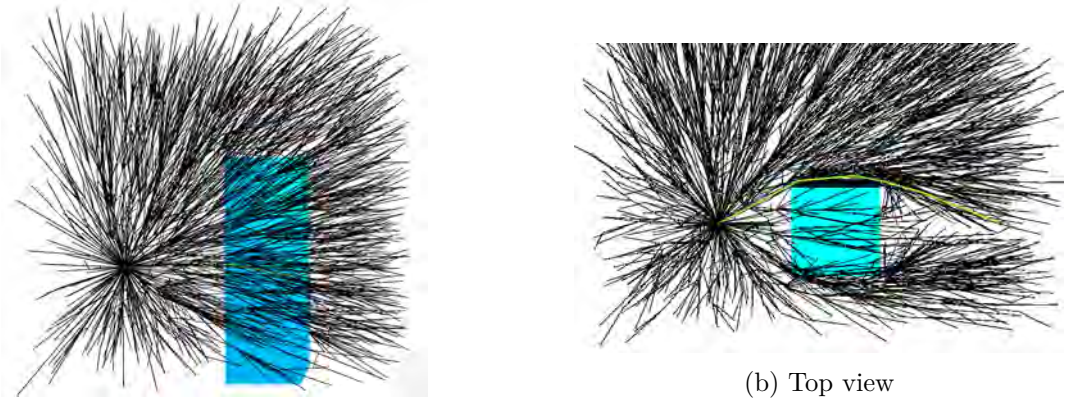
```
1: StartMortonCode = CoordinateToMC(start)
2: EndMortonCode = CoordinateToMC(end)
3: StartNode = MCTab.get(startMortonCode)
4: EndNode = MCTab.get(endMortonCode)
5: if StartNode ≠ NULL and EndNode ≠ NULL then
6:    if StartNode = EndNode then
7:       return True
8:    else
9:       middle = middleCoordinate(start,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

trajectory generation. The methodology proposed to compare FMT* and our algorithm is illustrated by the cases of one flight crossing a 3D cube of $1000 \times 1000 \times 1000$. For these tests, the orientations, the start point, and the final point are given. The obstacles are very simple in order to easily observe the cost of the shortest path. In these early simulation tests, the Dubins curves are not computed and the descent constraints are not considered. The experiments were carried out with a computer equipped on an i7-8550 processor and a RAM of 8 GB. The algorithm was tested on a dozen of simple scenarios with obstacles of several sizes to evaluate its computing-time efficiency. These scenarios were chosen because the optimal cost was known; therefore, the error was easily computed as: $error = \frac{c_{FMT} - c_{real}}{c_{real}}$, where c_{FMT} represents the

cost of the path computed by the algorithm and c_{real} is the cost of the optimal path. The first case corresponds to two points (the origin and destination) separated by a large obstacle. This example shows the difference between the FMT* tree and the tree of our algorithm. It illustrates the reduction in computation time. Then, we present a more complex scenario to show the avoidance of the obstacles. This scenario raises the limits of the algorithm if the sampling size is too small. After testing the algorithm on simple cases, it is tested on real cases to show that the proposed algorithm can take into account aeronautical constraints.

2.3.5.2 Simple case test

Figure 2.25 presents the results obtained with the FMT* algorithm. These two figures show that the FMT algorithm explored many irrelevant points. This implies that the number of points should be increased in order to compute a path that has a cost near the optimal cost. In this test case, with 3000 samples, the computing time is 589.6 ms and with an error was 2%. Figure 2.26 presents the tree and the optimal path computed by the proposed algorithm (improved FMT*). These figures show that the tree is less extended than the tree computed by the original FMT*. This enables us to reduce significantly the number of samples and therefore the path-computing time. Indeed, for the same error rate (2%) the computing time is 208.6 ms.



(a) Side view

(b) Top view

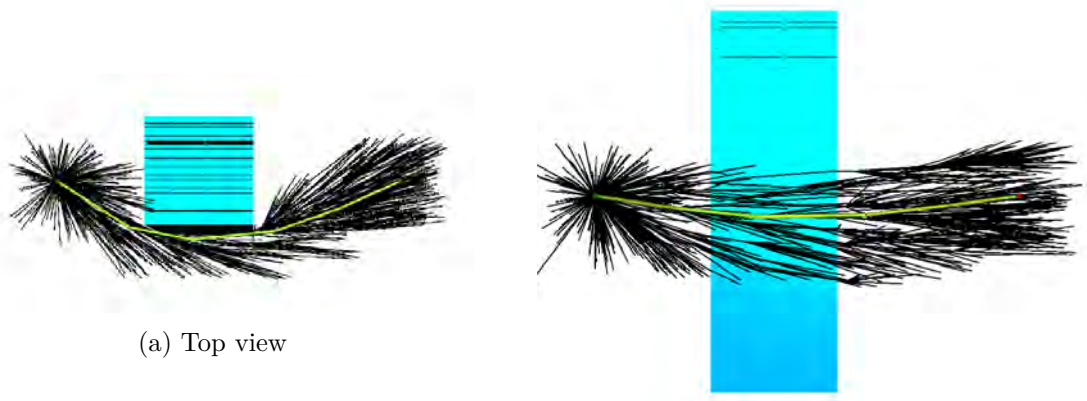
Figure 2.25: Example of the result of the FMT algorithm with one obstacle: number of samples = 3000, computing time = 589.6 ms, error = 2%. The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green. The computed trajectory passes behind the obstacle.

2.3.5.3 Multi-obstacles case test

Figure 2.27 shows another result obtained using the proposed algorithm in the presence of four different obstacles. This test was performed with a first step composed of 3000 samples and then a second step with 1500 points. This figure shows that the path avoids the obstacles horizontally and vertically.

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

Figure 2.28 shows an example of a result obtained using the algorithm with a small number



(a) Top view

(b) Side view

Figure 2.26: Example result of the improved FMT algorithm with large obstacle: first step number of samples = 1500, second step number of samples = 1000, computing time = 208.6 ms, error = 2%.The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green.

of samples for the first step (500 samples). The path is very far from the path in Figure 2.27. Indeed, it is 8% longer. To avoid this problem, the first sampling should have a higher number of samples than in the second sampling step. Indeed, the second sampling step is just used to refine the first solution. In the worst case, the new solution is the same as the first.

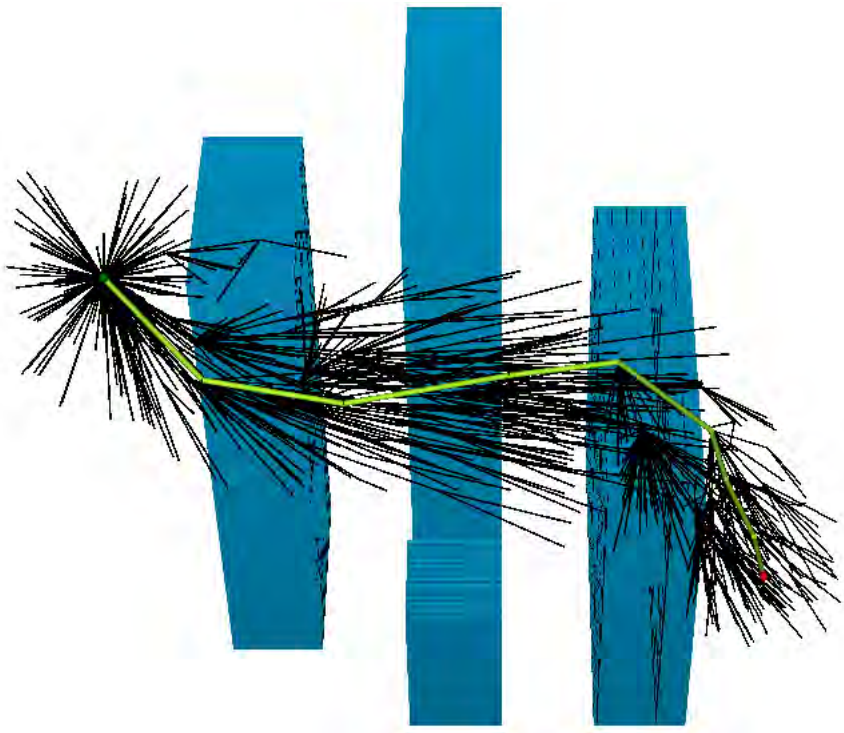


Figure 2.27: Example of the result of the improved FMT* algorithm with four different obstacles: first step number of samples = 3000, second step number of samples = 1500, computing time = 1123.6 ms. The generated graph is drawn in black, the obstacles in blue and the path (start point in green and end point in red) in green.

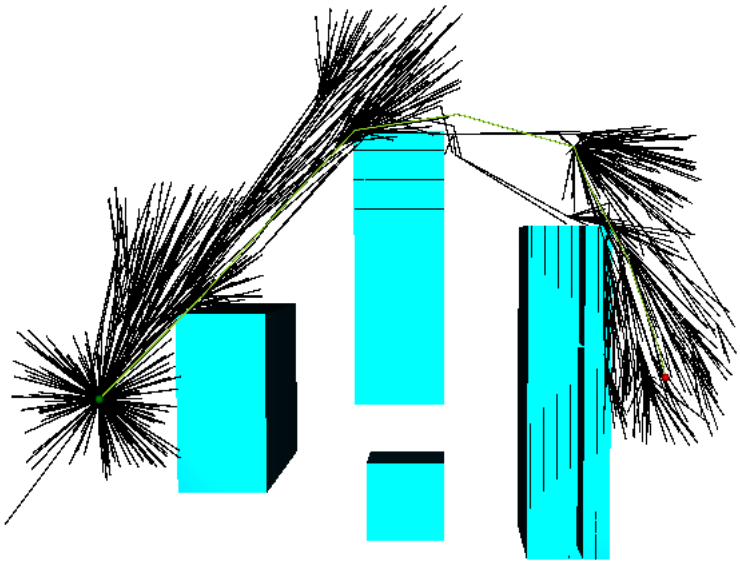


Figure 2.28: Example result of the improved FMT* algorithm with four different obstacles: first step: number of samples = 500, second step: number of samples = 1500. The generated graph is drawn in black, the obstacle in blue, and the path (start point in green and end point in red) in green.

2.3.5.4 Computing time results

Other tests were performed to demonstrate the speed of the proposed algorithm. The test conditions were presented in Section 2.3.5.1. Table 2.2 summarizes the results of the simulations. It shows the computing time, depending on the error. These tests show first that the proposed FMT* was clearly faster than the original FMT*. Indeed, the proposed algorithm reduced the computing time by three.

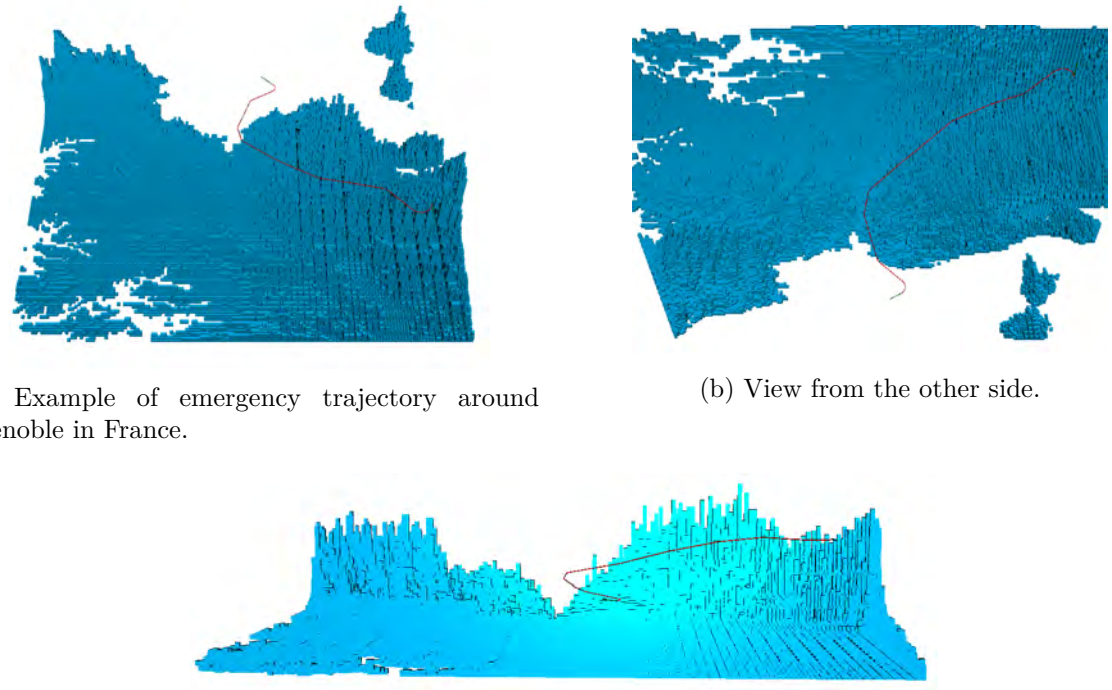
Error (%)	FMT*	Proposed FMT*
5	98.7	51.3
2	493.8	203.4
1	6896.5	2057.3

Table 2.2: Average computing time (ms) for scenarios between two points separated by a distance of 500 cells.

2.3.5.5 Real case test

After testing the algorithm on simple cases, it was tested on real cases. The data used for these tests were obtained from an altitude data map around Grenoble airport in France with a radius of 50 NM. The data were represented by a 3D cube of $1200 \times 1200 \times 500$. Altitude data are given at each 0.083 NM $\left(\frac{2 \times 50}{1200}\right)$. The precision of the altitude was 30 ft. After a data processing step to enlarge the obstacles, the computed octree had about 6 million leaves and the file size was equal to 80 MB. The number of cells was reduced by 120 thanks to the octree trick. Indeed, the initial cube contained 720 million cells ($1200 \times 1200 \times 500$). The absence of obstacles at high altitudes explains this significant reduction in the number of cells. They were therefore easily grouped

to create large free-space cells. As explained in the last part, this speeds up the free space checker algorithm, and consequently, the trajectory generation algorithm. As previously stated, the landing site, the curvature radius and descent rates were considered known. Figure 2.29 represents one studied scenario. The initial altitude is 10,000 ft, the initial heading is 180°, the final altitude is the ground altitude and the final heading is 315°. These figures show that the trajectory avoids the mountains near to the emergency position and seems smooth with the addition of Dubins curves. However, this smoothing significantly increases the computing time required for the generation of a trajectory but it was still reasonable. The average computing time is less than 10 seconds.



(a) Example of emergency trajectory around Grenoble in France.

(b) View from the other side.

(c) Other view of the trajectory to show the avoidance of the obstacle.

Figure 2.29: Trajectory generation example: The computed path is represented in red and the obstacles in blue.

2.3.6 Conclusion

This section has presented an adapted and accelerated version of the Fast marching Tree algorithm to design emergency trajectory. Simulation results show that the proposed method works well. Indeed, it computes a trajectory from the emergency location to the landing site in less than 10 seconds. Moreover, the proposed method is adaptable to any type of emergency. Indeed, the algorithm can take as input any cone of descent angle and radius of curvature. The next section presents a new approach based on Fast Marching methods.

2.4 Fast Marching Approach

The second studied approach is based on Fast Marching methods. These methods were first introduced by James Sethian in [179] for solving boundary value problems of the Eikonal Equation. They are based on the physical propagation of a wavefront in a given domain Ω, containing

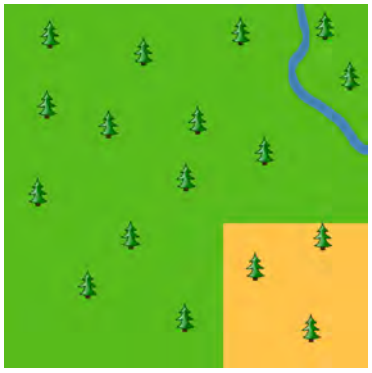
obstacles. The front $\delta\Omega$ fixes the minimum cost to reach any point in space from the source point. To propagate the front, the Eikonal equation has to be solved:

$$|\nabla T| = f, \tag{2.33}$$

where T is the cost function 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.

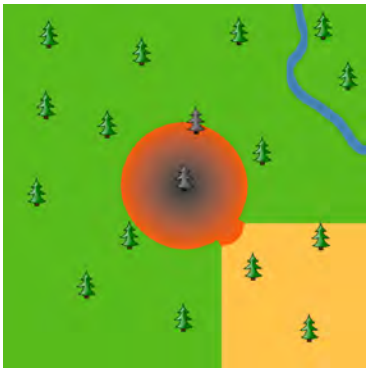
An analogy can be done with a forest fire, where some areas are moist (the fire propagation is slow) and others are dry (the fire propagation is fast). A forest fire simulation with the Fast Marching algorithm is presented in Figure 2.30.

In this example, the fire starts to spread uniformly from a single tree, making circles. Once the fire reaches dry land (orange area), it propagates faster in this direction, modifying the *level curves* or in other words, the shape of the propagation. Once the front reaches the water (in blue), which can be assimilated to an *obstacle*, the propagation stops in this direction. The propagation completely stops when all the available free space has been visited, meaning when there is no more grass to burn.

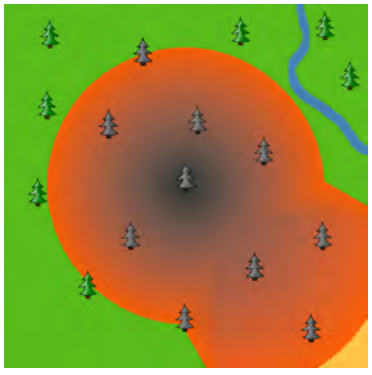


(a) Forest fire propagation - Step 1.

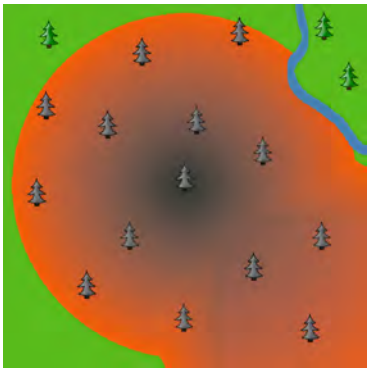
(b) Forest fire propagation - Step 2.



(c) Forest fire propagation - Step 3.



(d) Forest fire propagation - Step 4.



(e) Forest fire propagation - Step 5.

(f) Forest fire propagation - Step 6.

Figure 2.30: Forest fire propagation. The fire, the forest, the dry land and water are represented in red, green, orange and blue respectively.

Solving the Eikonal equation requires discretizing the domain into a mesh. All the points of the mesh are nodes. At each iteration, they can be in three states:

- *Far*: nodes not yet visited;
- *Considered*: nodes for which T has already been computed, and its value is not definitely fixed;
- *Accepted*: nodes for which T has already been computed, and its value is definitely fixed.

The Fast Marching algorithm can be described by the following steps:

1. Set the start node cost to 0 and it is *Considered*. All other nodes are *Far* and their costs are equal to ∞. Mark the start node as the *current node*.
2. Solve the Eikonal equation for all neighbors of the current node. They are now *Considered* and the current node is *Accepted*
3. Set the lowest cost node as *current node*.
4. If there are still *Considered* nodes, or the end node has not yet been reached, repeat Step 2.

This section proposes to compare 2D and 3D Fast Marching methods on simplex structures applied to emergency trajectory design. In geometry, a simplex is a generalization of the notion of a triangle or tetrahedron to arbitrary dimensions.

2.4.1 Fast Marching in 2D

First, we consider in 2D Fast Marching methods, although, the studied problem is in 3D. We simplify the problem by defining a descent plan to apply 2D Fast Marching methods. This choice is guided by the fact that the Fast Marching in 2D is faster than in 3D due to a smaller search space.

2.4.1.1 Grid Discretization

As explained in the introduction of this section, it is necessary to discretize the space to solve the Eikonal equation. A first way to discretize a 2D space is to use a constant step grid. Let (i, j) be the row and the column of the grid map (Figure 2.31), that corresponds to a position in the search space. Let $F_{i,j}$ be the "slowness" at the gridpoint (i, j).

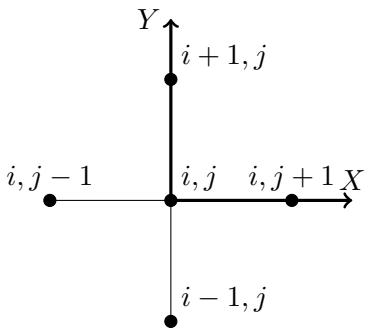


Figure 2.31: Grid discretization.

In the case of constant step grids, the discretization of the gradient ΔT according to Sethian [180] drives to the following equation:

$$\max(D_{ij}^{-x}T, 0)^2 + \min(D_{ij}^{+x}T, 0)^2 + \max(D_{ij}^{-y}T, 0)^2 + \min(D_{ij}^{+y}T, 0)^2 = F_{ij}^2, \tag{2.34}$$

where:

$$\begin{cases} D_{ij}^{-x} = \frac{T_{i,j} - T_{i-1,j}}{\Delta x} \\ D_{ij}^{+x} = \frac{T_{i+1,j} - T_{i,j}}{\Delta x} \\ D_{ij}^{-y} = \frac{T_{i,j} - T_{i,j-1}}{\Delta y} \\ D_{ij}^{+y} = \frac{T_{i,j+1} - T_{i,j}}{\Delta y} \end{cases}, \tag{2.35}$$

and Δx and Δy are the grid steps in the x and y directions respectively. The Eikonal equation can be rewritten as:

$$max\left(\frac{T - T_1}{\Delta x}, 0\right)^2 + max\left(\frac{T - T_2}{\Delta y}, 0\right)^2 = F_{i,j}^2, \tag{2.36}$$

where:

$$\begin{cases} T = T_{i,j} \\ T_1 = min(T_{i-1,j}, T_{i+1,j}) \\ T_2 = min(T_{i,j-1}, T_{i,j+1}) \end{cases}.$$

The front is supposed to be positive, therefore if the front wave has not already passed over the grid point (i, j), T has to be greater than T_1 and T_2. The previous equation can be written as follows:

$$\left(\frac{T - T_1}{\Delta x}\right)^2 + \left(\frac{T - T_2}{\Delta y}\right)^2 = F_{i,j}^2. \tag{2.37}$$

Once the propagation is performed and the destination is reached, the shortest path can be extracted through back propagation of the gradient from P_{end} to P_{start}, by solving

$$X(t) = -\nabla T \qquad \text{given } X(0) = P_{end}. \tag{2.38}$$

A simulation example is given in Figure 2.32, with two obstacles and where the propagation on the left side of the domain is two times slower than the propagation on the right. 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 vector which is orthogonal to the level curves.

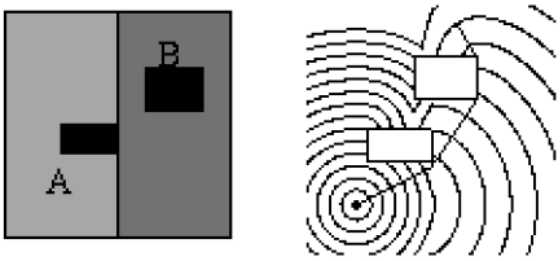


Figure 2.32: Two-dimensional navigation with constraints on variable domain [180, Fig. 21].

The Fast Marching propagation depends on the chosen discretization of the domain Ω. First, the propagation on a grid is presented, then it is compared to a propgation on a triangular mesh. Note that, in both cases, in the back propagation phase, the optimal trajectory (geodesic) is built in a pseudo-continuous manner without following the grid or mesh points.

Propagation over a grid Let A, B and C be three 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 2.33. In this figure, 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 (See Algorithm 4).

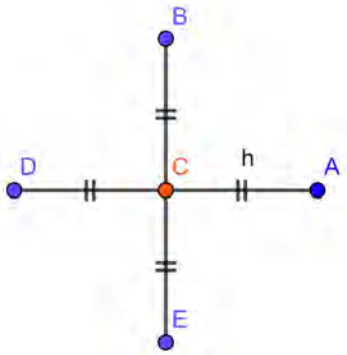


Figure 2.33: Update configuration on a gridpoint. C is the grid vertice to be updated which is neighbor of the grid vertices A, B, D and E.

Algorithm 4 Update on a Gridpoint Procedure

$T_1 \leftarrow$ 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 $T_2 \leq T(B)$
    $T(C) = T_2$
end if

2.4.1.2 Triangular Mesh

Another way to discretize the space is to generate a triangular mesh. To build such a mesh, we propose to first generate a balanced quadtree and then build the triangular mesh thanks to Delaunay triangulation.

As explained in Section 2.3.4.2, thanks to terrain data, a linear quadtree can be generated. It is possible to directly build the mesh by using a Delaunay triangulation. However, the Delaunay triangulation can create triangle with very acute angle due to the high size difference between two neighboring cells (See Figure 2.34). This kind of triangulation can create signifcant errors. That is why, it is necessary to balance the quadtree to reduce the size difference between neighbors.

Balancing a quadtree consists in ensuring a level difference of at most 1 between two neighboring leaves. In Figure 2.35c, the only leaf that needs to be divided is leaf (32, 1). The balancing operation results in removing this leaf and building four new leaf nodes (32, 2), (36, 2), (40, 2) and (44, 2) (see Figure 2.35d). In Figure 2.35c, leaves (10, 3) and (32, 1) had a difference level

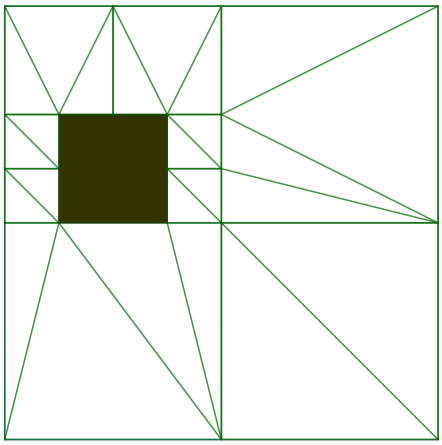


Figure 2.34: Delaunay triangulation composed of triangles with very acute angles.

of $3 - 1 = 2$. After balancing, the level difference between (10, 3) and (32, 2) is 1. The general algorithm to balance a quadtree is given in [184].

In order to quickly transform quadtree leaves from a grid into mesh cells (here, triangles), neighboring information is added to those stored in the linear quadtree. The idea is to create a mesh vertex whenever there is a level difference of 1 between two leaves. Thus, one is able to 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* [5], where each leaf is characterized by its Morton code MC, its level l and its level differences δ (see Figure 2.35e). Considering domain sizes, it can easily assume that MC is stored in a 4-byte signed integer and l in a byte.

To reduce as much as possible the memory space, δ can be stored in a byte. Indeed, after balancing the quadtree, the only possible values for a level difference are:

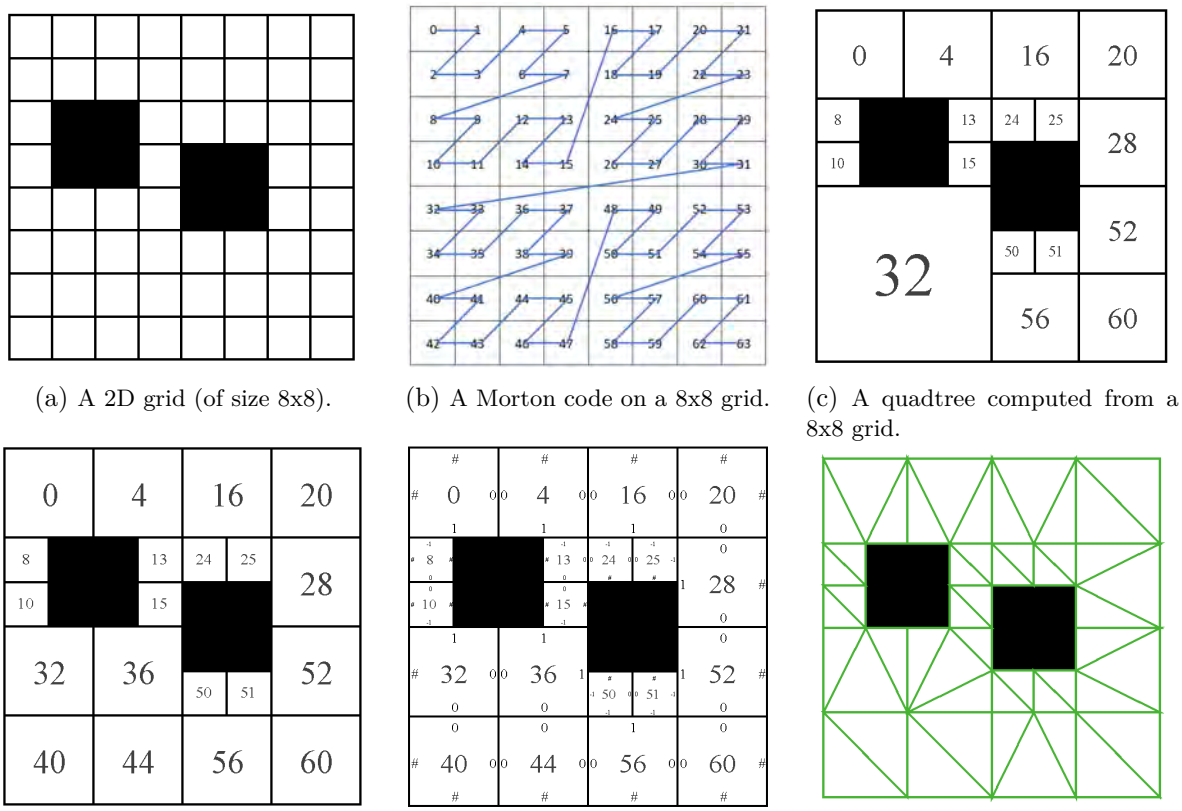
- '-1' representing a neighbor with a lower level (thus a larger neighbor),
- '0' representing a neighbor with the same level,
- '1' representing a neighbor with a higher level (thus a smaller neighbor),
- '#' representing a neighboring obstacle or grid limits.

Therefore, each level difference is coded with 2 bits, or 8 bits for the four cardinal directions. For instance, Figure 2.36 represents a quadtree node with a smaller neighbor at North, a larger neighbor at East, and grid limits at West.

Finally, a Delaunay triangulation is done [47]. This meshing has been proven adequate over balanced quadtrees in [171].

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, the edges to add for building the mesh are directly given by the Delaunay triangulation scheme (see Figure 2.37), in order to build two or more triangles inside this leaf. The building of the mesh is shown in Figure 2.35f. Hence, the triangulation is built over the balanced quadtree thanks to the level differences.

After building the triangular mesh, the propagation process must be defined. A method similar



(a) A 2D grid (of size 8x8).

(b) A Morton code on a 8x8 grid.

(c) A quadtree computed from a 8x8 grid.

(d) A balanced quadtree.

(e) A balanced quadtree with level differences.

(f) A triangular mesh adapted to the terrain.

Figure 2.35: Steps for building a Data structure from a grid.

Figure 2.36: Example of δ value for a quadtree node.

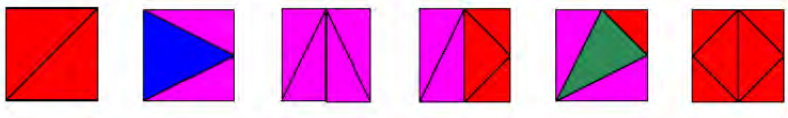


Figure 2.37: Triangles Generation. The sizes of the neighbors define the triangulation into a cell. For instance the second one corresponds to the case where the neighbor on the right is smaller.

to the one on the grids can be applied. However, neighbors are no longer located on a regular grid but are found on the vertices of irregular triangles.

Consider an acute triangle ABC with $T(A) \neq \infty$, *i.e.* A has already been updated. Such a configuration is represented in Figure 2.38. Once again 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 $t = T(C) - T(A)$.

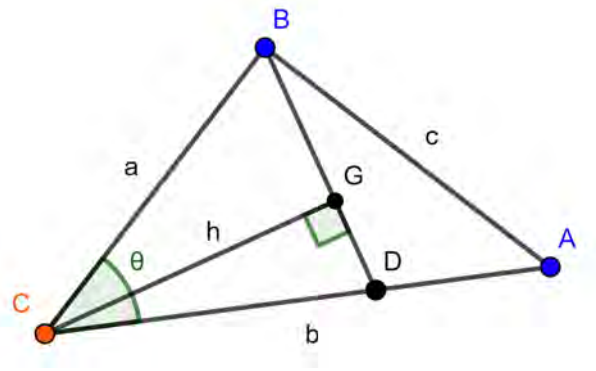


Figure 2.38: Update configuration on a meshpoint.

Two cases are studied:

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

The first case is trivial because the update comes from the edges: $T(B) = \min\left(T(B),\ cf_B + T(A)\right)$ and $T(C) = \min\left(T(C),\ bf_C + T(A)\right)$. The latter is more complex, please refer to the work of Kimmel and Sethian [120] for detailed calculations. In this case, the quadratic equation is written as follows:

$$\begin{aligned}(a^2 + b^2 - 2ab\cos\theta)t^2 + 2bu(a\cos(\theta) - b)t \\ + b^2(u^2 - F^2a^2\sin^2(\theta)) = 0.\end{aligned} \tag{2.39}$$

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

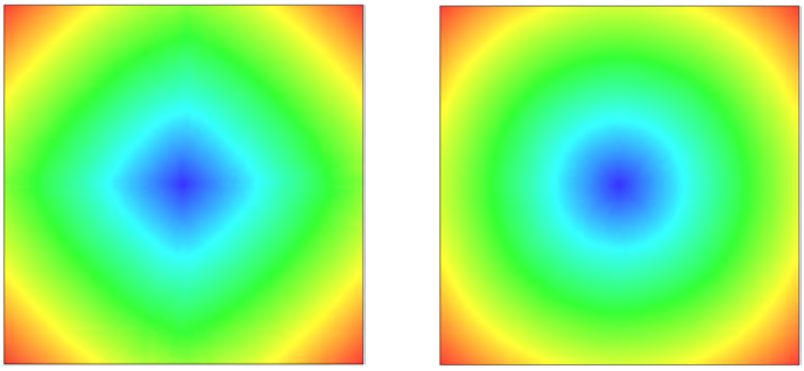
Algorithm 5 Update on a Meshpoint Procedure

1: **if** $u < t$ & $a\cos\theta < b(t-u)/t < a/\cos\theta$ **then**
2: $\quad T(C) = \min\left(T(C),\ t + T(A)\right)$
3: **else**
4: $\quad T(C) = \min\left(T(C),\ bf_C + T(A),\ af_C + T(B)\right)$
5: **end if**

Figure 2.39 shows the propagation of costs using the two update procedures. This figure shows that the Fast Marching method on a triangular mesh is more accurate than on a grid because costs are isotropically propagated, *i.e.* costs do not depend on any direction of propagation. Hence, since grid bias is more powerful on a grid than on a mesh, the method over a triangular mesh is the only one that is considered for the algorithm.

The final path is found thanks to the back propagation of gradients, from *EndCoordinate* to *StartCoordinate*. This paragraph supposes that a triangular mesh has been built (Note: the study over a simple grid follows the same ideas as described below).

The choice of the mesh finds its importance with gradients back propagation. 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 choice for these considerations: it ensures a minimum angle of 26.565° within each triangle [171], and it has the minimum number of triangles inside each node that connect all of the mesh vertices.



(a) Isocosts on a grid. (b) Isocosts on a triangular mesh.

Figure 2.39: Fast Marching fronts on a grid (a) and on a triangular mesh (b).

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, 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, the gradient must be interpolated. It can easily be done by computing the barycenter of the gradients linked with the two vertices of the considered edge (see Figure 2.40).

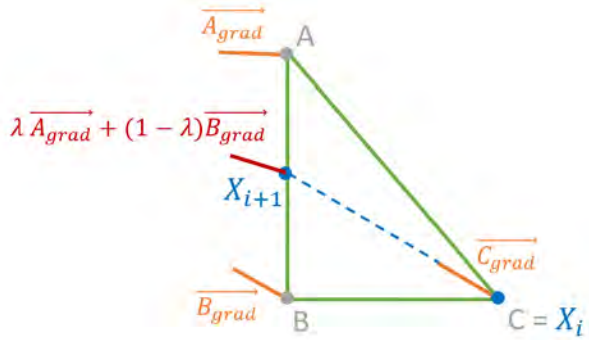


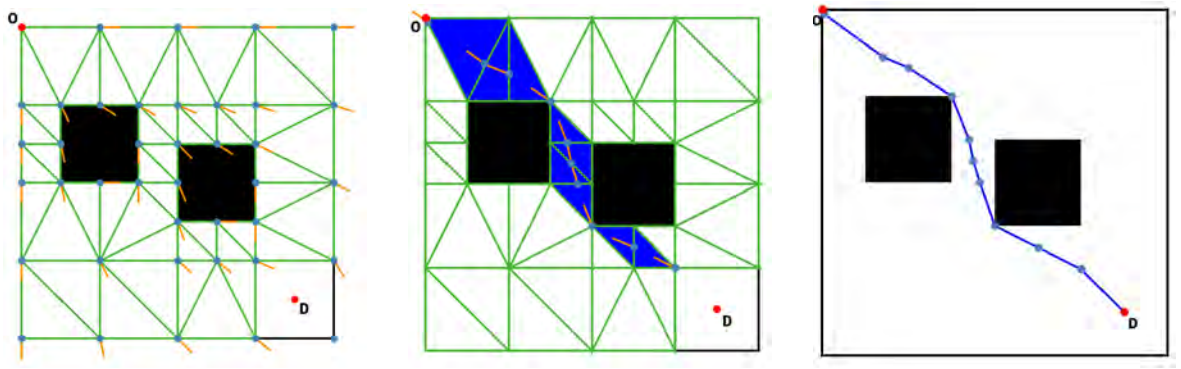
Figure 2.40: Back propagation of gradients on a triangle.

Figure 2.40 shows the 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 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 $\overrightarrow{C_{grad}}$ and the edge AB, hence the gradient has to be interpolated at that point. The interpolation parameter is $\lambda = \frac{||\overrightarrow{X_{i+1}B}||}{||\overrightarrow{AB}||}$, *i.e.* the ratio between the distance from X_{i+1} to B and the length of the edge AB. Thus, the gradient at X_{i+1} is $\lambda\overrightarrow{A_{grad}} + (1-\lambda)\overrightarrow{B_{grad}}$.

It is easy to show the consistency of the interpolation. If $X_{i+1} = A$, then $\lambda = 1$ and the gradient at X_{i+1} is $\overrightarrow{A_{grad}}$. Conversely, if $X_{i+1} = B$, then $\lambda = 0$ and the gradient at X_{i+1} is $\overrightarrow{B_{grad}}$.

The algorithm stops when it reaches *StartCoordinate*, and the final trajectory between *StartCoordinate* and *EndCoordinate* is found by linking the edge intersection points together. The forward propagation, the back propagation, and the trajectory generation phases are all illustrated with an example in the next paragraph.

The propagation of costs and gradients through the mesh occurs between *StartCoordinate* and *EndCoordinate*. Once visited, a mesh vertex is associated with a cost and a gradient according to Kimmel and Sethian's work [120]. The propagation stops when the quadtree leaf node containing the *EndCoordinate* is reached, as shown in Figure 2.41a. In this figure, *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.



(a) Propagation phase. (b) Back propagation phase. (c) The computed trajectory.

Figure 2.41: Optimal trajectory from O to D generated by the Fast Marching algorithm applied on a triangular mesh.

Once the propagation phase is over, the algorithm is ready to execute the back propagation phase. It mainly consists in interpolating gradients throughout the mesh, and taking care of special cases where the trajectory hits an obstacle. The back propagation phase is shown in Figure 2.41b, 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 Figure 2.41c.

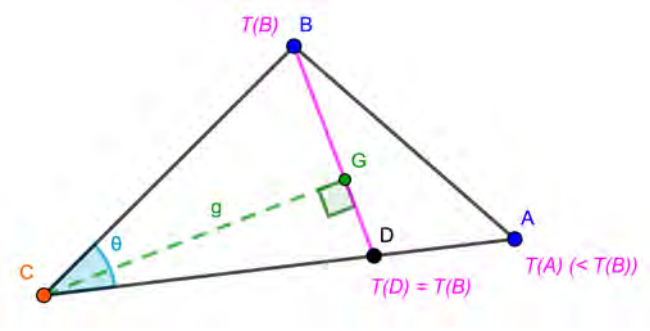
2.4.1.3 Wind

To improve the method, it is necessary to consider the wind. In Figure 2.42a, A, B, and C are some of the triangular mesh vertices. $\overrightarrow{GC}$ is the gradient associated with vertex C. This gradient is the direction of minimal cost, *i.e.* it is the direction that the aircraft follows as soon as it enters the triangle. Thereby, the gradient $\overrightarrow{GC}$ and the aircraft ground speed $\overrightarrow{GS}$ are colinear. This observation is crucial in order to develop a procedure for taking into account the wind. Also, note that the ground speed and the true airspeed $\overrightarrow{TAS}$ are colinear when there is no wind field. Thus, in absence of wind, the direction of minimal cost inside a triangle is the direction of the aircraft true airspeed. To consider the wind, the speed triangle has to be built. It is defined by the following equation:

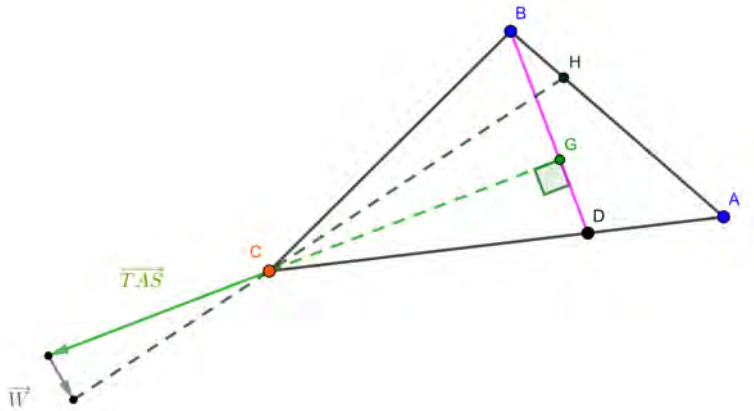
$$\overrightarrow{TAS} + \overrightarrow{W} = \overrightarrow{GS}. \tag{2.40}$$

Hence, to find the new direction of minimal cost in a wind field, this direction is rotated to match the direction of the ground speed $\overrightarrow{GS}$. This operation is shown in Figure 2.42b.

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



(a) Update on a triangle.



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

Figure 2.42: Wind consideration in the update process. The direction of optimal cost is modified thanks to the wind triangle.

vertex A and vertex B. The interpolation is formalized as follows:

$$T(H) = \lambda T(A) + (1-\lambda)T(B)\,, \text{ with } \lambda = \frac{||\overrightarrow{HB}||}{||\overrightarrow{AB}||}. \tag{2.41}$$

The interpolation procedure is shown in Figure 2.43. Finally, the cost at C is given by the

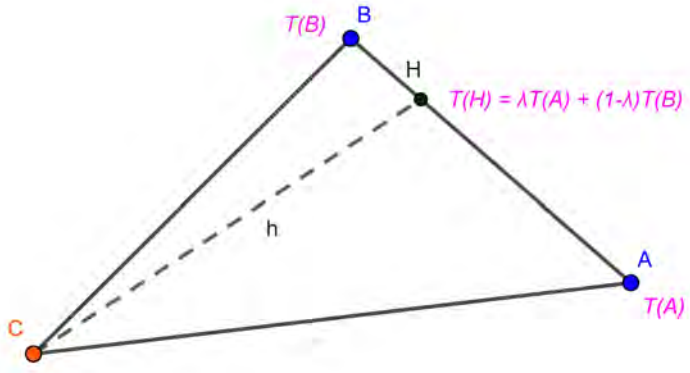


Figure 2.43: Update on a triangle with the wind.

following formula:

$$T(C) = T(H) + h/V, \tag{2.42}$$

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

To check the accuracy of the new procedure, the algorithm is tested on Zermelo's problem, described in the next section. In mathematical optimization, Zermelo's navigation problem, proposed by Ernst Zermelo in [203], is a classic optimal control problem that deals with a boat navigating on a body of water, originating from a point A to a destination point B. The boat sails at a given speed, and the goal is to derive the best possible control to reach B in the least possible time. The problem remains unchanged when considering an aircraft surrounded by air.

Although it is usually impossible to find an exact solution to this problem in most cases, the general case was solved by Zermelo himself in the form of a partial differential equation, known as Zermelo's equation, which can be numerically solved. Solutions without wind (or with constant wind) and with a wind gradient are shown in Figure 2.44.

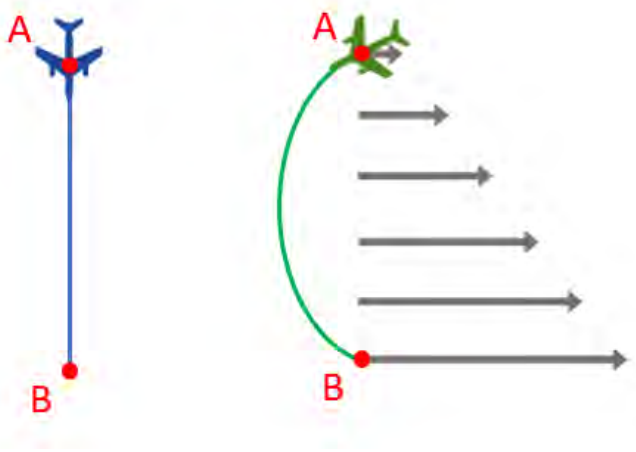


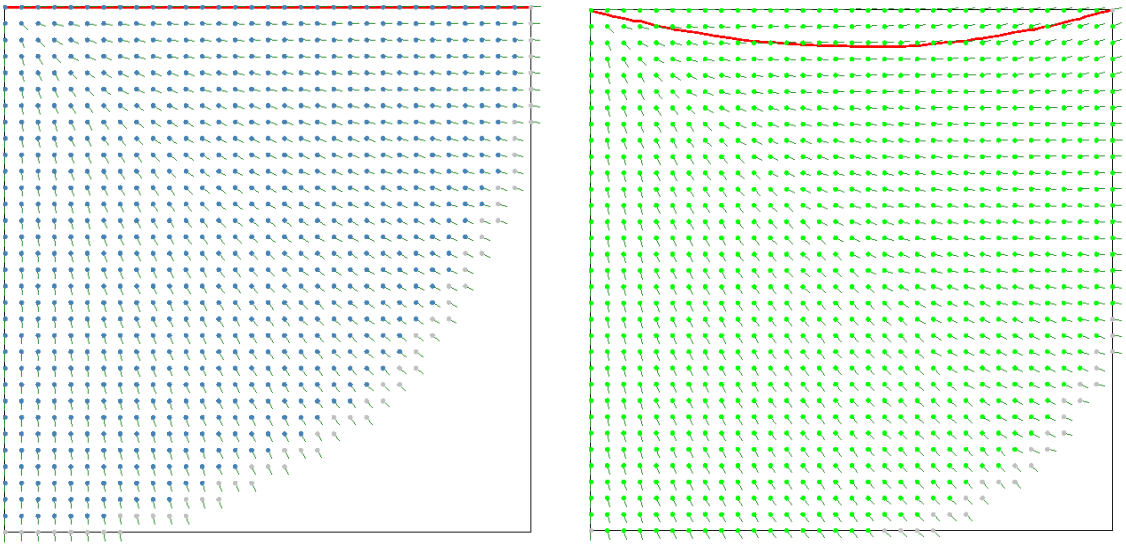
Figure 2.44: Solutions to Zermelo's problem with constant wind (left) and with linearly variable wind (right).

In the first case, the optimal trajectory is a straight line. This result is very intuitive, without wind or even with constant wind, the optimal direction remains the straight line: changing direction would waste time and lengthen the trajectory.

The second case is trickier to analyze. The straight line is no longer the shortest path. Instead, the aircraft has to go upstream near A at the beginning of the trajectory, losing time, only to regain this time going downstream when approaching B. As it is a wind gradient, it is more favorable to go upstream at the beginning and downstream at the end than to make the straight line. Thus, the optimal trajectory is curved.

The algorithm works without wind and computes the straight line. To test its operation in windy condition, the algorithm is tested on Zermelo's problem. Results are shown in Figure 2.45 with a 64×64 grid, trying to link the up-left corner to the up-right corner in a wind field that is 0 at the top of the figure, and 67% of the aircraft speed at the bottom of the figure.

With the same gradient, the algorithm can also be tested in the presence of obstacles. Results are shown in Figure 2.46. This figure shows that the optimal trajectory benefits from strong winds and reaches the arrival faster than in the "no wind" scenario. Thus, the improvement of the algorithm by taking into account the wind is validated.



(a) Without taking the wind into account. (b) By taking the wind into account.

Figure 2.45: Optimal trajectory generation in a wind field with the Fast Marching algorithm.

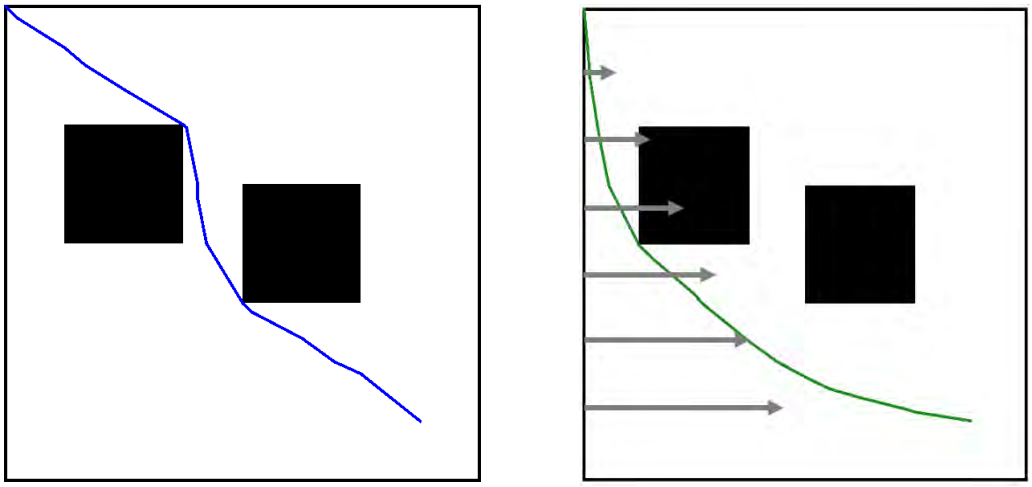


Figure 2.46: Optimal trajectory generation with obstacles in a wind field with the Fast Marching algorithm. On the left, the trajectory finds without taking the wind into account. On the right, the trajectory with a wind from the west and gets stronger the further south you go.

2.4.1.4 Path generation process

After presenting the 2D Fast Marching method and its adaptation to the wind constraint, this subsection presents the process for generating the emergency path. The process is the following:

- A descent plan is defined from the start position and the final point. Figure 2.47 shows an example of descent plan in the mountainous region of Grenoble (France),
- A balanced quadtree is then built,
- Delaunay triangulation is done to generate the mesh,
- 2D Fast Marching method is then applied to find the fastest path,

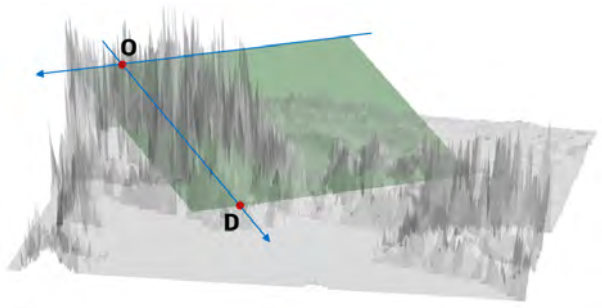


Figure 2.47: A glide slope in the mountainous region of Grenoble (France) computed thanks to O and D.

.

2.4.1.5 Results

This subsection presents the results given by the algorithm based on 2D Fast Marching method. The method is tested in the case of ASAP and ANSA emergencies in the mountainous region of Grenoble in France. For these scenarios, we are particularly interested in the computation time, the memory space and the flyability of the path. The data is a 12000×12000 grid containing the elevation of the terrain in a 50NM radius around Grenoble.

As Soon As Possible emergencies (ASAP) Results for the ASAP scenario are shown in Figure 2.48. In this figure, *StartCoordinate* is on the left, *EndCoordinate* on the right. The map corresponds to the glide slope of Figure 2.47.

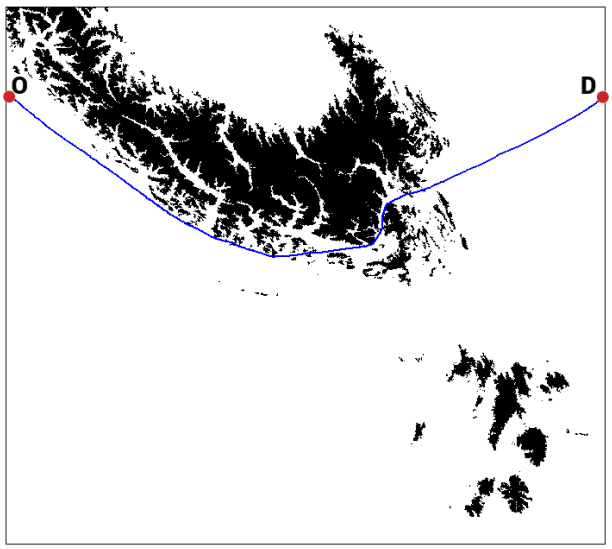


Figure 2.48: 2D trajectory generated in an ASAP scenario.

The blue trajectory goes through the mountains, in order to minimize its length as expected. 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, it is preferable to have a safe path, like in an ANSA scenario. Figure 2.49 shows the triangles generated during the simulation.

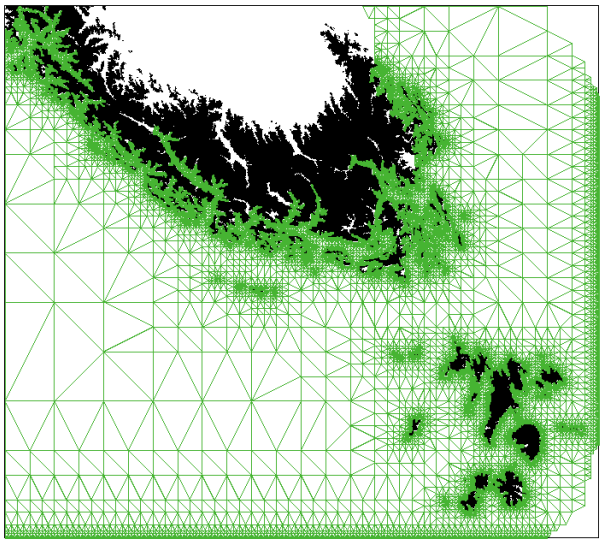
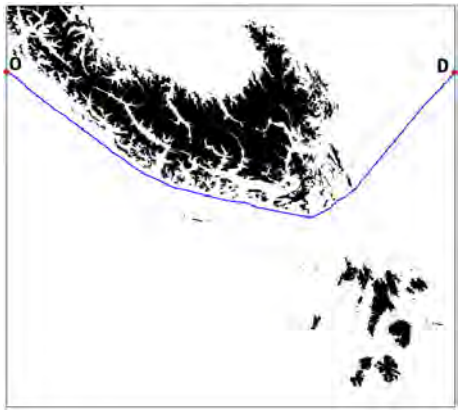
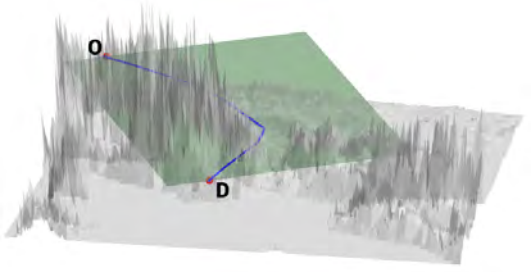


Figure 2.49: Triangles generated during the simulation.

At Nearest Suitable Airport (ANSA) Leaf nodes have a higher level near the obstacles. To avoid the trajectory passing near obstacles, it is proposed not to propagate the front in small cells except close to *StartCoordinate* and *EndCoordinate*. This improvement significantly reduces the number of visited cells and thus it decreases the computation time. Results for the ANSA scenario are shown in Figure 2.50.



(a) Horizontal view.



(b) 3D View

Figure 2.50: 3D trajectory from O to D generated with a 2D Fast Marching Algorithm over a single glide slope.

The blue trajectory gets around the mountains, in order to minimize the length of the trajectory while keeping it safe and flyable. This is the trajectory that 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 the final approach.

Compuation analysis Two of major issues of emergency trajectory design is the computation time and the memory space. Table 2.3a presents the computation time of each step of the algorithm. In both scenarios, the computation time is lower than 1 second. However, the ANSA scenario is faster because the front propagation does not visit small cells. Table 2.3b shows the

different sizes of the stored data, and Table 2.3c shows the number of nodes visited during the Fast Marching propagation. Low-sized structures are obtained, that can easily be embedded in a FMS. As expected, thanks to the improvement brought for the ANSA scenario, the number of visited nodes is divided by 5 while providing a trajectory of better quality than the ASAP scenario.

(a) 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

(b) Comparison on Stored Data Sizes (in Kilooctets)

	File Size
Terrain Data	491 649
Quadtree	225
Balanced Quadtree	302

(c) Comparison on the Number of Nodes Visited During Propagation

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

Table 2.3: Computation times and memory spaces.

2.4.1.6 Limitations of the 2D method

The method based on the 2D Fast Marching works well on many scenarios, but reaches its limits when the aircraft is too close to the landing site. Indeed, if the aircraft is just over the landing site when the emergency occurs, it is impossible to define a valid descent plan. Moreover, working over a single glide slope, the algorithm has to run multiple times to obtain a trajectory to all available landing sites. Finally, the method also does not allow a descent with variable glide slope, that could be insteresting to avoid obstacles. Therefore, although the method works quite well, it is necessary to generalize the method to 3D to respond to each type of emergency.

2.4.2 Fast Marching in 3D

This section presents the second trajectory generation algorithm based on 3D Fast Marching method. In contrast to the method illustrated in 2.4.1, the aircraft will not be constrained by a descent plane.

2.4.2.1 Tetrahedral Mesh

As for 2D Fast Marching, the first step is to build a structure to discretize the space. To do it, we propose generating octrees, the 3D equivalent of quadtrees. They will then be balanced to obtain tetrahedrons, the 3D equivalent of triangles.

The generation of an octree is similar to the one used for an quadtree. However, in 3D, the octree balancing is more complex. 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. Section 2.4.1 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 (see Section 2.4.1), the process called 1-*balance* was used, which corresponds to ensuring level differences between neighbors in the four cardinal directions. There exists another type of balance, the 2-*balance*, which ensures level differences with neighbors in the four cardinal directions and the four inter-cardinal directions. These are illustrated in Figure 2.51a and Figure 2.51b.

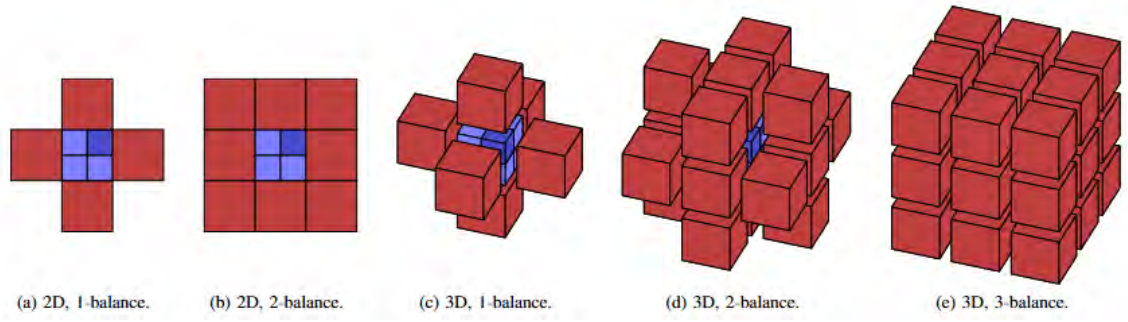
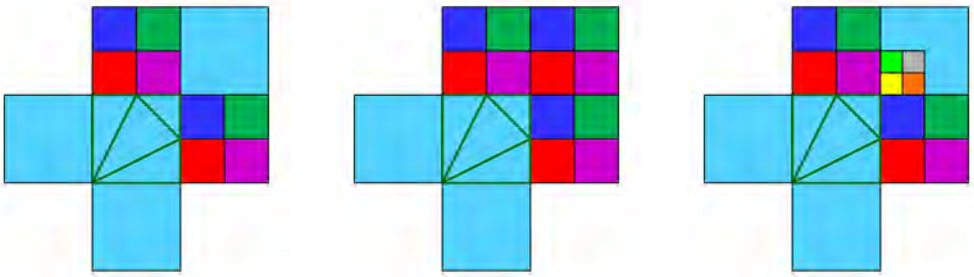


Figure 2.51: Ways to balance a quadtree (a and b) and an octree (c, d and e) [105, Fig. 5].

It is sufficient to make a 1-balance in two-dimensions, because inter-cardinal directions do not affect the possible quadtree node configurations. This is shown in Figure 2.52. This figure shows that no matter the level difference between the current node (at the center of each figure) and its NorthEast neighbor, the triangulation remains the same. Thus, considering 2-balance does not add any new configuration for the triangles.

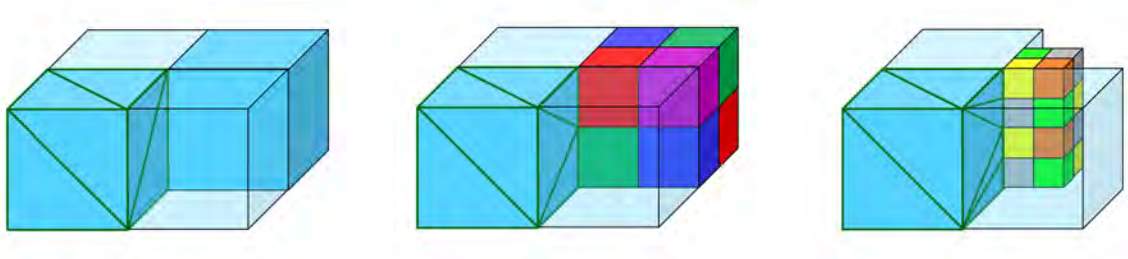


(a) $\delta = 0$. (b) $\delta = 1$. (c) $\delta = 2$.

Figure 2.52: Configuration of a quadtree node with a NorthEast neighbor with variable level difference δ. No matter δ, the northeast neighbor has no impact on the cell's triangulation.

In three-dimensions, a 1-balance can be made (see Figure 2.51c) 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 (generalization of triangulation in higher dimensions) varies according to the level difference between these two nodes. This is highlighted in Figure 2.53. This figure shows that the current node configuration changes when the level difference between the two nodes increases. Then, it is not enough to consider 1-balancing the octree. For the same reasons

as in two-dimensions, it is sufficient to 2-balance the octrees (see Figure 2.51d). A 3-balance, as in Figure 2.51e, would amount in balancing according to the inter-cardinal directions of order 2 (*e.g.* NorthEastUp). However, since the junction between such nodes are single vertices, the 3-balance does not add any new configuration, then is useless to consider it. Hence, it is necessary to make a 2-balance of the octree, which amounts in considering 18 directions (the 6 cardinals and the 12 inter-cardinals of order 1). This implies the study of $2^{18} = 262144$ configurations.



(a) $\delta = 0$. (b) $\delta = 1$. (c) $\delta = 2$.

Figure 2.53: Configuration of an octree node with a NorthEast neighbor with variable level difference δ. Unlike quadtrees, the tetrahedrization depends on the value of δ.

It is very difficult to analyze more than two hundred thousand configurations 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. This function returns, given a set of points and some options, the Delaunay tessellation in N dimensions (here, $N = 3$). The Delaunay tessellation 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 can be evaluated in order to check if they are all composed of acute tetrahedrons, the necessary condition that allows propagation in the Fast Marching algorithm. After computations, results show that about half the configurations are not suited for propagation, *i.e.* at least one of the tetrahedron of half the configurations are obtuse (they present a face with at least one obtuse angle). At first sight, this issue seems too hard to overcome. Nevertheless, we will see that we are able to reduce the number of configurations to evaluate and make them all acute. One first task is to remove redundant schemes. A shrewd eye can observe that 2^{18} configurations are not necessary to describe all the possible ones. 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, only 6210 configurations must be studied. This result is obtained in Appendix C. Again, these 6210 configurations are evaluated with the Python function `Delaunay`. As expected, about half of these configurations contain obtuse angles.

A second idea is to remove all symmetrical schemes. Symmetries appear with reflection or rotation of configurations, leading from one to another. An example is shown in Figure 2.54.

In this figure, only the first configuration is studied in order to find the Delaunay tessellation for all the other configurations. After removing all these symmetrical schemes, only 227 configurations are left to analyze. The Python function `Delaunay` gives 97 obtuse configurations and 130 acute configurations.

Finally, one has to consider these 130 acute configurations as *templates*. The goal is to manage to rebuild the 227 non-symmetrical configurations from these only 130 templates, to be able to shape every single octree node that can be encountered.

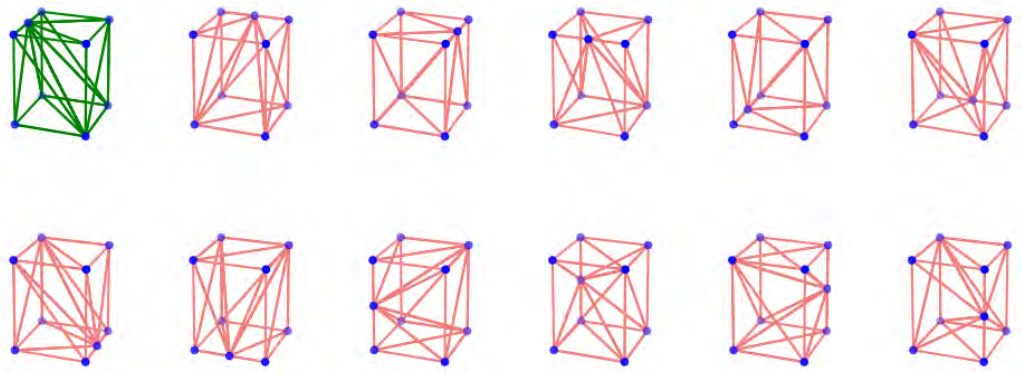


Figure 2.54: Symmetrical schemes for the scenario of smaller neighbors in 1 inter-cardinal direction.

To this end, an idea is to overlay good configurations in order to rebuild the obtuse ones. To illustrate this idea, an example is presented in Figure 2.55.

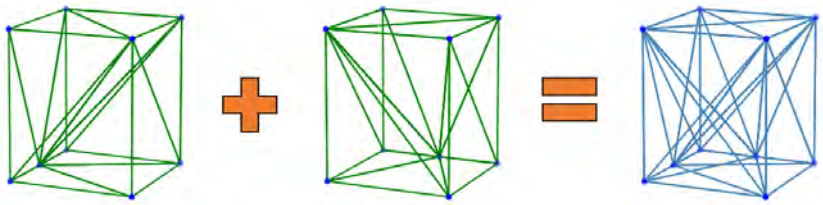


Figure 2.55: Building of a configuration (originally obtuse) from two instances of an acute template.

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 procedures of the Fast Marching algorithms, which only affect the vertices of the mesh. The more tetrahedrons, the better the update. A tetrahedron in the mesh should not be seen as a solid formed of four points, but rather as four triangle-vertex couples which serve the propagation.

To properly finish the building of the mesh, one must ensure 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 Figure 2.56 are acute.

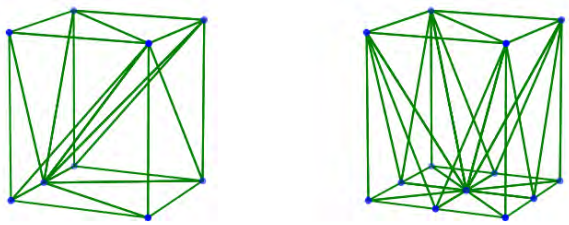


Figure 2.56: Templates n°2 and n°52.

Templates given in Figure 2.56 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 can

help rebuild all possible configurations. Moreover, since they are actually composed of acute tetrahedrons, thus all obtuse configurations can be rebuilt.

This demonstration shows that any obtuse configuration can be rebuilt from only two basic templates. However, actually 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, another secondary objective would be to minimize the number of tetrahedrons in each node. This is achieved by considering the level differences quantity δ, as represented in Figure 2.57.

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, Figure 2.57 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.

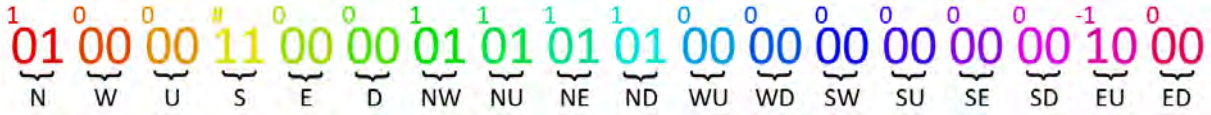


Figure 2.57: Example of δ value for an octree node. 2 bits are set for each direction. δ is therefore composed of 36 bits.

Considering this δ value, the procedure described in Algorithm 6 must be executed in order to find the proper template decomposition for an obtuse configuration.

Algorithm 6 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 C for more details on *activated* points). Thus, by following such a procedure, we add the smallest number of possible configurations in order to build 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 tetrahedrize each of its nodes, allowing updates for any mesh points with the Fast Marching algorithm. Each leaf is characterized by its Morton code m (8-byte signed integer), its level l (2 bytes), and its index i in the list of all possible configurations (4 bytes).

Each element in the list of all possible configurations refers to one or more of the 130 templates (stored in a 98 Ko file), and the operations to manage over these templates (rotations, symmetries).

To summarize, this section showed that an acute tetrahedrization can be built using the same ideas that were used in 2D. 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 in an aircraft.

2.4.2.2 Propagation process

The property that the minimal cost paths are orthogonal to the isocost curves is valid in any dimension. Hence, in 3D, the equation to solve is the 3D Eikonal equation written as follows:

$$|\nabla T| = f(x, y, z). \tag{2.43}$$

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

$$\begin{aligned} &\max\left(D_{ijk}^{-x}T, -D_{ijk}^{+x}T, 0\right)^2 \\ &+\max\left(D_{ijk}^{-y}T, -D_{ijk}^{+y}T, 0\right)^2 \\ &+\max\left(D_{ijk}^{-z}T, -D_{ijk}^{+z}T, 0\right)^2 = f_{i,j,k}^2 \end{aligned} , \tag{2.44}$$

where the forward and backward operators are given by:

$$\begin{aligned} D_{ijk}^{-x}T &= (T_{i,j,k} - T_{i-1,j,k})/\Delta x \qquad D_{ijk}^{+x}T = (T_{i+1,j,k} - T_{i,j,k})/\Delta x \\ D_{ijk}^{-y}T &= (T_{i,j,k} - T_{i,j-1,k})/\Delta y \qquad D_{ijk}^{+y}T = (T_{i,j+1,k} - T_{i,j,k})/\Delta y \\ D_{ijk}^{-z}T &= (T_{i,j,k} - T_{i,j,k-1})/\Delta z \qquad D_{ijk}^{+z}T = (T_{i,j,k+1} - T_{i,j,k})/\Delta z \end{aligned} , \tag{2.45}$$

where Δx, Δy and Δz are the cube steps. $T_{i,j,k}$ and $f_{i,j,k}$ are respectively the cost and the slowness at cubepoint (i, j, k).

Once again, the Fast Marching propagation depends on the chosen discretization of the domain Ω. In this manuscript, propagation on a cube will not be detailed because it is less efficient than propagation on a tetrahedral mesh (as explained in 2D in Section 2.4.1).

Consider an acute tetrahedron ABCD with $T(A) \neq \infty$, *i.e.* A has already been updated. Such a configuration is represented in Figure 2.58. The quadratic equation must be solved, to compute t such that $(t - u_{\max})^2 = h^2 f_D^2$, where $u_{\max} = T(C) - T(A)$ and $t = T(D) - T(A)$. Also, the value $u_{\min} = T(B) - T(A)$ is introduced.

The first case is trivial because the update comes from the edges: $T(B) = \min\left(T(B),\ \mathrm{AB} \times f_B + T(A)\right)$, $T(C) = \min\left(T(C),\ \mathrm{AC} \times f_C + T(A)\right)$ and $T(D) = \min\left(T(D),\ \mathrm{AD} \times f_D + T(A)\right)$. The second case corresponds to an update from the triangles ABC and ABD. Please refer to Section 2.4.1 for the detailed procedure. The third case

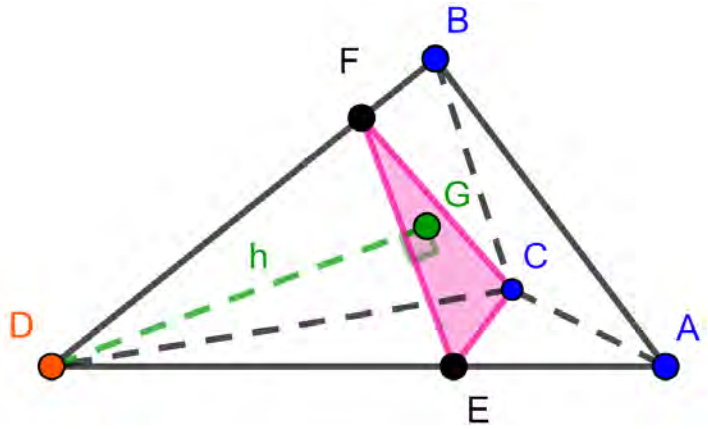


Figure 2.58: Update on a tetrahedral mesh.

corresponds to the update on a tetrahedron, for which all computations are made in Appendix D. In this case, the quadratic equation is written as follows:

$$\begin{aligned}&\left(\mathcal{A}_{\mathrm{BCD}}^2+\mathcal{A}_{\mathrm{ACD}}^2+\mathcal{A}_{\mathrm{ABD}}^2\right)t^2-2\left(\mathcal{A}_{\mathrm{ACD}}^2u_{\min}+\mathcal{A}_{\mathrm{ABD}}^2u_{\max}\right)t\\&+\mathcal{A}_{\mathrm{ACD}}^2u_{\min}^2+\mathcal{A}_{\mathrm{ABD}}^2u_{\max}^2-9f_D^2\mathcal{V}_{\mathrm{ABCD}}^2=0\end{aligned},\tag{2.46}$$

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

Hence, if the update at D comes from the tetrahedron, then one must solve equation (2.46) to compute the cost at D.

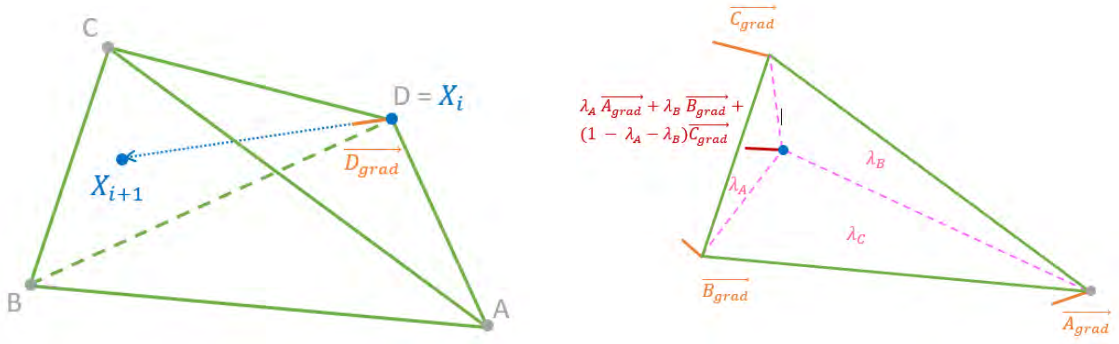
The final trajectory is found thanks to the back propagation of gradients, from *EndCoordinate* to *StartCoordinate*.

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 forward propagation phase. Thus, at each encounter with an edge or a triangle, the gradient must be interpolated.

On an edge, it is easily done by computing the barycenter of the gradients linked with the two vertices of the considered edge (see Section 2.4.1).

On a triangle, it is done by computing the barycenter of the gradients linked with the three vertices of the considered triangle (see Figure 2.59).

Figure 2.59 shows the interpolation of a gradient inside a tetrahedron ABCD. In Figure 2.59a, 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 forward 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. In Figure 2.59b, a barycentric interpolation is made. The interpolation parameters are $\lambda_A = \dfrac{\mathcal{A}_{X_{i+1}BC}}{\mathcal{A}_{\mathrm{ABC}}}$ 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.

Figure 2.59: Back propagation of gradients on a tetrahedron. X_{i+1} is at the intersection between $\overrightarrow{D_{grad}}$ and the triangle ABC. The gradient of X_{i+1} is then computed by a barycentric interpolation of the gradients of A, B and C.

As in 2D, it is easy to show the consistency of the interpolation. If $X_{i+1} = A$, then $\lambda_A = 1$, $\lambda_B = 0$ and the gradient at X_{i+1} is $\overrightarrow{A_{grad}}$. Same remarks apply for $X_{i+1} = B$ and $X_{i+1} = C$.

The algorithm stops when reaching *StartCoordinate*, and the final trajectory between *StartCoordinate* and *EndCoordinate* is found by linking the triangle intersection points together.

2.4.2.3 Operational Constraints

The Fast Marching methods generate the shortest path on a given mesh. However, the path is generally unflyable. To consider the aeronautical constraints, it is proposed to generate an approximate path that respects such constraints. This path is then used to build a guiding tube in which Fast Marching will be applied. Figure 2.60 shows an example of such a guiding tube.

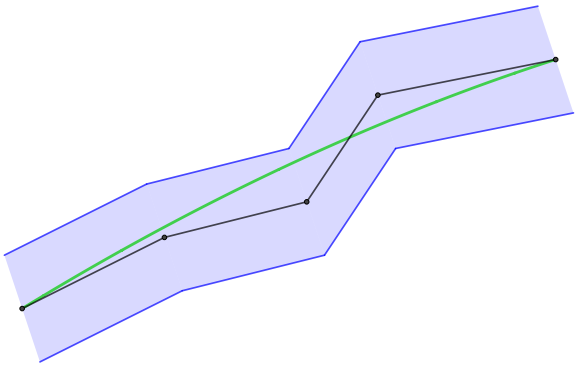
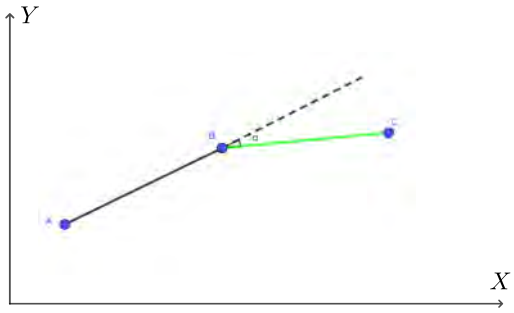


Figure 2.60: Building of a guiding tube: the approximate path is in black, the tube is the blue area and the path generated by the Fast Marching is in green.

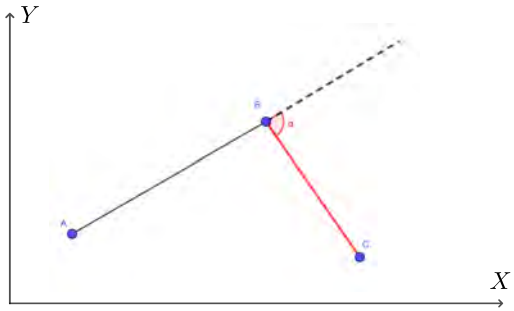
To build the approxiamte path, the Fast Marching Tree algorithm (presented in Section 2.3) is chosen for its adaptability to the constraints and its low computing time complexity.

This algorithm is able to take into account aeronautical constraints. Indeed, the algorithm has a *best neighbor* selection phase, with the possibility to discard neighbors that do not satisfy such constraints. It also adapts really well to a three-dimensional space. 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 Figure 2.61) 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 Figure 2.62).



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



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

Figure 2.61: Taking into account turning constraints. The angle α must not be too high to make the turn feasible.

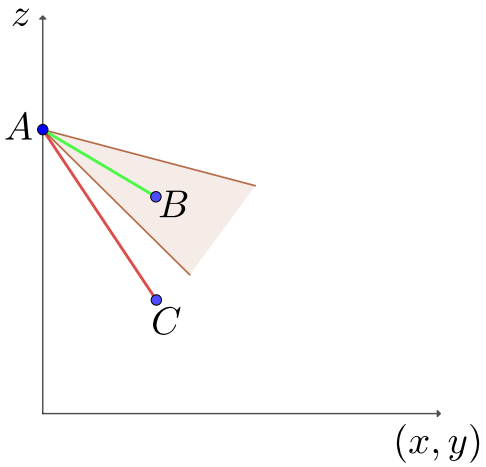


Figure 2.62: Taking into account descent constraints: the connection between the points A and B is possible because the segment is in the descent cone, but the connection AC is impossible.

2.4.2.4 Results

This subsection presents the results of the method based on 3D Fast Marching for an ANSA emergency in the mountainous region of Grenoble in France. Once again, we are insterested in the computation time, the memory space and the flyability of trajectories.

The scenario presented in this section is complementary to the two scenarios studied in 2D in Section 2.4.1, because it can only be tackled by the 3D algorithm (one single descent plan cannot guide the aircraft towards the nearest airport).

Unconstrained Fast Marching Algorithm First, to validate the method on tetrahedron, a simulation is run without taking into account aeronautical constraints. Figure 2.63 shows the result of the simulation. There is no obstacles between the start point and the end point, the solution is therefore a straight line. In order to satisfy the descent constraints, the aircraft would have to move in the mountain direction to lose altitude and land safely.

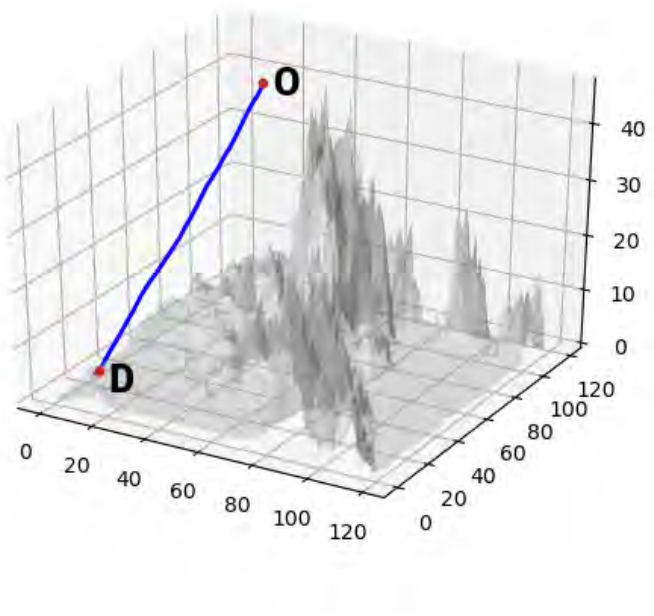
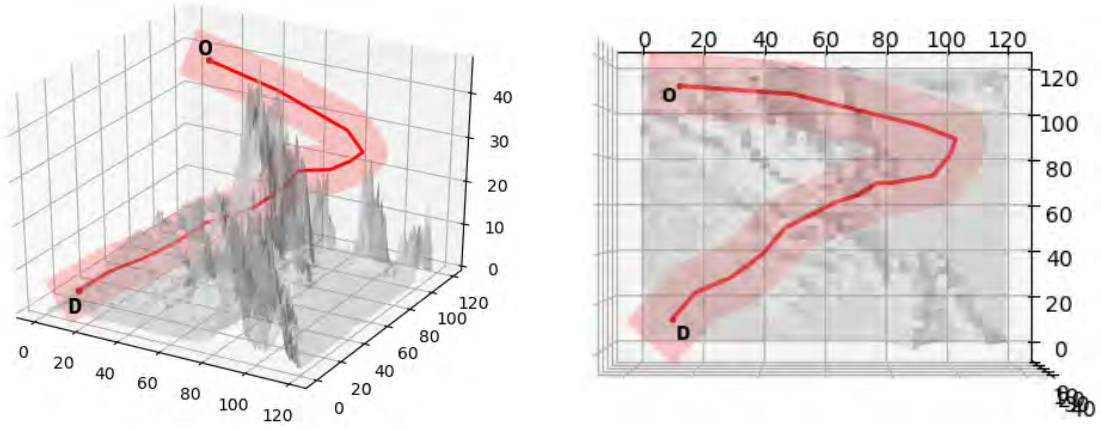


Figure 2.63: 3D trajectory (in blue) generated in an ANSA scenario (arb. unit).

Constrained fast marching algorithm For the second simulation, the flight path angle and turn radius constraints are considered. Figure 2.64 shows the path computed by the FMT* algorithm and the tube generated around it. The size of the tube is an input that can be a function of the domain size. Its size needs to povide a balance between two opposite objectives:

- The tube must not be too thin because it reduces the degrees of freedom of the method.
- The tube must also not be too large because otherwise the path computed by the fast marching would not satisfy the constraints.

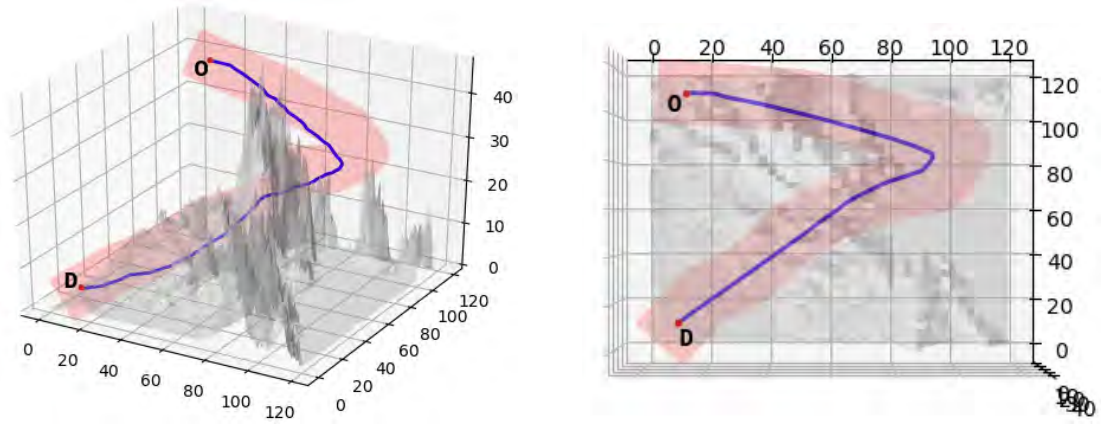
Figure 2.65 shows the trajectory computed by the constrained Fast Marching algorithm. This simulation highlights the advantages and drawbacks of the considered method. The computed



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

Figure 2.64: Approximate trajectory generated in an ANSA scenario with the FMT* algorithm (arb. unit).

trajectory is smooth, short, avoid obstacles and stays within the guiding tube. However, it is clear that the trajectory is not fully flyable. Indeed, it has a higher angle of descent than the FMT* path, which can potentially be out of the constraints. This problem can be solved by reducing the size of the guiding tube. However, the path could be too close to the approximate path and therefore far from the optimum. 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 size of the descent cone is reduced.

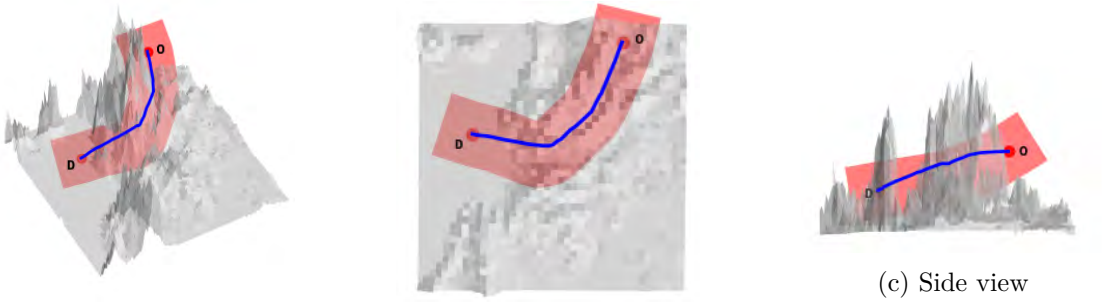


(a) Side view. (b) Top view.

Figure 2.65: Trajectory generated in an ANSA scenario with the Fast Marching algorithm (arb. unit). The red area is the tube generated around the trajectory computed by the FMT* algorithm.

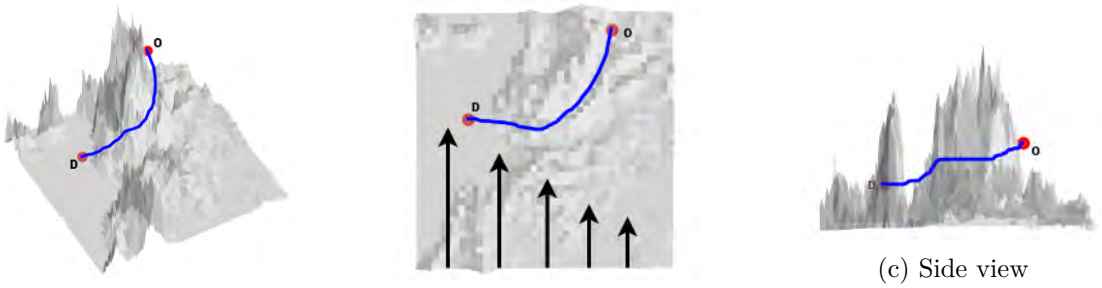
Wind fast marching algorithm Finally, the last simulation tests the algorithm with wind consideration. The wind field is represented by a static grid. The scenario is different from the previous one. The aircraft is not constrained by a minimum descent angle. Figure 2.66 shows the result obtained by the algorithm without considering the wind. Figure 2.67 presents, the trajectory computed by the proposed method taking into account the wind. In this scenario, the wind comes from the south. Its intensity decreases from west to east and is constant in the

vertical dimension (See Figure 2.67b). The wind being unfavorable, the solution computed with the wind is 10% more costly.



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

Figure 2.66: Trajectory generated without wind. The red area is the tube generated around the trajectory computed by the FMT* algorithm.



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

Figure 2.67: Trajectory generated with wind from the south (top view)

These tests show the efficiency of the 3D Fast Marching method but also its limits. Indeed, the medthod does not guarantee 100% satisfaction of the constraints.

Computation time analysis After studying the resulting trajectories, it is necessary to analyze the computation times and memory space. Tables 2.4a and 2.4b present these the associated results and summarize the computation time and the stored data size of each part of the algorithm for the scenario presented in the last paragraph. 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. As expected, the 3D extension increases the computation time of the Fast Marching algorithm. Indeed, the update procedure is more complex and the visited space is larger than in 2D. However, the restriction in the guiding tube reduces this increase. The algorithm is still very fast and computes a trajectory in less than 4 seconds, which is acceptable for an emergency trajectory generation.

2.4.3 Conclusion

This section has presented two adaptions of Fast Marching methods to emergency trajectory design. The methods were tested on several scenarios and seem to be promising. Indeed, they rapidly compute a trajectory and can manage some aeronautical constraints. However, some improvements have to be made in order to make the algorithm workable on an aircraft.

(a) Computation Times (in Milliseconds)

	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 Kilooctets)

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

Table 2.4: Computation time and memory usage results.

The implementation of functions (creation of octrees, balancing, Fast Marching...) must be improved in order to obtain a stronger method. Indeed, improvements are yet to be found with the octree tetrahedrization. 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. Moreover, currently the method does not guarantee a trajectory 100% flyable. The next section presents the SafeNcy system and more particularly the path generator that is based on the Rapidly-exploring Random Tree algorithm.

2.5 SafeNcy project

This section presents the works done during the european project SafeNcy. It mainly focuses on the works of the author of this manuscript.

2.5.1 Introduction

The SafeNcy (the safe emergency trajectory generator) project is a European project funded by CleanSky 2 (JU) under Grant Agreement No. 864771. This project is coordinated by the company FRACS (France Aviation Services) and the technical participants are CGX Aero, MetSafe, UPC (Universitat Politècnica de Catalunya), Eurocontrol, and ENAC (Ecole Nationale de l'Aviation Civile). The project started in November 2019 and finished in December 2022.

The SafeNcy project aims at designing, developing, testing, and validating a TRL5 (Technology readiness level 5) on-board function for the FMS capable to support the pilot in the decision-making process following an emergency situation with degraded and adverse conditions (See Figure 2.68).

Firstly, an emergency landing site selection algorithm has been developed, using a database of possible landing sites that is constantly updated via data-link connection with the airline Operating Control Centre (OCC) and which takes into account updated meteorological information and potential hazards. This landing site minimizes also the collateral effect of the emergency landing by selecting landing sites with a minimum population density and low risk propagation.

Secondly, an on-board trajectory generation prototype has been developed in order to compute the best emergency trajectory from the aircraft to the selected landing site(s). This algorithm takes into account the most up-to-date meteorological information, considers the degraded dynamics of the aircraft resulting from some failures, and has a robust design allowing real-time capabilities.

Finally, a thorough verification and validation campaign has been performed, developing an

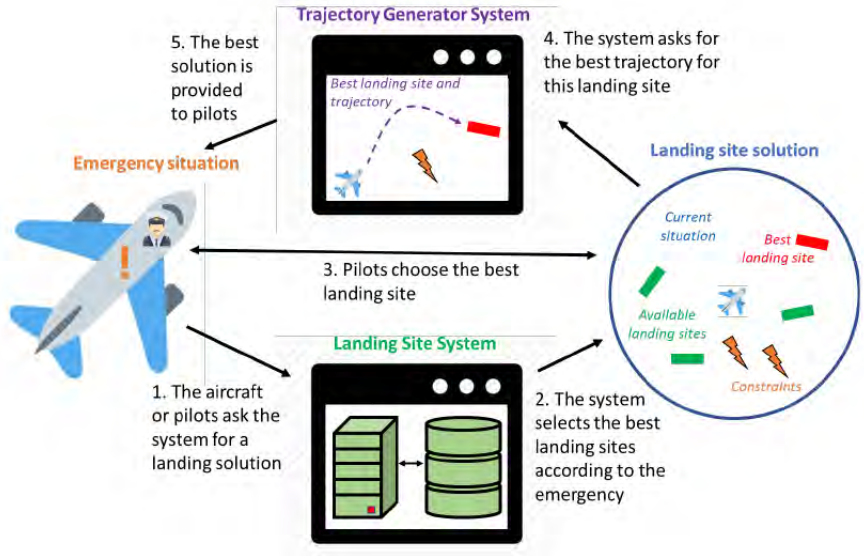


Figure 2.68: System Diagram.

ad-hoc simulator for testing purposes and integrating it with a fault detection and identification external block.

This manuscript presents all parts of the SafeNcy system to enable readers to understand the context. However, it focuses on the part about the path generator on which the author of this manuscript has mostly worked. The other parts are presented without going into details of the algorithms used.

2.5.2 Study cases

In the SafeNcy project, we defined several study cases. As described in the introduction of this chapter, there are three types of emergency: ANSA, ASAP, and TEFO. In this project, due to their specificities, two emergencies are considered and are separated from the 3 other types: FIRE and DEPRESSURIZATION.

- FIRE: it is considered as an ASAP emergency in which there is fire on board. The vertical profiles are the same as in the ASAP cases. However, as it will be explained in Section 2.5.3, the maximum range is different because it corresponds to a maximum flight time of 20 minutes.
- DEPRESSURIZATION: it is an ANSA emergency in which there is a loss of pressure in the cabin. This problem implies a modification of the vertical profile because the aircraft has to reach as soon as possible the flight level 100 to have enough oxygen.

This classification of the emergencies leads to the definition of vertical profiles, maximum range, and available landing sites. Moreover, the vertical profiles depend on the other following factors:

- Engines available: all engines out or at least one engine is operative;
- Approach procedure: does the landing site have a suitable IFR approach?
- Maneuverability: aircraft is fully or not fully maneuverable;

- Fuel on board: Is there any fuel left on board? This is only considered in the TEFO case.

In addition to these factors, in case no engines are available, the availability of fuel on board is also considered.

Finally, the list of vertical profiles considered in the project is the following:

1. TEFO + Fuel on board;
2. TEFO + No fuel on board;
3. ASAP/FIRE + IFR + Fully maneuverable;
4. ASAP/FIRE + IFR + Not fully maneuverable;
5. ASAP/FIRE + No IFR + Fully maneuverable;
6. ASAP/FIRE + No IFR + Not fully maneuverable;
7. ANSA + IFR + Fully maneuverable;
8. ANSA + IFR + Not fully maneuverable;
9. ANSA + No IFR + Fully maneuverable;
10. ANSA + No IFR + Not fully maneuverable;
11. DEPRESSURIZATION + IFR + Fully maneuverable;
12. DEPRESSURIZATION + IFR + Not fully maneuverable;
13. DEPRESSURIZATION + No IFR + Fully maneuverable;
14. DEPRESSURIZATION + No IFR + Not fully maneuverable.

2.5.3 Landing Site Selector

The first module of the SafeNcy system is the landing site selector which was developed by CGX Aero. The aim of the module is to provide more assistance to the flight crew in case of an in-flight emergency situation by providing a set of landing solutions where multiple factors are taken into account. The objective is to find the best compromise between limited on-ground damages and maximum survival chances for the aircraft passengers. In order to answer this need, the envisioned management process for landing sites consists of two distinct modules:

- **Data compilation**: it is used to integrate system worldwide data from multiple sources. More precisely, the purpose is to extract from a pre-flight database a list of potential landing sites (airport and off-airport) that can be used in an emergency situation. The result of this process is composed of a master database of landing sites and constraints (e.g., terrain elevation data). This database must contain as many as possible landing sites and consist of terrain elevation data of the entire planet to propose a solution in every situation.
- **Data exploitation**: this module provides on-board assistance to the pilot crew in case of emergency. It works during the flight and selects the best available landing sites and the relevant constraints for the pilots. The landing sites are extracted from the database created by the data compilation module.

2.5.3.1 Data Compilation Module

The goal of this module is to create a single database from different database sources. The architecture of the module is given in Figure 2.69.

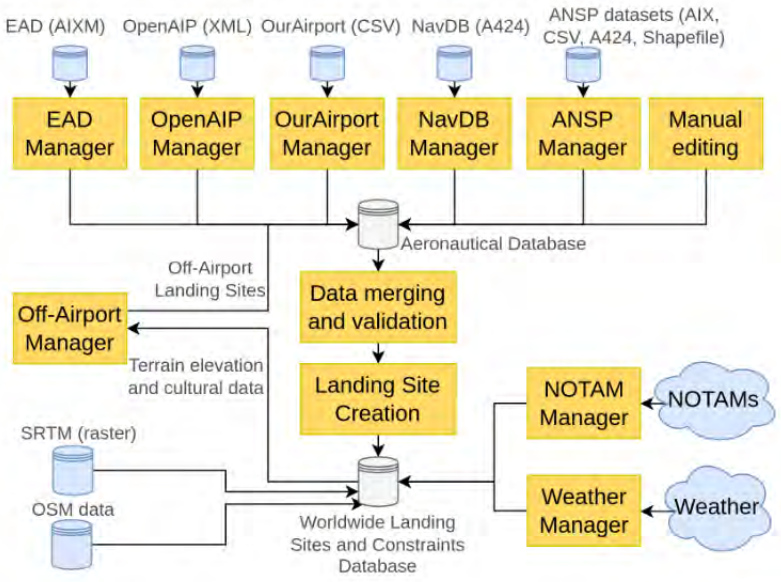


Figure 2.69: Architecture of the data compilation module.

Different types of inputs are required for the data compilation module. They are divided into 2 parts: static and dynamic:

- **Static**: these data do not change all along the flight.
  - **Geographical databases**: theses databases are used to generate off-airport landing sites thanks to information on buildings, areas to be avoided, terrestrial traffic lanes (highways, roads...), terrain types, natural features, and waterways. OpenStreetMap (OSM) has been used during the project.
  - **Terrain databases**: these databases provide terrain altitude information through digital elevation model (DEM) files. The terrain elevation is used in two uses. First, the compilation module exploits it to find flat terrains and therefore possible off-airport landing sites. The other use is done by the trajectory generator module which detects obstacles to be avoided thanks to this information. The source of this database is the Shuttle radar topography mission (SRTM) developed by NASA.
  - **Aeronautical databases**: these databases are composed of static aeronautical features necessary for the SafeNcy system such as:
    * Aerodromes features (e.g., runways and thresholds);
    * Radio Navigation aids;
    * Standard Terminal Arrival Route procedures;
    * Airspace and route names, geometries, and constraints;
    * Obstacles features.

    The sources of data are numerous. Air navigation service provider (ANSP) databases, navigation databases (NavDB) and European AIS Database (EAD) are mainly used.

- **Dynamic**: these data are updated all along the flight.
  - **Weather Data**:
    * **Navigational constraints**: temperature, pressure and wind speed/direction in a 4D grid format.
    * **Navigational obstacles**: significant meteorological (SIGMET), icing, lighting cells and thunderstorms. SIGMET gives the presence of expected or unexpected en-route weather phenomena. These different phenomena define areas that have to be taken into account by the trajectory generator module.
    * **Landing site weather information**: it is composed of meteorological aerodrome reports (METAR), pseudo-METAR, and automatic terminal information service (ATIS). METAR is an observation made at an aerodrome updated each hour. Pseudo-METAR is a METAR information determined by the national oceanic and atmospheric administration(NOAA) from the global forecast system (GFS). ATIS provides information in terminal areas. The weather data are given by MetSafe [145].
  - **Notice to air missions (NOTAM)**: states if the aircraft can land safely on the selected landing site.

All these databases are aggregated to create two complete databases: a worldwide landing site database and a worldwide static constraints database. The system is updated whenever new data is available and the process is done on the ground before the flight. These databases are then loaded into the aircraft to be used during the flight by the data exploitation module and the trajectory generator module.

2.5.3.2 Data Exploitation Module

The second module of the landing site selector is the data exploitation module. It works during the flight and has to select the available landing sites around the aircraft position and select information for the trajectory generation module. This module is divided into four parts:

- **The landing site selector**: this sub-module computes the best landing sites from the landing sites database, and using METAR and pseudo-METAR information;
- **The weather manager**: it downloads weather dynamic data;
- **The NOTAM manager**: it downloads the NOTAMSs;
- **A web service**: It is called by the data exploitation module and returns, from aircraft position and type of emergency, a list of suitable landing sites. It also provides the weather constraints for the trajectory generation module.

2.5.3.3 Landing site generation and selection mechanism.

In the SafeNcy system, the landing sites are separated into two categories: airports and off-airports landing sites. Airports are landing sites used in normal situations. They are the first choices of the pilots because they are designed for aircraft landings. In the case of an airport, a landing site corresponds to one runway with its direction. For instance, the Charles-De-Gaulle airport is composed of 4 runways, therefore, is consisted of 8 landing sites.

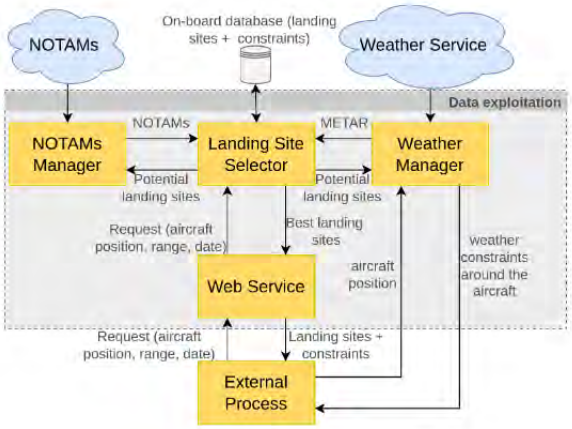


Figure 2.70: Architecture of the data exploitation module.

On the other hand, off-airports are landing sites that can be used only when no airport can be reached. An off-airport is determined by gathering geographical data and terrain elevation data. It is an area modeled by a strip of land that has some similarities with a runway. A terrain can be considered as an off-airports has the following characteristics:

- The terrain must be flat to avoid any problem at the time of the impact with the ground. It is to ensure the aircraft will not land on abrupt slope or fall into a ravine. The precision of the elevation must be very high to detect small obstacles that can damage the aircraft. The flatness is assessed using a terrain roughness index (TRI) [172]. The TRI evaluates the heterogeneity of the roughness of the terrain.
- The type of terrain must be considered because the terrain must not damage the aircraft, endanger the passengers or people on ground.
- The terrain must be long and wide enough to guarantee a safe landing. The landing site selector considers an off airport as a rectangle of at least a length of 1500m and a width of 100m.
- The path to land on an off-airport must cross an area clear of obstacles. In the SafeNcy project, the approach of an off-airport is protected on a slope at 3 degrees for at least 10NM from the threshold of the landing site.

For both types (airports and off-airports), a landing site is represented by the following features: identifier, name, airport name, country code, landing site class, landing site surface, landing closed, elevation, position, length, width, heading, and source.

All types of airports are classified into 6 classes, as shown in Table 2.5. The classification is done by taking into account the following characteristics: runway length, runway type (e.g, concrete, grass, gravel,...), approach type, ATC availability, security concern (e.g., landing site in a war zone), and passenger accommodation available (e.g., medical assistance).

Depending on the type of emergency (ANSA, ASAP, or TEFO), the landing site selector proposes different types of airports:

- ANSA: in this situation, the pilot must find the safest way to land. The passenger safety is not directly in danger, the pilot has therefore the time to choose a major airport. Consequently, airports of class 1 and 2 are considered in this case.

Class	Type	Characteristics
1	Airport	• Length of the runway > 2700m • Airport in navdatabase • IFR approach with at least 200ft minima or lower • ATC • Fire assistance • No security concern • Passenger accomodation available
2	Airport	• At least one runway > 2200m for big aircraft • Not necessarily included in navdatabase • Ar least an IFR non-precision approach procedure • At least one remote ATC or traffic information
3	Airport/runway/concret strip	• At least one concrete runway with length greater than the average landing distance for the aircraft category • Not in the aircraft navdatabase • No IFR approach procedure • At least remote ATC
4	Airstrip	• At least one runway with ,o solid obstacle in a 1200m radius after the touchdown area • Not in the aircraft navdatbase • No IFR approach procedure
5	Ditching	• Open sea or river • Within reachable distance to a place where emergency services are available
6	Forced landing	• Flat or almost-flat area with no obstacles • One direction identified providing the longest landing strip • With reachable distance to a place where emergency services are available

Table 2.5: Landing sites classes.

- ASAP: in this case, the pilot has to land as soon as possible. Landing sites of classes 1 to 4 are therefore proposed by the landing site selector.
- TEFO: in this case, the range of the aircraft is limited. In priority, the landing site selector proposes landing sites of classes 1 to 4. But, if they can not be reached, landing sites of classes 5 and 6 are also considered.

From all these data and the range estimated by the trajectory generation module, the data exploitation module selects potential landing sites. These landing sites are then ranked as follows:

- Landing sites are ranked according to their class (see Table 2.5).
- Landing sites of the same class are then sorted by order of importance that depends on: the distance as the crow flies from the aircraft to the landing site, the weather in the landing site and the contamination of the runway.

Figure 2.71 shows the ranges and selected airports depending on the emergency for a Boeing B777 flying over Karaganda at FL330 on a route from Moscow to Almaty. The ANSA range is represented in green and is the biggest one. In this case, the engines are operative. The limit of the achievable space is determined by the amount of fuel remaining. The depressurization range is in purple. In this case, the aircraft has to fly at a lower altitude to get enough oxygen. It burns more fuel, the range is, therefore, smaller than in ANSA. For the specific case of a fire (FIRE), in yellow, the limitation is the time. The aircraft can not fly more than 20 minutes to prevent that the fire from becoming too large and damaging the aircraft, which could be fatal. Finally, the range TEFO, in red, is the smallest one. In this case, the engines no longer work. The range corresponds to the maximum range when flying at the maximum lift-to-drag ratio. In this figure, the ASAP range is not represented. In this scenario, the landing sites for ASAP emergencies are within the ANSA range. However, as explained before, if there is no airport of class 1 or 2, the landing site selector proposes also lower class landing sites.

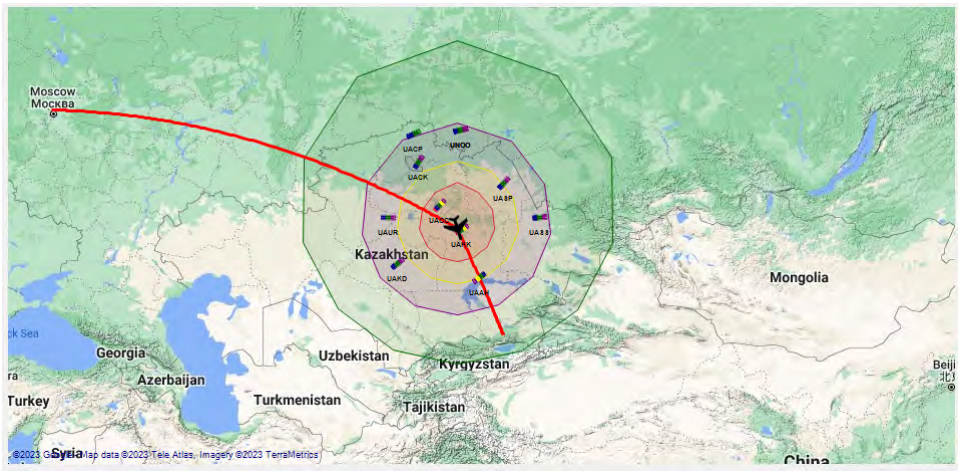


Figure 2.71: Ranges and landing sites available for a B777 from Moscow to Almaty flying over Karaganda at FL330.

As explained before, for each landing site in the range, information is given on the class, the heading, the elevation, dimensions, ILS frequency, and weather (See Figure 2.72).

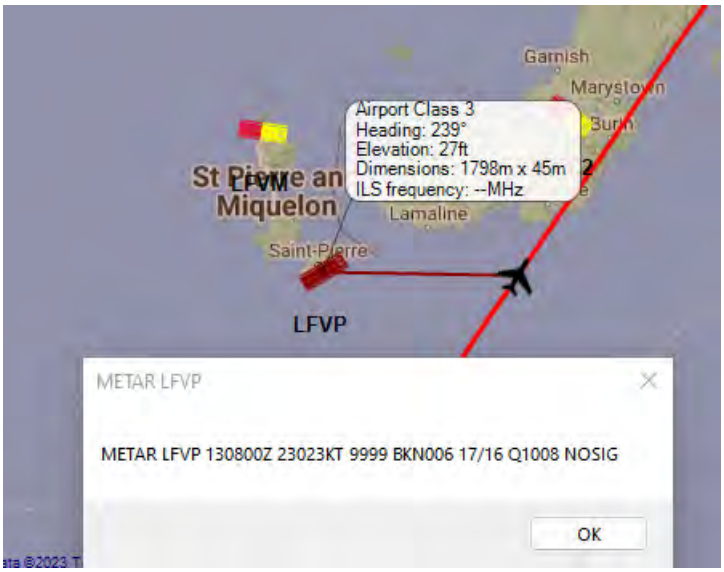


Figure 2.72: Saint-Pierre Airport (LFVP) runway information and METAR.

After selecting the potential landing sites, the SafeNcy computes the trajectory from the aircraft position to the landing sites. The trajectory generation module is presented in the next subsection.

2.5.4 Trajectory Generator

Now that the landing sites have been determined, the trajectories must be computed.

2.5.4.1 Methodology

The methodology to design a 4D emergency trajectory is shown in the diagram in Figure 2.73. It is composed of three different steps represented by the three following modules:

- **Path bounds generator**: in order to determine the path from the aircraft to a landing site, this module computes a set of flight path angles ($\gamma_{g,1}...\gamma_{g,n}$) and a set of minimum turn radius ($R_{min,1}...R_{min,n}$) depending on the altitude h and the track χ_g. This module considers as inputs, aircraft performance depending on the emergency type and weather—i.e., temperature, pressure, and wind. More details regarding the path bounds generator module can be found in Section 2.5.4.2.
- **Path generator**: this module takes as inputs the two sets generated by the path bounds generator, the terrain elevation, weather obstacle (thunderstorm, icing,...), and landing sites data. It generates the path linking the aircraft position and the landing site that avoids all the obstacles. The method is based on an adapted version of the Rapidly-Exploring Random Tree Star algorithm. More details regarding the path generator module can be found in Section 2.5.4.3.
- **Motion planner**: from the path and the distance to the landing site, this module generates a 4D trajectory with a Trajectory Prediction (TP), which refers to the process of computing a trajectory given a known sequence of control inputs. More details regarding the motion planner module can be found in Section 2.5.4.4.

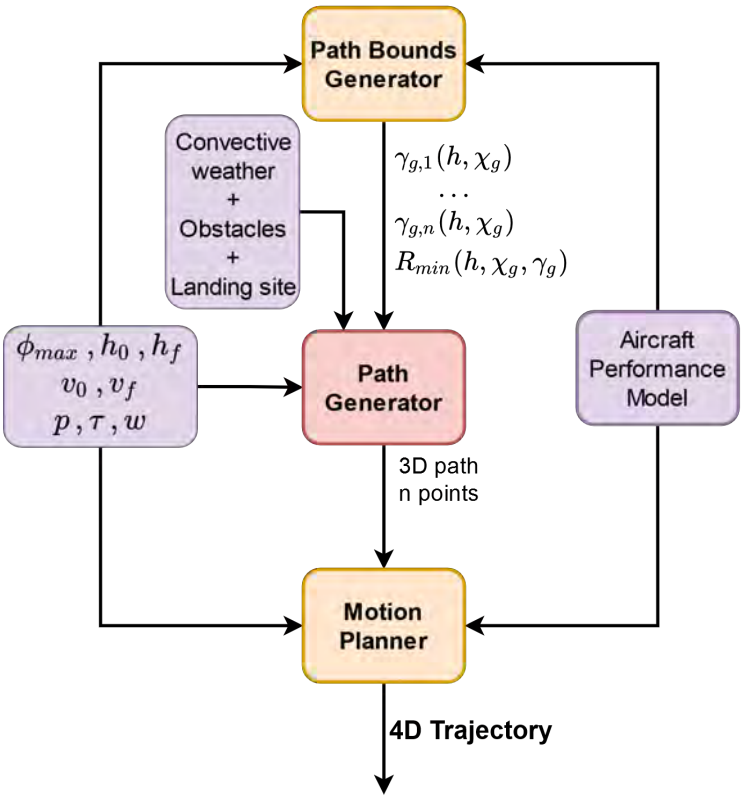


Figure 2.73: Generation process of the emergency trajectory. The trajectory generator is composed of 3 modules: Path Bounds Generator, Path Generator and Motion Planner. First, Path Bounds generator module provides flight path angles and minimum turn radius to the Path Generator. Then, the path generator computes a 3D path from Paths Bounds Generator outputs, weather, obstacles and landing site. Finally, the Motion Planner modifies the path to obtain a 4D trajectory.

2.5.4.2 Path bounds generator

For each profile defined in Section 2.5.2 and used by the motion planner (presented in Section 2.5.4.4), the path bounds generator defines a finite number of flight path angles and minimum turn radius as a function of the altitude and the track. These parameters are used by the path generator. They are computed as follows:

- Flight path angles (γ_1 to γ_n): during a descent trajectory except at the end, the flight path is continuously changing as a function of the altitude and the track. In order to speed up the computational time of the path generator, it is proposed to discretize the descent in several sections. On each section, the flight path angle is considered constant. The path bounds generator provides a flight path angle every 100 feet of altitude and every 10° of track.
- Minimum turn radius (R_{min}): it is achieved when the aircraft flies with the maximum bank angle and it is also affected by the current ground speed. The minimum turn radius depends on altitude, track, and ground flight path angle. If the aircraft is fully maneuverable, the bank angle is 30° and 15° otherwise.

This choice of representation of the descent profiles makes the two methods presented above unusable (Sections 2.3 and 2.4). Indeed, the FMT* algorithm cannot take into account constant

slopes because the probability of a connection is almost zero due to the fixed sampling that is generated at the initialization. As for the Fast Marching approach, the guiding tube cannot be defined because the FMT* does not work. Moreover, it is complicated to define the descent plan from these data. It was therefore necessary to develop a new algorithm which is based on the Rapidly-Exploring Random Tree Star (RRT*) algorithm.

2.5.4.3 Path Generator

This subsection presents the path generator which computes the 3D path from the aircraft position to the landing site. The path has to avoid all obstacles (terrain and weather) and respects the path bounds constraints. To generate this path, the module takes into account the terrain elevation, the weather obstacles, and the sets generated by the path bounds generator. In the context of an emergency, the main goal is to have a very fast path generation algorithm. This algorithm needs also to be adaptable to any type of emergency situation. In this project, it is therefore proposed to use an adapted version of the RRT* algorithm, which has a low complexity time and leads to very low computational times. Indeed, this is one of the main reasons why this algorithm was chosen, as when dealing with emergency situations, time is a very important factor to take into account. Moreover, the algorithm is asymptotically optimal, thus, leading to solutions that satisfy the safety-critical nature of the application presented in this work.

Rapidly-Exploring Random Tree The Rapidly-Exploring Random Tree (RRT) algorithm is part of the same family of algorithms as the FMT* (presented in Section 2.3) and PRM (presented in Section 1.2.6). As a reminder, these algorithms are composed of 4 main functions: Sampling, Nearest Neighbor, Near Vertices, and Collision Test functions. However, unlike the FMT*, in the RRT algorithm, the sampling is generated during the growth of the tree. At each iteration, a new point x_{rand} is randomly generated in the free space. After that, this point is steered towards its nearest point in the tree. Given two points x and y, the function *Steer*: $(x, y) \mapsto z$ returns a point z such that z is "closer" to y than x (See Figure 2.74). The point returned by this function is connected to the graph if there is no obstacle between it and its nearest point (See Figure 2.75a). This difference with the FMT* algorithm makes it more adaptable to the constraints of the path bounds generator. Indeed, in the FMT* algorithm, the probability of a connection between two nodes is almost zero with the constant flight path angle constraints. However, in the RRT algorithm, the function *Steer* can fix the altitude of the new point to satisfy this constraint.

Figure 2.76 shows an illustration of the tree growth and the pseudo code of the RRT algorithm is given in Algorithm 7.

An improvement of such algorithm is the RRT* algorithm [117]. It connects vertices in the same way as RRT. Moreover, at each step, it tries to improve the graph around this new vertex. If the addition of the new vertex can reduce the cost of vertices that are within a distance r (neighboring radius), a new connection replaces the previous connection (Figure 2.75b). The distance r is computed as a function of V, which is an open set composed of nodes, and a coefficient γ_{RRT*}, whose value depends on the search space. It is defined as follows:

$$r = (1 + \eta)\gamma_{RRT*} \left(\frac{\log(|V|)}{|V|} \right)^{1/d}, \tag{2.47}$$

where $\eta > 0$ and:

$$\gamma_{RRT*} = 2 \left(1 + \frac{1}{d} \right)^{1/d} \left(\frac{\mu(\chi_{free})}{\zeta_d} \right)^{1/d}. \tag{2.48}$$

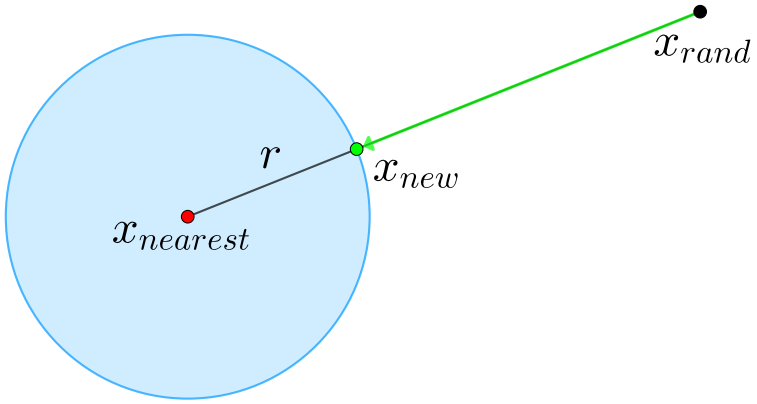
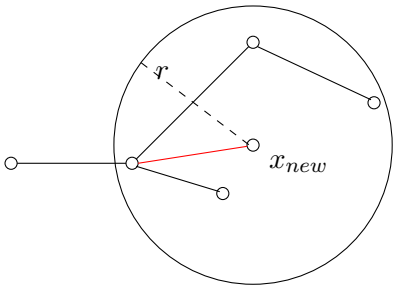
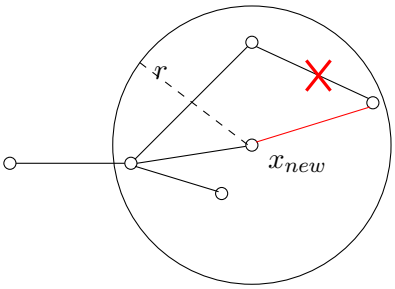


Figure 2.74: Illustration of the steering function: a point x_{rand} is randomly generated, then this point is steered to create a new point x_{new} that is close to $x_{nearest}$.



(a) New connection building: The new node x_{new} is connected to a neighboring node which minimizes its cost.



(b) Connection replacement: The cost of a neighbor is improved by the creation of the new node x_{new}. The edge which connected this node to the graph is deleted and a new edge is created

Figure 2.75: RRT*.

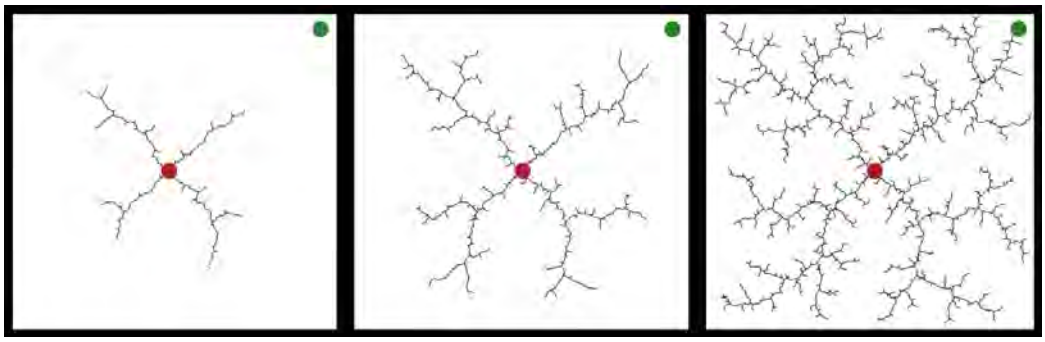


Figure 2.76: Representation of the graph at three different iterations of the RRT algorithm. The right subfigure is the final iteration, for which the initial node and the final node are now connected. (Initial node in red and final node in green) [128].

The pseudo code of the RRT* algorithm is given in Algorithm 8, the differences with the RRT algorithm are written in red.

Figure 2.77 shows, for the same number of iterations, the tree generated by the RRT algorithm (left) and the RRT* algorithm (right). This figure shows that with the phase of improvement, the RRT* tree is more ordered than the RRT and, therefore, the path computed by the RRT*

Algorithm 7 Rapidly exploring Random Tree

$V \leftarrow \{x_{start}\}; E \leftarrow \emptyset$
for $i = 1, ..., n$ **do**
  $x_{rand} \leftarrow \texttt{SampleFree}$
  $x_{nearest} \leftarrow \texttt{Nearest}(G = (V, E), x_{rand})$
  $x_{new} \leftarrow \texttt{Steer}(x_{nearest}, x_{new})$
  if $\texttt{ObstacleFree}(x_{nearest}, x_{rand})$ **then**
    $V \leftarrow V \cup \{x_{new}\}; E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
  end if
end for
return $G = (V, E)$

Algorithm 8 Rapidly Exploring Random Tree Star

$V \leftarrow \{x_{start}\}; E \leftarrow \emptyset$
for $i = 1, ..., n$ **do**
  $x_{rand} \leftarrow \texttt{SampleFree}$
  $x_{nearest} \leftarrow \texttt{Nearest}(G = (V, E), x_{rand})$
  $x_{new} \leftarrow \texttt{Steer}(x_{nearest}, x_{new})$
  if $\texttt{ObstacleFree}(x_{nearest}, x_{rand})$ **then**
    $V \leftarrow V \cup \{x_{new}\}$
    $X_{near} \leftarrow \texttt{Near}(G = (V, E), x_{new}, \min\{\gamma_{RRT*} \log(|V|/|V|)^{1/d}, \eta\}$
    $x_{min} \leftarrow x_{nearest}; c_{min} \leftarrow \texttt{Cost}(x_{neareast} + c(\texttt{Line}(x_{nearest}, x_{new}))$
    for all $x_{near} \in X_{near}$ **do**
      if $\texttt{CollisionFree}(x_{near}, x_{new})$ **and** $\texttt{Cost}(x_{new}) + c(\texttt{Line}(x_{near}, x_{new})) < c_{min}$ **then**
        $x_{min} \leftarrow x_{near}; \texttt{Cost}(x_{near}) + c(\texttt{Line}(x_{new}, x_{near}))$
      end if
    end for
    $E \leftarrow E \cup \{(x_{min}, x_{new})\}$
    for all $x_{near} \in X_{near}$ **do**
      if $\texttt{CollisionFree}(x_{new}, x_{near})$ **and** $\texttt{Cost}(x_{new}) + c(\texttt{Line}(x_{new}, x_{near})) < \texttt{Cost}(x_{near})$ **then**
        $x_{parent} \leftarrow \texttt{Parent}(x_{near})$
        $E \leftarrow (E \backslash \{(x_{parent}, x_{near})\} \cup \{(x_{new}, x_{near})\}$
      end if
    end for
  end if
end for
return $G = (V, E)$

algorithm is closer to the shortest path. Moreover, it has been proven that these changes make the RRT* algorithm asymptotically optimal.

As also explained in Section 2.3, the sampling-based path planning algorithms do not generate flyable paths. Therefore, it is necessary to modify this algorithm in order to take into account the minimum turn radius and the flight path angles computed by the path bound generator. To ensure it, it is proposed to use single turn curves.

Single Turn curve Dubins curves could be used to make trajectories flyable. However, the series of turns can be constraining for the pilots. It is better to generate curves with only one

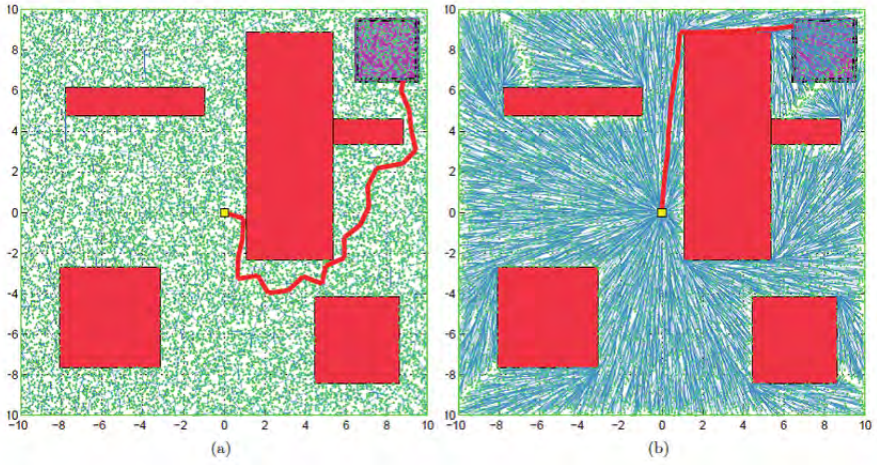


Figure 2.77: Comparison between the graph generated by the RRT algorithm (left) and the one generated by the RRT* algorithm (right) in presence of obstacle [117].

turn. This type of curve is composed of a first turn to the right or to the left and a straight line. Let $p_s = (x_s, y_s)$ and $p_f = (x_f, y_f)$ be the start and end position of the curve. Let $h_s \in [0, 360[$ be the start heading. The associated direction angle θ_s in radians is computed as follows:

$$\theta_s = \frac{\pi}{180}(90 - h_s) \mod 2\pi. \tag{2.49}$$

Let $p_s^C = (x_s^C, y_s^C)$ be the center of the circle. Let ϕ_1 be the turn angle and L be the length of the segment. It is determined as follows:

$$L = \sqrt{(x - x_s^C)^2 + (y - y_s^C)^2 - r^2}. \tag{2.50}$$

The position of the center of the circle p_s^C depends on the turn direction. It is computed as follows:

$$p_s^C = (x_s^C, y_s^C) = (x_s \pm r \sin\theta_s, y_s \mp r \cos\theta_s), \tag{2.51}$$

where the plus for x_s^C coordinate corresponds to a left turn and the minus to a right turn. On the other hand, for the y_s^C coordinate, the minus corresponds to a left turn and the plus to a right turn. The turn angle ϕ_1 is defined as follows :

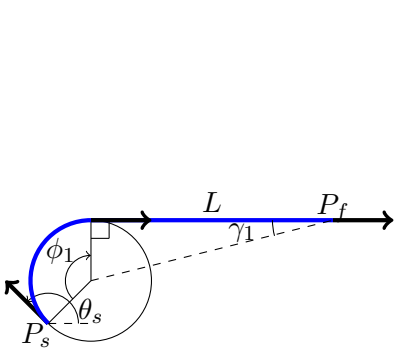
$$\phi_1 = \pm \left(\theta_s - \arctan \frac{y_f - y_c}{x_f - x_c} \right) + \psi_1 \mod 2\pi, \tag{2.52}$$

where the plus corresponds to a left turn and the minus to a right turn and:

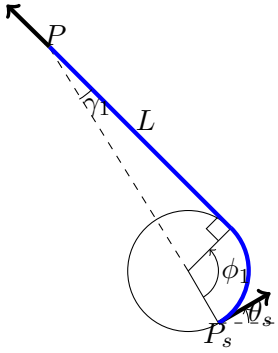
$$\psi_1 = \arcsin \frac{r}{\sqrt{(x - x_s^C)^2 + (y - y_s^C)^2}}. \tag{2.53}$$

Figure 2.78 shows the two types of single-turn curves: Right-Segment and Left-Segment.

Racetrack pattern Another feature has been added to the path generator: the addition of descent hippodromes (or racetracks) at the end of the paths, just above the final delivery point of the trajectory (See Figure 2.79). Such a feature has been requested by the expert pilot who was participating to the project. This pattern is used in the path generation algorithm when



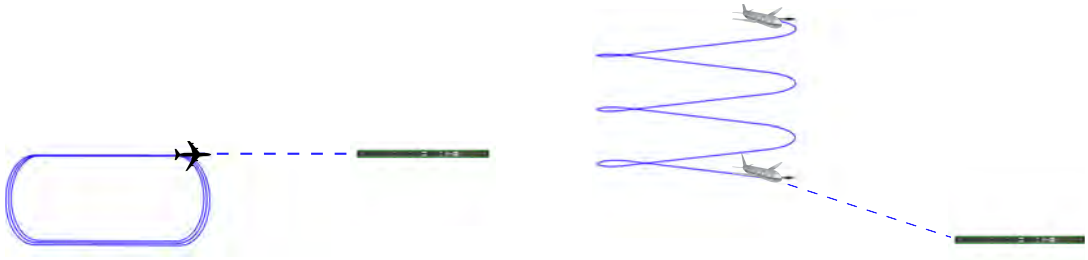
(a) Right-Segment curve: This curve is composed of a turn to the right of an angle ϕ_1 and a segment of length L



(b) Left-Segment curve: This curve is composed of a turn to the left of an angle ϕ_1 and a segment of length L

Figure 2.78: Single Turn Curve.

the aircraft is in a high-energy state and cannot perform a direct descent to the landing site. For instance, when an aircraft declares an emergency in the cruise phase, when it is flying over a suitable landing site for an emergency landing. In such a situation, the pilots would need first to perform a series of descending turns until it loses enough altitude to finish the landing. In order to reduce the pilots' workload, the shape of this procedure is the same as current racetrack procedures used in some initial approach segments [103]. They are consisting of a 180° track-change turn over a specific *anchor* fix, followed by a straight segment (typically between 1 and 2 minutes of flight), a second 180° turn, and another straight segment to overfly again the final point. This procedure is repeated until the desired altitude is reached. In the project racetrack patterns with straight segments between 4NM and 8NM (equivalent to around 1 and 2 minutes) have been considered. Those patterns are added if necessary. However, following the advice of the pilots, in the case of ANSA emergencies, a short (4NM) descent racetrack is added to the end of the path. This addition allows the pilots having 2 options. It can decide to follow the whole path. However, it can also accelerate the descent and not follow the racetrack. This decision can be guided by the worsening of the situation. For instance, an A320 declares an emergency following the loss of an engine, and during the descent, the second engine breakdowns. The emergency becomes TEFO and the aircraft trajectory must be modified.



(a) Horizontal View.

(b) Vertical view.

Figure 2.79: Hippodrome Descent.

Approach Procedure Entries to these racetrack patterns have also been considered, by following the guidelines of ICAO [103]. They are only taken into account in the case of emergencies with available thrust (i.e., at least one operative engine). The entry to the racetrack pattern is assumed to be performed depending on the course and according to the three entry sectors shown

in Figure 2.80. The procedure flown depending on the entry sector is inspired by guidelines of ICAO [103] and is as follows:

- Sector 1: on passing over the final point, the aircraft turns to a heading such that the track is parallel and opposite to the inbound track and then maintains this heading on a length equal to the length of the racetrack. Then, the aircraft turns to join the final point. If the aircraft does not reach the point at the desired altitude, it follows the descent racetrack.
- Sector 2: when the aircraft flies over the final point, it takes a heading such that the trajectory forms an angle of 30 degrees with the reverse of the approach trajectory and it follows this heading on the length of the racetrack. Then it turns to the right to join the final point and, as for Sector 1, follows the descent racetrack if necessary.
- Sector 3: if the aircraft does not reach the final point at the desired altitude, it directly follows the descent racetrack.

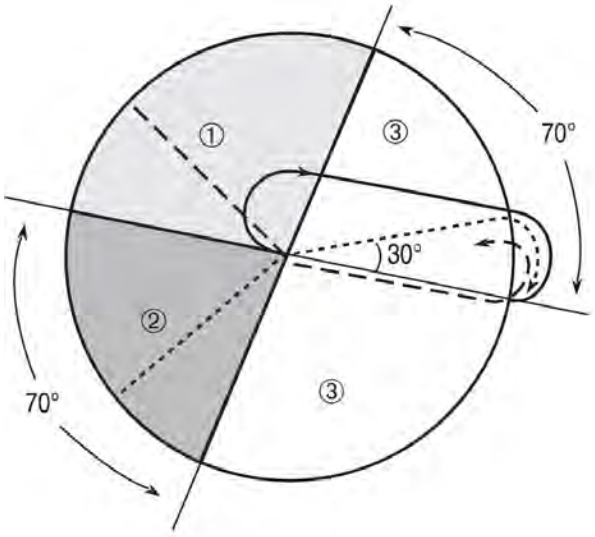


Figure 2.80: Entry procedures to the holding pattern [103].

Following the advice of the pilots, entries in the procedure are only considered for ANSA ASAP and DEPRESSURIZATION emergencies.

Weather Obstacle avoidance Only ANSA emergencies consider thunderstorms and icing areas as obstacles. In the case of their presence, the path has to avoid them with safety margins. In this project, the horizontal and vertical margins are 4NM and 2000ft respectively. This choice was made under the advice of the pilots who consider that an avoidance meneuver is a loss of time that could be fatal to the aircraft. For instance, in the case of a TEFO emergency, a suitable landing site could be unreachable due to weather obstacle avoidance.

Path Generator process The path generator process is composed of 2 steps:

1. For each landing site, test if the direct path is possible. Most of the time, this option is possible because there is no obstacle (mountain or thunderstorm). In this case, the path is only composed of a Dubins curve from the aircraft position to the final point. Then, there are two options:

- The aircraft can follow the descent profile to reach the point at the good altitude. In this case, nothing is added to the path.
- The aircraft can not reach the final point at the given altitude of the final point. In this case, one or several hippodromes are added to the path to arrive at the good altitude.

These tests significantly reduce the number of paths computed by the RRT* algorithm. As these tests are very fast, this significantly reduces the computation time. Moreover, according to the pilots of the project, the direct route is preferable to be close to the landing site as soon as possible.

2. If there are still paths to be computed, *i.e.* there are obstacles on the direct routes. The modified RRT* algorithm is running. A graph of curves is generated (See Figure 2.81). Finally, the final point is connected to the graph thanks to a Dubins curve (presented in Section 2.3) that minimizes the length of the path. Then, as for the first tests, one or several hippodromes are added if necessary.

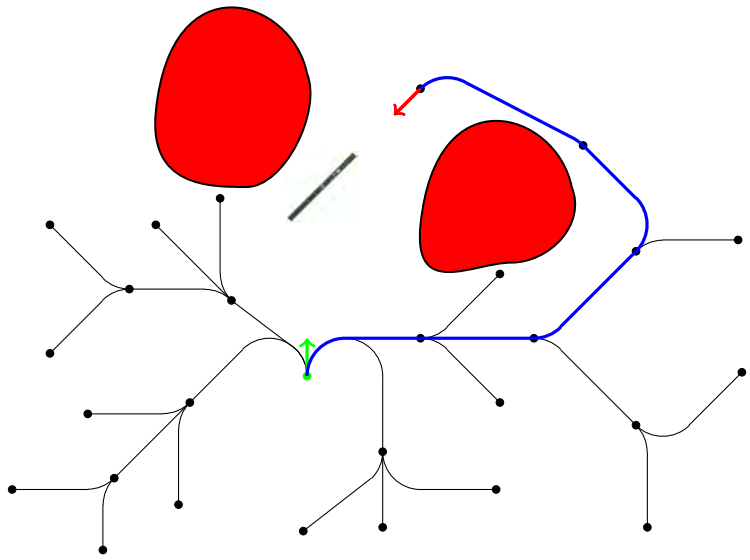


Figure 2.81: Illustration of the modified RRT* algorithm: A tree of curves is built by the algorithm and the shortest path (in blue) that avoids the obstacles (in red) is computed.

The path generated by the module is 100% flyable. It is modified by the motion planner to obtain a 4D trajectory.

2.5.4.4 Motion Planner

The role of the motion planner module is to generate a 4D trajectory by using the 3D path computed by the path generator, together with the current aircraft performance and weather. For each scenario of emergency, a predetermined profile is used to generate the 4D trajectory. The profile was presented in Section 2.5.2.

In this project, the TP logic used in both the path bounds generator and the motion planner considers the following aspects:

- Trajectory phases: the vertical profile is divided into a finite number of phases. Each phase is defined by an end condition and two aircraft intents.

- Aircraft intents: operational intents such as flying at constant Mach, CAS, altitude, throttle setting, flight path angle, etc.
- End condition: it is necessary to define the transition from one phase to another where intents might change and/or aerodynamic conditions might change.
- Trajectory computation: a 4D trajectory is specified given a sequence of phases, with the 2 intents per phase and the end condition specified.

The output of the motion planner is a 4D trajectory plan. This plan will be executed by the (auto) pilot.

After presenting the modules of the SafeNcy, the next subsection presents the results on several scenarios.

2.5.5 Results

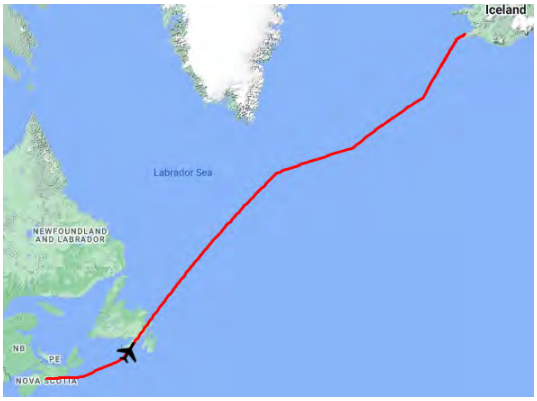
This subsection presents the results of the SafeNcy project. Five scenarios are presented and tested for different emergencies.

2.5.5.1 Scenarios

To validate the approach of the SafeNcy system, it is proposed to study the following scenario that is tested on one or several types of emergency:

- Scenario 1 (Oceanic): a ferry flight[3] from Halifax Airport (CYHZ) to Keflavik Airport (BIKF) with 15 people on board is considered (See Figure 2.82a). In this scenario, three different emergencies are tested: a failure of both engines, leading to a TEFO emergency with fuel on board (Study case 1) (in this case, the proposed emergency trajectory seeks the optimal flight conditions to attempt an engine relight); a total fuel leak, leading to a TEFO emergency with no fuel (Study case 2); and a fire on board without impact on the aircraft maneuverability (Study case 3). For the three cases, the emergency is assumed to happen when the aircraft flies over the south of the island of Newfoundland. Figure 2.82b shows the place of the emergency and the two ranges (FIRE in yellow and TEFO in red). The landing site selector provides three airports for a TEFO emergency: Saint Pierre Airport (LFVP), Winterland Airport (CCC2), and Miquelon Aerodrome (LFVM). LFVP is preferred to others because it is the highest-class airport, Class 3, featuring a 1,798-meter-long runway. The other suitable landing sites are of Class 4, with shorter runways (less than 1,000 meters). For other emergencies, two landing sites of Class 2 are available: John's International Airport (CYYT) and Gander International Airport (CYQX).
- Scenario 2 (LFLC departure): this scenario is a flight from Clermont-Ferrand Airport (LFLC) to Paris Charles de Gaulle Airport (LFPG) with 50 people on board (See Figure 2.83b). At FL80, while the aircraft is following the departure procedure, a bird strike occurs leading to a TEFO emergency with fuel on board (Study case 1). Contrary to scenario 1, the aircraft is at a very low altitude, therefore engine restart during the emergency landing is not considered. The location of the emergency is represented by a red cross in Figure 2.83a.

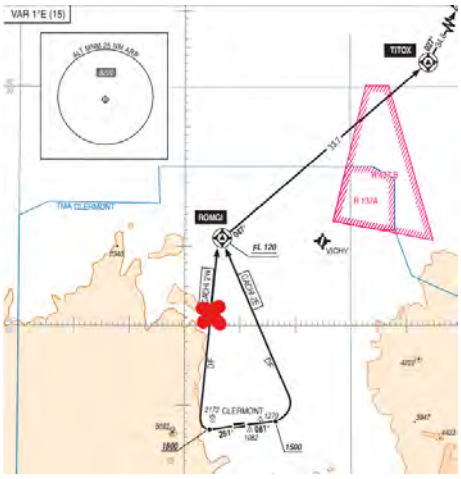
[3] A ferry flight, also known as positioning flight, generally refers to a non-revenue-generating flight operation that requires moving an aircraft from one place to another. One example is when an operator delivers an empty jet to its costumer who has ordered the plane for a charter journey.



(a) Flight Plan.

(b) FIRE and TEFO ranges in yellow and red respectively.

Figure 2.82: Scenario 1: Flight from Halifax Airport (CYHZ) to keflavik Airport (BIRK).



(a) Clermont-Ferrand Airport (LFLC) SID RNAV RWY 08 / RWY 26 and location of the emergency (Source: French AIP).

(b) Departure trajectory.

Figure 2.83: Scenario 2: TEFO emergency at the departure of a flight from Clermont-Ferrand Airport (LFLC) to Paris Charles de Gaulle Airport (LFPG).

- Scenario 3 (Mont Blanc): an Airbus A320 flies from Paris Charles de Gaulle Airport (LFPG) to Roma Fiumicino (LIRF). It declares an emergency over Mont Blanc at FL330. This scenario is tested in the case of an ASAP emergency. Figure 2.84 shows the aircraft position at the time of the emergency and the available landing sites. LSGS, LSTS, LFXA and LFLB are only used in the case of a TEFO emergency and are not considered in these tests because in the ASAP and ANSA ranges, there are several airports of Class 1 and 2. The others are suitable for every situation.

- Scenario 4 (Flight in stormy conditions): an Airbus A320 declares an emergency near Mont-de-Marsan in France (See Figure 2.85). In this scenario, the aircraft is confronted with a complicated weather situation. Indeed, big thunderstorm areas are present near Biarritz. Biarritz airport (LFBZ) is therefore unavailable. This scenario is tested on ANSA emergencies

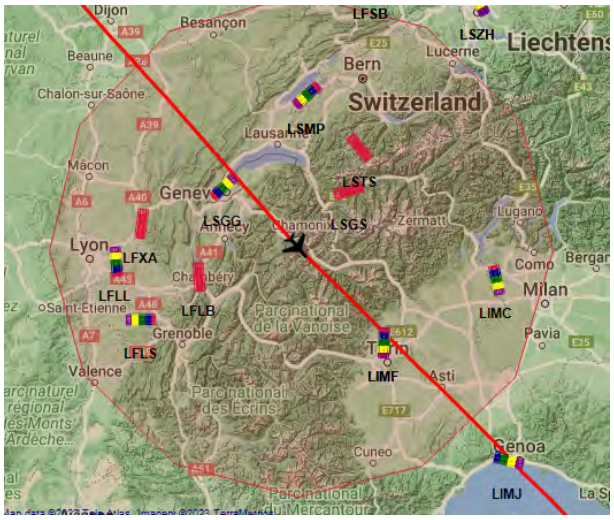


Figure 2.84: Scenario 3: Landing sites available around the aircraft position. LSGS, LSTS, LFXA and LFLB are in red because they are only used in the case of a TEFO emergency. The others are marked with all the emergency colors because they are suitable for every emergency situation.

to show the weather obstacles avoidance. As a reminder, only ANSA emergencies consider weather obstacles. In this scenario, the aircraft has to perform an avoidance maneuver to reach, for instance, Valladolid Airport (LEVD) and Zaragoza Airport (LEZG).

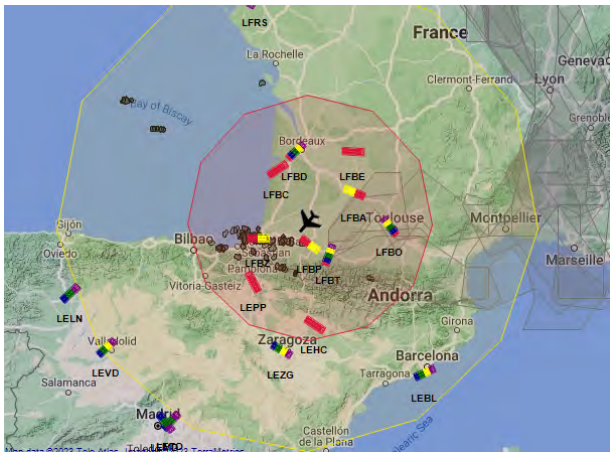


Figure 2.85: Scenario 4: Emergency near Mont-de-Marsan in France with the presence of thunderstorms (in grey) around LFBZ (Biarritz airport).

- Scenario 5 (Hudson): the last scenario is the well-known case of the sea landing of US Airways Flight 1549 on the Hudson River. This accident was presented in detail in the introduction of this chapter. As a reminder, the Airbus A320 lost its two engines at 2818ft because of a bird strike. This case is similar to Scenario 2, however, the altitude of the aircraft is even lower and the emergency is declared near a big city (New York). The number of landing sites is very small because apart from the Hudson River, no off-airport can be generated. This scenario only considers a return to LaGuardia Airport (LGA). The two runways (04/22 and 13/31) in both directions are considered as potential landing sites *i.e.* the algorithm will try to compute 4 trajectories. For this scenario, two tests are performed. The first one considers

an emergency trajectory computation just after the bird strike. The second one is running around 30 seconds later when the pilots decided to land on the Hudson River.

Thanks to these 5 scenarios, all types of emergencies are tested.

2.5.5.2 TEFO

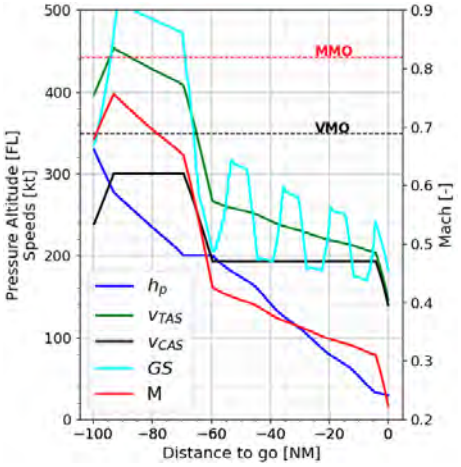
First, TEFO emergencies are tested on Scenario 1 and 2. In Scenario 1, the SafeNcy system is able to compute trajectories. Figure 2.86 shows the lateral and the vertical profile of the trajectories to Saint-Pierre Airport (LFVP) for the two considered case studies: TEFO emergency with and without fuel on board. In both cases, first the aircraft directly goes to the final point and, after that, it performs several racetracks in order to reach the desired final altitude at the delivery point. This point is located 5NM before the runway threshold at an altitude of 3250ft above the landing site. Concerning the vertical profile for the TEFO emergency with fuel on board (see Figure 2.86b), the aircraft accelerates while descending to the relight speed, which is equal to a CAS of 300kt. Then, when the aircraft reaches FL200, it flies at a constant altitude to quickly decelerate to the green dot speed. Finally, the aircraft resumes the descent at the green dot speed and finishes with a final deceleration to the approach speed with the landing configuration. On the other hand, in the case with no fuel on-board, the aircraft directly decelerates to the green dot speed. The end of the vertical profile remains the same. An easy way to see the effect of the wind in this scenario is to study the evolution of the GS. When the aircraft reaches the final point, it performs several racetracks to be at the desired altitude. These patterns imply a periodical heading change which translates into several oscillations of the GS.

Figure 2.87 shows the trajectory to Winterland Airport (CCC2) computed by the SafeNcy system for Scenario 1. This airport is of Class 4, therefore it will not be the first option proposed by the system. Indeed, LFVP is a landing site of Class 3 which means higher safety and therefore a better chance of a successful landing. The trajectories are still computed for each landing site because, in some critical situations (bad weather for instance), landing sites could not be reachable. These computations increase the running time of the SafeNcy system. Further study of the weather near airports could avoid unnecessary computations.

To summarize the results on this scenario, here is some additional information on trajectories by order of landing site preference:

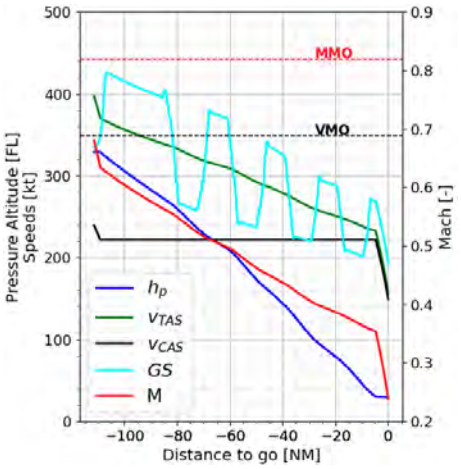
- TEFO with fuel on board:
  - LFVP: Class-3 landing site, 99NM of flight distance equivalent to a flight time of 21 minutes.
  - CCC2: Class-4 landing site, 87.5NM of flight distance equivalent to a flight time of 18 minutes.
- TEFO with no fuel board:
  - LFVP: Class-3 landing site, 111NM of flight distance equivalent to a flight time of 23 minutes.
  - CCC2: Class-4 landing site, 101NM of flight distance equivalent to a flight time of 22 minutes.
  - LFVM: Class-4 landing site, 113NM of flight distance equivalent to 24 minutes of flight time.

(a) Lateral view, TEFO with fuel on-board



(b) Vertical profile, TEFO with fuel on-board

(c) Lateral view, TEFO with no fuel on-board



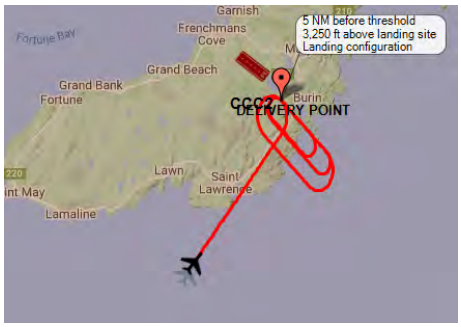
(d) Vertical profile, TEFO with no fuel on-board

Figure 2.86: Total Engine Flame Out (TEFO) for an A320 cruising at FL 330 south of Newfoundland (flight from Halifax CYHZ to Reykjavik BIRK); trajectories to Saint Pierre Airport (LFVP).

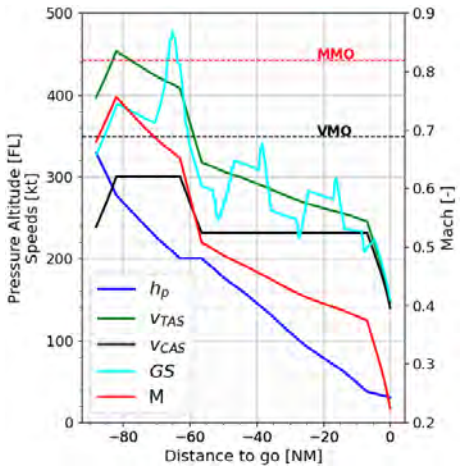
It is important to note that no trajectory was found to LFVM in the case of the TEFO emergency with fuel on board. This problem is due to the framework architecture and the interactions between the path generator and motion planner modules. For this kind of emergency, the algorithm needs several iterations to find a trajectory. This phenomenon can significantly increase the computation time that is critical for an emergency system. That is why, to limit this issue, when a sufficient percentage of trajectories are found the algorithm is stopped.

The second TEFO scenario is the departure from Clermont-Ferrand Airport (LFLC) (Scenario 2). Figure 2.88a shows the potential sites around the aircraft position for a TEFO emergency. In this scenario, only one airport is available, corresponding to the departure airport (LFLC) which is a landing site of Class 2 with a runway of 3,000 meters. The other landing sites are off-airports determined by the landing site selector as presented in Section 2.5.3. As explained in the previous scenario, the system is not able to compute a trajectory for all airports. Here, only 5 trajectories among 10 are computed. Figure 2.88b shows the resulting trajectories: in red, the trajectory to return to Clermont-Ferrand Airport, and in magenta, the trajectories leading the aircraft to off-airport landing sites.

In this scenario, the preference is to return to LFLC which is the only airport. The flight

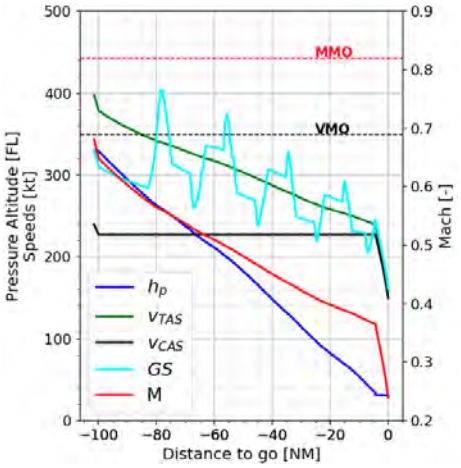


(a) Lateral view, TEFO with fuel on-board



(b) Vertical profile, TEFO with fuel on-board

(c) Lateral view, TEFO with no fuel on board



(d) Vertical profile, TEFO with no fuel on-board

Figure 2.87: Total Engine Flame Out (TEFO) for an A320 cruising at FL 330 south of Newfoundland (flight from Halifax CYHZ to Reykjavik BIRK); trajectories to Winterland Airport (CCC2).

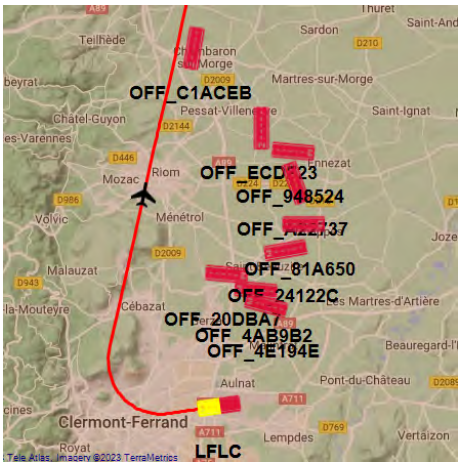
distance is 20.5NM which is equivalent to 5 minutes of flight time. In this simulation, the off-airport options are also selected to show the efficiency of the landing site selector. In a real flight, it is obvious for safety reasons that if an airport is available, the trajectory to LFLC would be the only solution.

For both scenarios, the computation time of the whole process (landing site selection and trajectory generation) is around 1 minute.

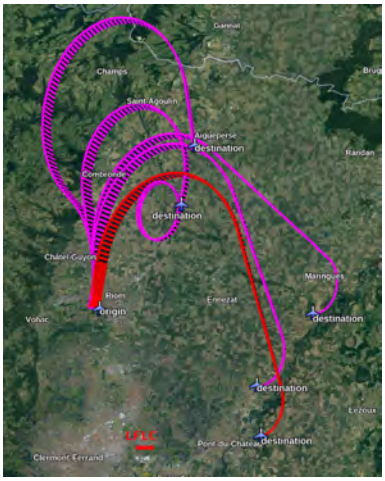
Thanks to this scenario, we show that the proposed system is able to compute trajectories in the case of a TEFO emergency. However, in some cases, all trajectories can not be computed.

2.5.5.3 ASAP

After testing TEFO emergencies, this subsection presents the result in the case of an ASAP emergency. The scenario is an emergency over the Mont-Blanc at FL330 during a flight from Paris Charles de Gaulle (LFPG) to Roma Flumicino (LIRF). Figure 2.89a shows the lateral view



(a) Available TEFO landing sites (airport and off-airport).



(b) Lateral view of emergency trajectories.

Figure 2.88: TEFO emergency for a representative narrow-body aircraft at FL80 after take-off from Clermont-Ferrand Airport (LFLC).

of the trajectory leading to Milan Malpensa Airport (LIMC). The aircraft performs a direct to the final point and follows the entry into the procedure. In this case, the aircraft enters by the sector 2 (see paragraph Approach procedure in Section 2.5.4.3). Figure 2.89b shows the vertical profile with terrain. This figure shows that the aircraft keeps sufficient safety margins with the terrain. Moreover, it shows that the aircraft descents until it reaches the altitude of 24500ft to fly as fastest as possible *i.e.* the altitude where the maximum operating limit speed (VMO) is the highest. When the aircraft is close enough to the airport, it begins its descent. In this case, the SafeNcy system is able to compute the trajectories for the 10 selected landing sites in less than 10 seconds. Figure 2.89 shows one of these 10 trajectories. The algorithm is very fast because the direct route is possible for each landing site. The RRT algorithm is therefore not run.

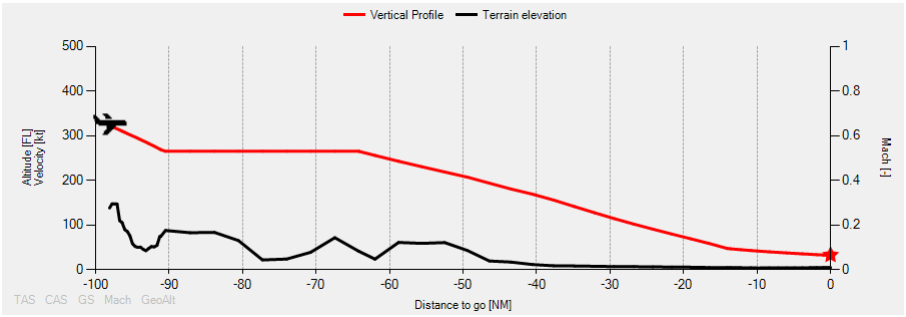
2.5.5.4 ANSA

The ANSA emergencies are tested on Scenario 4: emergency in the case of hazardous weather. Figure 2.90 shows the trajectories from the aircraft position to Valladolid Airport (LEVD) and Zaragoza Airport (LEZG). The computed trajectories avoid the thunderstorm areas (in grey) with a horizontal margin. It can be noted that the two trajectories have a descent racetrack as requested by the pilots. In this case, these airports will not be the first option because a trajectory to Toulouse-Blagnac Airport (LFBT) (Class-1 airport) is possible. This scenario illustrates that the SafeNcy system is able to generate trajectories that avoid weather obstacles. The computation time to compute the 6 ANSA trajectories is a little bit higher due to the obstacles avoidance (around 15 seconds).

2.5.5.5 FIRE

The FIRE emergencies are tested on Scenario 1. As for TEFO emergencies, the trajectories to Saint-Pierre Airport (LFVP), Winterland Airport (CCC2), and Miquelon Airport (LFVM) are computed. Indeed the TEFO range is included in the FIRE range. As shown in Figure 2.82b, several airports, such as St. John's International Airport (CYHZ), can be reachable in case of

(a) Lateral view



(b) Vertical profile with terrain

Figure 2.89: ASAP emergency for an A320 cruising at FL330 over Mont Blanc (flight from Paris Charles de Gaulle LFPG to Roma Fiumicino LIRF); trajectory to Milan Malpensa Airport (LIMC).

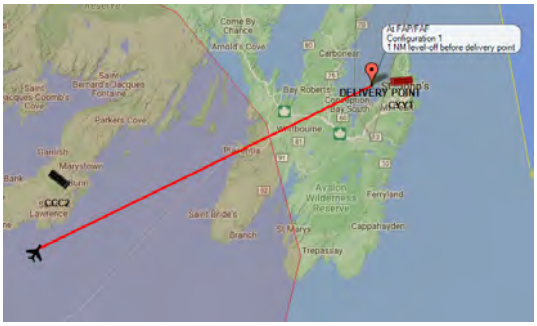
(a) Horizontal view of the trajectory to Valladolid Airport (LEVD).

(b) Horizontal view of the trajectory to Zaragoza Airport (LEZG).

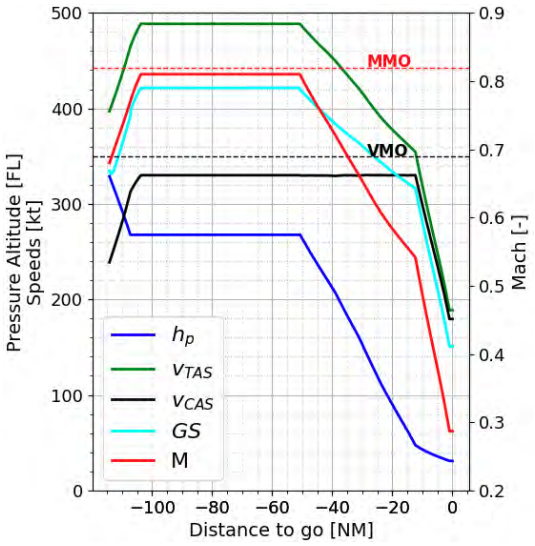
Figure 2.90: At Nearest Suitable Airport (ANSA) emergency for an A320 at FL330 near the Pyrenees with the presence of stormy areas (in grey).

a FIRE emergency. Figure 2.91a shows the lateral view of the trajectory leading the aircraft to CYYT. In this case, no descent racetrack is necessary to reach the desired altitude at the delivery point. Moreover, contrary to ANSA emergency, no racetrack is added. Figure 2.91b shows the vertical profile of the trajectory. The profile is the same as an ASAP emergency: the aircraft descents until it reaches the altitude of 24500ft to fly as fastest as possible *i.e.* the

altitude where the maximum operating limit speed (VMO) is the highest. When the aircraft is close enough to the airport, it begins its descent.



(a) Horizontal view



(b) Vertical profile

Figure 2.91: Fire emergency for an A320 cruising at FL 330 south of Newfoundland (flight from Halifax CYHZ to Reykjavik BIRK); trajectories to St. John's International Airport (CYYT).

2.5.5.6 Hudson case study

To conclude the tests of the SafeNcy system, we are interested in the well-known case of the Hudson River accident. Unfortunately, the path bounds generator does not work when the emergency is declared at a very low altitude (less than 5000ft). To overcome this problem, we studied the real flight to generate similar data as the one provided by the path bound generator. From the video in [65], we can see that the trajectory of the US Airways 1549 was composed of 2 phases. The first phase was a turn to the left during which the aircraft lost a lot of altitude (1500ft in around 40s). This loss of altitude is explained by the fact that during a turn an aircraft without engines must increase its descent gradient to maintain sufficient airspeed and avoid stalling. The second phase is a straight trajectory during which the aircraft lost less altitude. To be close to reality, in the tests, the descent angle during a turn is equal to 6° and during a straight line is 2°. As described in the presentation of this scenario, two tests are performed. Figure 2.92 shows the result of the first simulation. The path generator computes trajectories to runways 13 and 22 in less than 500ms. The second simulation does not generate a path. This coincides with the report of the investigation that showed that at the time of the decision, it was too late to return to Laguardia Airport. This test is obviously very far from reality due to the unrealistic data but it shows that the path generator works for emergencies at very low altitudes.

2.5.5.7 Future improvements

These tests have shown the efficiency of the SafeNcy system to generate emergency trajectories for different types of emergencies. However, some improvements must be done to eventually integrate the system into an FMS. Indeed, in some TEFO emergencies, due to its architecture, SafeNcy can not compute all trajectories. Moreover, in some cases, the computation time can

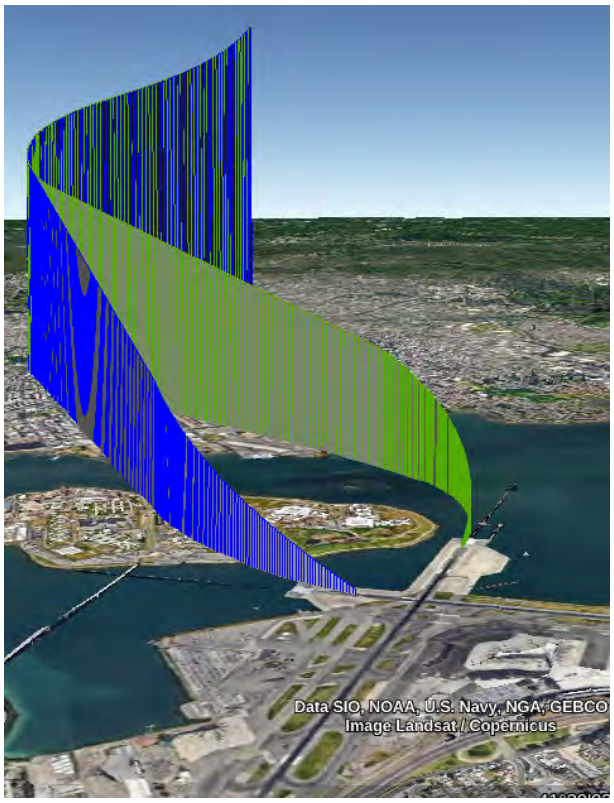


Figure 2.92: Trajectories to return to Laguardia Airport.

be a little bit high (more than 30 seconds) to allow instantaneous reaction. To reduce this time, it could be interesting to reduce the number of considered potential landing sites. Indeed, it is obvious that it is useless to generate 10 trajectories. Because, in addition to increasing the computation time, it is unintelligible to the pilots to have so many trajectories displayed on their screens. However, in order to reduce the number of landing sites considered, it is necessary to guarantee that the system computes a trajectory, which is not actually the case. The final conclusion of the project is still very positive with enthusiastic feedback from the pilots on the trajectories obtained.

2.6 Conclusions and perspectives

In this chapter, we presented the work done related to the emergency trajectory design. This work was guided by the European project: SafeNcy. Three methods have been explored during this thesis: the Fast Marching Tree algorithm, Fast Marching methods, and the Rapidly-exploring Random Tree algorithm. The two first methods show promising results and open the way for use on other trajectory optimization problems. However, for the emergency trajectory design, they show their limitations. Indeed, the Fast Marching methods do not guarantee to have a 100% flyable trajectory and the Fast Marching Tree can not satisfy the constraints chosen in the SafeNcy project. The Rapidly-exploring Random Tree algorithm was therefore chosen as the method for integration into the SafeNcy system. This algorithm was adapted thanks to single-turn curves and Dubins curves to consider curvature constraints. Moreover, this method works regardless of the descent profile, *i.e.* the type of emergency situation. Moreover, other operational features have been considered in the algorithm. Descent racetrack patterns have been added in case of over-energy and ANSA situations. The approach procedures have been

also taken into account in the case of ASAP, ANSA, and DEPRESSURIZATION situations. The SafeNcy system was tested on all types of emergencies. It seems to work very well in many scenarios. Indeed, it rapidly computes several trajectories for each type of emergency. However, in some specific situations (TEFO), the system does not generate a trajectory for all available landing sites. The conclusion of the project is nonetheless very positive with many encouraging feedbacks from the pilots. Finally, the conclusion on the research on this problem is really positive with satisfactory results for all methods. Moreover, the first use of the FMT* algorithm in aviation looks very promising for future research into aircraft trajectory design. In fact, the next chapter presents work, a local conflict free trajectories generation with weather hazards avoidance based on the FMT* algorithm.

Chapter 3

Local Conflict Free Trajectories Generation with Weather Hazards Avoidance

In this chapter, the work related to local conflict free trajectories generation with weather hazards avoidance is presented. This research aims to develop a collaborative solution framework to help pilots to avoid weather obstacles without creating conflicts. The goal is to cure a complex situation to avoid making it critical and being potentially in emergency.

3.1 Introduction

For the last decades, the European air traffic management system has remained the near same while traffic levels were increasing steadily towards saturation conditions. In 2004, the European Commission started the Single European Sky in order to overcome the airspace segmentation and its negative effect on airspace capacity. The purpose of this project is to structure space and navigation services at European level. A similar program, Next-Gen, was launched in the United States. Since then, European and US authorities have been concerned with the interoperability between these future traffic management systems. While improved coordination and information sharing through collaborative decision making (CDM) between the Air Traffic Management (ATM) actors has already been implemented, new concepts to be made effective on a longer term have been considered. Among them, 4D trajectory and Free-Flight open new perspectives to optimize air traffic management [23, 61, 62, 102]. During a flight, an aircraft can encounter unexpected events that can disrupt the flight plan: weather obstacles, conflicts between aircraft, or "mechanical" failures. In the last case, the pilots often have to land as soon as possible. This problem has been addressed in Chapter 2. In the two other cases, the pilots led by air traffic controllers have to find solutions to avoid hazardous areas or to solve conflicts. The resolution should be done fast, that is why these solutions are solely optimal. Moreover, in some very critical situations, it could be difficult to rapidly react and this can impact aircraft safety. Thunderstorms and convective areas are the source of many hazards for aircraft. Indeed, thunderstorms and convective areas are the cause of turbulence and even accidents. For instance, in 1977 the Southern Airways Flight 242 crashed which caused 63 deaths on board and 9 on the ground. Flying from Huntsville-Madison County Jetport to Hartsfield Atlanta International Airport, the flight crew was forced to land on Georgia State Route 381 in New Hope, Georgia USA, after losing thrust on both engines in a severe thunderstorm [152]. Although many safety

systems have been added to aircraft, conflicts can also lead to accidents. For instance, on July 1^{st} 2002, a Boeing B757-200 and a Tupolev TU154M collied in mid-air over Überlingen [34]. The official investigation by the German Federal Bureau of Aircraft Accident Investigation (BFU) (German: Bundesstelle für Flugunfalluntersuchung) identifies a number of shortcomings on the part of the Swiss air traffic control service responsible for the sector concerned as the main cause of the collision, as well as ambiguities in the procedures for using the Traffic Alert and Collision Avoidance System (TCAS). This highlights the importance of developing decision support tools to facilitate the work of controllers and pilots and thus guarantee safety.

The objective of this research is to develop a solution framework to design, for a foreseen set of flights, efficient trajectories taking into account weather conditions while avoiding traffic conflicts. The aim is also to maximize the exchange of information (position, speed, route,...) between aircraft to enhance communication between them.

This chapter is organized as follows: first, Section 3.2 presents two safety systems involved in the problem studied. Then, Section 3.3 analyzes the studied problem. Section 3.4 presents the solution framework to generate local conflict free trajectories with weather hazards avoidance. Then, first simulation results on a designed case are presented in Section 3.6. A discussion on the fairness of the method is carried out in Section 3.7. Finally, a fairer method is presented in Section 3.8

3.2 Flight guidance alarms

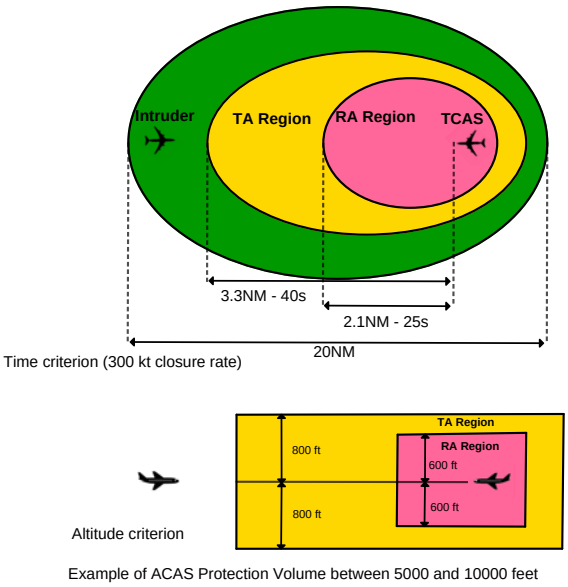
While performing the flight plan, the aircraft trajectory may present local changes to avoid different threats. Guidance alarms have been progressively implemented on board transportation aircraft. Today, the main guidance alarms on board transport aircraft are relative to the threat of collision with the ground, of traffic collision with flying devices, and of crossing weather hazards areas [149]. The Ground Proximity Warning System (GPWS) is of little interest in the present study since it generates visual and aural warnings in abnormal flight situations in the vicinity of the ground. Its alarm is activated by an excessive rate of descent, a too high ground closure rate, altitude loss after take-off or go-around, an unreasonable gap with glide slope at landing, and abnormal landing configuration. Alarms related to possible traffic collisions and too bad weather conditions along the track are of direct interest here.

3.2.1 The Traffic Collision Avoidance System

The Traffic Collision Avoidance System (TCAS) generates visual and aural warnings as well as resolution advice. The TCAS interrogates every second the transponder of neighboring aircraft, "intruders", flying in the nearby (maximum range of 30-40 NM, maximum difference of altitude of 9900 feet). From the received information, it determines for each intruder the relative bearing, its range and closure rate, and its relative altitude. This information may be integrated into the navigation display (ND) in mode ARC (see Figure 3.1a).

Then, the TCAS computes the trajectories of the intruders, the Closest Point of Approach (CPA), and the estimated time (TAU) to reach the CPA. Each time there is a collision threat, aural and visual alerts, and advisories are activated. The TCAS optimizes vertical orders to ensure a sufficient separation between trajectories and to guarantee a minimal vertical speed variation, considering all intruders. When the separation of aircraft diminishes, Traffic Alerts (TA) are sent to the pilot and when separation continues to decrease, Resolution Advices (RA) are produced (see Figure 3.1b) to help the pilot to take manual action.

(a) TCAS Traffic Information on ND.



(b) TCAS Protection, alarms and resolutions volumes: a protected volume of airspace surrounds each TCAS equipped aircraft. The size of the protected volume depends on the altitude, speed, and heading of the aircraft involved in the encounter.

Figure 3.1: TCAS.

Some TCAS systems are already able to cooperate so that the RAs given to the pilots are coordinated. International regulations indicate that in the case of a conflict between RAs and the ATC directions, the RAs of the TCAS have priority.

3.2.2 The Weather Radar and Predictive Wind Shear System

While the Predictive Wind Shear System operates only at low altitudes, the Meteo Radar can be active all along the flight. In weather mode, it allows visualizing the precipitation areas with color codes to represent their intensities, from black (lower intensity), green, amber, and red (higher intensity), as well as the turbulence areas in magenta. The weather radar has today a mere alert function towards the pilot with respect to immediate bad weather conditions. Comparing the planned trajectory with the position of the detected weather hazards, it will be to the pilot to decide to modify or not, the course of the aircraft. Figure 3.2 presents an example of Meteo Radar information available on the navigation display in mode ARC.

From above it appears that these two alarm systems are independent from the automatic guidance function developed by the Flight Management System (FMS) although they use the planned trajectory locally for prediction as can be seen in Figures 3.1a and 3.2. Today, when both threats are present, TCAS traffic information can be given in the pilot ND while local weather information can be given in the co-pilot ND.

The goal of this study is to propose a decentralized coordinated weather and traffic avoidance algorithm for several aircraft flying near a weather area and to avoid the potential activation of TCAS. It is therefore interesting to develop a collaborative tool in order to propose a safe

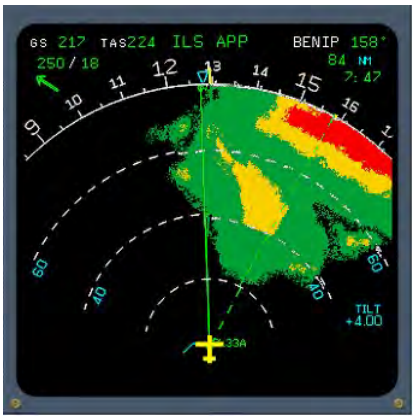


Figure 3.2: Meteo Radar information display. The colored areas correspond to cloudy areas. Those in green do not pose a safety problem. On the other hand, those in red have to be avoided.

trajectory for each aircraft and communicate it to the others. This system will use the weather radar to detect the position of obstacles.

3.3 Problem Analysis

In this section, an analysis of the problem is made. The considered problem is composed of two sub-problems: the local deconflicting problem and the weather avoidance problem.

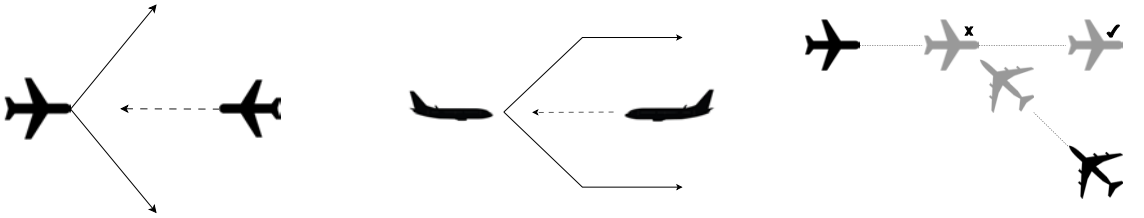
3.3.1 Analysis of the Deconflicting problem

From the mathematical programming point of view, the considered problem has been tackled as a constraint satisfaction (zero traffic conflict) problem with performance requirements (limited modifications from the reference flight plan). Here, the concept of an exact solution for this problem is not clear for the following reasons:

- No global objective is clearly defined. It would be possible to define one but this would create problems of equity between the different flights and subsequently airlines. Making it fair would imply adding new constraints and therefore increasing complexity.
- The separation constraints induce, through the Euclidean norm, the non convexity of the feasible domain and then the possible existence of local minima if an optimization criterion is introduced.
- The dimension of this problem can be extremely large when considering real case applications (tens of flights with tens of waypoints) and computation time may become an issue.

A conflict can be resolved by using one or a combination of the three following maneuvers:

- Change the horizontal trajectory of the aircraft (See Figure 3.3a). This action is preferred by air traffic controllers. They order the pilots to change their heading by giving an accurate heading or a turn of a given angle.
- Change the flight level of the aircraft (See Figure 3.3b).
- Change the speed of the aircraft (See Figure 3.3c).



(a) Heading avoidance manoeuvre.

(b) Vertical avoidance manoeuvre.

(c) Speed avoidance manoeuvre.

Figure 3.3: Different types of avoidance manoeuvre.

3.3.2 Analysis of the weather avoidance problem

When hazardous weather is detected by the radar, it is essential to avoid it. However, its position is uncertain. Indeed, during the avoidance maneuver, the thunderstorm zone moves and can impact the aircraft's trajectory. In most cases, pilots decide to go upwind in order to anticipate the storm's movement. Indeed, a thunderstorm moves most of the time in the direction of the wind. Consequently, during the avoidance maneuver, the obstacle moves away from the trajectory. Due to its dynamicity, the trajectory must be regularly updated to ensure avoidance in case of uncertain storm movement. Unlike conflict resolution, the only way to avoid a stormy area is mainly by using a horizontal maneuver. The goal is to find an efficient solution to avoid a weather obstacle *i.e.* a solution that minimizes the deviation time of the maneuver.

In this work, we propose to design, for a set of flights, efficient horizontal thunderstorm avoidance, and conflict avoidance trajectories. The next section presents the proposed algorithm.

3.4 Resolution Algorithm

The proposed algorithm is based on the Fast Marching Tree algorithm [108] which was presented in Section 2.3. To consider the dynamic of the weather obstacle, the algorithm consists of a succession of Fast Marching Tree propagation with an adaptive sampling to regularly update the trajectory.

3.4.1 Adaptation to dynamic version for one aircraft

To take into account the dynamic of the weather obstacle, it is necessary to have a dynamic version of the Fast Marching Tree algorithm. We propose to regularly update the trajectory. Let Δt be the trajectory update time step.

In the following, the trajectory of an aircraft a will be noted P_a and will be composed of a set of 3D (2D + time) points $p_{a,i}$ and is defined as follows:

$$P_a = (p_{a,1}, p_{a,2}, ..., p_{a,n}), \tag{3.1}$$

where:

$$\forall i \in \{1, 2, ..., n\},\ p_{a,i} = \begin{pmatrix} x_{a,i} \\ y_{a,i} \\ t_{a,i} \end{pmatrix}, \tag{3.2}$$

such as:

$$\begin{cases} t_{a,1} = 0 \\ t_{a,i+1} = t_{a,i} + \dfrac{\sqrt{(x_{a,i+1} - x_{a,i})^2 + (y_{a,i+1} - y_{a,i})^2}}{GS_{a,i}} \end{cases}, \tag{3.3}$$

where $GS_{a,i}$ is the ground speed of aircraft a between the points $p_{a,i}$ and $p_{a,i+1}$ defined as follows:

$$GS_{a,i} = TAS_a \sqrt{\left(\cos\psi_{a,i} + \frac{W_{x,i}}{TAS_a}\right)^2 + \left(\sin\psi_{a,i} + \frac{W_{y,i}}{TAS_a}\right)^2}, \tag{3.4}$$

where TAS_a is the true airspeed of aircraft a assumed constant, $W_i = (W_{x,i}, W_{y,i})$ the mean wind speed between the points $p_{a,i}$ and $p_{a,i+1}$ computed as follows:

$$W_i = \left(\frac{W_x(p_{a,i}) + W_x(p_{a,i+1})}{2}, \frac{W_y(p_{a,i}) + W_y(p_{a,i+1})}{2}\right), \tag{3.5}$$

and $\psi_{a,i}$ the direction angle defined as follows:

$$\psi_{a,i} = atan2(y_{a,i+1} - y_{a,i}, x_{a,i+1} - x_{a,i}). \tag{3.6}$$

As explained in Section 3.3.2, most of the time pilots decide to go upwind. To promote upwind maneuver, the thunderstorm area is enlarged in the wind direction (See Figure 3.4). The stronger the wind, the bigger the obstacle. In the case where avoidance from the right is as costly as from the left, the addition of such an enlarged area forces the upwind maneuver. However, if the upwind avoidance cost is too high, the maneuver will be done downwind. Stormy areas are also extended all around by 10NM to satisfy separation distance and to ensure safe avoidance.

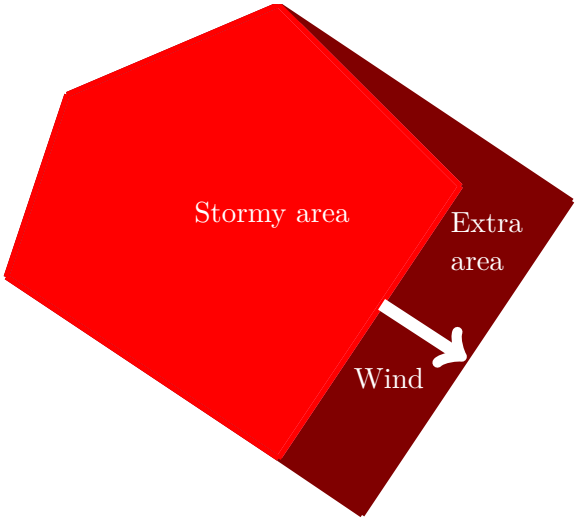


Figure 3.4: Enlargement of the storm area in the wind direction.

The trajectory generation algorithm is composed of several steps and works as follows:

1. A reference trajectory P_a^{init} is computed by using the FMT* algorithm with a random sampling of the whole space (See Figure 3.5). In this step, the obstacle is considered static and its size is defined as explained above.

2. From this first trajectory, an estimated travel time t_a^{max} can be computed. It corresponds to the duration of the avoidance maneuver. In most cases, this value overestimates the real

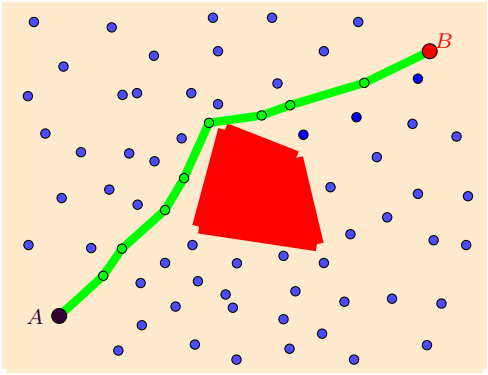


Figure 3.5: Obstacle Avoidance: the FMT* algorithm computes an obstacle (in red) avoidance trajectory between A and B by using random sampling (in blue).

avoidance time. Indeed, during the avoidance maneuver, the trajectory could be shortened because the thunderstorm moves away. The number of update iterations N_u is therefore defined as follows:

$$N_u = \left\lfloor \frac{t_a^{max}}{\Delta t} \right\rfloor . \tag{3.7}$$

3. The aircraft flies the previously computed trajectory during a time Δt. Then, this trajectory is updated. A new sampling is generated around the previous trajectory (See Figure 3.6) and the weather is updated. The new starting point is the current position of the aircraft. Samples are randomly positioned around points of the previous trajectory at a distance less than a radius r. This radius is defined from the time of the new starting point *i.e.* from the update iteration index. The closer the time is to t_a^{max} the smaller the radius r_i is. It is defined as follows:

$$\forall i \in [1, N_u], r_i = \frac{1}{N_u}(r_{final} - r_{first})\ i + r_{first}, \tag{3.8}$$

where r_{first} is the radius of the first update process, r_{final} is the final radius and i is the update iteration index. The radii r_{first} and r_{final} should not be very large, because they are just used to refine the initial trajectory. In the following, the initial radius will be 10NM, and the final radius 1NM. The radius decreases with time as the position of obstacles becomes more certain. Moreover, near the end point, it is not necessary to change the trajectory. The number of samples $n_{s,i}$ is also reduced during the process. It is defined as follows:

$$\forall i \in [1, N_u], n_{s,i} = \frac{1}{N_u}(n_{final} - n_{first})\ i + n_{first}, \tag{3.9}$$

where n_{first} is the number of samples of the first update process and the n_{final} is the final number of samples. At each update process, a new trajectory is computed with these new inputs and becomes the new reference. This process is repeated until the aircraft reaches its final point, *i.e.* the avoidance maneuver is finished. In most cases, regular updating reduces the cost of the trajectory as the obstacle moves further away. However, due to the uncertainties of the meteorological phenomenon, it is possible that the storm will get closer. In this case, the trajectory time is longer, so the maximum time t_a^{max} is modified (N_u is therefore also) to guarantee an update every Δt until the end. In both cases, Δt must be small enough to avoid an overly conservative trajectory in one case and to move far enough away from the obstacle in the other case. With a short time interval, a local modification, *i.e.* just a sampling around the previous trajectory, is sufficient because during this period the obstacle moves only slightly.

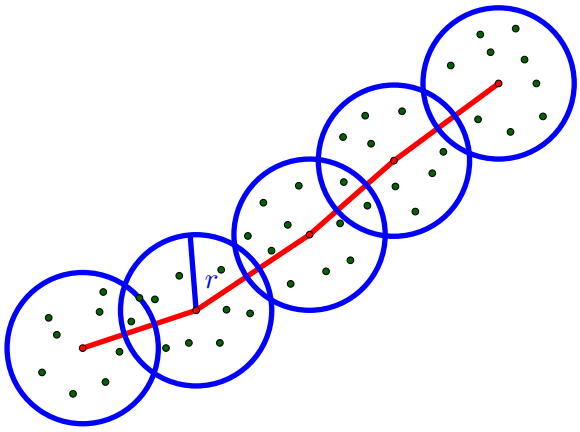
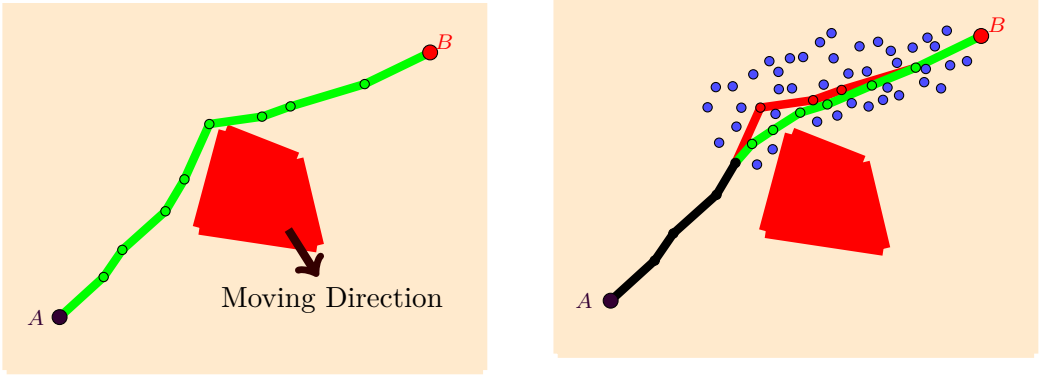


Figure 3.6: Sampling in a radius r around each point of a trajectory.

In most cases, pilots decide to go upwind in order to anticipate the storm's movement (see Figure 3.7a). Indeed, a thunderstorm moves most of the time in the direction of the wind. During the avoidance maneuver, the obstacle moves away from the trajectory. By the process explained above, the proposed algorithm can improve the trajectory to reduce the avoidance time cost (See Figure 3.7b).



(a) Initial path

(b) New path: the aircraft followed the path in red and then at a given iteration (the black route), the algorithm finds a better path in green to avoid the obstacle.

Figure 3.7: Improvement of the trajectory with the new sampling.

This section presented the algorithm to generate dynamically an avoidance trajectory. The next section presents the collaborative method to generate trajectories for a set of flights.

3.5 Collaborative trajectory generation algorithm

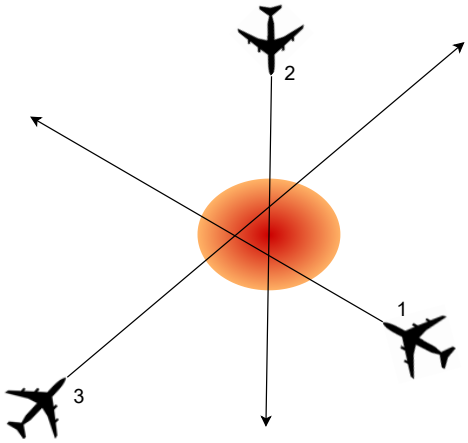
After presenting the dynamic algorithm to generate avoidance trajectory, this section presents the collaborative approach for designing efficient avoidance trajectories for a set of flights. The proposed approach works as follows:

- The aircraft are all forced to avoid obstacles. However, some are closer to the stormy areas

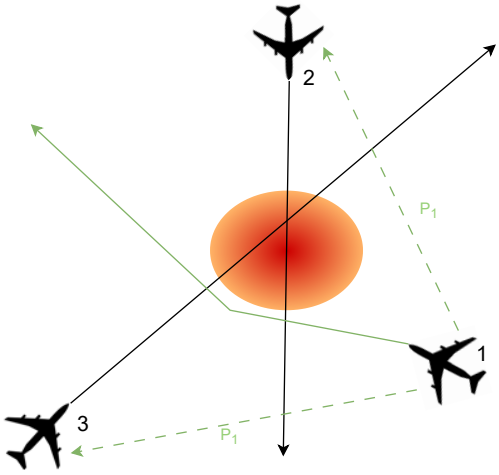
than others and must therefore react more quickly. It is proposed to treat aircraft in order of proximity to the obstacle: first-come, first-served basis. Proximity to the obstacle is determined by estimating the time before entering the storm zone if the aircraft is following its flight plan. In our case, the straight line between the start and end points will be considered to estimate this time. The position of the obstacle is considered fixed for this calculation.

- The first aircraft has just to go around the stormy areas and communicates its avoidance trajectory to the nearby aircraft which will have then to consider it as an additional moving obstacle. The next aircraft computes its trajectory and sends it to the others etc. In other words, the aircraft ranked at position n has to avoid the first $n-1$ aircraft.

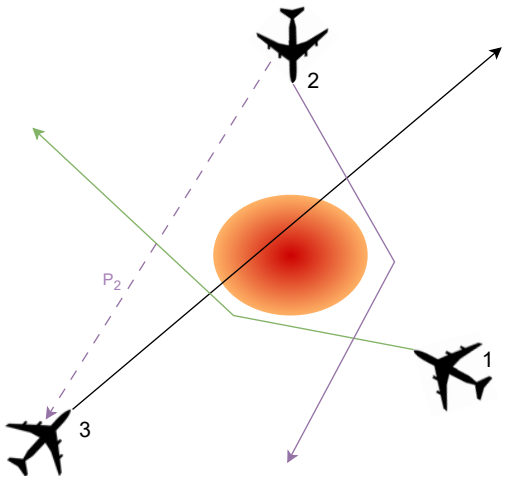
Figure 3.8 illustrates the proposed approach with three aircraft.



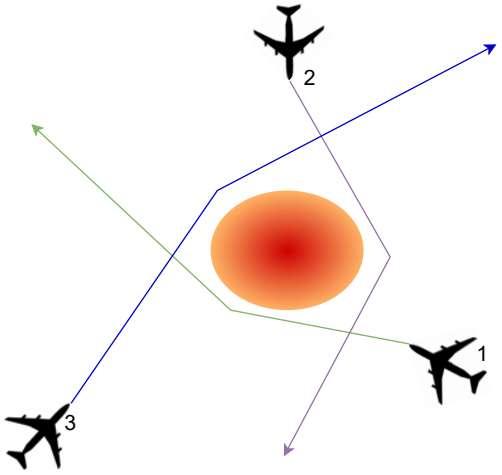
(a) Initial situation: the aircraft are sorted in order of proximity to the obstacle.



(b) The aircraft closest to the obstacle computes its trajectory and provides it to the two other aircraft.



(c) The second aircraft computes its trajectory that avoids the obstacle and the trajectory of aircraft 1 and provides it to the last aircraft.



(d) The last aircraft computes its trajectory that avoids the obstacle and the trajectories of aircraft 1 and 2.

Figure 3.8: Illustration of the collaborative method with 3 aircraft.

Before presenting the details of the algorithm (Algorithm 9), some sets and variables must be introduced. χ_{obs} represents the obstacle space. This set is firstly composed of stormy areas χ_{obs}^{w}

and then completed with intruders trajectories. P is the set of aircraft trajectories. Let P^{last}, the set of trajectories computed at the previous iteration. Let $P_{i,j}$, be the point j of the trajectory i. As explained above, the proposed collaborative algorithm (Section 4.4) gives priority to the aircraft closest to weather obstacle space χ^w_{obs}. The first step computes the trajectory of this priority aircraft. It is then considered as an obstacle by other aircraft and χ_{obs} is updated (Line 9). The following aircraft have to avoid stormy areas and previously computed trajectories. The conflict detection is done during the FMT* propagation. If at nearby times, a potential connection is close to obstacles (less than 10NM) or intersects a trajectory, it is not created *i.e.* there is a point on the segment and a point on the trajectory which are within 10NM of each other at a given time. Figure 3.9 shows an example of conflict detected during the FMT* propagation phase. At the end of this first process, the "initial" trajectories are computed. However, as explained before, the obstacle will move and the trajectories have to be updated (Lines 15-27). The aircraft trajectories are therefore recomputed in the same order based on their new position and the updated stormy areas by using the previously presented method. This process is repeated until each aircraft has reached its ending point.

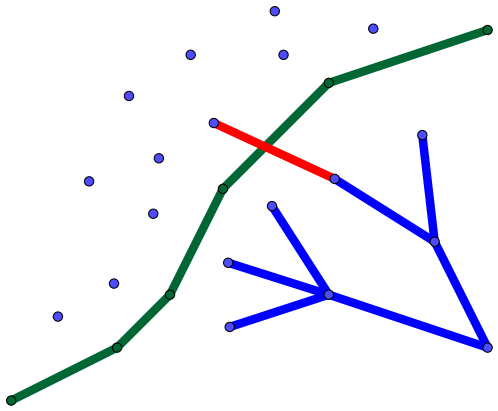


Figure 3.9: Conflict detection: During the creation of the new tree (in blue), a potential connection (in red) intersects the trajectory (in green) of another aircraft. The conflict is only detected if the intersection takes place at close times *i.e.* there is a point on the segment and a point on the trajectory which are within 10NM of each other at a given time.

This collaborative approach is tested on a designed study case, the first results are given in the next section.

3.6 First Results

3.6.1 Case study

The proposed methodology is illustrated by the case of four close flights crossing an area of 200 × 200 NM². This case study displays a rather high degree of difficulty with different opportunities for loss of separation (LOS) or storm area crossings. For each flight is given the entry point, the exit point, a common flight level, and the adopted true airspeed. Each trajectory is updated every 30 seconds. The algorithm was implemented in Java and the experiments were carried out with a computer equipped with an i5-10310U processor and a RAM of 8GB. Only four aircraft are considered because in practice it is highly unlikely that more planes are at a similar altitude in such a small area. The initial situation is shown in Figure 3.10. The red area is the obstacle

Algorithm 9 Collaborative Generation of Conflict Free Trajectories with Weather Hazards Avoidance

$X_{start} \leftarrow \{p_1^{start}, p_2^{start}, ..., p_n^{start}\}$
$X_{end} \leftarrow \{p_1^{end}, p_2^{end}, ..., p_n^{end}\}$
$O \leftarrow \chi_{obs}^{w}$
$T \leftarrow \emptyset$
for $i \in \{1, ..., n\}$ **do**
  $V \leftarrow \{p_i^{start}\} \cup SampleFree(nbSamples)$
  $P_i = FMT(p_i^{start}, p_i^{end}, V, \chi_{obs})$
  $P \leftarrow P \cup P_i$
  $\chi_{obs} \leftarrow \chi_{obs} \cup P_i$
end for
$P^{last} \leftarrow P$
$X_{start} \leftarrow \{P_{1,2}^{last}, P_{2,2}^{last}, ..., P_{n,2}^{last}\}$
$t_{max} = \max_{P_i \in P} t_i^{max}$
$t = 0$
while $t < t_{max}$ **do**
  $t = t + \Delta t$
  $k = \frac{t}{\Delta t}$
  $P \leftarrow \emptyset$
  $\chi_{obs} \leftarrow \chi_{obs}^{w}$
  for $i \in \{1, ..., n\}$ **do**
    $V \leftarrow Sampling(k, P_i^{last})$
    $P_i \leftarrow FMT(p_i^{start}, p_i^{end}, V, \chi_{obs})$
    $P \leftarrow P \cup P_i$
    $\chi_{obs} \leftarrow \chi_{obs} \cup P_i$
  end for
  $P^{last} \leftarrow P$
  $X_{start} \leftarrow \{P_{1,2}^{last}, P_{2,2}^{last}, ..., P_{n,2}^{last}\}$
end while

area and the green circles correspond to aircraft separation norms. If two circles overlap, the two planes are in conflict. In this case, the purple aircraft has priority because it is the closest to the obstacle. The priority order is 1 = purple (bottom right), 2 = orange (middle right), 3 = yellow (bottom left), and 4 = blue (top left). The wind comes from the north and is constant, the storm will therefore mainly move to the south. The scenario presented in this work was chosen to show the impact of the presence of stormy areas. Indeed, in the absence of obstacles, the planes will not be in conflict (See Table 3.1 and Figure 3.11).

3.6.2 Simulations

The addition of the storm area creates several conflicts. Figure 3.12 illustrates a simulation without considering conflicts during the graph generation. It shows that, at two different times, two circles overlap (circles in red). Two conflicts therefore occur, if the conflict detection is not considered in the optimization process.

Figure 3.13 displays the positions of aircraft at different times computed by the proposed algorithm. This figure shows that the separation distances are respected and aircraft avoid safely

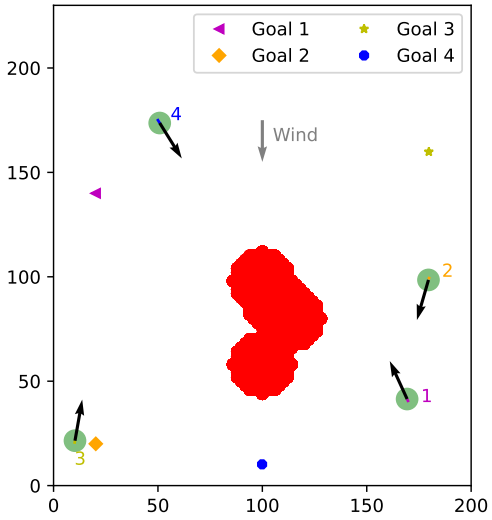


Figure 3.10: Initial Conditions: 4 aircraft are considered around the storm area and the wind comes from the north. Green circles correspond to the safety zone of each aircraft and the vectors are the aircraft speed vectors.

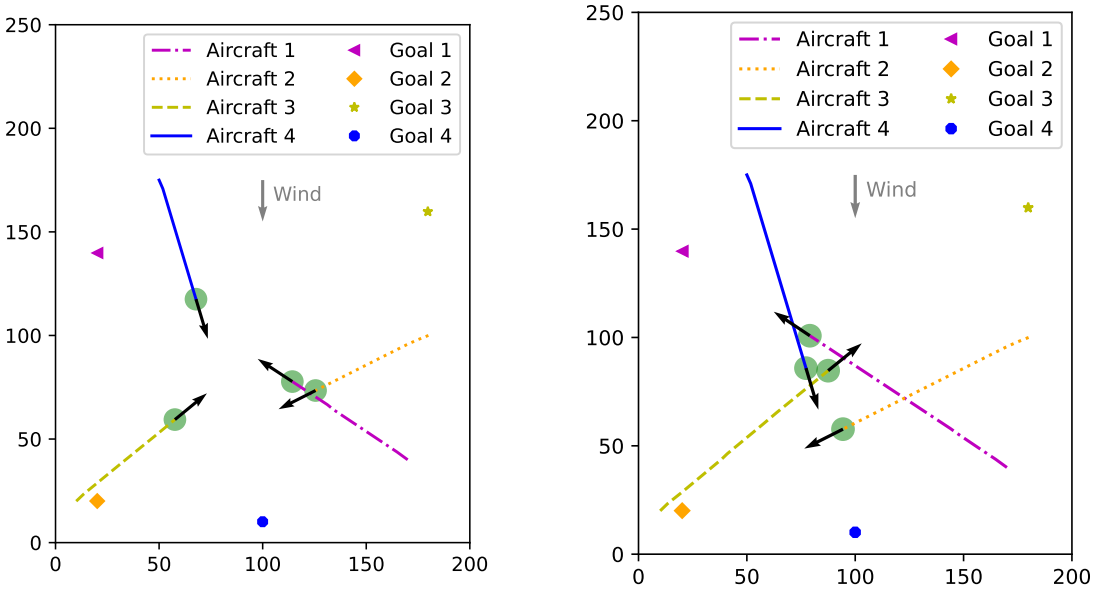


Figure 3.11: Simulations without considering weather: no conflict detected.

the weather obstacle. However, the blue trajectory has significantly deviated. This first result seems to show that the proposed method is unfair.

	Route 1	Route 2	Route 3	Route 4
Without considering obstacle or conflict	1355	1402	1733	1295
Without obstacle and considering conflict	1355	1402	1733	1295
Considering obstacles and not considering conflict	1438	1501	1770	1306
Considering obstacles and conflict	1438	1525	1976	1459

Table 3.1: Duration in seconds of trajectories depending on constraints.

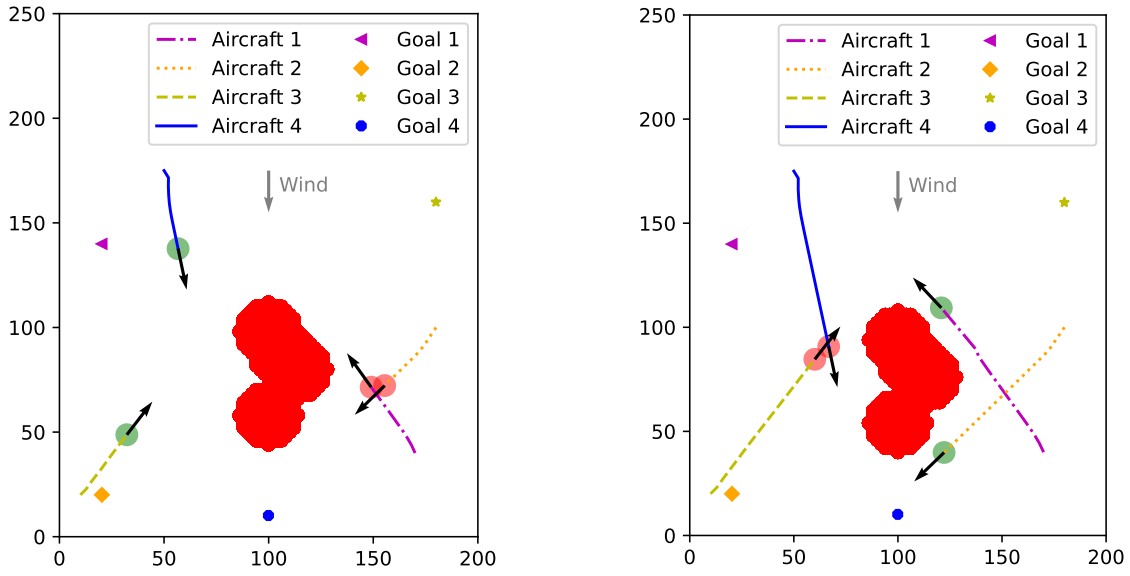


Figure 3.12: Conflicts detection: at two different times, two aircraft are at a distance lower than 10NM. Their safety zone appears in red.

Table 3.1 summarizes four different simulations and shows the duration in seconds of each trajectory. The first simulation computes the aircraft trajectories without considering obstacles or conflicts. The second simulation shows that aircraft are not in conflict without the presence of the stormy area. Indeed, the time costs are the same as without taking conflicts into account. The two last tests highlight the creation of conflicts due to the stormy area. The conflicts are solved by the proposed strategy, however, in this case, routes 3 and 4 are impacted and penalized. Indeed, their time cost is significantly higher when considering conflicts. This problem is discussed in Section 3.7.

3.6.3 Computation time

In the case of stormy areas avoidance by using the weather radar, it is necessary to have an efficient algorithm in terms of computing time. Table 3.2 shows the computing time of the first trajectories depending on the number of samples. A sampling composed of 1000 or 2000 points represents a good compromise between trajectories accuracy and computation time. Indeed, with these numbers of samples, the time taken to compute a trajectory is less than a second, while the result is close to the fastest route. If the number of samples is too small, the error in comparison with the fastest route may be high, and regular updating of the trajectory will not be able to reduce it (as shown in Section 2.3).

Initial Free Space Sampling	First Trajectories Computing Time (ms)
1000	585
2000	973
5000	9465
10000	183526

Table 3.2: First Trajectories computing time depending on number of samples.

After computing the initial trajectory, it is necessary to quickly update it. Table 3.3 shows

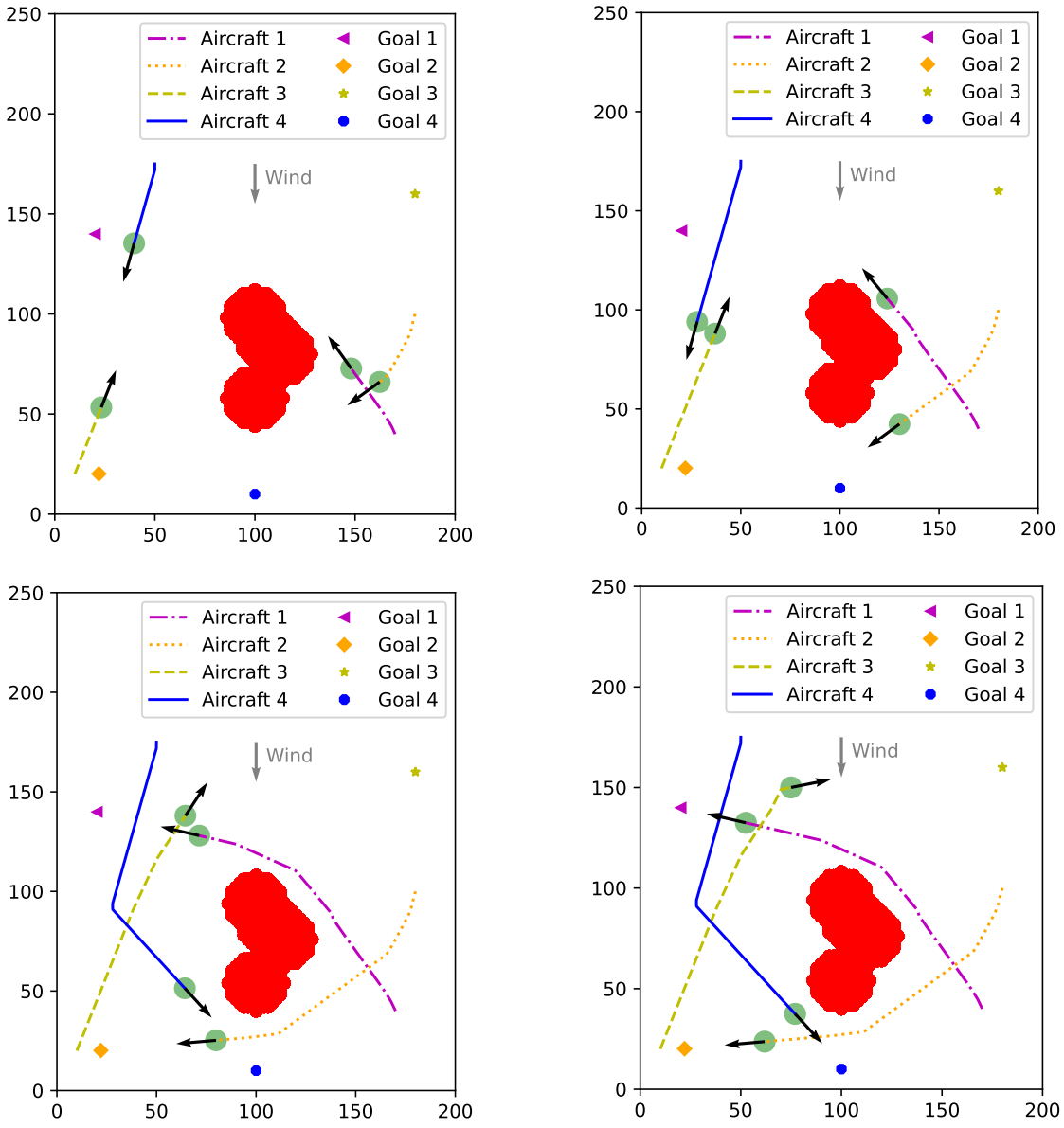


Figure 3.13: Results of the proposed method at 4 different times (First number of samples = 1000, First Update number of samples = 500). All aircraft avoid the storm area and no conflict is created. However, the blue route is very deviated.

the update process average computing time and the average improvement in terms of time cost depending on the number of samples of the first iteration. These results express that the update process is very fast and 500 samples seem to be enough to improve trajectories *i.e* reduce the time cost. This number of samples can be changed depending on the number of aircraft, the trajectory update step Δt, and the computational performance. In an aircraft, the computation will certainly be slower. A larger number of samples could hamper real-time use.

The first results seem promising. Indeed, the proposed method computes trajectories in less than 1 second and updates them in less than 100 milliseconds. It also ensures safe avoidance of stormy areas. However, some aircraft trajectories have much larger deviations. This problem is addressed in the following section.

First Sampling	Update Process Average computing time (ms)	Average improvement (%)
250	20.4	3
500	107.9	3.5
1000	1095	3.75

Table 3.3: Update process average computing time and average improvement (%) depending on the first sampling.

3.7 Fairness tests

The results previously presented show that the proposed method can be extremely unfair. To highlight this phenomenon, it is proposed to test the algorithm with all flight permutations σ *i.e* the order of priority of aircraft. In the considered scenario, 24 tests have been performed. Table 3.4 presents the deviation time $\delta t_{a,\sigma}$ of each aircraft a for all different permutations. The deviation time is computed as follows:

$$\delta t_{a,\sigma} = t_{a,\sigma} - t_a, \tag{3.10}$$

where $t_{a,\sigma}$ is the time cost of the trajectory of aircraft a for the permutation σ and t_a is the time cost of the trajectory of aircraft a considering weather but without considering conflicts *i.e.* the cost of the trajectory where the aircraft a has priority. It is proposed to use the maximum time deviation as a selection criterion. The best permutation σ_{min} is defined as follows:

$$\sigma_{min} = \arg\min_{\sigma} \max_{a} \delta t_{a,\sigma}. \tag{3.11}$$

Results show that the first-come first-served basis is not the best permutation. Indeed, in this situation, aircraft 2,3, and 4 are deviated and delayed by 24, 206, and 153 seconds, respectively. On the other hand, permutation $(4, 3, 1, 2)$ is the best one because the highest deviation time is only 26 seconds for aircraft 2.

3.8 New collaborative method

The new proposed method is based on the same exploration principle. The difference is that all permutations are computed. Although the time complexity of these computations is very high, the computing time of the initial process being low (less than 1 second) and the number of aircraft being small, the computation of all trajectory permutations is therefore feasible in a limited time. It is nevertheless proposed to speed up this computation by parallelizing it along the different planes. Consider N_a aircraft around a stormy area. The improved method works as follows:

- Firstly, each aircraft computes its best trajectory to avoid the stormy areas without considering the other flights. Each one sends its trajectory to the $N_a - 1$ other aircraft.
- At the next iteration, each aircraft computes its trajectory in case it is in the second position. An aircraft a_i has received a set of trajectories $\{P_1, P_2, ..., P_{i-1}, P_{i+1}, ..., P_{N_a}\}$. It computes for each received trajectory P_j, its own trajectory considering P_j as an obstacle. It then sends it to the other flights.
- To summarize the process, at an iteration $i > 0$, each aircraft has to compute N_i trajectories. This value corresponds to the number of trajectories where an aircraft a is at the i^{th} position.

	Route 1	Route 2	Route 3	Route 4
(1,2,3,4)	0	24	206	153
(1,2,4,3)	0	24	206	153
(1,3,2,4)	0	24	206	153
(1,3,4,2)	0	24	206	0
(1,4,3,2)	0	24	206	0
(1,4,2,3)	0	24	206	0
(2,1,3,4)	109	0	137	1
(2,1,4,3)	109	0	137	1
(2,3,1,4)	119	0	4	21
(2,3,4,1)	119	0	4	21
(2,4,3,1)	109	0	97	0
(2,4,1,3)	109	0	97	0
(3,2,1,4)	119	4	0	21
(3,2,4,1)	119	4	0	21
(3,1,2,4)	4	25	0	341
(3,1,4,2)	4	26	0	28
(3,4,1,2)	4	26	0	28
(3,4,2,1)	119	5	0	28
(4,2,3,1)	109	0	18	0
(4,2,1,3)	103	0	141	0
(4,3,2,1)	109	0	18	0
(4,3,1,2)	6	26	18	0
(4,1,3,2)	4	23	206	0
(4,1,2,3)	4	23	206	0

Table 3.4: Deviation time in second depending on aircraft permutation. The best permutation is in red.

To compute this number, we have to take $i-1$ elements among the N_a-1 other aircraft. There are $\begin{pmatrix} N_a-1 \\ i-1 \end{pmatrix}$ cases. Then, the permutations of these $i-1$ elements have to be computed; this corresponds to $(i-1)!$. At each iteration, each aircraft has to compute $(i-1)!\begin{pmatrix} N_a-1 \\ i-1 \end{pmatrix}$ trajectories.

During the whole process, each aircraft will therefore have to compute:

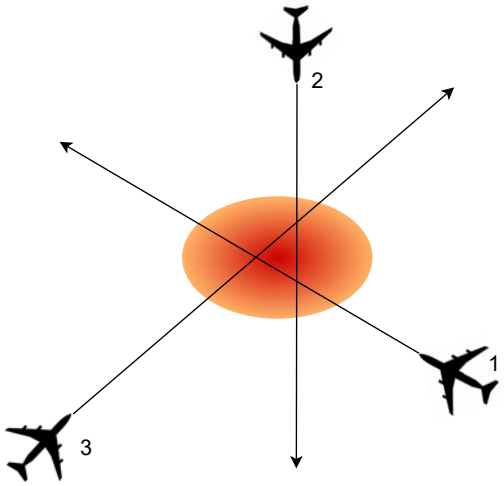
$$\sum_{i=1}^{N_a}(i-1)!\begin{pmatrix} N_a-1 \\ i-1 \end{pmatrix} \tag{3.12}$$

trajectories. This number is equal to:

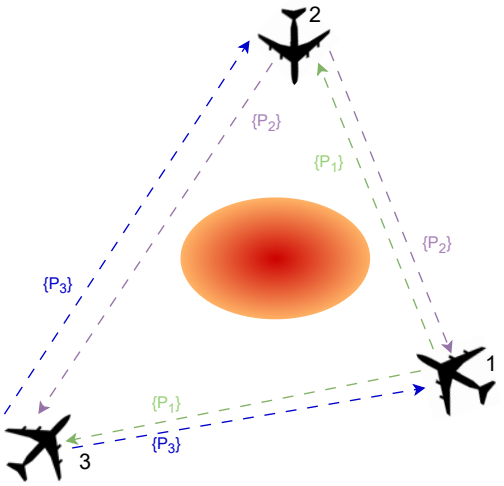
$$(N_a-1)!\sum_{i=0}^{N_a-1}\frac{1}{i!}. \tag{3.13}$$

An example of the process with 3 aircraft is presented in Figure 3.14.

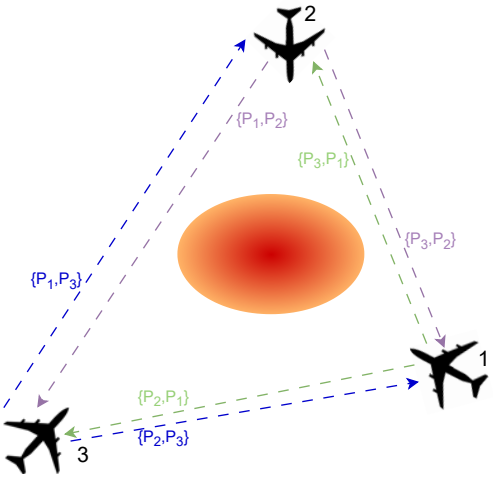
In this example, each aircraft computes 5 trajectories instead of 6 (3!). At the first iteration, each aircraft computes its trajectory without considering the others and sends it to them (1 computed



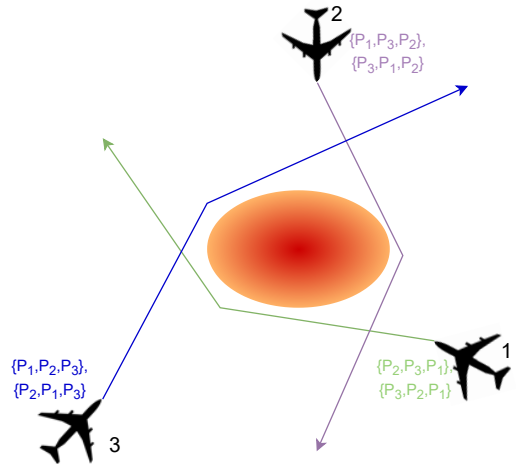
(a) Initial situation.



(b) Each aircraft computes its best trajectory to avoid stormy areas without considering conflicts. Each one provides its trajectory to the two other aircraft.



(c) Each aircraft computes its trajectory in case it is in second position and provides them to the others. For instance, the aircraft 1 computes two trajectories: the one where the aircraft 2 has priority and the one where the 3 has priority. The first is sent to the aircraft 3 and the second to the aircraft 2.



(d) Each aircraft computes its trajectory in case it is in last position. All permutations are computed. The best solution, *i.e.* the solution with the lowest maximum deviation, is selected.

Figure 3.14: Illustration of the new collaborative approach with 3 aircraft.

trajectory per aircraft). Then, from the two received trajectories, each aircraft computes its trajectory considering itself in the second position. It then sends a trajectory to one aircraft and the second to the other (2 computed trajectories per aircraft). For example, the aircraft a_2 received $\{P_1\}$ and $\{P_3\}$ and computes its trajectory in the two cases (a_1 has priority and a_3 has priority). Then, it sends $\{P_1, P_2\}$ to a_3 and sends $\{P_3, P_2\}$ to a_1. Finally, the three aircraft compute their trajectory considering themselves in the third position (2 computed trajectories per aircraft). The best permutation is then selected. For 4 planes, the gain becomes more

important, 16 computations are done instead of 24 (4!). The time complexity of the process is reduced to $O((N_a - 1)!)$ (See Table 3.5).

Number of aircraft	2	3	4	5
Permutations	2	6	24	120
Computations	2	5	16	65

Table 3.5: Permutations and computations depending on the number of aircraft.

Going back to the previous case study with 1000 initial samples and 500 update samples as parameters, the new process is able to compute the best permutation in less than 3 seconds $\left(16 \times \frac{585ms}{4}\right)$. Figure 3.15 shows the result obtained by the new approach. It is clear that trajectories are much closer to their optimum *i.e* their trajectory without considering conflicts.

In the considered case study, the new proposed process is slower than the previous one, however, it clearly reduces the maximal time deviation since it shortens the deviation of each trajectory while it is also fairer. Considering the exhaustivity of the developed search process, it can be expected that these good properties are maintained in other stormy traffic situations.

3.9 Conclusion

This study has addressed the simultaneous design, from the ground and on-board perspectives, of conflict free flight trajectories for a set of flights crossing areas subject to winds and meteorological hazards. A first approach, based on a dynamic version Fast Marching Tree algorithm and the first-come first-served basis, has been developed. According to the numerical results, this approach produces conflict free solutions but sometimes with a high degree of unfairness between aircraft. Consequently, the method has been improved by developing a collaborative process to compute all permutations of flight priority. Additional results show that the resulting method induces more fairness and reduces significantly the time deviation of each aircraft. This study opens the way to the integration of guidance protections (TCAS and weather hazards protection) into FMS with in-flight automatic resolution of potential traffic conflicts and weather hazard avoidance. This remains an important subject for future research.

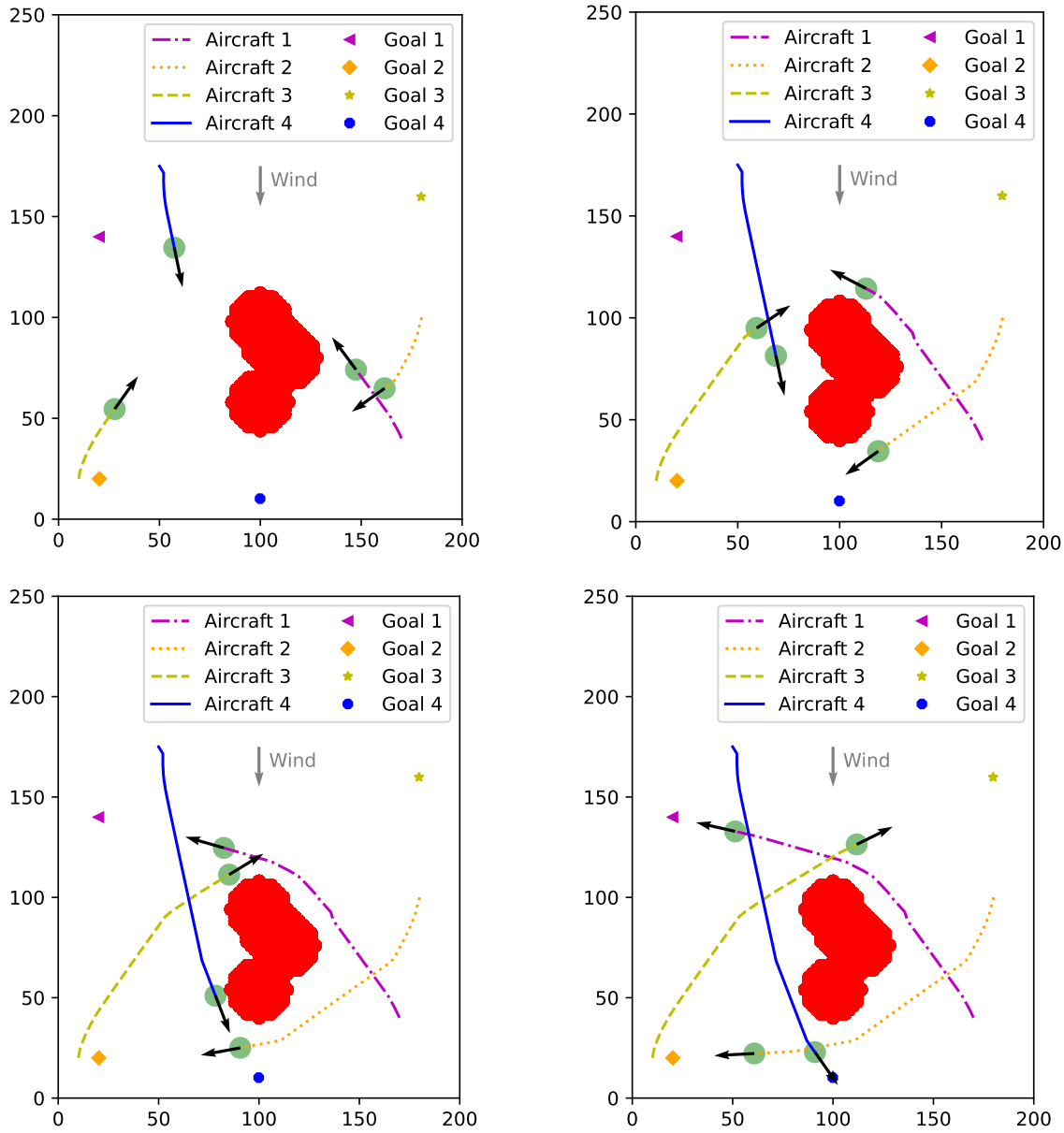


Figure 3.15: Results of the best permutation (4,3,1,2) (First number of samples = 1000, First Update number of samples = 500). All aircraft avoid the storm area, no conflict is created and the routes are little deviated.

Chapter 4

Departure and Arrival Routes Design Near Large Airports

After presenting, in the two previous chapters, work related to methods solving critical situations at the tactical level, this chapter deals with the design of departure and arrival procedures. The design of these routes, at the strategic level, is essential to guarantee safety around large airports. In this work, we propose to automatically generate these routes satisfying separation constraints but also minimizing the associated noise impact.

4.1 Introduction

Air traffic is constantly increasing, which implies a risk of congestion in the near future, thus affecting its performances [2]. The areas most affected by this growth are the Terminal Maneuvering Areas (TMA) (Terminal Control Areas in USA and Canada) which are the airspace bottlenecks. A TMA is a designated area of controlled airspace surrounding major airports where there is a high volume of traffic. To facilitate departures and arrivals of a large number of aircraft, the airports have designed specific routes called procedures. They provide paths that aircraft should follow from the runway to the en-route airspace (or the other way around). The procedures are very regulated by the Procedures for Air Navigation Services (PANS) documents such as [98]. The publication of these procedures is primarily to ensure the safety of aircraft on approach and departure by establishing vertical and horizontal margins between routes. The second objective is to reduce the workload of air traffic controllers. Indeed, these procedures reduce the number of potential conflicts between aircraft and facilitate the sequencing of arrivals. If these procedures are not followed, this can lead to accidents. This was notably the case of three flights in the past: Alitalia Flight 112 in 1972 [142], Japan Air Lines Flight 472 in 1972 [72], and Eastern Air Lines Flight 212 in 1974 [157]. Moreover, some approaches are non-compliant with the requierements of the operational documentation. These phemenoma have been identified by Jarry et al. [110, 111]. These observations highlight the importance to design optimal procedures to decrease congestion and try to reduce the number of atypical approaches. The remaining of this section presents the navigational aids and instruments that can be used near airports and the different types of procedures. Then, the objectives of this work are described. Next, Section 4.3 presents the modelling of SID/STAR routes. Section 4.4 describes the proposed algorithm based on a Simulated Annealing. Finally, Section 4.5 presents the results on the Paris TMA study case.

4.1.1 The navigational aids and instruments

Before describing the procedures, it is necessary to define some categories of navigational equipment.

- The *Distance Measuring Equipment (DME)* is a radio navigation technology that measures the distance between an aircraft and a ground station by timing the propagation delay of radio signals in the frequency band.
- The *VHF Omnidirectional Range (VOR)* is a short-range radio navigation system that gives the position of the aircraft relatively to a radial selected by the pilot.
- The *Tactical Air Navigation System (TACAN)* is a combination of a VOR and DME, with more accuracy. It is more used by the military than civilian aircraft.
- The *Non-Directional Beacon (NDB)* is a radio transmitter that does not include inherent directional information. Contrary to VOR, NDB signals follow the curvature of the Earth, they can therefore be perceived at much greater distances.
- The *Intrument Landing System (ILS)* is a precision radio navigation system that helps pilots during the approach in the case of bad weather or at night. It is composed of a Localizer and a Glide.
- A *Localizer (LOC)* is located beyond the runway and provides horizontal guidance to stay in the runway axis.
- A *Glide* provides to the aircraft the optimal descent slope to the runway .
- The *Transponder Landing System (TLS)* is a precision landing system that can be used in cases ILS does not work. For instance, in mountain areas or terrains with large obstacles.
- The *Inertial Navigation System (INS)* or Inertial Measurement Unit (IMU), is an embedded system that gives information to the aircraft on its position, speed, and orientation. It does not use exterior equipment. It just needs the aircraft's last known parameters, a computer, accelerometers, and gyroscopes.
- The *Global Navigation Satellite System(GNSS)* is a satellite-based radio navigation system that uses trilateration to compute its position, speed, and time on a receiver with high precision. It needs at least four "visible" satellites to work well. The GPS (USA), GLONASS (Russia), Galileo (Europe), and Beidou-2 (China) are some examples of GNSS.

4.1.2 Categories of procedures

Due to the high complexity of the design of the procedures, they are currently drawn by experts by hand. There are two types of procedures. The first is called Standard Instrument Departure (SID). A SID route is a flight procedure followed by aircraft immediately after take off from an airport until reaching the en-route airspace. The other type is Standard Terminal Arrival Route (STAR). A STAR is a flight route that connects the last en-route waypoint to the Initial Approach Fix (IAF) which marks the transition to the approach control area. It could also connect to the Final Approach Fix (FAF) which is the last point before the runway's threshold. A procedure is composed of a set of points with information and requirement about altitude. Usually, procedures also consist of one or several level flights. However, it is possible to perform a "continuous" procedure (Continuous Climb Operations or Continuous Descent Operations),

i.e. without a flight step at a constant altitude. Although this type of procedure significantly reduces fuel consumption and noise abatement, it requires additional margins. It is therefore difficult to use in heavy traffic conditions close to big airports.

4.1.2.1 Conventional procedures

Conventional procedures use only ground equipment. Their name is associated with the equipment they use. For instance, a STAR procedure can be VOR/DME or DME/DME. They are designed by a sequence of segments linking devices on the ground. Their precision is relatively low. For instance, a DME has a minimum error range of ± 400m. This low precision implies additional margins to ensure safety. Moreover, aircraft following conventional procedures can be scattered around the nominal path due to their low accuracy equipment.

4.1.2.2 RNAV procedures

The Area Navigation (RNAV) is part of the Performance Based Navigation (PBN). PBN is defined by its operational requirements and can use VOR or DME. An RNAV procedure is defined by waypoints which can be geographic fixes or points given in latitude and longitude coordinates. The denomination of an RNAV corresponds to the degree of accuracy available. For instance, in a system called “RNAV 1", the accuracy is 1NM around the desired path at least 95% of the flight. During the en-route phase “RNAV 10” is used and on the other hand, in the terminal areas, “RNAV 5,2 or 1” are prefered. An example of an RNAV procedure is given in Figure 4.1.

4.1.2.3 RNP procedures

The Required Navigation Performance (RNP) is also part of the PBN category. It was introduced by the ICAO in reference document 9613 [96]. As RNAV, its name is defined by its acceptable lateral error. The difference with RNAV is its reliability by on-board performance monitoring and alerting. A specific type of RNP is RNP AR, for Authorization Required [97]. This type of navigation is more precise, with RNP values going from 0.3 to 0.1 NM. This navigation is only used during the approach part of a flight and is only possible with authorization from the safety authority. This difference allows to perform complex maneuvers during landing. For example, it is possible to perform a curve route instead of a straight line route. Figure 4.2 shows the difference between conventional, RNAV, and RNP procedures.

4.1.2.4 Point Merge

In 2010, EUROCONTROL introduced in [60] a new concept called *point merge*. The goal of this concept is to decrease congestion near large airports. Usually, air traffic controllers have to deviate aircraft temporarily during their approach to merge STARs and manage the aircraft sequence (this is called *vectoring*). In some specific situations, they can also request a speed change. The *point merge* (Figure 4.3) works as follows:

- A waypoint is considered as the point merge where two STARs must merge.
- A cone is defined from the point merge and expand to the beginning of the STARs.

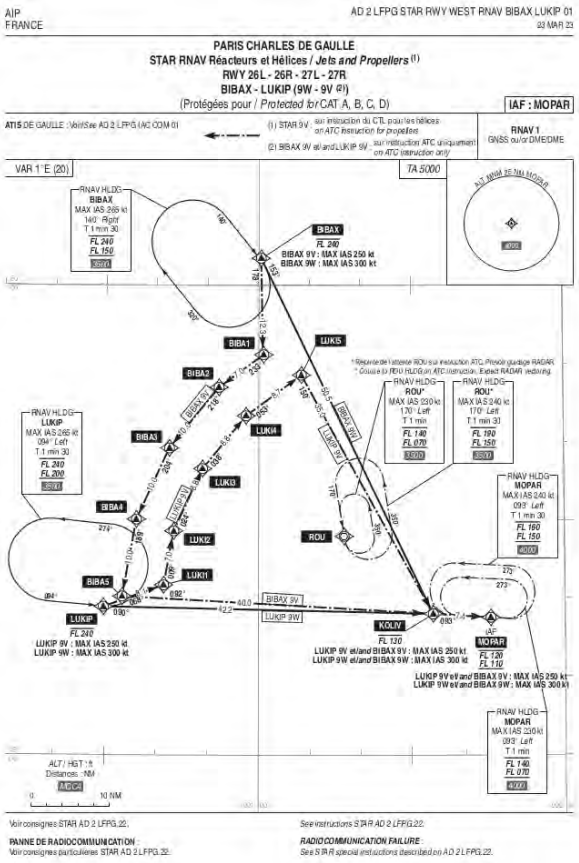


Figure 4.1: A Paris Charles de Gaulle airport RNAV procedure from [49].

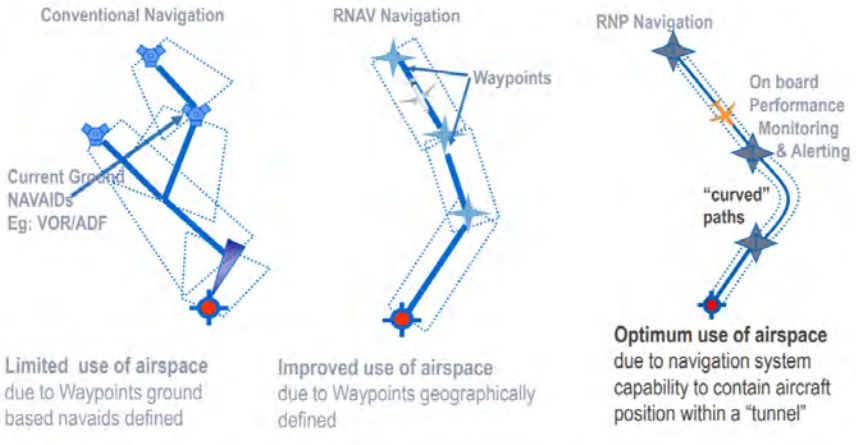


Figure 4.2: Comparison between conventional, RNAV and RNP procedures (from [3]).

- The cone is partitioned by a set of circles centered at the merge point and at different distances from it. The distance corresponds to the level of congestion: the higher the congestion, the greater the distance. This partitioning defines a set of layers.
- When an aircraft arrives on a specific leg, it must remain on that leg until the controller requests the pilot to go directly to the point merge.

This concept improves the sequencing of landings compared to the vectoring. It has already

implemented in 38 airports through 19 countries and 4 continents, such as Istanbul, Shanghai, Tokyo, Mexico or Sao Paulo.

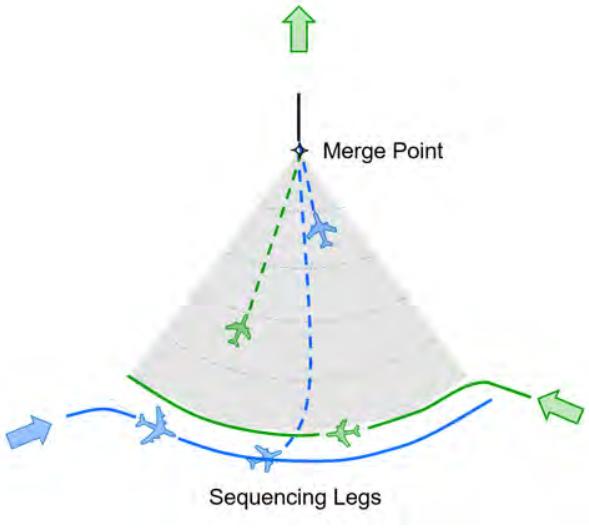


Figure 4.3: Illustration of the Point Merge concept (from [59])

The design of departure and arrival procedures is a very complex topic, due to the huge number of specific constraints. In this work, we propose to automatically design these procedures by developing an optimization process.

4.2 Objectives of this work

According to the operational requirements [1], SIDs and STARs are designed manually, and no automation has been introduced yet. Most of the airports are very close to big cities. As a result, the aircraft have to fly over dense residential areas. This can cause noise pollution for the population near the airports. The objective of this study is to automatically design safe SIDs and STARs procedures that avoid as many population areas as possible while being as short as possible. The problem is very constrained, which makes its resolution very complex. The constraints considered in this study are the following:

- **Obstacle avoidance**: This constraint is the most critical. Indeed, the designed routes have to absolutely avoid obstacles to guarantee safety. Depending on the type of instruments used to navigate, the safety margins can be different. Indeed, in the case of a Required Navigation Performance (RNP) procedure, these margins are smaller than in the case of a standard procedure.
- **Route separation**: In order to avoid conflicts between aircraft in the terminal area, routes have to be sufficiently distant from each other. They can be separated vertically (usually 1000ft) or horizontally (generally 3NM). In the case of vertical separation, it can require the use of level flights. However, those level flights have to be minimized.
- **Merging points separation**: In the case of SIDs, this constraint is not considered. However, in the case of STARs, many routes cannot merge on the same spot because the air traffic controllers' workload would be too high. Due to a large number of potential conflicts, safety would be compromized. To let controllers the time to anticipate, two merge points must not be too close to each other and be far from the entry/exit points of the TMA.

- **Noise abatement**: In some cities, it is prohibited to fly over some populated areas. In this case, the constraint is considered an obstacle avoidance problem. When flying over a population area, it is preferable to reduce as much as possible the flyover time in such area for noise abatement issues.
- **Trajectory flyability**: The SIDs and STARs design has to take into account the aircraft limitations to ensure that they will be able to follow the procedures. For example, aircraft cannot climb and descend too fast or make a turn too sharp.

4.3 Modelling

This section presents the mathematical model of the TMA routes optimization problem. These routes are based on Dubins curve and single-turn curve presented in Sections 2.3 and 2.5.

4.3.1 Input data

In this study, the following inputs are considered:

- Position and orientation of each runway in the TMA.
- All obstacles in the TMA (mountains, buildings, military zones...) defining the obstacles space χ_o. The SID/STAR routes have to avoid this obstacles space with vertical and horizontal safety margins. The obstacles are therefore enlarged by these given margins.
- The population density of a set of cities and an aircraft noise model.
- The entry and exit points of the TMA called gates which are noted $\{g_1, ..., g_n\}$. Each gate is defined by a 2D point and an altitude range.
- A minimum turn radius r_{min} to avoid too sharp turns.
- A minimum and maximum descent angle $\alpha_{min,LD}$ and $\alpha_{max,LD}$ in degrees for STAR routes and a minimum and maximum climb angle $\alpha_{min,TO}$ and $\alpha_{max,TO}$ in degrees for SID routes. As a matter of fact, the designed procedure may be used by any kind of aircraft with different climbing rates or descending rates. Those rates depend on the aircraft type, the aircraft weight, the wind, the temperature, the airlines' rules for operating aircraft, etc.
- The maximum number n^{LF}_{max} of level flights and their minimum and maximum length (l^{LF}_{min} and l^{LF}_{max}).

4.3.2 Routes design

In this study, it is proposed to design routes thanks to Dubins curves and single-turn curves. The design generates 111 routes for connecting pairs of points in the TMA. Those routes represent a set of alternative options for which one will be selected by the optimization process.

4.3.2.1 Alternative Routes

In this study, these two types of curves are used to design routes. The route connecting the point p_s and the point p_f is composed of a single turn curve and a Dubins curve. Therefore, this

route consists of three turns and two segments (see Figure 4.4). Let φ be the direction angle of the straight line which links the start point p_s to the final point p_f defined as follows:

$$\varphi = \arctan \frac{y_f - y_s}{x_f - x_s}. \tag{4.1}$$

Let d be the distance between the two points computed as follows:

$$d = \sqrt{(x_f - x_s)^2 + (y_f - y_s)^2}. \tag{4.2}$$

The proposed method generates 111 different trajectories. The first one is the direct trajectory and the others are defined using an intermediate point p defined as follows:

$$p = (x, y) = (x_s + \frac{k_1 d}{12}\cos\varphi - \frac{k_2 d}{30}\sin\varphi, y_s + \frac{k_1 d}{12}\sin\varphi + \frac{k_2 d}{30}\cos\varphi), \tag{4.3}$$

where:

$$k_1 \in \{1, 10\}, k_2 \in \{-5, 5\}. \tag{4.4}$$

Based on those two discrete sets, the number of options is given by: $10 \times 11 + 1 = 111$. Figure 4.4 shows the route defined using the intermediate point p where $k_1 = 4$, $k_2 = 5$ and $\varphi = 0$ defined as follows:

$$p = (x, y) = (x_s + \frac{d}{3}, y_s + \frac{d}{6}). \tag{4.5}$$

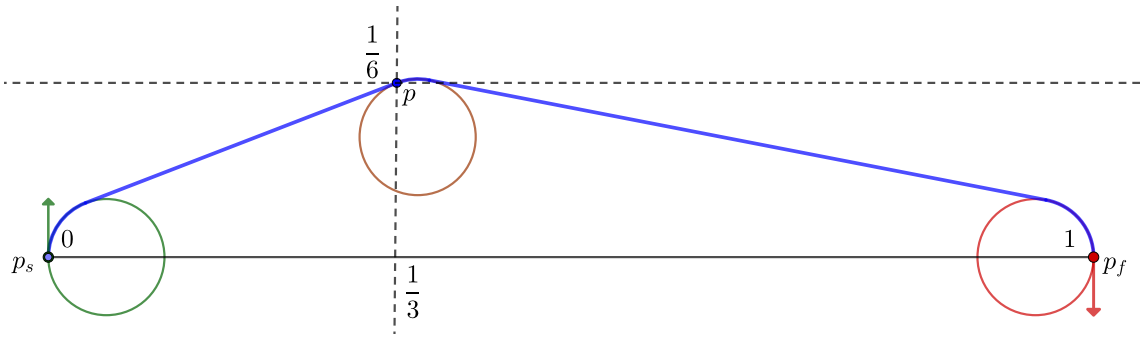


Figure 4.4: Route pattern with $k_1 = 4$, $k_2 = 5$ and $\varphi = 0$.

4.3.3 Procedures Design

A TMA consists of a set of procedures for each runway. In this work, we propose for each procedure to discretize the space by an indexed set of rings P which represents circles centered at the departure point (SID) or the arrival point (STAR) of the procedure. These rings are equidistantly distributed (See Figures 4.5a and 4.6a). Merging points and spitting points will be located on those rings.

Before introducing the specifications of each procedure, it is necessary to define some sets and notations. Let $\mathcal{P}$ be the indexed set of procedures. Let Γ_i be the routes set of a procedure i. For all $\gamma_j \in \Gamma_i$, γ_j is associated with the gate g_j. For each procedure, the routes are ordered in descending order of traffic flow. For a given route γ, let $\rho^m \in P$ be the merging ring and γ^m be the route on which γ is merged. From the horizontal path of γ^m and a merging ring, the position of the merging point on the ring can be determined. It is computed by finding the intersection between the route γ^m and the circle associated with the merging ring. For a given route, only routes with lower flows can merge on it. In other words, for all i and $j \leq i$, there can be no

$\rho^m \in P$ such that γ_i merges with route γ_j. In the case of STAR design, to avoid a heavy air traffic controller workload, only two routes can merge at a point. The decision variables of the problem are the routes to be designed. For a given route, the decision is decomposed into three sub-decisions, the first one is on the horizontal profile, the second one is about the merging and the third one is on the vertical profile. The horizontal profile is defined by the route set defined in Section 4.3.2.1. The merging decision is composed of two pieces of information the merging route and the merging ring. The vertical decision defines the presence, the location and the length of a level-off. A level-off is defined by a start ring $\rho^{LF,start}$ and an end ring $\rho^{LF,end}$. In the case of a SID, if a level-off is added between two rings, the altitude of the level-off is fixed by the mean climb ratio (blue profile in Figure 4.5). If the aircraft climbs faster (red profile), it has to stay at the level-off for a longer duration. On the other hand, if the climbing is slower (green profile), then the level-off is shorter or not needed (in some specific cases, the green profile could not have a level-off phase). As for SID, the STAR level off altitude is defined by the mean profile. If a level-off is added, all profiles have to reach the altitude at a given point (ring 3 in Figure 4.6) and then have to follow the mean profile until the FAF point.

To summarize the procedures model, each procedure decision is composed of a set of sub-decisions for each route. A route decision of a procedure i is defined as follows:

$$\begin{pmatrix} k_1 \\ k_2 \\ \gamma^m \\ \rho^m \\ x^{LF} \\ \rho^{LF,start} \\ \rho^{LF,end} \end{pmatrix}, \tag{4.6}$$

where $k_1 \in [1, 10]$ and $k_2 \in [-5, 5]$ define the horizontal route as details in Section 4.3.2.1, $\gamma^m \in \Gamma_i$ is the merging route (the first route merges into itself), ρ^m is the merging ring (the first route merges at the ring 0), x^{LF} is a boolean indicating whether a level-off is considered and $\rho^{LF,start}$ and $\rho^{LF,end}$ are the start ring and the end ring of the level flight. If a route γ_2 merges on a route γ_1, γ_2 shares the beginning of the route γ_1 from the circle centre to the merging ring ρ^m. The end of the route is defined by the variables k_1 and k_2 from the merging point to the entry/exit gate of the TMA.

4.3.4 Constraints

The two constraints of this problem are obstacle avoidance and route separation. These constraints are added to the objective function as a penalty. These constraints are represented by the following functions:

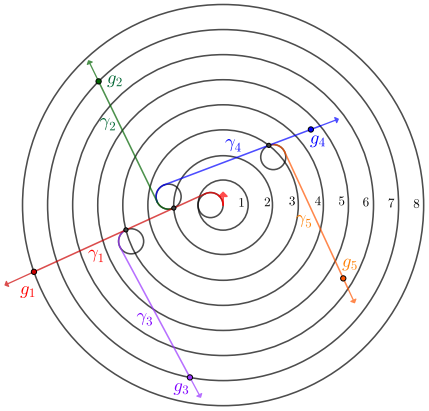
- The obstacle avoidance penalty function is defined as follows:

$$f_{obs} = \sum_{\gamma \in \bigcup_{i \in \mathcal{P}} \Gamma_i} \sum_{o \in \chi_o} c_{\gamma,o}, \tag{4.7}$$

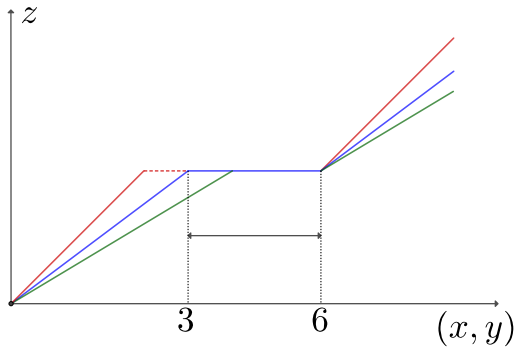
  where $\bigcup_{i \in \mathcal{P}} \Gamma_i$ is the set of all routes and $c_{\gamma,o}$ is equal to 1 if the route γ is in conflict with the obstacle o and 0 otherwise.

- Conflict: The conflict cost is the total number of conflicts between routes.

$$f_c = \sum_{(\gamma_1,\gamma_2) \in (\bigcup_{i \in \mathcal{P}} \Gamma_i)^2} \sum_{(p_j \in \gamma_1, p_k \in \gamma_2)} c^{\gamma_1,\gamma_2}_{p_j,p_k}, \tag{4.8}$$

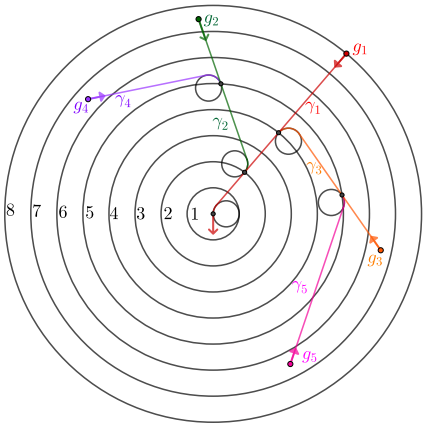


(a) Horizontal profile: γ_2 merges on γ_1 at ring 2, γ_3 merges on γ_1 at ring 4, γ_4 merges on γ_1 at ring 2, γ_5 merges on γ_4 at ring 3. In this case, straight routes have been selected among the 111 options to connect pairs of points.

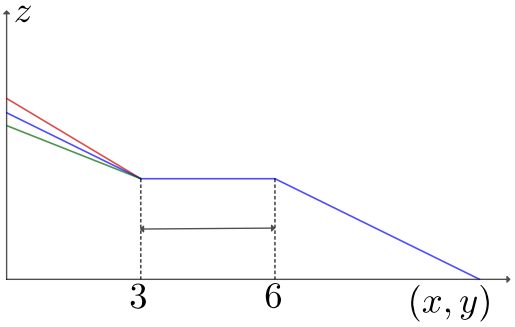


(b) Vertical profile of a route γ: a level off is added between rings 3 and 6.

Figure 4.5: SID routes: the procedure is composed of 5 gates.



(a) Horizontal profile: γ_2 merges on γ_1 at ring 2, γ_3 merges on γ_1 at ring 4, γ_4 merges on γ_2 at ring 5, γ_5 merges on γ_3 at ring 5. In this case, straight routes have been selected among the 111 options to connect pairs of points.



(b) Vertical profile of a route γ: a level off is added between rings 3 and 6.

Figure 4.6: STAR routes: the procedure is composed of 5 gates.

where $c_{i,j}^{\gamma_1,\gamma_2}$ is equal to 1 if point p_j from route γ_1 is in conflict with point p_k from route γ_2 and 0 otherwise.

The evaluation is carried out by discretizing the routes and obstacles in a 3D cube with a horizontal step of 3NM and a vertical step of 1000 feet. A conflict is detected if two points of two different routes are in neighbouring cells and their distance is lower than 3NM.

Operational constraints were also added to the problem:

- For STAR procedures, only two routes can merge at the same point. This constraint reduces the number of potential conflicts at the merging point and therefore reduces the workload of air traffic controllers.

- The turn radius r is equal to the minimum turn radius r_{min} (2NM in this study).
- The length and the number of the level flights must be minimized. The length should respect the following constraint:

$$\forall i \in \mathcal{P}, \forall \gamma \in \Gamma_i, l_{min}^{LF} \leq (\rho_\gamma^{LF,end} - \rho_\gamma^{LF,start}) l_i \leq l_{max}^{LF}, \tag{4.9}$$

  where l_i is the length between two rings of a procedure i.

4.3.5 Noise model

To define the noise abatement, it is necessary to have population data in the TMA and a noise cost function depending on the altitude.

4.3.5.1 Population data

The population data is extracted from different sources. City position data are extracted from [126], city population data are extracted from [99], and city area surface data are given by Observatoire des Territoires [100]. From this set of data, a population grid is built. A city is considered as a square of area equal to the city surface in which the population is uniformly distributed. Figure 4.7 shows an example of such grid. The black points are the cities and the color gradient represents the population density in a cell. Figure 4.8 represents the population density in the Paris TMA.

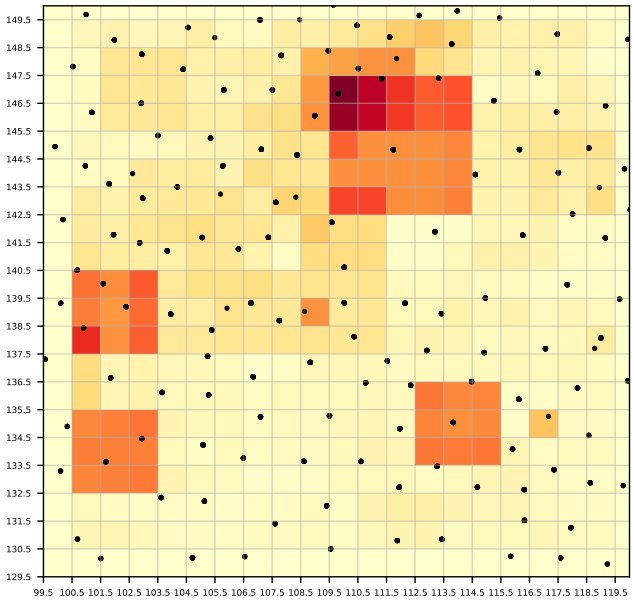


Figure 4.7: An extraction of population density grid. Black points represent the cities and the colour gradient represents the population density.

To take into account the noise impact of the SID/STAR routes, the number of people impacted by the procedures weighted by the noise intensity has to be minimized. To evaluate the noise cost

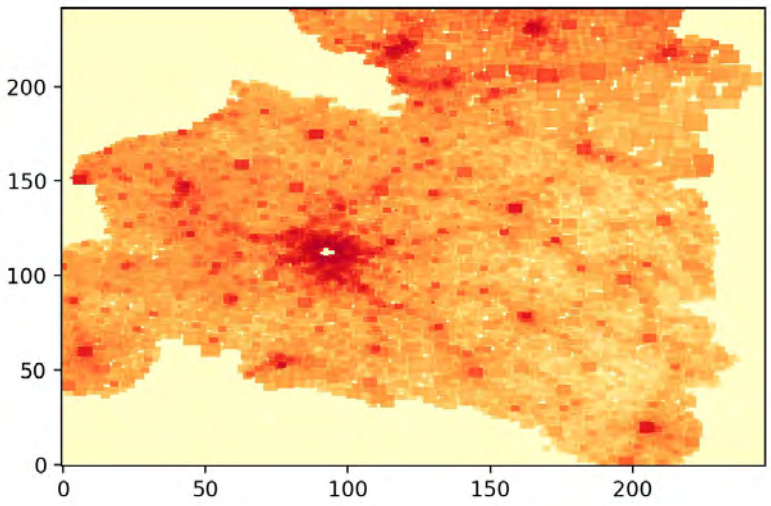


Figure 4.8: Population density around Paris Charles-de-Gaulle airport.

of a route, such a route has first to be discretized by using the 3D Bresenham's line algorithm [31, 32, 141] (presented in Section 2.3). A route is therefore composed of a set of 3D cells (i, j, z) where (i, j) is the cell in the population grid and z is the altitude of the route at that point. Then, for each cell composing the route, the noise cost is evaluated thanks to the following noise model:

$$c_n(i, j, z) = \eta(i, j) \max\left(100 - 6\frac{\ln(z/3)}{\ln 2}, 0\right), \tag{4.10}$$

where $\eta(i, j)$ is the population density in cell (i, j). Therefore, the noise cost of all routes can be computed as follows:

$$f_{noise} = \sum_{\gamma \in \bigcup_{i \in \mathcal{P}} \Gamma_i} \sum_{(i,j,z) \in \gamma} c_n(i, j, z). \tag{4.11}$$

4.3.6 Objective function

In this study, the objective function is composed of four different criteria that have to be minimized. The two first criteria represent the constraints of obstacle avoidance f_{obs} and conflict f_c. The third criterion represents f_{noise} represents the noise abatement. Note that the noise criterion f_{noise} is replaced by a noise function f^*_{noise} normalised by the average noise per NM of all routes. This change guarantees homogeneity between the noise function and the distance function. The last function is linked to the lengths of routes. It is defined as follows:

$$f_{dist} = \sum_{\gamma \in \bigcup_{i \in \mathcal{P}} \Gamma_i} l(\gamma), \tag{4.12}$$

where $l(\gamma)$ is the length of route γ. Finally, the objective can be formulated as follows:

$$\begin{cases} \min & \beta f_{obs} + \lambda f_c + \alpha f^*_{noise} + (1 - \alpha) f_{dist} \\ s.t & \text{Limited turn constraint} \\ & \text{Merge constraint} \\ & \text{Level-off constraint} \end{cases},$$

where β and λ are a high value coefficient to guarantee a feasible solution *i.e.* a solution that avoids obstacles without conflict. These values are estimated to be twice the sum of all route straight line distances.

4.4 Resolution Algorithm

4.4.1 Simulated Annealing

In this work, it is proposed to use the Simulated Annealing (SA) algorithm. It is introduced by Kirkpatrick et al. [121] and is born from an analogy with the physical annealing of materials. The process is composed of two steps, a heating phase of the solid to a very high temperature and a slow cooling phase to reach a solid state of minimum energy [46].

One of the main features of Simulated Annealing is its ability to accept transitions that degrade the objective function. At the beginning of the process, the value of the temperature T is high, which makes it possible to accept transitions with high criterion degradation, and thereby explore the state space thoroughly. As T decreases, only the transitions improving the criterion, or with a low criterion deterioration, are accepted. Finally, when T tends to zero, no deterioration of the criterion is accepted, and the SA algorithm behaves like a Monte Carlo algorithm.

The SA algorithm works as follows:

- Initialization: An initial temperature T is given and a first solution is generated.
- Cooling loop:
  - The objective function is evaluated and the result is denoted y_i;
  - A neighbor of the current solution is generated;
  - Computation of the objective function y_j of the neighbor;
  - If $y_j < y_i$, the new solution is accepted and considered as the new current point. If not, it can be accepted with a probability $\exp \frac{y_i - y_j}{T}$;
  - The temperature is decreased;
- End: The process is finished when the temperature reaches a small value.

This section presents the implementation of the Simulated Annealing for our problem. The solution generation and the neighbourhood operator are described.

4.4.2 Solution generation

The proposed algorithm generates the routes one by one, by decreasing order of traffic flow. A solution is composed of a decision vector for each procedure. As a reminder, each procedure decision is composed of a set of decisions for each route. A route decision for a procedure i is defined as follows:

$$\begin{pmatrix} k_1 \\ k_2 \\ \gamma^m \\ \rho^m \\ x^{LF} \\ \rho^{LF,start} \\ \rho^{LF,end} \end{pmatrix}. \tag{4.13}$$

4.4.3 Neighbour generation

To explore the state space efficiently, it is necessary to define a neighbourhood operator. For this purpose, the neighbourhood of a given point in the state space is explored by the mean of two operators. The first one explores the neighborhood locally by performing minor modifications to adjust the solution. The second one, on the other hand, significantly modifies the solution in order to avoid staying in the same state space domain. To generate a neighbour, the algorithm can modify the decisions by one of the five parameters:

- The route of connection;
- The merging ring of a route;
- The horizontal route;
- The level-off;
- The selection of the merging rings *i.e.* the merging rings that can be chosen to merge routes.

A neighbour is therefore created by randomly selecting and changing one or more of these parameters on the current solution. The solutions change more locally as the iterations progress and the probability of changing several parameters decreases.

4.5 Simulation Results

The solution presented in this chapter has been implemented and tested for the design of multiple routes. The proposed methodology is first tested on Paris Charles-De-Gaulle airport SIDs/STARs design taken from the literature [209]. Then, the results with different values of α (trade-off coefficient between distance and noise) are compared. Finally, the Paris Orly airport has been added to the scenario. All the tests were run on a 1.70GHz Intel Core i5-10310U processor with 8GB of RAM on a Linux operating system.

4.5.1 Paris Charles-De-Gaulle case study

This case study is based on the benchmark presented in [209] (See Tables 4.1 to 4.3 and Figure 4.9) in which the problem is addressed with a Branch and bound coupled with a Simulated Annealing. This case study is also solved in [39] by using constraints graph generation and Simulated Annealing. In the first approach [209], a noise model is not considered. In the second approach [39], the results seem to show long, and unsmooth routes (see Figure 4.10).

threshold	take-off/landing	threshold		FAF	
		position	altitude	position	altitude
09R	take-off	(49°01'14.22"N, 2°30'47.01"E)	370	×	×
08L	take-off	(48°59'44.47"N, 2°33'09.88"E)	338	×	×
27R	landing	(49°01'36.10"N, 2°33'42.09"E)	392	(49°02'43.7"N, 2°55'22.1"E)	3392
26L	landing	(48°59'41.56"N, 2°36'08.77"E)	316	(49°00'35.3"N, 2°53'23.1"E)	3316

Table 4.1: Paris CDG Airport: characteristics of the threshold, and the FAFs associated with STARs [209] for the west configuration.

In this scenario, Paris city is considered as an obstacle represented by a circle of radius 5NM centred in (48°51'18.27"N, 2°20'49.60"E). The following additional data are used in this instance:

route	SID/STAR	threshold	traffic load	entry/exit point
1	STAR	27R	12.77%	DINAN
2	SID	09R	10.82%	DIKOL
3	SID	08L	10.72%	OKASI
4	STAR	27R	8.7%	DPE
5	SID	08L	7.44%	AGOPA
6	STAR	26L	7.33%	BENAR
7	STAR	26L	7.16%	RLP
8	STAR	27R	6.97%	MOPIL
9	SID	09R	5.63%	OPALE
10	STAR	26L	5.61%	DJL
11	SID	09R	4.9%	LASIV
12	SID	09R	4.76%	NURMO
13	SID	09R	3.38%	EVX
14	SID	08L	2.33%	LGL
15	STAR	27R	1.48%	LUKIP

Table 4.2: Paris CDG airport: traffic load and entry/exit points [209].

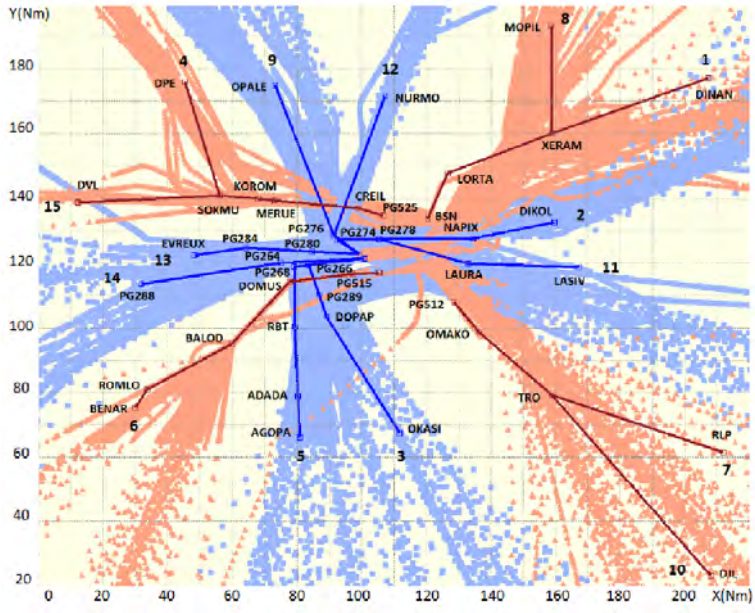


Figure 4.9: One day radar data of Paris CDG airport, and principal SIDs and STARs.

- 12 discretization circles (rings) are considered for each procedure. The distance between them is therefore equal to one-twelfth of the distance from the centre to the closer entry/exit point.
- Minimum turn radius: $r_{min} = 2$NM.
- Minimum and maximum taking-off slopes: $\alpha_{min,TO} = 7\%(\sim 4°)$, $\alpha_{max,TO} = 11\%(\sim 6.3°)$.
- Minimum and maximum landing slopes: $\alpha_{min,LD} = 1.6\%(\sim 0.92°)$, $\alpha_{max,LD} = 4.2\%(\sim 2.4°)$.
- The altitude ranges of entry/exit points are not input. They are determined from the route horizontal profiles and minimum and maximum slopes.

γ_i	threshold	starting point		ending point
		(x_{A_i}, y_{A_i})	H_{A_i}	(x_{B_i}, y_{B_i})
γ_2	09R	(49°01'14.22"N, 2°30'47.01"E)	370	(49°08'15.0"N, 4°02'57.0"E)
γ_9				(49°53'59.1"N, 1°53'06.3"E)
γ_{11}				(48°54'03.3"N, 4°13'37.0"E)
γ_{12}				(49°49'34.0"N, 2°45'19.0"E)
γ_{13}				(49°01'42.7"N, 1°12'50.6"E)
γ_3	08L	(48°59'44.47"N, 2°33'09.88"E)	338	(48°05'00.0"N, 2°46'40.0"E)
γ_5				(48°05'00.0"N, 2°00'35.0"E)
γ_{14}				(48°47'26.2"N, 0°31'49.0"E)
γ_1	27R	(49°02'54.9" N, 2°59'03.2"E)	3392	(49°49'55.0"N, 5°19'53.0"E)
γ_4				(49°55'31.4"N, 1°10'14.3"E)
γ_8				(50°08'52.0"N, 4°06'28.0"E)
γ_{15}				(49°18'57.0"N, 0°29'47.0"E)
γ_6	26L	(49°00'35.3"N, 2°53'23.1"E)	3316	(48°15'11.0"N, 0°44'40.0"E)
γ_7				(47°54'22.7"N, 5°14'57.0"E)
γ_{10}				(47°16'14.8"N, 5°05'50.4"E)

Table 4.3: CDG airport: Starting and ending points of the routes to be designed from [209].

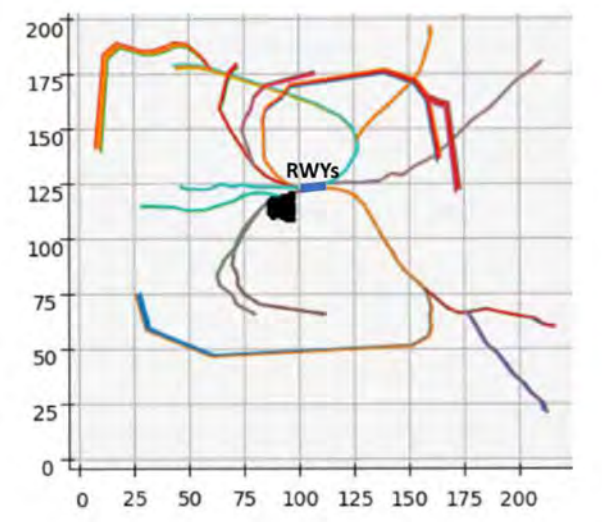


Figure 4.10: Result with the method proposed in [39]

The algorithm runs between 45min and 1h for the CDG scenario depending on the choice of the number of transitions of the Simulated Annealing. A result without considering the noise ($\alpha = 0$) is given in Figures 4.11 and 4.12. They show that the method generates short procedures without conflict and avoiding obstacles. Figure 4.11 shows that a level-off is added to one route of a STAR procedure. This is because the STAR route intersects two SID routes. Without this level-off, the routes would be in conflict.

Table 4.4 shows the comparison between the results obtained in the previous work [209] and our results. The results show that the proposed method computes shorter routes (about 2.5% shorter). Our method is slower but in [209], the positions of merge points are not optimized. Moreover, as the computation time criterion is not essential, the proposed algorithm is efficient in the case of not considering the noise.

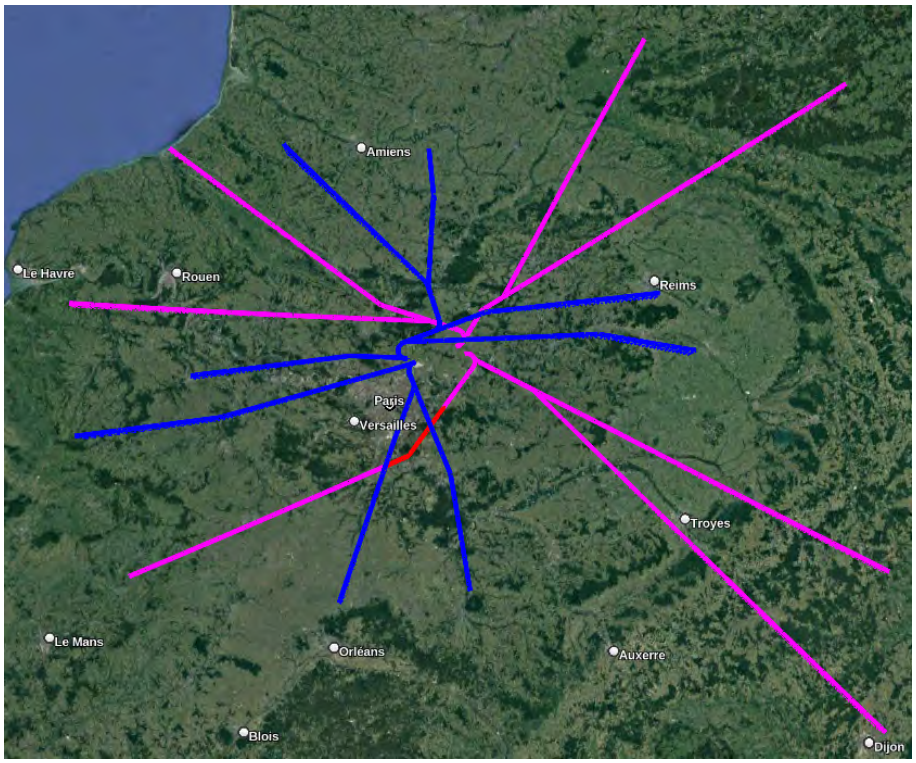


Figure 4.11: Results without considering noise ($\alpha = 0$). The SID routes and STAR routes are represented in blue and pink respectively. The level-off is in red.

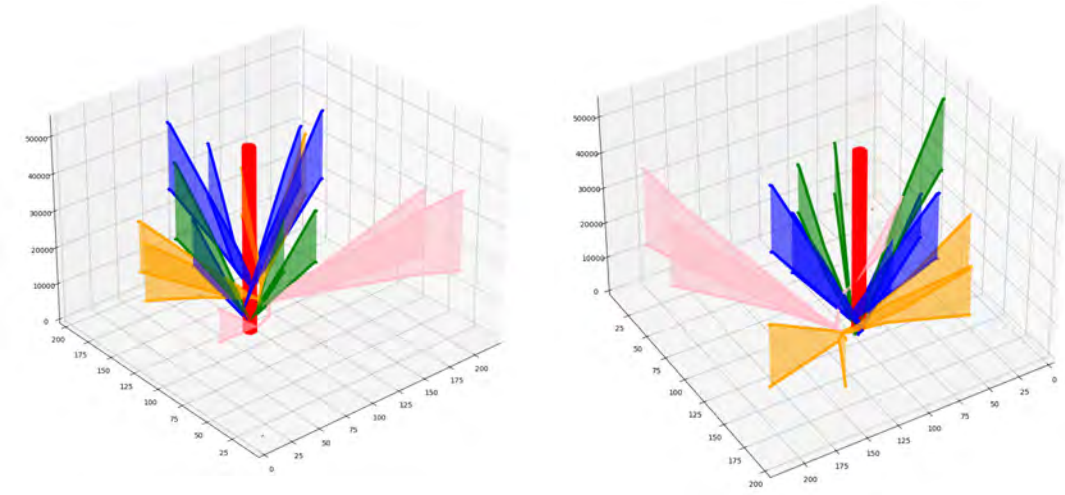


Figure 4.12: 3D views of the designed procedures (Paris city is the big red tube).

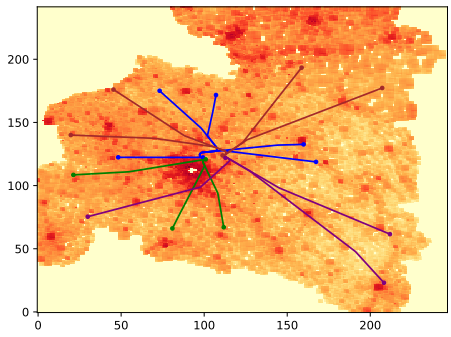
Method	B&B [209]	SA [209]	Proposed Method
Total length of routes (NM)	1313.49	1295.51	1262.31
Computation Time	9.8s	20 min	1h

Table 4.4: Comparison between the results obtained in [209] and our results.

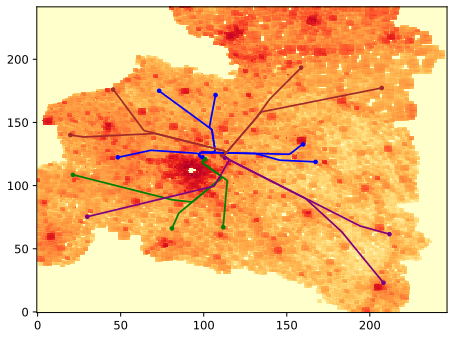
After comparing our algorithm with the literature, the following section presents a case study which takes into account the noise abatement.

4.5.2 Route optimization taking into account the noise abatement.

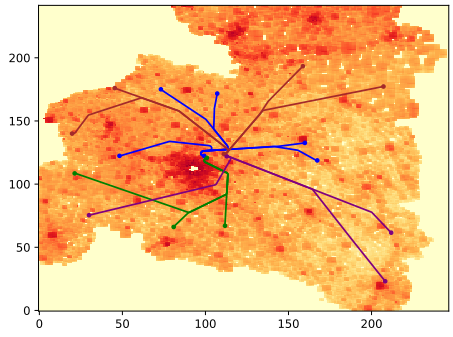
In this work, a noise model is considered. This subsection compares the results based on the coefficient α. Figures 4.13a to 4.13c show 3 different results with 3 values of α, 0, 0.2 and 0.4 respectively. As expected, the higher the α coefficient, the longer the routes and the more they avoid high population areas (in red in the figures). This is reflected in Figure 4.14 which shows a significant decrease in noise and an increase in distance. In the case of $\alpha = 0.2$, the route total length is 1333.75NM which corresponds to an increase of 5.6% compared to the case without noise considerations. However, the noise cost is reduced by almost 65%. This α value makes little difference to the routes lengths but significantly reduces the associated noise impact. In this case, the routes mainly avoid highly populated cities (in bright red in the figures). However, by increasing again the value of α, the noise reduction becomes insignificant. For example, in the case of $\alpha = 0.4$,the length of routes increases by 85NM (compared to the case of $\alpha = 0.2$), the noise reduction is only 14%. In this case, the routes also avoid medium-sized cities (in light red in the figures). This results in more than 12% increase in route length compared to shortest routes i.e. an increase of 150NM. These procedures do not seem to be acceptable as they are too long and therefore involve a significant increase in fuel consumption which causes pollution that should not be neglected. Figure 4.15 shows noise and distance costs for several values of α. This figure summarizes the previous explanations and shows the value of α that seems to offer a good compromise between distance and noise: $\alpha = 0.2$. This value will be used for the test of the final scenario which considers the Orly airport.



(a) Result routes with $\alpha = 0$.



(b) Result routes with $\alpha = 0.2$.



(c) Result routes with $\alpha = 0.4$.

Figure 4.13: Results comparison depending on α value.

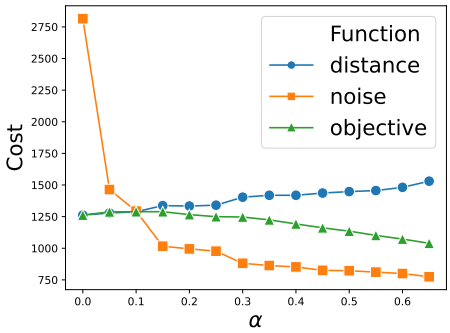


Figure 4.14: Evolution of noise, distance, and objective functions depending on alpha value.

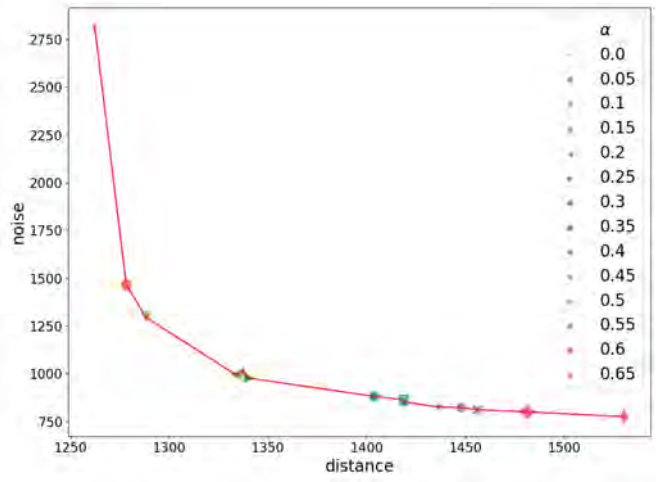


Figure 4.15: Noise and distance costs of the result solution for several α values. This figure represents also the pareto front of the two criteria (noise and distance).

4.5.3 CDG and Orly Instance

To improve the scenario of CDG airport, it is proposed to add the Orly airport which has about 600 daily flights [87]. This airport is therefore not negligible in the Paris TMA study. The runway and FAF position are given by [50] (See Table 4.5). To determine the entry/exit points of the Orly procedures, we used the data from [51]. This data provide the number of flights on July 12, 2019 depending on the entry/exit positions (see Figure 4.16). From this data, we estimate navigation points that fit with trajectory flows (See Table 4.6). Table 4.7 summarizes the scenario of Orly airport.

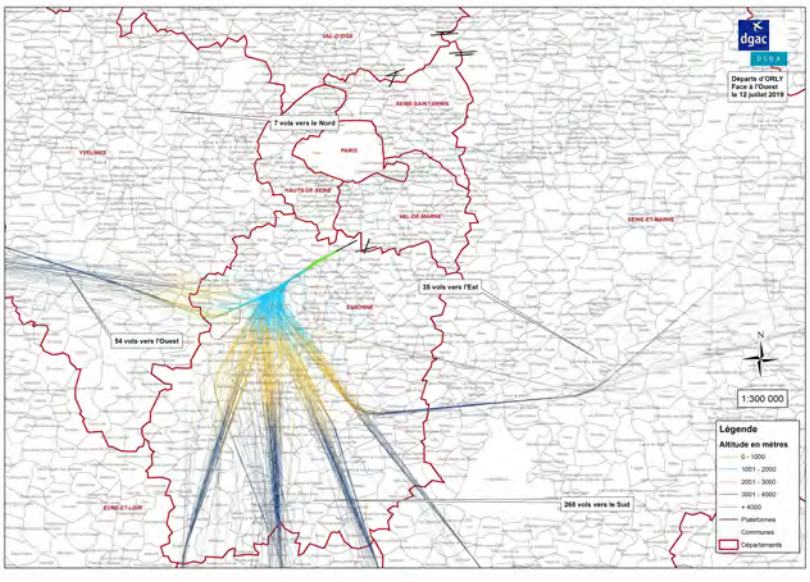


Figure 4.16: Number of departure flights on 12 July 2019. The colors represent the altitudes in meters. Green (0m to 1000m), Blue (1001m to 2000m), orange (2001m to 3000m), dark blue (3001m to 4000m) and grey (higher than 4000).

threshold	take-off/landing	threshold		FAF	
		position	altitude	position	altitude
06	take-off	(48°43'16.49"N, 2°19'13.85"E)	291	×	×
25	landing	(48°43'34.98"N, 2°23'47.01"E)	291	(48°45'48.7"N, 2°35'56.4"E)	3291

Table 4.5: Paris Orly Airport: characteristics of the threshold, and the FAFs associated with STARs.

route	SID/STAR	threshold	Number of flights	entry/exit point
1	STAR	25	200	SOTIP
2	SID	06	160	ERIXU
3	STAR	25	153	VALNU
4	SID	06	108	PILUL
5	SID	06	54	PIGOP
6	SID	06	35	VATRI
7	STAR	25	9	OKVIG
8	SID	06	7	BOXXO

Table 4.6: Paris Orly airport: number of flights and entry/exit points determined from data of 12^{th} July 2019 in [51].

γ_i	threshold	starting point		ending point
		(x_{A_i}, y_{A_i})	H_{A_i}	(x_{B_i}, y_{B_i})
γ_2	06	(48°43'16.49"N, 2°19'13.85"E)	291	(48°05'0"N, 2°15'35"E)
γ_4				(48°05'0"N, 3°02'53"E)
γ_5				(48°41'9"N, 1°20'20"E)
γ_6				(48°47'36"N, 4°03'30"E)
γ_8				(49°26'56"N, 2°00'35"E)
γ_1	25	(48°45'48.7"N, 2°35'56.4"E)	3291	(48°07'52.9"N, 1°27'35.6"E)
γ_3				(48°04'54"N, 3°16'0"E)
γ_7				(48°54'52.4"N, 3°54'9.9"E)

Table 4.7: Orly airport: Starting and ending points of the routes to be designed for the two threshold 06 and 25.

For the simulation, the obstacles and data are the same as in the Paris CDG instance. The algorithm runs for about 2h. The results of the simulation are given in Table 4.8 and Figure 4.18. This new scenario is much more complex with many crossings, which means longer routes to avoid conflict. This is confirmed by the total length of routes for CDG procedures, which is increased by around 70NM. In addition, several level-offs had to be added to guarantee separation between the routes. Although the complexity of the scenario, the proposed algorithm is able to compute short routes that also consider noise impact. The Google Earth view shows that the routes avoid big cities such that Troyes, Rouen, and Versailles. They also avoid the Parisian suburb as much as possible.

	CDG	Orly
Total length of routes (NM)	1405.5	531.3

Table 4.8: Total length of routes for the two studied airports.

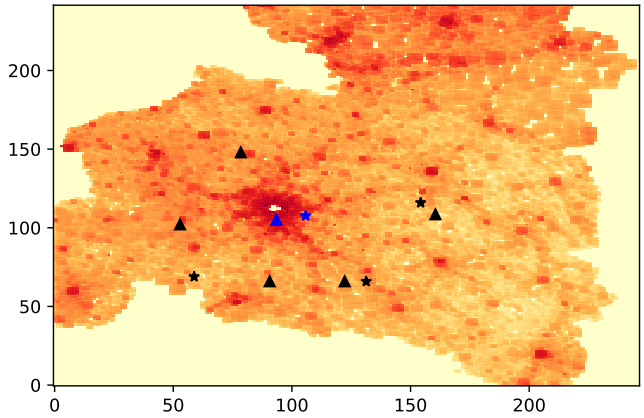


Figure 4.17: Starting points (blue) and ending points (black) of procedures for threshold 06 (triangles) and 25 (stars).

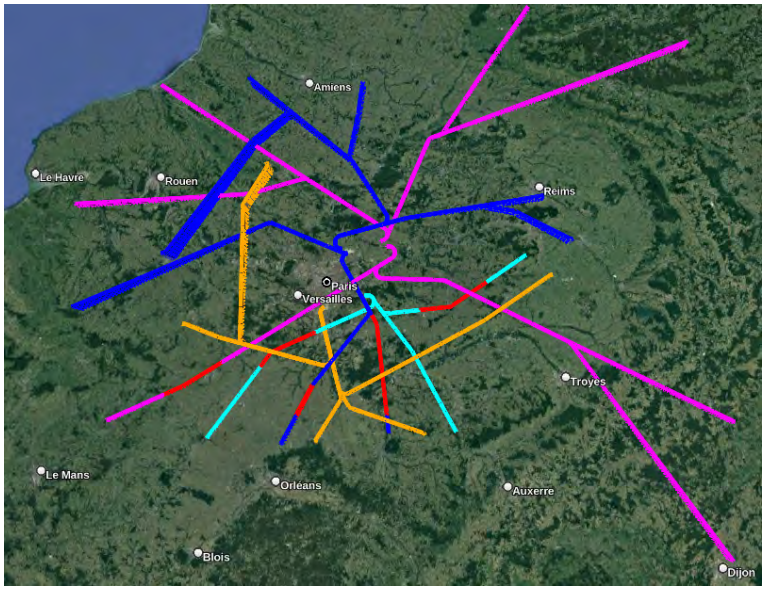


Figure 4.18: SID/STAR procedures of CDG and Orly airports. The SID and STAR routes of the CDG airport are in blue and pink respectively. The SID and STAR routes of the Orly Airport are in orange and cyan respectively. The level-off are in red.

4.6 Conclusion

This chapter has addressed the problem of the automatic design of departure/arrival procedures for large airports considering noise abatement. After an introduction of the context, the method based on Simulated Annealing and Dubins curves is presented. It was first tested on a well-know case study without considering noise abatement: Paris Charles-de-Gaulle airport. The results show that the proposed method computes smoother and shorter routes than those generated in previous works. Then, in the same study case, the noise abatement was considered. A compromise between noise and distance has been proposed to take into the operational constraints. Finally, the Orly airport was added to the previous scenario to test the proposed method in a more complex case. The results show that the algorithm finds SID/STAR procedures, consid-

ering noise impact, that avoid conflict and obstacles. This chapter has presented work on the strategic design of routes for the beginning and the end of a flight. The next chapter proposes to strategically generate cruise alternative trajectories to respond to unforeseen events.

Chapter 5

Alternative Trajectories Generation

Chapters 2 and 3 of this thesis presented work allowing the development of decision support tools when an unexpected event occurs. This chapter proposes to develop a solution to prevent these unexpected events by generating in advance alternative trajectories. This chapter is organized as follows: First, Section 5.1 presents the reason for development of an alternative trajectories generator. Section 5.2 details the proposed algorithm. Finally, in Section 5.3, the preliminary test results on contrails avoidance are presented.

5.1 Introduction

The day of the flight, the airline computes and publishes the optimized flight plan. The optimization takes into account the aircraft's performance and the expected weather. This flight plan is followed as much as possible by the pilots. However, during a flight, an aircraft can be confronted with unexpected events that can disrupt the smooth operation of the flight: weather obstacles, conflicts between aircraft or "mechanical" failures. In the last case, pilots have to find a way to land as soon as possible. This topic has been addressed in Chapter 2. In the other cases, pilots led by air traffic controllers have to perform a maneuver to solve conflicts, or avoid hazardous areas. The problem must be solved quickly to avoid a critical situation that could lead to an accident. This was the case of the Air France flight 447 [35], during which the pilots did not avoid a zone of strong turbulence. This caused a failure of the Pitot probes, thus disturbing the speed measurements of the aircraft. This resulted in a fatal stall for the passengers and crew. A more anticipated reaction would have made it possible to carry out a maneuver to avoid the stormy area. Even when the situation is resolved, the solution is often sub-optimal because it is quickly determined by pilots. It is then necessary to develop tools to prevent this kind of event. We propose to address the problem by automatically generating, a set of alternative cruise trajectories to rapidly find a satisfactory solution. In addition to guaranteeing safety, air transport is facing a new major challenge: reducing its environmental impact. Indeed, aviation is responsible for about 3-5% of total global warming [131]. Its impact is composed of CO_2 effects but also from non-CO_2 effects. Among these non-CO_2 effects, condensation trails (contrails) have the higher warming power [131]. Indeed, if they persist and form cirrus clouds, contrails can have a warming effect. However, contrails areas are still difficult to predict [82]. Ice-supersaturated areas are the most likely to be favorable to persistent contrails Irvine and Shine [104]. These areas can therefore be considered hard obstacles such like thunderstorm areas, or be considered as soft obstacles areas where the crossing time must be limited. In this study, it is proposed to test the proposed approach on the avoidance of contrails areas by considering

them hard obstacles. Contrail areas are very similar to thunderstorm areas. Moreover, they are more complex to avoid than stormy areas because they are larger. Although contrails do not impact safety, their avoidance is very interesting to study.

Most of the works in the literature focus on the generation of only one avoidance trajectory for a given situation. In contrast, in this study, we propose to generate several trajectories to prevent any unexpected events such as weather obstacles or contrails areas.

The objective of this research is to create a solution framework to design, for a given flight, efficient alternative cruise trajectories to be able to solve any problem related to an unexpected event. This tool proposes, from the set of alternative trajectories, a solution to effectively avoid an obstacle and thus ensure safety. At the end of the maneuver, the pilot could decide to resume its initial flight plan or continue to follow the alternative route. In this work, the alternative trajectories must satisfy the following requirements:

- Alternative trajectories should not be too far (fuel is limited) from the initial flight plan.
- Alternatives trajectories should not be too similar to each other.
- An alternative trajectory has at least one significant maneuver compared to the initial flight plan.
- At least one of the alternative paths has to avoid the obstacles.

5.2 Alternative Path Generation

5.2.1 Graph Generation

In the following paragraphs, a graph G is defined by a set of nodes V and a set of edges E. To generate several alternative trajectories, it is proposed to use the Rapidly-exploring Random Graph (RRG) algorithm.

5.2.1.1 Rapidly-Exploring Random Graph

The RRG algorithm works in the same way as RRT (presented in Section 2.5.4.3). It is composed of five main functions: sampling, nearest neighbor, near vertices, steer, and collision test functions. As a reminder of the RRT algorithm, at each iteration, a new node x_{new} is added to V and an edge is created between the new node, and its nearest neighbor $x_{nearest}$ in the graph. In the RRG algorithm, other connections are added, which creates a graph and not a tree. For all vertices $x_{near} \in V$ that are at a distance lower than a radius r from x_{new}, a new edge is added to E. The radius r depends on the cardinal of V and is defined as follows:

$$r = \min\{\eta, \gamma_{RRG}(\log(|V|)/|V|)^{1/d}\}, \tag{5.1}$$

where:

$$\gamma_{RRG} > 2(1 + 1/d)^{1/d}(\mu(\chi_{free})/\zeta_d)^{1/d}, \tag{5.2}$$

with d the dimension of the space ($d = 2$ in this study), $\mu(\chi_{free})$ is the Lebesgue measure of the obstacle-free space and ζ_d is the volume of the unit ball in the d-dimensional euclidean space. Figure 5.1 shows an iteration of the RRG algorithm. As for the RRT* algorithm, it is proven to be asymptotically optimal. The pseudo code of the RRG algorithm is given in Algorithm 10.

Thanks to the graph obtained by the RRG and knowing the predecessors of each node, the paths from the origin to the destination can be easily determined.

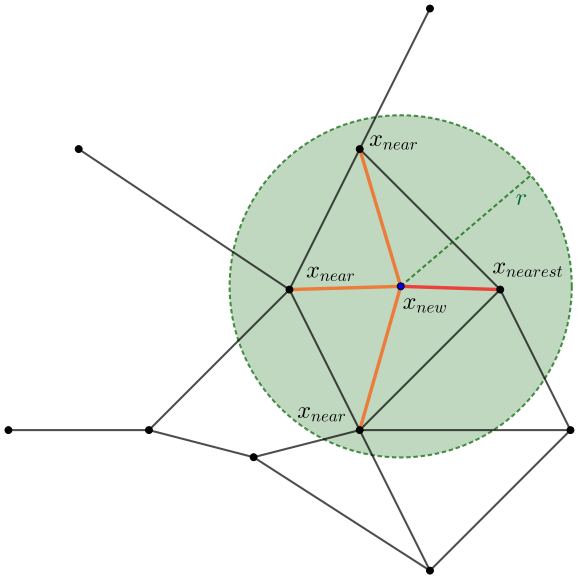


Figure 5.1: RRG local optimization iteration: First, a connection linking x_{new} and its nearest neighbor $x_{nearest}$ is created (in red) and then, x_{new} is connected to the points x_{near} at a distance lower than r (in green).

Algorithm 10 Rapidly-exploring Random Graph [117].

$V \longleftarrow \{x_{init}\}$; $E \longleftarrow \emptyset$
for $i = 1, ..., n$ **do**
  $x_{rand} \longleftarrow \texttt{SampleFree}_i$
  $x_{nearest} \longleftarrow \texttt{Nearest}(G = (V, E), x_{rand})$
  $x_{new} \longleftarrow \texttt{Steer}(x_{nearest}, x_{rand})$
  if $\texttt{ObstacleFree}(x_{nearest}, x_{new})$ **then**
    $X_{near} \longleftarrow \texttt{Near}(G = (V, E), x_{new}, \min\{\eta, \gamma_{RRG}(\log(|V|)/|V|)^{1/d}\})$
    $V \longleftarrow V \cup \{x_{new}\}$; $E \longleftarrow E \cup \{(x_{nearest}, x_{new}), (x_{new}, x_{nearest})\}$
    for all $x_{near} \in X_{near}$ **do**
      if $\texttt{CollisionFree}(x_{near}, x_{new})$ **then**
        $E \longleftarrow E \cup \{(x_{near}, x_{new}), (x_{new}, x_{near})\}$
      end if
    end for
  end if
end for
return $G = (V, E)$

5.2.2 Improvements

In this study, it is proposed to adapt the RRG algorithm to the alternative trajectories generation. To avoid too short maneuvers and therefore too similar trajectories, it is proposed to modify the *near* function. As a reminder, this function returns the closest neighbors. If a neighbor is very close to the new point, it creates a very short connection. To avoid this phenomenon, two points at a distance lower than a radius r_{min} are not considered neighbors. This radius is defined as follows:

$$r_{min} = r\alpha \left(1 - \frac{1}{i}\right), \tag{5.3}$$

where i is the iteration number and α is a coefficient in $[0, 1]$. The higher α is, the higher the radius r_{min} is, and therefore, the higher the connection lengths are. However, the connection linking x_{new} and $x_{nearest}$ is kept to preserve the optimal connection. Figure 5.2 shows this change, only the points in the “donut" zone (in green) are considered neighbors. In this example, the second closest neighbor is therefore not connected.

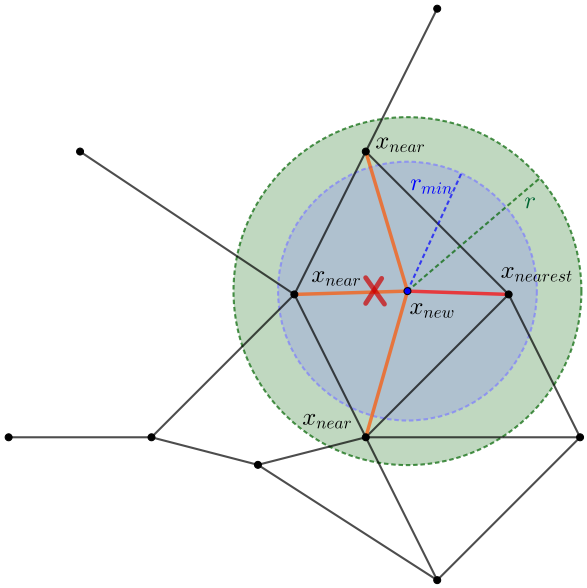


Figure 5.2: RRG local optimization iteration with “donut": Connection linking x_{new} and $x_{nearest}$ is created as usual but the edge with the second closest point is not added to the graph.

Then, to avoid too long trajectories, it is proposed to modify the sampling method. Most of the time, the samples are randomly generated in the whole space. To reduce the sampling space, it is proposed to sample around the straight line linking the start and the final positions. Figure 5.3 shows the proposed sampling area. This area is defined by the distance d_s to the straight line computed as follows:

$$d_s = \frac{d}{\beta}\left(1 - \sqrt{\frac{|d_o - d/2|}{d/2}}\right), \tag{5.4}$$

where d is the distance from the origin O to the destination D, d_o is the distance of the straight line from the origin and β is a coefficient in $\mathbb{N}^*$. This sampling reduces the graph propagation area. The number of samples can thus be lower and this reduces the computation time of the graph.

5.2.3 Clustering

After generating alternative paths thanks to the RRG algorithm, it can be noted that a lot of nodes are very close to each other (see Figure 5.4a). Therefore, it is proposed to merge “neighbor” nodes into only one centroid node (see Figure 5.4b). The chosen clusterization method is the k-means algorithm because it is an unsupervised method with only one parameter which is the number of clusters. Besides, there are no outliers in the data. Indeed, all the points belong to at least one path because they have been generated by the RRG algorithm. Figure 5.5 shows the clusterization for the Paris-Toulouse flights with three sizes of clusters (20, 70, and 130). When the number of clusters is low (Figure 5.5a), distant points can be grouped together. In the second case (Figure 5.5b), the paths are smoother than for the previous clustering and there

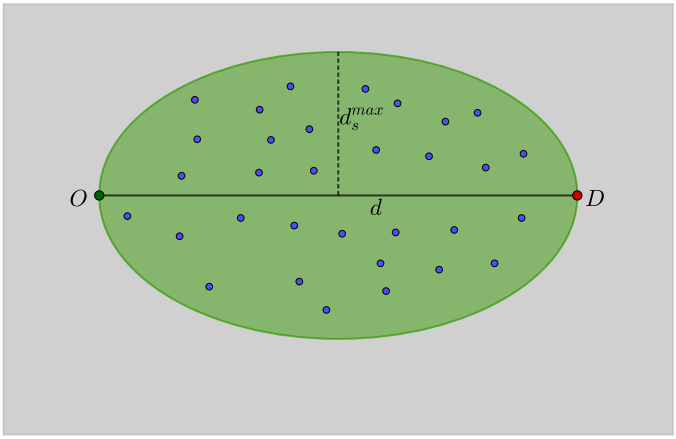
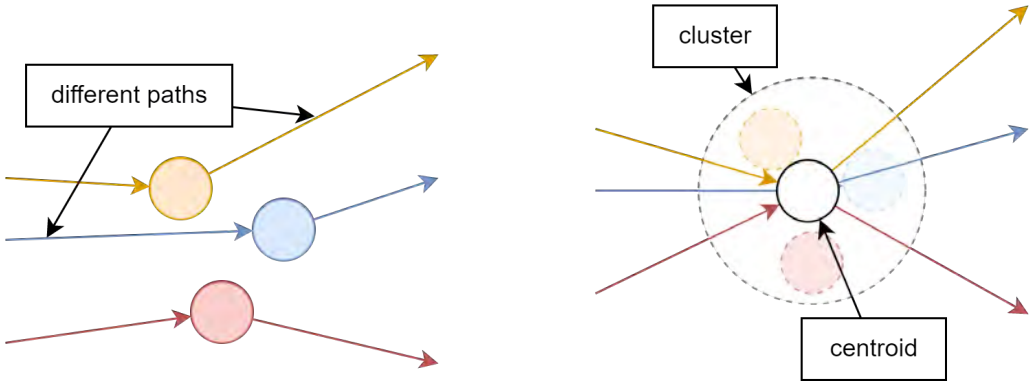


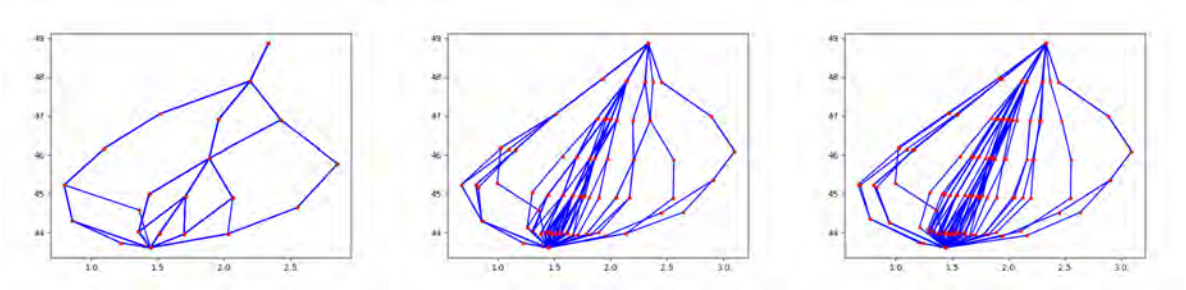
Figure 5.3: The sampling area (in green) is defined around the straight line linking the origin O and the destination D by the distance d_s.

is less overlapping after replacing the points by their centroids. In the last case (Figure 5.5c), the paths seem shorter but are too similar due to the sizes of clusters. A trade-off has to be done here between similarity and precision. The next step consists in removing similar paths by using a dedicated metric.



(a) Before clustering (b) After clustering

Figure 5.4: Three spatially close nodes from three different paths merged into one centroid before/after clustering.



(a) Clustering with 20 clusters (b) Clustering with 70 clusters (c) Clustering with 130 clusters.

Figure 5.5: K-means clustering of the points generated by RRG algorithm for three different number of clusters (20, 70 and 130) for the Paris-Toulouse trip

5.2.4 Similarity post processing

After generating paths thanks to the RRG algorithm and clustering nodes, the next step consists in removing similar paths. Similar paths (red paths in Figure 5.6) are not relevant because they may be similarly impacted by the obstacles. Moreover, two equivalent routes do not provide additional information and are not considered as alternatives. The idea behind this process is therefore to detect similar alternative paths among those generated and merge them.

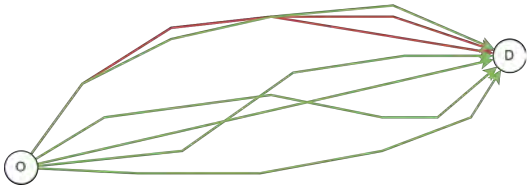


Figure 5.6: Post treatment on generated paths by RRG algorithm to detect and remove similar paths (red paths) and keep alternative paths (green paths).

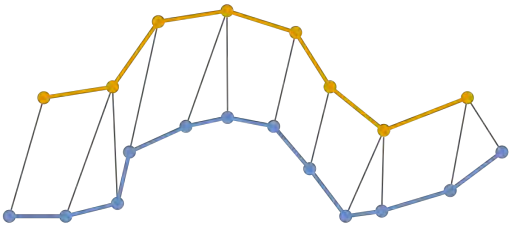


Figure 5.7: DTW point matching process between two trajectories (blue and orange) with different lengths.

The similarity metric uses in this study is the Dynamic Time Warping (DTW). This method can compare two temporal sequences with potentially different speeds, and sizes. The algorithm computes the optimal matches between these two sequences. Let consider two paths $\mathbf{x} = (x_i)_{0 \leq i \leq n}$ and $\mathbf{y} = (y_j)_{0 \leq j \leq m}$ connecting a departure airport to an arrival airport *i.e.* $x_0 = y_0$ and $x_n = y_m$. The DTW of $\mathbf{x}$ and $\mathbf{y}$ is then defined by:

$$DTW(\mathbf{x}, \mathbf{y}) = min_{\mu \in \mathcal{M}_{\mathbf{x},\mathbf{y}}} \sum_{(i,j) \in \mu} d(x_i, y_j), \tag{5.5}$$

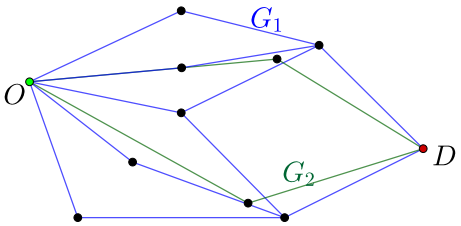
where $\mathcal{M}_{\mathbf{x},\mathbf{y}}$ is the set of possible matching between paths $\mathbf{x}$ and $\mathbf{y}$; d is a distance function between two points.

The solution algorithm is based on dynamic programming. It computes the best pair matching between the points of both trajectories greedily based on the DTW distance (Figure 5.7). The similarity of these trajectories is the sum of the best pair matching distances.

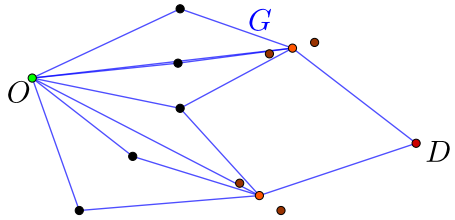
5.2.5 Proposed Method

After presenting different steps of the method, this subsection summarizes the alternative trajectories generation process (See Figure 5.8). The process is the following:

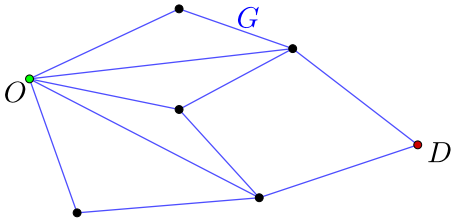
1. n_g graphs are generated with different values of maximum neighborhood radius η. This step computes short and long deviations from the flight plan to avoid small or big obstacles as fast as possible. An example with two graphs is given in Figure 5.8a.
2. A clusterization of the n_g graphs is made to obtain a bigger graph with more edges. The clusterization phase is illustrated in Figure 5.8b.
3. Similar paths are removed from the graph thanks to the similarity metrics DTW (See Figure 5.8c).
4. Finally, if obstacles appear, only paths that avoid them are kept (See Figures 5.8d and 5.8e).



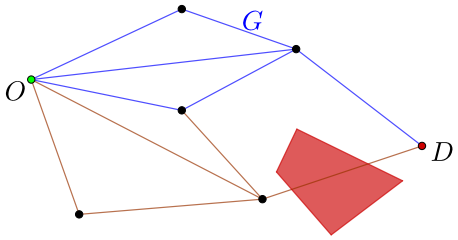
(a) Graphs generation: G_1 is generated with a small radius and G_2 with a long one.



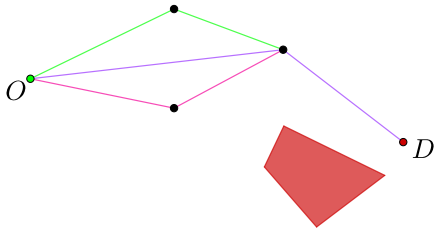
(b) Graphs clusterization: points are merged to create new points in orange.



(c) Removal of similar paths.



(d) Removal of paths passing through the obstacle.



(e) Selection of avoidance paths

Figure 5.8: Alternative trajectories generation process.

5.3 Results

In this work, the case study is the avoidance of contrail areas. These areas are not considered as “hard" obstacles *i.e.* obstacles that jeopardize the safety of the aircraft. However, it would be better to avoid them to reduce the non CO_2 effects of air traffic. But an avoidance maneuvering would release more CO_2. A compromise based on long term impact has to be found between both options. It seems that a mix between the two options would lead to the optimal solution. Contrail zone avoidance is a complex problem, in this study we focus on avoiding them like they were hard obstacles. A contrail area can be vertically or horizontally avoided. In this study, we only consider lateral avoidance during the cruise. Indeed, in operation, flight level change is rarely performed to avoid obstacles.

5.3.1 Contrails data

Relative humidity and temperature data are extracted from ECMWF [54] for two hours (6 am and 7 am UTC) on January 2, 2022, at Flight Level 300 (300 hPa pressure level). They have been extracted on a 2D-grid with a resolution of 0.1 degrees on latitude and longitude, from latitude 37.5 to latitude 55.4 and from longitude -12 to longitude 16. Based on these data, areas favorable to persistent contrails are computed thanks to the following process, as usually done in the literature (see [185]). First, on a given point, the *Schmidt-Appleman criterion* is applied

to determine if a contrail can be formed: a contrail is formed if the relative humidity of the air in liquid water is above a threshold $RH_w \geq r_{min}$ where

$$r_{min} = \frac{G(T - T_c) + e_{sat}^{liq}(T_c)}{e_{sat}^{liq}(T)}, \tag{5.6}$$

$e_{sat}^{liq}(T)$ is the saturation vapor pressure over water, T_c is the estimated threshold temperature (in Celsius degrees) for contrail formation at liquid saturation. T_c is computed as follows:

$$T_c = -46.46 + 9.43\log(G - 0.053) + 0.72\log^2(G - 0.053), \tag{5.7}$$

where $G = \frac{EI_{H_2O}C_pP}{\epsilon Q(1-\eta)}$, $EI_{H_2O} = 1.25$ is the water vapor emission index, $C_p = 1004\ J.kg^{-1}.K^{-1}$ is the heat capacity of the air, P is the ambient pressure (in Pascals), $\epsilon = 0.6222$ is the ratio of the molecular masses of water and dry air, $Q = 43 \cdot 10^6 J.kg^{-1}$ is the specific heat of combustion, and $\eta = 0.3$ is the average propulsion efficiency of a commercial aircraft. Figure 5.9 presents the persistent contrails areas of January 2, 2020, at 7 am at FL300.

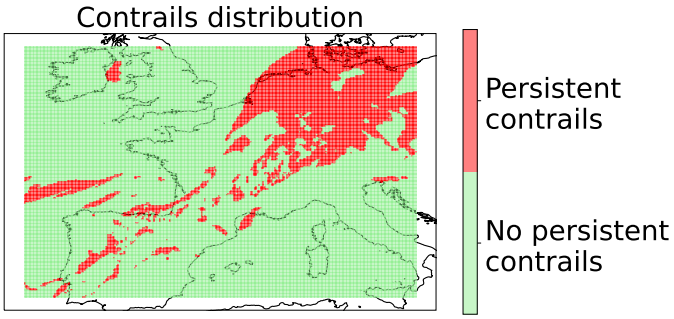


Figure 5.9: Contrails data map.

5.3.2 Alternative trajectories results

From this data, it is proposed to study two flight cases (London-Toulouse and Paris-Toulouse) to evaluate the method. In both cases, only the cruise phase is considered and the aircraft flies at FL300. Table 5.1 shows the parameters of the algorithm used for all tests. 15 graphs have been generated by the RRG algorithm with η values between 0.5 and 2.

Parameters	η_{min}	η_{max}	n_g	α	β	$n_{samples}$
Values	0.5	2	15	0.75	3	50,000

Table 5.1: Algorithm parameters.

5.3.2.1 Paris-Toulouse case

As explained in Section 5.2, the first step is to generate several graphs. Figure 5.10 shows the alternative trajectories generated by the RRG algorithm with the parameters given in Table 5.1. In this scenario, 1,972 trajectories are computed. This number is obviously too high and several trajectories are very similar. Then, the graphs are clusterized. Figure 5.11 shows the paths after the clusterization. Their number remains the same but the number of nodes is significantly reduced. After this step, several paths are very similar, some of them should be removed by the

similarity post-treatment. Figure 5.12 shows the result after this step. The resulting trajectories can be used throughout the flight in case of unexpected events. It is proposed to study the use of these routes in the case of contrails avoidance.

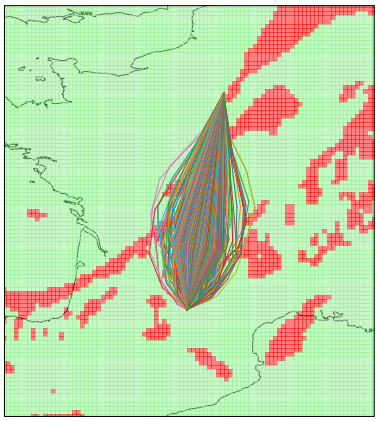


Figure 5.10: Paris-Toulouse alternative trajectories generated by the RRG algorithm.

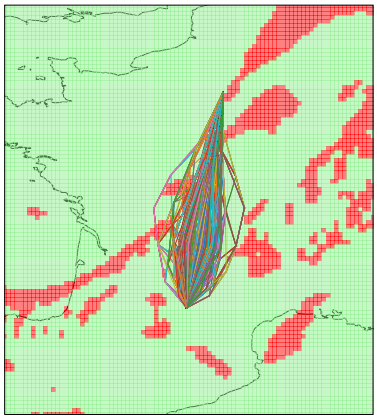


Figure 5.11: Paris-Toulouse alternative trajectories after clusterization (140 clusters).

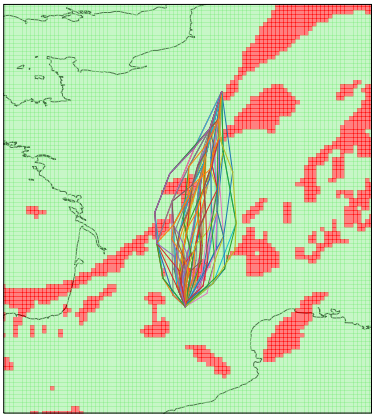


Figure 5.12: Paris-Toulouse alternative trajectories after similarity post-treatment.

The last step is to keep only contrails avoidance trajectories. Figures 5.13 and 5.14 present the resulting paths for Toulouse-Paris at 6 am and 7 am. The orange path is the shortest path from the origin to the destination. The blue paths are the alternative. In these scenarios, the pilots have several options. They can first decide to follow the shortest path and then if a new event occurs, they can follow a blue path just for the maneuver and then return to the optimal path. They can also choose to follow this route to the destination. In Figure 5.14, it can also be noted that the algorithm proposes obstacle avoidance from the right or from the left. However, it only offers one option if one maneuver is significantly longer than the other (See Figure 5.13). Table 5.2 summarizes the number of paths at each step of the proposed algorithm. Table 5.3 presents the computation time of each step of the process. The generation of the alternative trajectories takes 198.5 seconds, which is totally acceptable for a pre-flight computation. The selection of avoidance trajectories is very fast and can be therefore easily done during the flight.

To validate the results achieved by the methodology proposed in the study. We compute the proportion of alternative paths connecting the Paris-Toulouse airports depending on their deviation from the shortest path (see Figure 5.15). This figure also includes the paths that avoid obstacles for the scenario of Figure 5.13. The more the alternative paths (light-blue bars) devi-

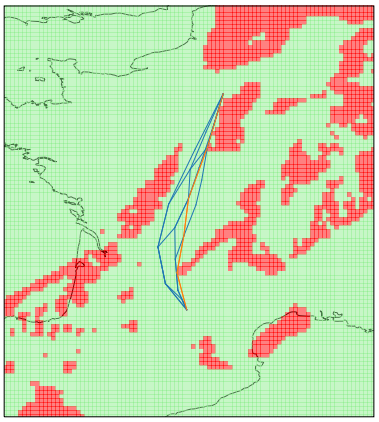


Figure 5.13: Paris-Toulouse contrails avoidance trajectories at 6 am.

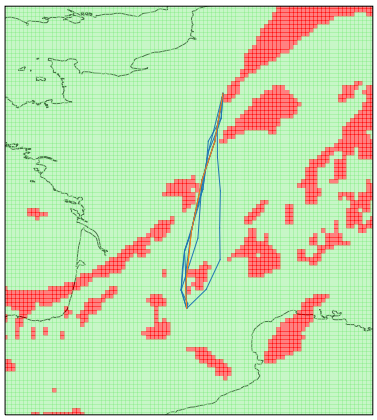


Figure 5.14: Paris-Toulouse contrails avoidance trajectories at 7 am.

	RRG	KMeans	Post	Avoidance	
				6am	7am
Number of paths	1972	1972	64	8	7

Table 5.2: Number of paths at each algorithm step.

	RRG	KMeans	Post	Avoidance Paths
Computation Time (s)	191	1	6.5	0.019

Table 5.3: Computation time in seconds of each algorithm step.

ate from the shortest path the lesser they are. We can remark that most of the paths are close to the shortest path. Around 60% of alternative trajectories have a deviation of less than 4%. The figure shows that the obstacles can be avoided in different ways. For instance, the pilot can decide to follow the shortest avoidance path (deviation between 1% and 2%). However, he can also prefer to follow a longer trajectory to perform a safer maneuver to reduce the risk due to the uncertainties.

5.3.2.2 London-Toulouse case

To complete the validation of this preliminary approach, the London-Toulouse flight is tested. Figure 5.16 shows the alternative paths generated by the proposed algorithm for the London-Toulouse flight at 7 am.

According to the results, the method seems promising. Indeed, for several case studies, the algorithm is able to generate multiple alternative trajectories. Moreover, these trajectories can be used to avoid obstacles and offer several options to pilots. However, further study on the flyability of alternative trajectories would improve the quality of the results. Moreover, in the context of contrails avoidance, a compromise between CO_2 and non-CO_2 effects is essential for an operational system. This implies a study of vertical avoidance trajectory generation. This work was obviously a preliminary work for the development of alternative flight plans generation.

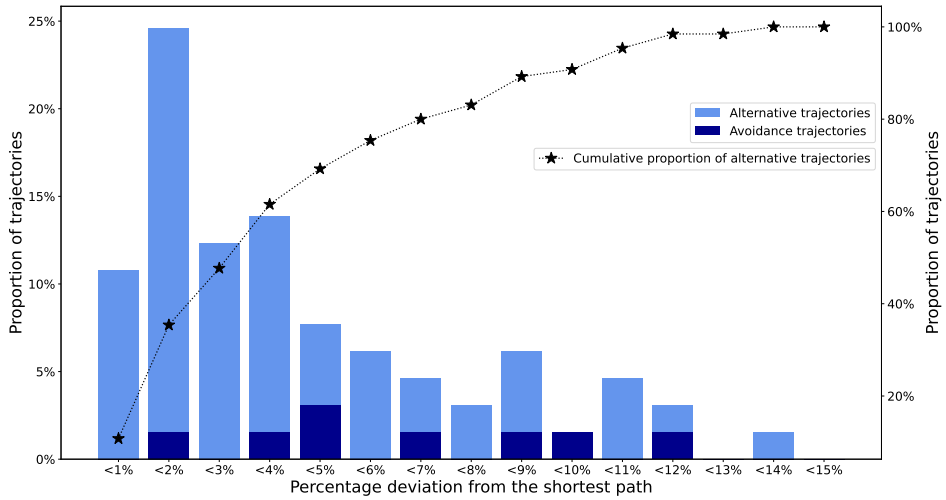


Figure 5.15: Alternative and avoidance trajectories proportion depending on the deviation from the shortest path linking Paris and Toulouse. The avoidance trajectories are for the scenario of Figure 5.13.

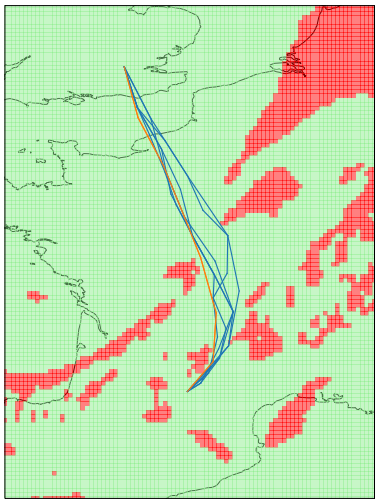


Figure 5.16: London-Toulouse contrails avoidance trajectories at 7 am.

5.4 Conclusion

This work has addressed the generation of alternative cruise trajectories to handle any type of unexpected event. The proposed solution is based on an adapted version of the RRG algorithm, k-means clusterization, and DTW similarity metric. Simulation results based on two different flights have illustrated the proposed approach. According to the numerical results, this approach generates alternative trajectories with diversity and considers some aeronautical operational constraints. The proposed method can be used in different ways. For instance, it can be relevant in the case of the presence of persistent condensation trails or for thunderstorms avoidance. They can also be used as an alternative to the optimal flight plan. The proposed method seems promising, but for long-haul flights, a dynamic version with updated alternative trajectories would be necessary. Moreover, vertical alternative trajectories could be added to this framework.

This study opens the way to the development of a complete flight system composed of an alternative cruise trajectory system and an emergency trajectory generation tool like the one developed in the SafeNcy project (presented in Section 2.5). This tool would propose alternative cruise trajectories to avoid unexpected events during a flight and trajectories in case of an emergency. Each point of the generated graph would have a set of alternatives and one or more emergency trajectories. This system could enhance the safety of the flights. The pilot could thus react to any type of event, be it a simple weather event or an emergency. This system would be a combination of “cure” and “prevention”.

Chapter 6

Conclusions and perspectives

6.1 Review of the work

During this thesis, four main topics have been studied: emergency trajectory design, weather hazards avoidance trajectory, tactical conflict detection and resolution problem, procedures design, and alternative trajectories generation. For the first problem, three different methods have been tested: the Fast Marching Tree algorithm, Fast Marching methods, and the Rapidly-exploring Random Tree. The first method has been used for the first time in an aviation application. Its adaptability to the aircraft trajectory has been proven and it seems promising for future research. The Fast Marching methods in 2D and 3D have also been applied to the emergency trajectory design. According to the results, these methods are limited because it is difficult to guarantee a 100% flyable trajectory. However, the major contribution was more theoretical by the development of tetrahedral structures from octrees to apply 3D Fast Marching method. Finally, the Rapidly-exploring Random Tree algorithm has been used for the integration into the SafeNcy system. The results of the project were very promising for the future development of an emergency system integrated into the FMS. Indeed, the system generates trajectories for all emergencies defined in the project, and feedback from the pilots on the trajectories obtained was very positive. However, some improvements should be added to be really integrated. In addition to this main work on emergency trajectories, other studies have been carried out. They were guided by the desire to adapt the methods used for emergency trajectories for other aviation problems. The FMT* algorithm was dynamically adapted to deal with the problem of avoiding weather obstacles in real time. Moreover, in this study, an aircraft collaborative method has also been implemented to have a fair algorithm which is essential in this kind of problem. Then, two research studies on the development of prevention tools were covered. The first one was on the design of safe and low noise departure and arrival procedures near large airports. Finally, a study on the generation of alternative trajectories was conducted. The goal was to create a solution framework to design, for a given flight, efficient alternative cruise trajectories to ensure obstacle avoidance. The proposed method was based on another sampling-based path planning algorithm: the Rapidly-exploring Random Graph. It has been tested on the problem of contrails avoidance and looks promising. To summarize, the main contributions of this thesis are the following:

- Adaptation of sampling-based path planning algorithms to aviation problems;
- Creation of a tetrahedral structure to apply 3D Fast Marching method for the emergency trajectory design problem;

- Participation in the SafeNcy system development;
- Development of a dynamic version of the Fast Marching Tree algorithm;
- Procedures design of the Paris TMA considering noise abatement;
- Preliminary work on alternative trajectories generation.

6.2 Other works

During this thesis, not presented in this manuscript, three other works have been carried out. These researches were linked with the methods used. The first one was done with a master student at ENAC and deals with the computation of optimal trajectories for light soaring aircraft. The goal of this research was to adapt the Fast Marching Tree algorithm to soaring. It is extended in such a way it can deal with differential constraints associated with light soaring aircraft, operating in a convective atmosphere, in a field of wind, and close to reliefs. Simulations show that the method is well adapted to plan the path in a 3D wind field with complex kinematics motion while avoiding terrain collision. An example of a result is given in Figures 6.1 and 6.2. This work leads to a publication at the EuroGNC 2022 conference.

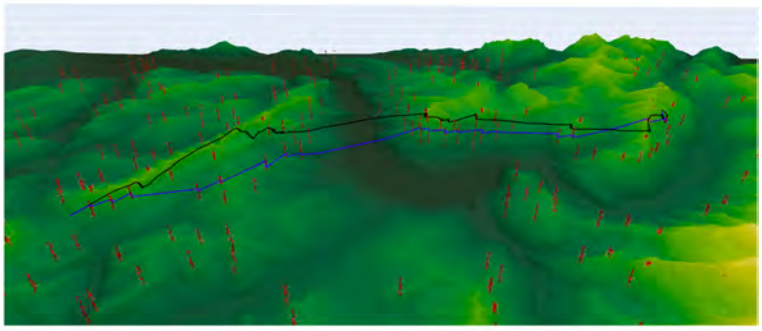


Figure 6.1: Comparison between a computed (blue) and a flown trajectory (black) for the first scenario, red quivers represent thermals.

The second work, done with another Ph.D. student, aims to show that the FMT* algorithm can be adaptable to cruise long-haul flights by considering wind and great circle distances. This study leads to a publication at the DASC 2023 conference.

The last work, still in progress, aims to adapt the Fast Marching Tree algorithm to the avoidance of contrails. The study, published at the DASC2023 conference, demonstrates the efficient adaptability of FMT* for wind constraints when considering great circle distances. In addition, previous works have shown the performance of FMT* in environments constrained by hard obstacles. We therefore propose an adaptation of this method to the case of contrails, considered as soft obstacles. The first results are promising and a paper will be submitted to a conference.

6.3 Discussion and perspectives

For each problem, the results showed that there is till room for improvement.

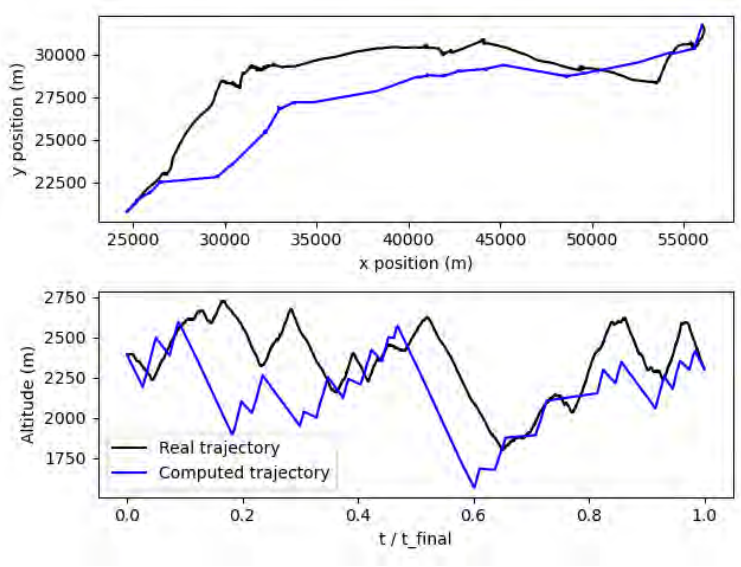


Figure 6.2: Comparison of the longitudinal trajectories and the altitude profiles for the first scenario.

6.3.1 Emergency trajectory design

In this thesis, we compared three methods. We concluded that the RRT* algorithm seems the more appropriate. The results of the SafeNcy project are very positive, but the following improvements could be investigated:

- Reduce the computing time of some scenarios;
- Ensure that the system finds a trajectory for all suitable landing sites;
- Limit the number of computed trajectories (3 for instance) to reduce the computation time.

6.3.2 Weather hazards avoidance trajectory

To integrate this system, the following steps have to be done:

- Test the algorithm on real cases of hazardous weather;
- Test the algorithm on cases with more aircraft so that it can potentially be used in denser areas such as TMAs; this improvement will require the algorithm to be adapted in 3D;
- Test whether the collaborative method can work with on-board aircraft tools.

6.3.3 SID/STAR design

The results of this work are positive but there are still some points to be worked out before the method can be fully validated:

- Test the proposed method on another TMAs (New York, Bangkok, Geneva, ...);
- Develop a more realistic noise model;
- Compare with other methods such as artificial evolution.

6.3.4 Alternative trajectories generation

This study is the least complete, with many areas for improvement:

- Generate vertical alternative trajectories;
- Consider operational constraints (trajectory flyability, airspace sectors, etc);
- Develop a dynamic version for long-haul flights;
- Test on real cases of hazardous weather;
- Test on cases with several aircraft in conflict.

Despite these areas for improvement, this thesis work has opened the way to the development of systems to improve safety. Firstly, by studying two critical situation resolution problems. Then, by developing methods for predicting routes to avoid this type of situation. The last subject dealt with in this thesis is the least accomplished, but a very promising subject for the future, as it would enable us to significantly reduce the number of critical situations and therefore accidents. However, this is a very complicated subject to tackle because, as Albert Einstein said: “Intellectuals solve problems, geniuses prevent them”.

Glossary

A

ACC Area Control Center.

AFP Adaptive Flight Planning.

ANSA At Nearest Suitable Airport.

ANSP Air Navigation Service Provider.

ASAP As Soon As Possible.

ATC Air Traffic Control.

ATIS Terminal Information Service.

ATM Air Traffic Management.

C

CDM Collaborative Decision Making.

CPA Closest Point of Approach.

D

DEM Digital Elevation Model.

DME Distance Measuring Equipment.

DTW Dynamic Time Warping.

E

EAD European AIS Database.

F

FAF Final Approach Fix.

FMT* Fast Marching Tree.

G

GFS Global Forecast System.

GNSS Global Navigation Satellite System.

GPWS Ground Proximity Warning System.

GS Ground Speed.

I

IAF Initial Approach Fix.

ICAO International Civil Aviation Organization.

IFR Instrument Flight Rules.

ILS Instrument Landing System.

INS Inertial Navigation System.

IP Integer Programming.

L

L/D Lift-to-Drag.

LOC Localizer.

LSL Left-Segment-Left Dubins curve.

LSR Left-Segment-Right Dubins curve.

M

MBVP Multi-point Boundary Value Problem.

MC Morton Code.

N

NavDB Navigation databases.

ND Navigation Display.

NDB Non-Directional Beacon.

NLP Nonlinear Programming Problem.

NM Nautical Miles.

NMPC Nonlinear Model Predictive Control.

NOAA National Oceanic and Atmospheric Administration.

NOTAM Notice of air missions.

O

OCC Operating Control Centre.

OCP Optimal Control Problem.

ODE Ordinary Differiential Equation.

P

PANS Procedures for Air Navigation Services.

PBN Performance Based Navigation.

PCA Principal Component Analysis.

PMP Pointryagin's Maximum Principle.

PRM Probabilistic RoadMaps.

R

RA Resolution Advices.

RNAV Area Navigation.

RNP Required Navigation Performance.

RoC Rate of Climb.

RoD Rate of Descent.

RRG Rapidly-Exploring Random Graph.

RRT Rapidly-Exploring Random Tree.

RSL Right-Segment-Left Dubins curve.

RSR Right-Segment-Right Dubins curve.

S

SA Simulated Annealing.

SID Standard Instrument Departure.

SIGMET Significant meteorological.

SRTM Shuttle Radar Topography Mission.

STAR Standard Terminal Arrival Route.

T

TA Traffic Alerts.

TACAN Tactical Air Navigation System.

TAS True Air Speed.

TBVP Two-point Boundary Value Problem.

TCAS Traffic Collision Avoidance System.

TEFO Total Engine Flame Out.

TLS Transponder Landing System.

TMA Terminal Maneuvering Area.

TP Trajectory Prediction.

U

UTC Coordinated Universal Time.

V

VHF Very High Frequency.

VMO Maximum operating limit speed.

VOR VHF Omnidirectional Range.

Publications

Journal Publications

- Guitart, Delahaye, and Feron. “An accelerated dual fast marching tree applied to emergency geometric trajectory generation." Aerospace 9.4 (2022): 180.
- Guitart *et al.* “Collaborative Generation of Local Conflict Free Trajectories With Weather Hazards Avoidance." IEEE Transactions on Intelligent Transportation Systems (2023).
- Guitart *et al.* “Multi criteria methodology for aircraft trajectory planning algorithm selection: A survey.” Submitted to IEEE Transactions on Intelligent Transportation Systems.
- Guitart *et al.* “SID/STAR Design Optimization considering noise impact by Simulated Annealing: the Paris case study” Submitted to IEEE Transactions on Aerospace and Electronic Systems.
- Ligny *et al.* “A 3D Fast Marching method on tetrahedral meshes built from octrees and its application to aircraft emergency trajectory design considering wind.” Submitted to ACM Transactions on Mathematical Software.
- Saez *et al.* “An Automated Airborne Support Tool for Aircraft Emergencies: Selection of Landing Sites and 4D Diversion Trajectories.” Submitted to IEEE Transactions on Intelligent Transportation Systems.

Conference Publications

- Sáez *et al.* “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). 2021.
- Ligny *et al.* “Aircraft Emergency Trajectory Design: A Fast Marching Method on a Triangular Mesh." Fourteenth USA/Europe Air Traffic Management Research and Development Seminar. 2021.
- Bonin *et al.* “Optimal path planning for soaring flight." Conference on Guidance Navigation and control (CEAS EuroGNC 2022). 2022.
- Sáez *et al.* “An Automated Emergency Airport and Off-Airport Landing Site Selector." 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC). IEEE, 2022.
- Demouge, Guitart and Delahaye. “Fast Marching Tree applied to geodesic trajectories in presence of uncertain wind: a day of flights in Europe study." IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC). IEEE, 2023.

- Demouge *et al.* “An adapted Fast Marching Tree for contrails mitigation: A short-and long-range flight study." 26th IEEE International Conference on Intelligent Transportation Systems ITSC 2023. IEEE, 2023.
- Lebegue *et al.* “Aircraft cruise alternative trajectories generation: a mixed RRG-clustering approach." EAI INTSYS 2023 - 7th EAI International Conference on Intelligent Transport Systems. EAI, 2023.

Bibliography

[1] "Aircraft Operations" International Civil Aviation Organization Doc. 8168. Montreal, Quebec, Canada, 2006.

[2] *EUROCONTROL Long-Term Forecast: IFR Flight Movements 2010-203*, pages pp. 9–12. EUROCONTROL, December 2010.

[3] Compatibility in clearances issued. In *International Federation of Air Traffic Controllers' Associations (IFACTA) 58th annual conference*, 2019. URL `https://www.ifatca.org/wp-archive/2019-conchal/b55-compatibility-in-clearances.pdf`.

[4] Airbus. Could the humble dragonfly help pilots during flight? URL `https://www.airbus.com/en/newsroom/stories/2023-01-could-the-humble-dragonfly-help-pilots-during-flight`.

[5] K. Aizawa, K. Motomura, S. Kimura, R. Kadowaki, and J. Fan. Constant Time Neighbor Finding in Quadtrees: An Experimental Result. *2008 3rd International Symposium on Communications, Control and Signal Processing*, pages 505–510, 2008.

[6] V. Akgün, E. Erkut, and R. Batta. On finding dissimilar paths. *European Journal of Operational Research*, 121(2):232–246, Mar. 2000.

[7] A. Alonso-Ayuso, L. F. Escudero, and F. J. Martín-Campo. Collision avoidance in air traffic management: A mixed-integer linear optimization approach. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):47–57, 2010.

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

[9] E. M. Atkins, I. A. Portillo, and M. J. Strube. Emergency Flight Planning Applied to Total Loss of Thrust. *Journal of Aircraft*, 43(4):1205–1216, 2006.

[10] Aviation Benefits Beyond Borders. Global fact sheet, 2020. URL `https://aviationbenefits.org/media/167144/abbb20_factsheet_global.pdf`.

[11] Z. Baklouti. *Système de planification de chemins aériens en 3D :préparation de missions et replanification en cas d'urgence*. Phd thesis, Université Polytechnique Hauts de France, 2018.

[12] E. Bakolas, Y. Zhao, and P. Tsiotras. Initial guess generation for aircraft landing trajectory optimization. In *AIAA Guidance, Navigation, and Control Conference*. AIAA, 2011.

[13] R. Bartels, J. Beatty, and B. Barskyn. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Computer graphics. Morgan Kaufmann, 1998.

[14] V. Becerra. Psopt optimal control solver user manual. Technical report, 2011.

[15] J. Bellingham, Y. Kuwata, and J. How. Stable receding horizon trajectory control for complex environment. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. AIAA, 2003.

[16] R. Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[17] M. Berger and B. Gostiaux. *Differential geometry : Manifolds, Curves and Surfaces*. Springer-Verlag, 1988.

[18] J. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998.

[19] J. Betts and W. Huffman. Sparse optimal control software SOCS. Technical report, Mathematics and Engineering Analysis Technical Document MEALR-085, Boeing Information and Support Services,The Boeing Compan, 1997.

[20] J. Betts and W. Huffman. Mesh refinement in direct transcription methods for optimal control. *Optimal Control Applications and Methods*, 19(1):1–21, 1998.

[21] J. Betts, N. Biehn, S. Campbell, and W. Huffman. Compensating for order variation in mesh refinement for direct transcription methods. *Journal of Computational and Applied Mathematics*, 125:147–158, 2000.

[22] J. T. Betts and E. J. Cramer. Application of direct transcription to commercial aircraft trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 18(1):151–159, 1995.

[23] L. Bianco, P. Dell'Olmo, and A. R. Odoni. *New concepts and methods in air traffic management*. Springer Science & Business Media, 2001.

[24] A. Bicchi and L. Pallottino. On optimal cooperative conflict resolution for air traffic management systems. *IEEE Transactions on Intelligent Transportation Systems*, 1(4): 221–231, 2000.

[25] T. Binder, L. Blank, W. Dahmen, , and W. Marquardt. Grid refinement in multiscale dynamic optimization. Technical report, RWTH Aachen, 2000.

[26] Birkhoff and C. de Boor. Piecewise polynomial interpolation and approximation. In *Proceeding of the General Motors Symposium of 1964*. General Motors, 1964.

[27] N. A. S. Board. Aircraft accident report 92-11, el al flight 1862, boeing 747-258f 4x-axg, bijlmermeer, amsterdam, october 4 1992, 1994. URL `https://reports.aviation-safety.net/1992/19921004-2_B742_4X-AXG.pdf`.

[28] Boeing. Commercial market outlook 2019 - 2038, 2018. URL `https://www.boeing.com/resources/boeingdotcom/commercial/market/commercial-market-outlook/assets/downloads/cmo-sept-2019-report-final.pdf`.

[29] J. Borenstein, Y. Koren, et al. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991.

[30] F. Borrelli, D. Subramanian, A. U. Raghunathan, and L. T. Biegler. Milp and nlp techniques for centralized trajectory planning of multiple unmanned air vehicles. In *2006 American Control Conference*, pages 6–pp. IEEE, 2006.

[31] J. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM*, 20(2):100—106, 1977.

[32] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25—30, 1965.

[33] R. A. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):224–233, 1985.

[34] Bundesstalle für Flugunfalluntersuchung. Investigation report, 2004. URL `https://reports.aviation-safety.net/2002/20020701-1_B752_A9C-DHL_T154_RA-85816.pdf`.

[35] Bureau d'Enquete et Analyse (BEA). Rapport final: Accident survenu le 1^{er} juin 2009 à l'airbus a330-203, immatriculé f-gzcp, exploité par air france, vol af 447 rio de janeiro-paris, 2012. URL `https://bea.aero/docspa/2009/f-cp090601/pdf/f-cp090601.pdf`.

[36] S. Cafieri and N. Durand. Aircraft deconfliction with speed regulation: new models from mixed-integer optimization. *Journal of Global Optimization*, 58(4):613–629, 2014.

[37] A. Chakravarty. Four-dimensional fuel-optimal guidance in the presence of winds. *Journal of Guidance, Control, and Dynamics*, 8(1):16–22, 1985.

[38] H. K.-C. Chang and J.-L. Liu. A linear quadtree compression scheme for image encryption. 10(4):279–290. ISSN 09235965. doi: 10.1016/S0923-5965(96)00025-2. URL `https://linkinghub.elsevier.com/retrieve/pii/S0923596596000252`.

[39] J. Chevalier. *Optimisation des routes de départ et d'arrivée aux approches des grands aéroports*. PhD thesis, 2020. URL `http://www.theses.fr/2020TOU30092`. Thèse de doctorat dirigée par Maréchal, Pierre et Sbihi, Mohammed Mathématiques et Applications Toulouse 3 2020.

[40] T. Chondrogiannis, P. Bouros, J. Gamper, and U. Leser. Alternative routing: K-shortest paths with limited overlap. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, New York, NY, USA, 2015. Association for Computing Machinery. URL `https://doi.org/10.1145/2820783.2820858`.

[41] ChrisnHouston. Trajet du vol us airways 1549 le 15 janvier 2009, Feb 2019.

[42] Cirrus Aircraft. Vision jet: The next evolution is here. URL `https://cirrusaircraft.com/aircraft/vision-jet/`.

[43] Cordis and Cleansky. Safency - the safe emergency trajectory generator. URL `https://cordis.europa.eu/project/id/864771`.

[44] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.

[45] S. Deane. Fear of flying statistics, trends & facts (2022 data), 2022. URL `https://www.stratosjets.com/blog/fear-of-flying-statistics-trends-facts/`.

[46] D. Delahaye, S. Chaimatanan, and M. Mongeau. Simulated annealing: From basics to applications. In *Handbook of metaheuristics*, pages 1–35. Springer, 2019.

[47] B. N. Delaunay. Sur la Sphère Vide. *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Mathematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[48] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec. 1959.

[49] DIRCAM. Lfpg - paris charles de gaulle, 2022. URL `https://www.dircam.dsae.defense.gouv.fr/images/Stories/Doc/MIAC1/miac1_paris_cdg_lfpg.pdf`.

[50] Direction de la Circulation Aérienne militaire (DIRCAM). Lfpo - paris orly, 2022.

[51] Direction Générale de l'Aviation Civile (DGAC). Présentation illustrée des flux de trajectoires en région ile-de-france, 2019.

[52] N. Durand. *Optimisation de trajectoires pour la résolution de conflits aériens en route*. PhD thesis, INPT, 1996.

[53] N. Durand and J.-M. Alliot. Ant colony optimization for air traffic conflict resolution. In *Atm seminar 2009, 8th usa/europe air traffic management research and developpment seminar*, 2009.

[54] ECMWF. Ecmwf datasets, 2020. URL `https://www.ecmwf.int/en/forecasts/access-forecasts/access-archive-datasets`.

[55] P. Enright and B. Conway. Discrete approximation to optimal trajectories using direct transcription and nonlinear programming. *Journal of Guidance, Control, and Dynamics*, 15:994–1002, 1992.

[56] D. Eppstein. Z-order curve, mar 2010.

[57] H. Erzberger, T. A. Lauderdale, and Y.-C. Chu. Automated conflict resolution, arrival management, and weather avoidance for air traffic management. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 226 (8):930–949, 2012. doi: 10.1177/0954410011417347. URL `https://doi.org/10.1177/0954410011417347`.

[58] H. Erzberger, T. Nikoleris, R. A. Paielli, and Y.-C. Chu. Algorithms for control of arrival and departure traffic in terminal airspace. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 230(9):1762–1779, 2016. doi: 10.1177/0954410016629499. URL `https://doi.org/10.1177/0954410016629499`.

[59] Eurocontrol. Point merge. URL `https://www.eurocontrol.int/concept/point-merge`.

[60] Eurocontrol. Point merge integration of arrival flows enabling extensive rnav application and continuous descent - operational services and environment definition, 2010. URL `https://www.eurocontrol.int/sites/default/files/2019-06/point-merge-osed-v2.0-2010_1.pdf`.

[61] Eurocontrol. Free route developments in europe. Feb 2012.

[62] Eurocontrol. Sesar demonstration initial 4d trajectory. 2014.

[63] Eurocontrol. European aviation in 2040 - challenges of growth - annex 4: Network congestion, 2018. URL `https://www.eurocontrol.int/sites/default/files/2019-05/challenges-of-growth-2018-network-congestion-21122018.pdf`.

[64] G. Ewing. *Calculus of Variations with Applications*. Norton, New York 1969, reprinted by Dover, 1985.

[65] exosphere3d. Flight 1549 3d reconstruction, hudson river ditching jan 15, 2009, 2009. URL `https://www.youtube.com/watch?v=tE_5eiYn0D0`.

[66] A. F. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[67] A. Fallast and B. Messnarz. Automated trajectory generation and airport selection for an emergency landing procedure of a CS23 aircraft. *CEAS Aeronautical Journal*, 8(3): 481–492, sep 2017.

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

[69] Farin, Gerald, and H. Dianne. *The Essentials of CAGD*. A K Peters, Ltd, 2000.

[70] G. Farin. *Curves and surfaces for computer aided geometric design. A practical guide*. Academic Press, 1993.

[71] E. M. Feron, O. Sanni, M. Mote, D. Delahaye, T. Khamvilai, M. Gariel, and S. I. Saber. *Ariadne: A common-sense thread for enabling provable safety in air mobility systems with unreliable components*. doi: 10.2514/6.2022-0057. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2022-0057`.

[72] M. Finlay. Japan air lines flight 472: The douglas dc-8 that landed at the wrong airport, 2022. URL `https://simpleflying.com/japan-airlines-douglas-dc-8-wrong-airport/`.

[73] G. Foderaro, V. Raju, and S. Ferrari. A cell decomposition approach to online evasive path planning and the video game ms. pac-man. In *2011 IEEE International Symposium on Intelligent Control*, pages 191–197. IEEE, 2011.

[74] T. Fraichard and C. Laugier. Path-velocity decomposition revisited and applied to dynamic trajectory planning. In *[1993] Proceedings IEEE International Conference on Robotics and Automation*, pages 40–45. IEEE, 1993.

[75] J. Gammell, S. Srinivasa, and T. Barfoot. Bit* : Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation*, 2015.

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

[77] I. Gargantini. An effective way to represent quadtrees. 25(12):905–910. ISSN 0001-0782. doi: 10.1145/358728.358741. URL `https://doi.org/10.1145/358728.358741`.

[78] D. Gianazza and N. Durand. Separating air traffic flows by allocating 3D-trajectories. In *DASC 2004, 23rd Digital Avionics Systems Conference*, pages pp 2.D.4 – 21–13, Salt Lake City, United States, Oct. 2004. IEEE. doi: 10.1109/DASC.2004.1391275. URL `https://hal-enac.archives-ouvertes.fr/hal-00938063`.

[79] D. Gianazza and N. Durand. Assessment of the 3D-separation of Air Traffic Flows. In *ATM 2005, 6th USA/ Europe Air Traffic Management Research and Development Seminar*, page pp xxxx, Baltimore, United States, June 2005. URL `https://hal-enac.archives-ouvertes.fr/hal-00938079`.

[80] D. Gianazza, N. Durand, and N. Archambault. Allocating 3D-trajectories to air traffic flows using A* and genetic algorithms. In *CIMCA 2004, international conference on Computational Intelligence for Modelling, Control and Automation*, page pp xxxx, Gold Coast, Australia, July 2004. URL `https://hal-enac.archives-ouvertes.fr/hal-01020102`.

[81] P. Gianfranco. Shortest paths for Dubins vehicles in presence of via points. *IFAC Conference Paper Archive*, 52(8):295—300, 2019.

[82] K. Gierens, S. Matthes, and S. Rohs. How well can persistent contrails be predicted? *Aerospace*, 7(12), 2020.

[83] B. Girardet, L. Lapasset, D. Delahaye, and C. R. . Wind-optimal path planning: Application to aircraft trajectories. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1403–1408, Singapore, dec 2014. IEEE.

[84] Q. Gong, F. Fahroo, and I. Ross. Spectral algorithm for pseudospectral methods in optimal control. *Journal of Guidance, Control, and Dynamics*, 31(3):460–471, 2008.

[85] D. González-Arribas, M. Soler, M. Sanjurjo-Rivo, M. Kamgarpour, and J. Simarro. Robust aircraft trajectory planning under uncertain convective environments with optimal control and rapidly developing thunderstorms. *Aerospace Science and Technology*, 89:445–459, 2019.

[86] T. A. Granberg, T. Polishchuk, V. Polishchuk, and C. Schmidt. Automatic Design of Aircraft Arrival Routes with Limited Turning Angle. In M. Goerigk and R. Werneck, editors, *16th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2016)*, volume 54 of *OpenAccess Series in Informatics (OASIcs)*, pages 9:1–9:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-021-7. doi: 10.4230/OASIcs.ATMOS.2016.9. URL `http://drops.dagstuhl.de/opus/volltexte/2016/6533`.

[87] Groupe ADP. Combien y-a-t-il de décollages et d'atterrissages chaque année dans nos aéroports ?, 2019.

[88] H. Haghighi, D. Delahaye, and D. Asadi. Performance-based emergency landing trajectory planning applying meta-heuristic and dubins paths. *Applied Soft Computing*, 117:108453, 2022.

[89] C. Hargraves and S. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics*, 15:994–1002, 1992.

[90] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, 1968.

[91] T. Hearth, M. *Scientific Computing, An Introductory Survey*. Computer graphics. McGraw-Hill, 2002.

[92] D. Hentzen, M. Kamgarpour, M. Soler, and D. González-Arribas. On maximizing safety in stochastic aircraft trajectory planning with uncertain thunderstorm development. *Aerospace Science and Technology*, 79:543–553, 2018. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2018.06.006. URL https://www.sciencedirect.com/science/article/pii/S1270963818300956.

[93] E. Hernández Romero, A. Valenzuela Romero, and D. Rivas Rivas. Probabilistic aircraft conflict detection and resolution considering wind uncertainty. *7th SESAR Innovation Days (2017), pp. 1-8.*, 2017.

[94] X.-B. Hu, C. Zhang, G.-P. Zhang, M.-K. Zhang, H. Li, M. Leeson, and J.-Q. Liao. Finding the k shortest paths by ripple-spreading algorithms. *Engineering Applications of Artificial Intelligence*, 87:103229, 01 2020.

[95] W. Huifang, P. Lucia, and B. Antonio. Motion planning for Formations of Dubins Vehicles. In *Proceedings of the 49th IEEE Conference on Decision and Control*, Atlanta, Georgia, USA, 2010.

[96] ICAO. Performance-based navigation (pbn) manual, third edition, 2008. URL https://www.icao.int/sam/documents/2009/samig3/pbn%20manual%20-%20doc%209613%20final%205%2010%2008%20with%20bookmarks1.pdf.

[97] ICAO. Required navigation performance authorization required (rnp ar) procedure design manual, 2009. URL https://www.icao.int/meetings/pbn-symposium/documents/9905_cons_en.pdf.

[98] ICAO, PANS OPS. Construction of visual and instrument flight procedures, 2006.

[99] Insee. Historique des populations communales, 2020. URL https://www.insee.fr/fr/statistiques/3698339.

[100] Insee. Superficie: Occupation du sol - paysages, 2021. URL https://www.observatoire-des-territoires.gouv.fr/superficie.

[101] International Civil Aviation Organization. Rules of the air and air traffic services. 1996.

[102] International Civil Aviation Organization. Procedures for air navigation services,air traffic management, fifteen edition. 2007.

[103] International Civil Aviation Organization (ICAO). Aircraft Operations, Volume I, Flight Procedures. Technical report, Montréal, Quebec, Canada, 2006.

[104] E. A. Irvine and K. P. Shine. Ice supersaturation and the potential for contrail formation in a changing climate. *Earth System Dynamics*, 6(2):555–568, 2015.

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

[106] S. Jain and P. Tsiotras. Multiresolution-based direct trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 31(5):1424–1436, 2008.

[107] S. Jain and P. Tsiotras. Trajectory optimization using multiresolution techniques. *Journal of Guidance, Control, and Dynamics*, 31(5):1424–1436, 2008.

[108] L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast Marching Tree: a Fast Marching Sampling-Based Method for Optimal Motion Planning in Many Dimensions. *arXiv:1306.3532 [cs]*, Feb. 2015. URL `http://arxiv.org/abs/1306.3532`. arXiv: 1306.3532.

[109] M. Jardin and A. Bryson. Neighboring optimal aircraft guidance in winds. *Journal of Guidance, Control, and Dynamics*, 24:710–715, 2001.

[110] G. Jarry, D. Delahaye, F. Nicol, and E. Feron. Aircraft atypical approach detection using functional principal component analysis. *Journal of Air Transport Management*, 84:101787, 2020. ISSN 0969-6997. doi: https://doi.org/10.1016/j.jairtraman.2020.101787. URL `https://www.sciencedirect.com/science/article/pii/S0969699719303266`.

[111] G. Jarry, D. Delahaye, S. Puechmorel, and E. Féron. Real Time Aircraft Atypical Approach Detection for Air Traffic Control. working paper or preprint, Mar. 2021. URL `https://hal-enac.archives-ouvertes.fr/hal-03163779`.

[112] H. Jeffreys and B. S. Jeffreys. *Methods of Mathematical Physics*. Cambridge University Press, 1988.

[113] M. Kamgarpour, V. Dadok, and C. Tomlin. Trajectory generation for aircraft subject to dynamic weather uncertainty. In *49th IEEE Conference on Decision and Control (CDC)*, pages 2063–2068. IEEE, 2010.

[114] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The international journal of robotics research*, 5(3):72–89, 1986.

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

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

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

[118] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 500–505. IEEE, 1985.

[119] O. Khatib and J. Le Maitre. Dynamic controlof manipulators operating in a complex environment. In *On Theory and Practice of Robots and Manipulators, 3rd CISM-IFToMM Symp*, volume 267, 1978.

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

[121] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[122] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[123] S. Koenig, C. Tovey, and Y. Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1-2):253–279, 2003.

[124] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*. *Artificial Intelligence*, 155 (1-2):93–146, 2004.

[125] A. Kunio, M. Koyo, K. Shintaro, K. Ryosuke, and F. Jia. Constant time neighbor finding in quadtrees : An experimental result. In *2008 3rd International Symposium on Communications, Control and Signal Processing*, Saint Julian's, Malta, 2008.

[126] LaPoste. Base officielle des codes postaux, 2022. URL `https://www.data.gouv.fr/fr/datasets/base-officielle-des-codes-postaux/`.

[127] F. Large, S. Sekhavat, Z. Shiller, and C. Laugier. Towards real-time global motion planning in a dynamic environment using the nlvo concept. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 607–612. IEEE, 2002.

[128] S. LaValle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[129] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[130] J. Le Ny, E. Feron, and E. Frazzoli. On the Dubins Traveling Salesman Problem. *IEEE Transactions on Automatic Control*, 57(1), 2012.

[131] D. Lee, D. Fahey, A. Skowron, M. Allen, U. Burkhardt, Q. Chen, S. Doherty, S. Freeman, P. Forster, J. Fuglestvedt, A. Gettelman, R. De León, L. Lim, M. Lund, R. Millar, B. Owen, J. Penner, G. Pitari, M. Prather, R. Sausen, and L. Wilcox. The contribution of global aviation to anthropogenic climate forcing for 2000 to 2018. *Atmospheric Environment*, 244:117834, 2021.

[132] C. Letondal, C. Hurter, R. Lesbordes, J.-L. Vinot, and S. Conversy. Flights in my hands: coherence concerns in designing strip'tic, a tangible space for air traffic controllers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2175–2184, 2013.

[133] M. Liang, D. Delahaye, and P. Maréchal. Integrated sequencing and merging aircraft to parallel runways with automated conflict resolution and advanced avionics capabilities. *Transportation Research Part C: Emerging Technologies*, 85:268–291, 2017. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2017.09.012. URL `https://www.sciencedirect.com/science/article/pii/S0968090X17302528`.

[134] M. Lindhé, T. Keviczky, and K. H. Johansson. Multi-robot path following with visual connectivity. In *2011 Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 1466–1471. IEEE, 2011.

[135] F. Lingelbach. Path planning using probabilistic cell decomposition. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 1, pages 467–472. IEEE, 2004.

[136] H. Liu, C. Jin, B. Yang, and A. Zhou. Finding top-k shortest paths with diversity. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):488–502, 2017.

[137] W. Liu and I. Hwang. Probabilistic trajectory prediction and conflict detection for air traffic control. *Journal of Guidance, Control, and Dynamics*, 34(6):1779–1789, 2011. doi: 10.2514/1.53645. URL `https://doi.org/10.2514/1.53645`.

[138] W. Liu and I. Hwang. Probabilistic aircraft midair conflict resolution using stochastic optimal control. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):37–46, 2013.

[139] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[140] P. LU. Regulation about time-varying trajectories: Precision entry guidance illustrated. *Journal of Guidance, Control, and Dynamics*, 22:784–790, 1999.

[141] M. L. V. Pitteway. Algorithm for drawing ellipses or hyperbolae with a digital. *The Computer Journal*, 10(3):282—289, 1967.

[142] R. A. Marretta. *Unconventional Aeronautical Investigatory Methods: The Case of Alitalia Flight AZ 112*. Cambridge Scholars Publishing, 2021.

[143] Y. Matsuno, T. Tsuchiya, J. Wei, I. Hwang, and N. Matayoshi. Stochastic optimal control for aircraft conflict resolution under wind uncertainty. *Aerospace Science and Technology*, 43:77–88, 2015. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2015.02.018. URL `https://www.sciencedirect.com/science/article/pii/S1270963815000759`.

[144] T. McLain and R. Beard. Trajectory planning for coordinated rendezvous of unmanned air vehicles. In *AIAA Guidance, navigation, and control conference and exhibit*, page 4369, 2000.

[145] MetSafe. The weather made simple, 2021. URL `ttps://metsafeatm.com/services`.

[146] A. Meyer. Xavion, 2014. URL `https://xavion.com/`.

[147] S. Meyniel. Le 15 avril 1959 dans le ciel : Raid paris – dijon de la caravelle "lorraine" en vol plané. URL `https://www.air-journal.fr/2017-04-15-le-15-avril-1959-dans-le-ciel-raid-paris-dijon-de-la-caravelle-lorraine-en.html`.

[148] J.-M. Mirebeau. *Numerical schemes for anisotropic PDEs on cartesian grid domains*. thesis, Université Paris-Sud XI, May 2018. URL `https://tel.archives-ouvertes.fr/tel-01811841`.

[149] F. Mora-Camino. Avionique, lectures notes (in french). 1995.

[150] G. M. Morton. A computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. Technical report, IBM, Ottawa, Canada, 1966.

[151] National Safety Council Injury Facts. Deaths by transportation mode, 2021. URL `https://injuryfacts.nsc.org/home-and-community/safety-topics/deaths-by-transportation-mode/`.

[152] National Transportation Safety Board. NTSB Aircraft Accident Report - Southern Airways Inc., DC-9-31, N1335U, New Hope, Georgia, April 4, 1977.

[153] National Transportation Safety Board. Loss of thrust in both engines after encountering a flock of birds and subsequent ditching on the hudson river us airways flight 1549 airbus a320-214, n106us weehawken, new jersey january 15, 2009, 2010. URL `https://www.ntsb.gov/investigations/accidentreports/reports/aar1003.pdf`.

[154] H. K. Ng, B. Sridhar, S. Grabbe, and N. Chen. Cross-polar aircraft trajectory optimization and the potential climate impact. In *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, pages 3D4–1. IEEE, 2011.

[155] Nigeria's Accident Investigation Bureau. Report on the accident to adc airlines, boeing 737-2b7 registration 5n-bfk at tungar madaki, abuja on 29th october, 2006, 2008. URL `https://reports.aviation-safety.net/2006/20061029-0_B732_5N-BFK.pdf`.

[156] NTSB. Descent below visual glidepath and impact with seawall asiana airlines flight 214 boeing 777-200er, hl7742 san francisco, california july 6, 2013. URL `https://www.ntsb.gov/investigations/accidentreports/reports/aar1401.pdf`.

[157] NTSB. Aircraft accident report - eastern air lines, inc., douglas dc-9-31, n8984e, charlotte, north carolina, september 11, 1974, 1975. URL `https://www.ntsb.gov/investigations/AccidentReports/Reports/AAR7509.pdf`.

[158] NTSB. Response to final aircraft accident investigation report ethiopian airlines flight 302 boeing 737-8 max, et-avj ejere, ethiopia march 10, 2019, 2023. URL `https://www.ntsb.gov/investigations/Documents/ResponsetoEAIBfinalreport.pdf`.

[159] K. O. *Dynamic control in the operational space of manipulator robots in the presence of obstacles.* PhD thesis, ENSAE, 1980.

[160] H. Oberle and W. Grimm. BNDSCO - A program for the numerical solution of optimal control problems. Technical report, Institute for Flight System Dynamics, German Aerospace Research Establishment Oberpfaffenhofen, 1989.

[161] T. Ohtsuka. Quasi-Newton-type continuation method for nonlinear receding horizon control. *Journal of Guidance, Control, and Dynamics*, 24:685–692, 2002.

[162] L. Pallottino, E. M. Feron, and A. Bicchi. Conflict resolution problems for air traffic management systems solved with mixed integer programming. *IEEE transactions on intelligent transportation systems*, 3(1):3–11, 2002.

[163] Panish Shea Boyle Ravipudi. Aviation and plane crash statistics, 2020. URL `https://www.psbr.law/aviation_accident_statistics.html`.

[164] J. Pannequin, A. Bayen, I. Mitchell, H. Chung, and S. Sastry. Multiple aircraft deconflicted path planning with weather avoidance constraints. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 6588, 2007.

[165] V. Polishchuk. Generating arrival routes with radius-to-fix functionalities. In *7th International Conference on Research in Air Transportation (ICRAT 2016)*. Drexel University Philadelphia, 2016.

[166] L. Pontryagin, V. Boltyanski, R. Gamkrelidze, and E. Mischenko. *The Mathematical Theory of Optimal Processes.* New York, NY: Interscience, 1962.

[167] M. Prandini, J. Hu, J. Lygeros, and S. Sastry. A probabilistic approach to aircraft conflict detection. *IEEE Transactions on intelligent transportation systems*, 1(4):199–220, 2000.

[168] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.

[169] J. Ramsay and B. Silverman. *Functional Data Analysis.* Springer Series in Statistics. Springer, 2005. ISBN 9780387400808.

[170] A. Rao, D. Benson, and G. Huntington. User's manual for GPOPS version 4.x: A matlab package for software for solving multiple-phase optimal control problems using hp-adaptive pseudospectral methods. Technical report, 2011.

[171] 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.

[172] S. J. Riley, S. D. DeGloria, and R. Elliot. Index that quantifies topographic heterogeneity. *intermountain Journal of sciences*, 5(1-4):23–27, 1999.

[173] E. Rimon and D. E. Koditschek. Exact robot navigation using cost functions: the case of distinct spherical boundaries in e/sup n. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 1791–1796. IEEE, 1988.

[174] I. Ross. 'user's manual for DIDO: A MATLAB application package for solving optimal control problems. Technical report, Naval Postgraduate School, 2005.

[175] R. Russell and L. Shampine. A collocation method for boundary value problems. *Numerische Mathematik*, 19:13–36, 1972.

[176] G. Satyanarayana, D. C. Manyam, A. L. Von Moll, and Z. Fuchs. Shortest Dubins Path to a circle. In *AIAA Scitech 2019 Forum*, San Diego, California, USA, 2019.

[177] A. Schwartz. *Theory and Implementation of Numerical Methods Based on Runge-Kutta Integration for Solving Optimal Control Problems.* PhD thesis, Université Montpellier II (France), 1996.

[178] D. B. Seenivasan, A. Olivares, and E. Staffetti. Multi-aircraft optimal 4d online trajectory planning in the presence of a multi-cell storm in development. *Transportation Research Part C: Emerging Technologies*, 110:123–142, 2020.

[179] J. A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, Feb. 1996.

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

[181] J. A. Sethian and A. Vladimirsky. Ordered upwind methods for static Hamilton–Jacobi equations. *Proceedings of the National Academy of Sciences*, 98(20):11069–11074, Sept. 2001.

[182] H. Seywald and E. Cliff. Neighboring optimal control based feedback law for the advanced launch system. *Journal of Guidance, Control, and Dynamics*, 17:1154–1162, 1994.

[183] Simple Flying. Azores glider: The story of air transat flight 236. URL `https://simpleflying.com/azores-glider-the-story-of-air-transat-flight-236/`.

[184] R. Sivan. *Surface modeling using quadtrees.* PhD thesis, Center for Automation, University of Maryland, College Park, MD, 1996.

[185] M. Soler, B. Zou, and M. Hansen. Flight trajectory design in the presence of contrails: Application of a multiphase mixed-integer optimal control approach. *Transportation Research Part C: Emerging Technologies*, 48:172–194, Nov. 2014.

[186] B. Sridhar, H. Ng, and N. Chen. Aircraft trajectory optimization and contrails avoidance in the presence of winds. *Journal of Guidance, Control, and Dynamics*, 34:1577–1583, 2011.

[187] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE international conference on robotics and automation*, pages 3310–3317. IEEE, 1994.

[188] A. Stentz et al. The focussed d* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[189] S. Summers, M. Kamgarpour, J. Lygeros, and C. Tomlin. A stochastic reach-avoid problem with random obstacles. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 251–260, 2011.

[190] L. Talarico, K. Sörensen, and J. Springael. The k-dissimilar vehicle routing problem. *European Journal of Operational Research*, 244(1):129–140, 2015.

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

[192] C. Taylor, S. Liu, C. Wanke, and T. Stewart. Generating diverse reroutes for tactical constraint avoidance. *Journal of Air Transportation*, 26(2):49–59, 2018.

[193] I. Ulrich and J. Borenstein. Vfh/sup*: Local obstacle avoidance with look-ahead verification. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 3, pages 2505–2511. IEEE, 2000.

[194] A. L. Visintini, W. Glover, J. Lygeros, and J. Maciejowski. Monte carlo optimization for conflict resolution in air traffic control. *IEEE Transactions on Intelligent Transportation Systems*, 7(4):470–482, 2006.

[195] K. Wang, J. Xu, K. Song, Y. Yan, and Y. Peng. Informed anytime bi-directional fast marching tree for optimal motion planning in complex cluttered environments. *Expert Systems with Applications*, 215:119263, 2023. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.119263. URL `https://www.sciencedirect.com/science/article/pii/S0957417422022813`.

[196] R. Weiss. The application of implicit Runge-Kutta and collocation methods to boundary value problems. *Mathematics of Computation*, 28:449–464, 1974.

[197] P. Williams. Application of pseudospectral methods for receding horizon control. *Journal of Guidance, Control, and Dynamics*, 27:310–314, 2004.

[198] J. Xia, Z. Jiang, H. Zhang, R. Zhu, and H. Tian. Dual fast marching tree algorithm for human-like motion planning of anthropomorphic arms with task constraints. *IEEE/ASME Transactions on Mechatronics*, 26(5):2803–2813, 2021. doi: 10.1109/TMECH.2020.3047476.

[199] J. Xu, K. Song, D. Zhang, H. Dong, Y. Yan, and Q. Meng. Informed anytime fast marching tree for asymptotically optimal motion planning. *IEEE Transactions on Industrial Electronics*, 68(6):5068–5077, 2021. doi: 10.1109/TIE.2020.2992978.

[200] H. Yan, F. Fahroo, and I. Ross. Real-time computation of neighboring optimal control laws. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. AIAA, 2002.

[201] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17 (11):712–716, 1971.

[202] Y.-H. Yu and Y.-T. Zhang. Collision avoidance and path planning for industrial manipulator using slice-based heuristic fast marching tree. *Robotics and Computer-Integrated Manufacturing*, 75:102289, 2022. ISSN 0736-5845. doi: https://doi.org/10.1016/j.rcim.2021.102289. URL `https://www.sciencedirect.com/science/article/pii/S0736584521001691`.

[203] E. Zermelo. Über das Navigationsproblem bei ruhender oder veränderlicher Windverteilung. *Zeitschrift für Angewandte Mathematik und Mechanik*, 11(2):114–124, 1931.

[204] Y. Zhao. *Efficient and robust aircraft landing trajectory optimization*. PhD thesis, Georgia Institute of Technology, 2012.

[205] Y. Zhao. Efficient and robust aircraft landing trajectory optimization. *undefined*, 2012. URL `https://www.semanticscholar.org/paper/Efficient-and-robust-aircraft-landing-trajectory-Zhao/eabc8d79d22e8a2e194d7e29ebceb2bd703c9781`.

[206] Y. Zhao and P. Tsiotras. Time-optimal parameterization of geometric path for fixed-wing aircraft. In *Infotech@Aerospace*. AIAA, 2010.

[207] Y. Zhao and P. Tsiotras. Density functions for mesh refinement in numerical optimal control. *Journal of Guidance, Control, and Dynamics*, 34(1):271–277, 2010.

[208] Y. Zhao and P. Tsiotras. Stable receding horizon trajectory control for complex environment. In *American Control Conference*. ACC, 2011.

[209] J. Zhou. *Optimal Design of SIDs/STARs in Terminal Maneuvering Area*. PhD thesis, Université Toulouse 3 Paul Sabatier, 2017.

[210] J. Zhou, S. Cafieri, D. Delahaye, and M. Sbihi. Optimization of Arrival and Departure Routes in Terminal Maneuvering Area. In *ICRAT 2014, 6th International Conference on Research in Air Transportation*, page pp xxxx, Istanbul, Turkey, May 2014. URL `https://hal-enac.archives-ouvertes.fr/hal-01002807`.

[211] J. Zhou, S. Cafieri, D. Delahaye, and M. Sbihi. Optimizing the Design of a Route in Terminal Maneuvering Area Using Branch and Bound. In *EIWAC 2015, 4th ENRI International Workshop on ATM/CNS*, volume 420 of *Lecture Notes in Electrical Engineering*, pages pp 171–184, Tokyo, Japan, Nov. 2015. Springer. doi: 10.1007/978-4-431-56423-2. URL `https://hal-enac.archives-ouvertes.fr/hal-01503222`.

Appendix A

Dubins curve

A.1 Left Segment Left

In the case of LSL Dubins curve, the following constraint is defined:

$$L_{\phi_1}(S_l(L_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.1}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha + \phi_1 + \phi_2 = \beta \\ x_2 = x_1 + r_1 \sin(\alpha + \phi_1) - r_1 \sin\alpha + l\cos(\alpha + \phi_1) + r_2 \sin\beta + r_2 \sin(\alpha + \phi_1) \\ y_2 = y_1 - r_1 \cos(\alpha + \phi_1) + r_1 \cos\alpha + l\sin(\alpha + \phi_1) - r_2 \cos\beta + r_2 \cos(\alpha + \phi_1) \end{cases} \tag{A.2}$$

This system can be rewritten as follows:

$$\begin{cases} \alpha + \phi_1 + \phi_2 = \beta \\ \Delta X = (r_1 - r_2)\sin(\alpha + \phi_1) + l\cos(\alpha + \phi_1) \\ \Delta Y = (r_2 - r_1)\cos(\alpha + \phi_1) + l\sin(\alpha + \phi_1) \end{cases} \tag{A.3}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1 \sin\alpha - r_2 \sin\beta \\ \Delta Y = y_2 - y_1 - r_1 \cos\alpha + r_2 \cos\beta \end{cases} \tag{A.4}$$

By squaring the two previous equations, we obtain:

$$\begin{cases} \Delta X^2 = (r_1 - r_2)^2 \sin^2(\alpha + \phi_1) + l^2 \cos^2(\alpha + \phi_1) \\ \qquad\qquad + 2(r_1 - r_2) l \sin(\alpha + \phi_1)\cos(\alpha + \phi_1) \\ \Delta Y^2 = (r_2 - r1)^2 \cos^2(\alpha - \phi_1) + l^2 \cos^2(\alpha + \phi_1) \\ \qquad\qquad - 2(r_1 - r_2) l \sin(\alpha + \phi_1)\cos(\alpha + \phi_1) \end{cases} \tag{A.5}$$

By summing the two, the unknown ϕ_1 disappears, and the following equation is obtained:

$$\Delta X^2 + \Delta Y^2 = (r_2 - r_1)^2 + l^2 \tag{A.6}$$

Finally, the straight line length can be computed as follows:

$$l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \tag{A.7}$$

To compute ϕ_1, $\tan(\alpha + \phi_1)$ is determined by modifying Equation (A.3). The new system is the following:

$$\begin{cases} \Delta X(r_1 - r_2) = (r_1 - r_2)^2 \sin(\alpha + \phi_1) + l(r_1 - r2)\cos(\alpha + \phi_1) \\ \Delta Y l = l(r_2 - r_1)\cos(\alpha + \phi_1) + l^2 \sin(\alpha + \phi_1) \end{cases} \tag{A.8}$$

$$\Delta X(r_1 - r_2) + \Delta Y l = ((r_1 - r_2)^2 + l^2)\sin(\alpha + \phi_1) \tag{A.9}$$

$$\sin(\alpha + \phi_1) = \frac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X^2 + \Delta Y^2} \tag{A.10}$$

$$\cos(\alpha + \phi_1) = \frac{\Delta X l + \Delta Y(r_2 - r_1)}{\Delta X^2 + \Delta Y^2} \tag{A.11}$$

$$\tan(\alpha + \phi_1) = \frac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X l + \Delta Y(r_2 - r_1)} \tag{A.12}$$

Finally, ϕ_1, l and ϕ_2 are computed as follows:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X l + \Delta Y(r_2 - r_1)}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \\ \phi_2 = \beta - \arctan\left(\dfrac{\Delta X(r_1 - r_2) + \Delta Y l}{\Delta X l + \Delta Y(r_2 - r_1)}\right) \end{cases} \tag{A.13}$$

A.2 Right Segment Right

In the case of RSR Dubins curve, the following constraint is defined:

$$R_{\phi_1}(S_l(R_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.14}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha - \phi_1 - \phi_2 = \beta \\ x_2 = x_1 - r_1\sin(\alpha - \phi_1) + r_1\sin\alpha + l\cos(\alpha - \phi_1) - r_2\sin\beta + r_2\sin(\alpha - \phi_1) \\ y_2 = y_1 + r_1\cos(\alpha - \phi_1) - r_1\cos\alpha + l\sin(\alpha - \phi_1) + r_2\cos\beta - r_2\cos(\alpha - \phi_1) \end{cases} \tag{A.15}$$

The demonstration is similar to the previous one.

$$\begin{cases} \alpha - \phi_1 - \phi_2 = \beta \\ \Delta X = (r_2 - r_1)\sin(\alpha - \phi_1) + l\cos(\alpha - \phi_1) \\ \Delta Y = (r_1 - r_2)\cos(\alpha - \phi_1) + l\sin(\alpha - \phi_1) \end{cases} \tag{A.16}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1\sin\alpha + r_2\sin\beta \\ \Delta Y = y_2 - y_1 + r_1\cos\alpha - r_2\cos\beta \end{cases} \tag{A.17}$$

$$l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \tag{A.18}$$

$$\sin(\alpha - \phi_1) = \frac{\Delta X(r2 - r1) + \Delta Y l}{\Delta X^2 + \Delta Y^2} \tag{A.19}$$

$$\cos(\alpha - \phi_1) = \frac{\Delta X l + \Delta Y(r_1 - r_2)}{\Delta X^2 + \Delta Y^2} \tag{A.20}$$

ϕ_1, l and ϕ_2 are computed as follows:

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(r_2 - r_1) + \Delta Y l}{\Delta X l + \Delta Y(r_1 - r_2)}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 - r_1)^2} \\ \phi_2 = -\beta + \arctan\left(\dfrac{\Delta X(r_2 - r_1) + \Delta Y l}{\Delta X l + \Delta Y(r_1 - r_2)}\right) \end{cases} \tag{A.21}$$

A.3 Left Segment Right

In the case of RSL Dubins curve, the following constraint is defined:

$$R_{\phi_1}(S_l(L_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.22}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha + \phi_1 - \phi_2 = \beta \\ \Delta X = (r_2 + r_1)\sin(\alpha + \phi_1) + l\cos(\alpha + \phi_1) \\ \Delta Y = -(r_2 + r_1)\cos(\alpha + \phi_1) + l\sin(\alpha + \phi_1) \end{cases} \tag{A.23}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1\sin\alpha + r_2\sin\beta \\ \Delta Y = y_2 - y_1 - r_1\cos\alpha - r_2\cos\beta \end{cases} \tag{A.24}$$

The demonstration is similar to the two previous ones.

$$l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 + r_1)^2} \tag{A.25}$$

$$\sin(\alpha + \phi_1) = \frac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X^2 + \Delta Y^2} \tag{A.26}$$

$$\cos(\alpha + \phi_1) = \frac{\Delta X l - (r_1 + r_2)\Delta Y}{\Delta X^2 + \Delta Y^2} \tag{A.27}$$

$$\tan(\alpha + \phi_1) = \frac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y} \tag{A.28}$$

ϕ_1, l and ϕ_2 are computed as follows:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 + r_1)^2} \\ \phi_2 = -\beta + \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \end{cases} \tag{A.29}$$

A.4 Right Segment Left

In the case of RSL Dubins curve, the following constraint is defined:

$$R_{\phi_1}(S_l(L_{\phi_2}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.30}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha - \phi_1 + \phi_2 = \beta \\ \Delta X = -(r_2 + r_1)\sin(\alpha - \phi_1) + l\cos(\alpha - \phi_1) \\ \Delta Y = (r_2 + r_1)\cos(\alpha - \phi_1) + l\sin(\alpha - \phi_1) \end{cases} \tag{A.31}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1\sin\alpha - r_2\sin\beta \\ \Delta Y = y_2 - y_1 + r_1\cos\alpha + r_2\cos\beta \end{cases} \tag{A.32}$$

The demonstration is similar to the previous demonstrations.

$$l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 + r_1)^2} \tag{A.33}$$

$$\sin(\alpha - \phi_1) = \frac{-\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X^2 + \Delta Y^2} \tag{A.34}$$

$$\cos(\alpha - \phi_1) = \frac{\Delta X l + (r_1 + r_2)\Delta Y}{\Delta X^2 + \Delta Y^2} \tag{A.35}$$

$$\tan(\alpha - \phi_1) = \frac{-\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l + (r_1 + r_2)\Delta Y} \tag{A.36}$$

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \\ l = \sqrt{\Delta X^2 + \Delta Y^2 - (r_2 + r_1)^2} \\ \phi_2 = \beta - \arctan\left(\dfrac{\Delta X(r_1 + r_2) + \Delta Y l}{\Delta X l - (r_1 + r_2)\Delta Y}\right) \end{cases} \tag{A.37}$$

A.5 Left Right Left

In the case of LRL Dubins curve, the following constraint is defined:

$$L_{\phi_1}(R_{\phi_2}(L_{\phi_3}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.38}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha + \phi_1 - \phi_2 + \phi_3 = \beta \\ x_2 = x_1 + r_1\sin(\alpha + \phi_1) - r_1\sin\alpha - r_2\sin(\alpha + \phi_1 - \phi_2) \\ \qquad + r_2\sin(\alpha + \phi_1) + r_3\sin\beta - r_3\sin(\alpha + \phi_1 - \phi_2) \\ y_2 = y_1 - r_1\cos(\alpha + \phi_1) + r_1\cos\alpha + r_2\cos(\alpha + \phi_1 - \phi_2) \\ \qquad - r_2\cos(\alpha + \phi_1) - r_3\cos\beta + r_3\cos(\alpha + \phi_1 - \phi_2) \end{cases} \tag{A.39}$$

The previous can be rewritten as follows:

$$\begin{cases} \alpha + \phi_1 - \phi_2 + \phi_3 = \beta \\ \Delta X = (r_1 + r_2)\sin(\alpha + \phi_1) - (r_2 + r_3)\sin(\alpha + \phi_1 - \phi_2) \\ \Delta Y = -(r_1 + r_2)\cos(\alpha + \phi_1) + (r_2 + r_3)\cos(\alpha + \phi_1 - \phi_2) \end{cases} \tag{A.40}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 + r_1\sin\alpha - r_3\sin\beta \\ \Delta Y = y_2 - y_1 - r_1\cos\alpha + r_3\cos\beta \end{cases} \tag{A.41}$$

To determine ϕ_2, the principle of the demonstration is similar to Circle-Segment-Circle demonstrations.

$$\begin{cases} \alpha + \phi_1 - \phi_2 + \phi_3 = \beta \\ \Delta X^2 = (r_1 + r_2)^2 \sin^2(\alpha + \phi_1) + (r_2 + r_3)^2 \sin^2(\alpha + \phi_1 - \phi_2) \\ \qquad - 2(r_1 + r_2)(r_2 + r_3) \sin(\alpha + \phi_1) \sin(\alpha + \phi_1 - \phi_2) \\ \Delta Y^2 = (r_1 + r_2)^2 \cos^2(\alpha + \phi_1) + (r_2 + r_3)^2 \cos^2(\alpha + \phi_1 - \phi_2) \\ \qquad - 2(r_1 + r_2)(r_2 + r_3) \cos(\alpha + \phi_1) \cos(\alpha + \phi_1 - \phi_2) \end{cases} \tag{A.42}$$

$$\Delta X^2 + \Delta Y^2 = (r_1 + r_2)^2 + (r_2 + r_3)^2 - 2(r_1 + r_2)(r_2 + r_3) \cos \phi_2 \tag{A.43}$$

$$\phi_2 = \arccos\left(\frac{(r_1 + r_2)^2 + (r_2 + r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1 + r_2)(r_2 + r_3)}\right) \tag{A.44}$$

ϕ_1 is computed by determining the value of $\tan(\alpha + \phi_1)$.

$$\begin{cases} \alpha + \phi_1 - \phi_2 + \phi_3 = \beta \\ \Delta X = ((r_1 + r_2) - (r_2 + r_3) \cos \phi_2) \sin(\alpha + \phi_1) + (r_2 + r_3) \sin \phi_2 \cos(\alpha + \phi_1) \\ \Delta Y = (r_2 + r_3) \sin p \sin(\alpha + \phi_1) + (-(r_1 + r_2) + (r_2 + r_3) \cos \phi_2) \cos(\alpha + \phi_1) \end{cases} \tag{A.45}$$

$$\sin(\alpha + \phi_1) = \frac{\Delta X((r_1 + r_2) - (r_2 + r_3) \cos \phi_2) + \Delta Y(r_2 + r_3) \sin \phi_2}{\Delta X^2 + \Delta Y^2} \tag{A.46}$$

$$\cos(\alpha + \phi_1) = \frac{\Delta X(r_2 + r_3) \sin \phi_2 + \Delta Y(-(r_1 + r_2) + (r_2 + r_3) \cos \phi_2)}{\Delta X^2 + \Delta Y^2} \tag{A.47}$$

$$\tan(\alpha + \phi_1) = \frac{\Delta X((r_1 + r_2) - (r_2 + r_3) \cos \phi_2) + \Delta Y(r_2 + r_3) \sin \phi_2}{\Delta X(r_2 + r_3) \sin \phi_2 + \Delta Y(-(r_1 + r_2) + (r_2 + r_3) \cos \phi_2)} \tag{A.48}$$

Finally, the three angles ϕ_1, ϕ_2 and ϕ_3 are computed as follows:

$$\begin{cases} \phi_1 = -\alpha + \arctan\left(\dfrac{\Delta X((r_1 + r_2) - (r_2 + r_3) \cos \phi_2) + \Delta Y(r_2 + r_3) \sin \phi_2}{\Delta X(r_2 + r_3) \sin \phi_2 + \Delta Y(-(r_1 + r_2) + (r_2 + r_3) \cos \phi_2)}\right) \\ \phi_2 = \arccos\left(\dfrac{(r_1 + r_2)^2 + (r_2 + r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1 + r_2)(r_2 + r_3)}\right) \\ \phi_3 = \beta - \arctan\left(\dfrac{\Delta X((r_1 + r_2) - (r_2 + r_3) \cos \phi_2) + \Delta Y(r_2 + r_3) \sin \phi_2}{\Delta X(r_2 + r_3) \sin \phi_2 + \Delta Y(-(r_1 + r_2) + (r_2 + r_3) \cos \phi_2)}\right) \\ \qquad + \arccos\left(\dfrac{(r_1 + r_2)^2 + (r_2 + r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1 + r_2)(r_2 + r_3)}\right) \end{cases} \tag{A.49}$$

A.6 Right Left Right

In the case of RLR Dubins curve, the following constraint is defined:

$$L_{\phi_1}(R_{\phi_2}(L_{\phi_3}(x_1, y_1, \alpha))) = (x_2, y_2, \beta) \tag{A.50}$$

This constraint leads to the following equations system:

$$\begin{cases} \alpha - \phi_1 + \phi_2 + \phi_3 = \beta \\ \Delta X = -(r_1 + r_2) \sin(\alpha - \phi_1) + (r_2 + r_3) \sin(\alpha - \phi_1 + \phi_2) \\ \Delta Y = (r_1 + r_2) \cos(\alpha - \phi_1) - (r_2 + r_3) \cos(\alpha - \phi_1 + \phi_2) \end{cases} \tag{A.51}$$

where:

$$\begin{cases} \Delta X = x_2 - x_1 - r_1 \sin\alpha + r_3 \sin\beta \\ \Delta Y = y_2 - y_1 + r_1 \cos\alpha - r_3 \cos\beta \end{cases} \tag{A.52}$$

The demonstration is similar to the previous one.

$$\phi_2 = \arccos\left(\frac{(r_1+r_2)^2 + (r_2+r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \tag{A.53}$$

$$\sin(\alpha - \phi_1) = \frac{\Delta X(-(r_1+r_2) + (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin\phi_2}{\Delta X^2 + \Delta Y^2} \tag{A.54}$$

$$\cos(\alpha - \phi_1) = \frac{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y((r_1+r_2) - (r_2+r_3)\cos\phi_2)}{\Delta X^2 + \Delta Y^2} \tag{A.55}$$

$$\tan(\alpha - \phi_1) = \frac{\Delta X(-(r_1+r_2) + (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y((r_1+r_2) - (r_2+r_3)\cos\phi_2)} \tag{A.56}$$

The three angles ϕ_1, ϕ_2 and ϕ_3 are computed as follows:

$$\begin{cases} \phi_1 = \alpha - \arctan\left(\dfrac{\Delta X(-(r_1+r_2) + (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y((r_1+r_2) - (r_2+r_3)\cos\phi_2)}\right) \\ \phi_2 = \arccos\left(\dfrac{(r_1+r_2)^2 + (r_2+r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \\ \phi_3 = -\beta + \arctan\left(\dfrac{\Delta X((r_1+r_2) - (r_2+r_3)\cos\phi_2) + \Delta Y(r_2+r_3)\sin\phi_2}{\Delta X(r_2+r_3)\sin\phi_2 + \Delta Y(-(r_1+r_2) + (r_2+r_3)\cos\phi_2)}\right) \\ \qquad\quad + \arccos\left(\dfrac{(r_1+r_2)^2 + (r_2+r_3)^2 - \Delta X^2 - \Delta Y^2}{2(r_1+r_2)(r_2+r_3)}\right) \end{cases} \tag{A.57}$$

Appendix B

Heading Constraints over a Single Glide Slope

The objective of this appendix is to re-express the aeronautical constraints operating on an aircraft when it descends over a single glide slope.

The original aeronautical constraints obtained in Section 2.4.1 were:

$$\pi \subset \mathcal{GS} \tag{B.1a}$$

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

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

In the special case where the engines are lost, the aircraft is limiting by a minimum negative flight path angle, forcing the aircraft to go down. The new constraints are:

$$\pi \subset \mathcal{GS} \tag{B.2a}$$

$$\gamma_{\min} \leq \gamma \leq \gamma_{\max}^{\text{gliding}} \tag{B.2b}$$

$$\left|\frac{\partial \sigma}{\partial x}\right| \leq \frac{1}{r_{\min}} \tag{B.2c}$$

An idea is to express the two first constraints in terms of heading σ. Indeed, working with an algorithm in two dimensions, it is difficult to verify equation (B.2b) because the glide slope is fixed. Thus, one must find the relationship between the heading angle σ and the flight path angle γ in order to ensure that (B.2b) is satisfied.

Consider $\overrightarrow{v} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$ the true airspeed of the aircraft and γ_0 the fixed glide slope. As shown on Figure B.1, we have $\overrightarrow{d} = \begin{pmatrix} \cos\gamma_0 \\ 0 \\ -\sin\gamma_0 \end{pmatrix}$ and $-\overrightarrow{z_a} = -\begin{pmatrix} \sin\gamma_0 \\ 0 \\ \cos\gamma_0 \end{pmatrix}$. To simplify calculations, assume that the descent direction of the glide slope $\overrightarrow{d}$ is oriented towards North, thus in that direction $\sigma = 0$.

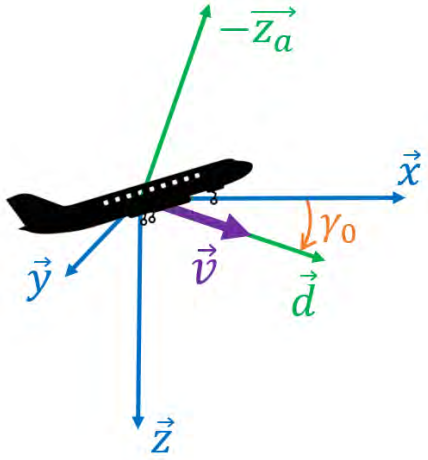


Figure B.1: Aircraft in the Earth frame, where $\overrightarrow{x}$ is oriented towards North. $\overrightarrow{d}$ is the direction of the glide slope of negative constant angle γ_0 in the $(\overrightarrow{x}, \overrightarrow{z})$-plane.

Let σ be the heading of the aircraft. The airspeed $\overrightarrow{v}$ is oriented along $\overrightarrow{x_a}$, which is defined as a rotation of $\overrightarrow{d}$ by an angle σ around the vector $-\overrightarrow{z_a}$.

With mathematical formalism:

$$\begin{aligned}
\overrightarrow{x_a} &= \mathrm{rot}_{-\overrightarrow{z_a}}(\overrightarrow{d}, \sigma) && \text{(B.3)}\\
&= \mathcal{M}_{\mathrm{rot}}(-\overrightarrow{z_a}, \sigma) \cdot \overrightarrow{d} && \text{(B.4)}\\
&= \begin{pmatrix} \sin^2\sigma(1-\cos\sigma)+\cos\sigma & \cos\gamma_0\sin\sigma & \cos\gamma_0\sin\gamma_0(1-\cos\sigma) \\ -\cos\gamma_0\sin\sigma & \cos\sigma & \sin\gamma_0\sin\sigma \\ \cos\gamma_0\sin\gamma_0(1-\cos\sigma) & -\sin\gamma_0\sin\sigma & \cos^2\gamma_0(1-\cos\sigma)+\cos\sigma \end{pmatrix} \cdot \begin{pmatrix} \cos\gamma_0 \\ 0 \\ -\sin\gamma_0 \end{pmatrix} && \text{(B.5)}\\
&= \begin{pmatrix} \cos\gamma_0\cos\sigma \\ -\sin\sigma \\ -\sin\gamma_0\cos\sigma \end{pmatrix} && \text{(B.6)}
\end{aligned}$$

Hence the airspeed of the aircraft is expressed as follows in the Earth frame

$$\overrightarrow{v} = ||\overrightarrow{v}|| \begin{pmatrix} \cos\gamma_0\cos\sigma \\ -\sin\sigma \\ -\sin\gamma_0\cos\sigma \end{pmatrix} \tag{B.7}$$

The flight path angle γ is defined as the angle between $\overrightarrow{x_a}$ and the horizontal plane, thus the angle between $\overrightarrow{v}$ and the horizontal airspeed $\overrightarrow{v_h}$ defined as

$$\overrightarrow{v_h} = ||\overrightarrow{v}|| \begin{pmatrix} \cos\gamma_0\cos\sigma \\ -\sin\sigma \\ 0 \end{pmatrix} \tag{B.8}$$

To express γ, one has to compute $\overrightarrow{v} \times \overrightarrow{v_h}$ and $\overrightarrow{v} \cdot \overrightarrow{v_h}$. Hence,

$$\overrightarrow{v} \times \overrightarrow{v_h} = ||\overrightarrow{v}||^2 \begin{pmatrix} -\sin\gamma_0\cos\sigma\sin\sigma \\ -\cos\gamma_0\sin\gamma_0\cos^2\sigma \\ 0 \end{pmatrix} \tag{B.9}$$

and then, by taking the norm

$$||\overrightarrow{v} \times \overrightarrow{v_h}|| = ||\overrightarrow{v}||^2 \sin\gamma_0 \cos\sigma \sqrt{\cos^2\gamma_0 \cos^2\sigma + \sin^2\sigma} \tag{B.10}$$

Also,

$$\overrightarrow{v} \cdot \overrightarrow{v_h} = ||\overrightarrow{v}||^2 (\cos^2\gamma_0 \cos^2\sigma + \sin^2\sigma) \tag{B.11}$$

Finally, the flight path angle γ is expressed as

$$\gamma = \tan^{-1}\left[\frac{||\overrightarrow{v} \times \overrightarrow{v_h}||}{\overrightarrow{v}} \cdot \overrightarrow{v_h}\right] \tag{B.12}$$

$$= \tan^{-1}\left[\frac{\sin\gamma_0 \cos\sigma}{\sqrt{\cos^2\gamma_0 \cos^2\sigma + \sin^2\sigma}}\right] \tag{B.13}$$

$$= \sin^{-1}[\cos\sigma \sin\gamma_0] \tag{B.14}$$

To rapidly check the consistency of this relation, one can check easy scenarios:

- $\gamma_0 = 0°$: then $\gamma = 0° \quad \forall\sigma$, hence the aircraft has a constant altitude.
- $\sigma = 0°$: then $\gamma = \gamma_0$, hence the aircraft has a constant descent rate (a straight line on the glide slope).
- $\sigma = 90°$: then $\gamma = 0°$, hence the aircraft is not descending on the glide slope.

The evolution of γ as a function of σ is traced in Figure B.2.

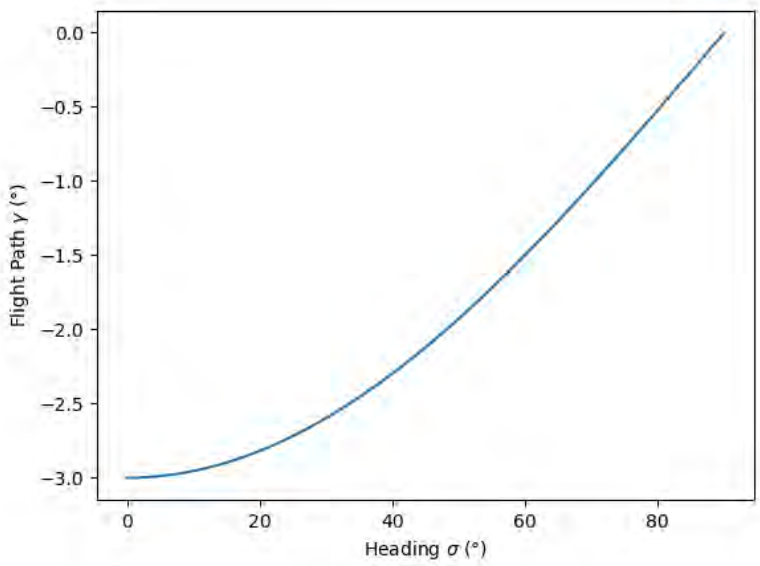


Figure B.2: The flight path angle γ versus the heading angle of the aircraft σ with $\gamma_0 = -3°$.

The above relations are rewritten to express the heading angle σ as a function of the flight path angle γ

$$\cos\sigma = \frac{\sin\gamma}{\sin\gamma_0} \tag{B.15}$$

Thereby, the constraint acting on the heading angle σ is:

$$\cos\sigma \geq \frac{\sin\gamma_m}{\sin\gamma_0} \tag{B.16}$$

with

$$\gamma_m = \begin{cases} -\gamma_0 & \text{if } \gamma_{\max} \geq -\gamma_0 \\ \gamma_{\max} & \text{if } \gamma_{\max} \leq -\gamma_0 \end{cases} \tag{B.17}$$

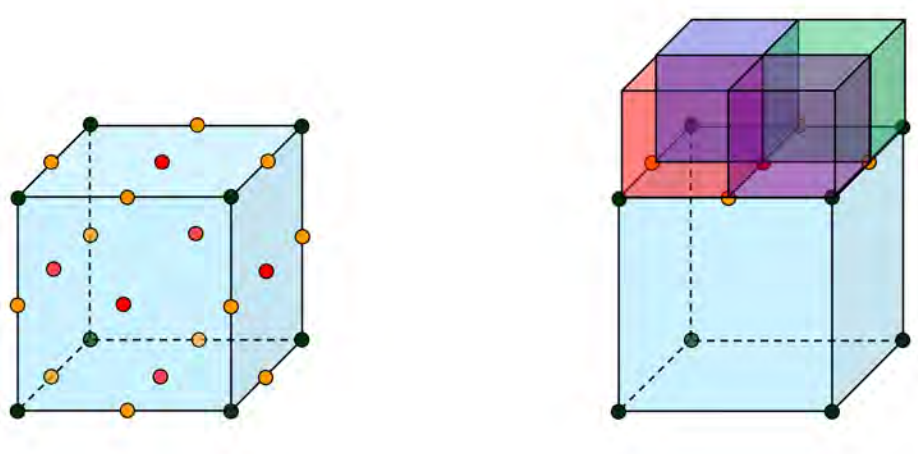
Again, when the engines are lost, $\gamma_{\max}^{\text{gliding}}$ replaces $\gamma_{\max}$.

Appendix C

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 Figure C.1a.



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

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

Figure C.1: Notion of *activated points*.

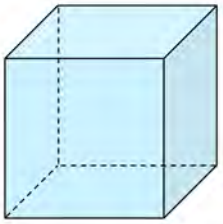
At first glance, there are $2^{F+E} = 2^{18}$ configurations possible for a node, whether each of its points are activated or not (its $N = 8$ vertices are always part of the mesh, thus are always activated).

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 Figure C.1b.

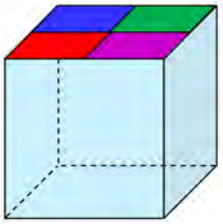
This observation reduces strongly the number of configurations to study. Below are presented the only 10 cases to consider in order to enumerate all the possible configurations exactly once. For each case, the number and relative positions of neighbors in cardinal directions of considered node is set.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



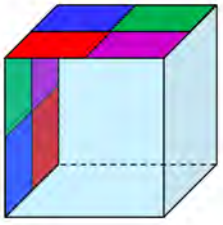
- Number of free edges: 12.
- Number of symmetrical configurations: 1.
- Total number of configurations : $1 \times 2^{12} = 4096$.

Case 2: Neighbors in 1 cardinal direction



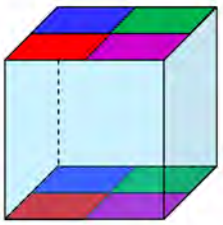
- Number of free edges: 8.
- Number of symmetrical configurations: $F = 6$.
- Total number of configurations : $6 \times 2^8 = 1536$.

Case 3: Neighbors in 2 adjacent cardinal directions



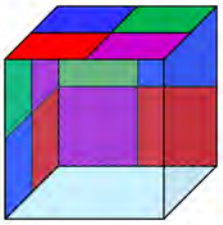
- Number of free edges: 5.
- Number of symmetrical configurations: $E = 12$.
- Total number of configurations : $12 \times 2^5 = 384$.

Case 4: Neighbors in 2 opposite cardinal directions



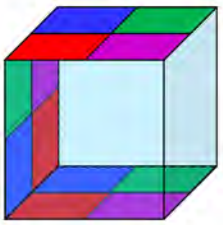
- Number of free edges: 4.
- Number of symmetrical configurations: $F/2 = 3$.
- Total number of configurations : $3 \times 2^4 = 48$.

Case 5: Neighbors in 3 adjacent cardinal directions



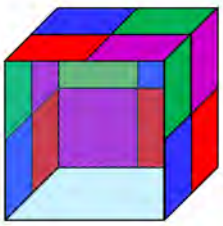
- Number of free edges: 3.
- Number of symmetrical configurations: $V = 8$.
- Total number of configurations : $8 \times 2^3 = 64$.

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



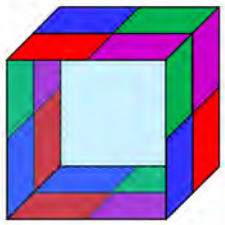
- Number of free edges: 2.
- Number of symmetrical configurations: $2 \times F = 12$.
- Total number of configurations : $12 \times 2^2 = 48$.

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



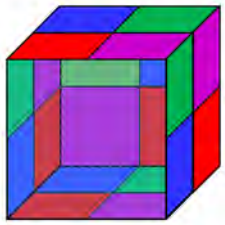
- Number of free edges: 1.
- Number of symmetrical configurations: $E = 12$.
- Total number of configurations : $12 \times 2^1 = 24$.

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



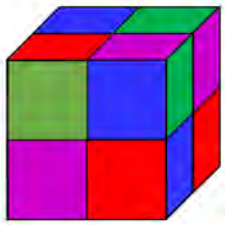
- Number of free edges: 0.
- Number of symmetrical configurations: $F/2 = 3$.
- Total number of configurations : $3 \times 2^0 = 3$.

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.

Appendix D

Update Procedure on a Tetrahedron

The objective of this appendix is to extend the update procedure established by Kimmel and Sethian in [120] 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 Figure D.1.

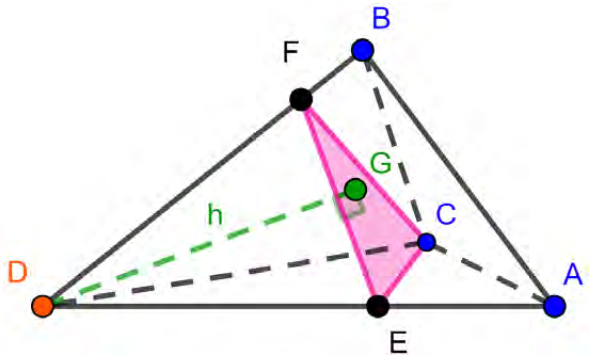


Figure D.1: Given the tetrahedron ABCD, such that $u_{\min} = T(B) - T(A)$ and $u_{\max} = T(C) - T(A)$, find $T(D) = T(A) + t$ such that $(t - u_{\max})/h = F$.

This means that we search for t such that

$$\frac{t - u_{\max}}{h} = F. \tag{D.1}$$

By similarity:

$$\frac{T(D) - T(A)}{DA} = \frac{T(E) - T(A)}{EA} \implies \frac{t}{DA} = \frac{u_{\max}}{EA} \implies \boxed{DE = DA\,\frac{t - u_{\max}}{t}} \tag{D.2}$$

$$\frac{T(D) - T(B)}{DB} = \frac{T(F) - T(B)}{FB} \implies \frac{t - u_{\min}}{DB} = \frac{u_{\max} - u_{\min}}{FB} \implies \boxed{DF = DB\,\frac{t - u_{\max}}{t - u_{\min}}} \tag{D.3}$$

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

$$h = \frac{|\overrightarrow{n} \cdot \overrightarrow{CD}|}{||\overrightarrow{n}||} \tag{D.4}$$

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

$$\overrightarrow{n} = \overrightarrow{CE} \times \overrightarrow{CF} \tag{D.5}$$

From equations Equations (D.2) and (D.3), $\overrightarrow{CE}$ and $\overrightarrow{CF}$ are expressed as functions of t:

$$\boxed{\overrightarrow{CE} = \overrightarrow{CD} + \frac{t - u_{\max}}{t}\,\overrightarrow{DA}} \tag{D.6a}$$

$$\boxed{\overrightarrow{CF} = \overrightarrow{CD} + \frac{t - u_{\max}}{t - u_{\min}}\,\overrightarrow{DB}} \tag{D.6b}$$

Equation (D.5) is developed as:

$$\boxed{\overrightarrow{n} = \frac{t - u_{\max}}{t - u_{\min}}(\overrightarrow{CD} \times \overrightarrow{DB}) + \frac{t - u_{\max}}{t}(\overrightarrow{DA} \times \overrightarrow{CD}) + \frac{(t - u_{\max})^2}{t(t - u_{\min})}(\overrightarrow{DA} \times \overrightarrow{DB})} \tag{D.7}$$

Hence, the numerator of Equation (D.4) is rewritten:

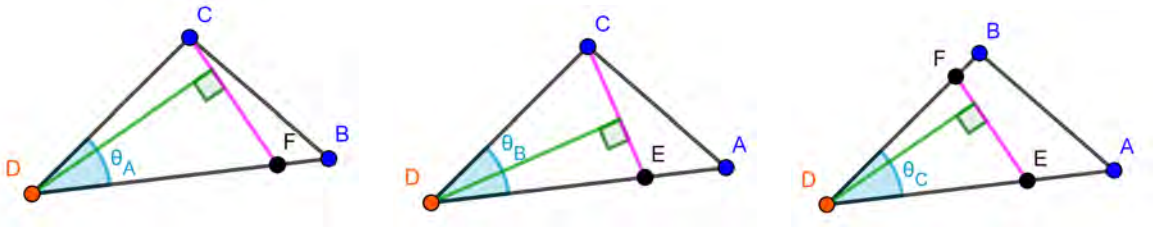
$$\left|\overrightarrow{n} \cdot \overrightarrow{CD}\right| = \frac{6(t - u_{\max})^2}{t(t - u_{\min})}\mathcal{V}_{\text{ABCD}} \tag{D.8}$$

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

The denominator of equation Equation (D.4) can also be developed:

$$||\overrightarrow{n}|| = 2(t - u_{\max})\sqrt{\frac{\mathcal{A}_{\text{BCD}}^2}{(t - u_{\min})^2} + \frac{\mathcal{A}_{\text{ACD}}^2}{t^2} + \frac{(t - u_{\max})^2}{t^2(t - u_{\min})^2}\mathcal{A}_{\text{ABD}}^2} \tag{D.9}$$

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 D.2).



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

Figure D.2: Triangles in the tetrahedron.

Replacing (D.8) and (D.9) in (D.4), then in (D.1):

$$\mathcal{A}_{\text{BCD}}^2 t^2 + \mathcal{A}_{\text{ACD}}^2(t - u_{\min})^2 + \mathcal{A}_{\text{ABD}}^2(t - u_{\max})^2 = 9F^2\mathcal{V}_{\text{ABCD}}^2 \tag{D.10}$$

We end up with the quadratic equation for t:

$$\boxed{\left(\mathcal{A}_{\text{BCD}}^2 + \mathcal{A}_{\text{ACD}}^2 + \mathcal{A}_{\text{ABD}}^2\right)t^2 - 2\left(\mathcal{A}_{\text{ACD}}^2 u_{\min} + \mathcal{A}_{\text{ABD}}^2 u_{\max}\right)t + \mathcal{A}_{\text{ACD}}^2 u_{\min}^2 + \mathcal{A}_{\text{ABD}}^2 u_{\max}^2 - 9F^2\mathcal{V}_{\text{ABCD}}^2 = 0} \tag{D.11}$$

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

$$DC\cos\theta_A < DF < \frac{DC}{\cos\theta_A} \tag{D.12a}$$

$$DC\cos\theta_B < DE < \frac{DC}{\cos\theta_B} \tag{D.12b}$$

$$DF\cos\theta_C < DE < \frac{DF}{\cos\theta_C} \tag{D.12c}$$

Now in terms of t:

$$\boxed{DC\cos\theta_A < DB\,\frac{t-u_{\max}}{t-u_{\min}} < \frac{DC}{\cos\theta_A}} \tag{D.13a}$$

$$\boxed{DC\cos\theta_B < DA\,\frac{t-u_{\max}}{t} < \frac{DC}{\cos\theta_B}} \tag{D.13b}$$

$$\boxed{DB\cos\theta_C < DA\,\frac{t-u_{\min}}{t} < \frac{DB}{\cos\theta_C}} \tag{D.13c}$$

The update procedure is given as follows:

Algorithm 11 Update on a 3D Meshpoint Procedure

Solve Equation (D.11). Note t the solution.
if $t > u_{\max}$ and all 3 conditions of Equation (D.13) 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 Equation (D.13c) 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