TÉCNICO LISBOA

A Geometric Method for Static and Dynamic Collision Avoidance for UAVs in a 3D Environment

Carolina Pereira Pinheiro

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Prof. Afzal Suleman
Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. Fernando José Parracho Lau
Supervisor: Prof. Afzal Suleman
Member of the Committee: Prof. Homayoun Najjaran

December 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

Firstly I would like to thank Professor Afzal Suleman for the opportunity to come to Canada and do my thesis here, as well as everybody at CfAR that gave their insight and helpful advice throughout my stay at the centre, especially Stephen Warwick and Sean Bazzocchi. I would also like to thank Professor Rodrigo Ventura for his insightful advice on the thesis direction.

I would also like to thank my parents Ermelinda and Victor and my brothers Gonçalo and Manuel, not only for their support on my decision to come to Canada, but also during all of my university years and beyond.

A special thank you to my hometown friends as well, hoping that we will continue extinguishing each others fires for years to come.

To all the friends I made at Técnico, thank you for making these last 5 years a lot more memorable and enjoyable. A special thanks to Benny for the constant music and playlist sharing, to Joões, Inês and Nuno for the pointless arguments at the Civil cafeteria and to the Bombarral group for the incredible memories and constant banter. A warm thank you to the RED and the futsal team too, for making this experience more than just school work.

A special thank you to all of the friends that I made over here, you elevated my experience in Victoria and made me feel welcomed 8000km away from home, I'll miss you dearly and hopefully we will see each other again.

Thank you as well to the group that accompanied me on this adventure, especially Inês, António and Luís for being the best flatmates that I could ask for and for all the gleeful dinners that we shared together.

Finally, I need to thank Inês again, for being the best friend that I could've asked for. For following me (or letting me follow her) from Lisbon, to Barcelona and to Victoria. For always being by my side, figuratively and literally, for being able to match my energy and for all of the advice and life lessons.

Abstract

Generating real-time solutions to avoid dynamic threats that are on a collision course with an Unmanned Aerial Vehicle (UAV) is a challenging task. This thesis presents the development of a framework with three integrated main blocks - path generation, collision detection and cost estimation - to find an adequate path in a 3D environment which safely avoids both the impending threat and other threats present in the environment. The path generation block uses a cubic spline method to generate an initial group of candidate paths that is then expanded to 3D space by using a rotation matrix. Next, this group of candidate paths are evaluated for collision detection and any candidate paths that have the possibility to collide with any threats are discarded. Lastly, a function that estimates a cost determines the optimal solution for collision avoidance. The cost function takes into account the environment around the candidate path and the performance of the UAV. The parameters selected were normalized according to their distribution across a multitude of scenarios and their weights are tuned to ensure a well balanced cost function. The results show that the framework is able to find solutions for most situations with promising run-times, but several improvements and modifications to the implementation and tests still need to made before field deployment.

Keywords: collision avoidance, static and dynamic obstacles, 3D environment, cost function, online computation

Resumo

Gerar soluções capazes de evitar ameaças dinâmicas que estão em rota de colisão com um Veículo Aéreo Não Tripulado num cenário on-line é uma tarefa complicada. Esta tese tenta abordar esta questão desenvolvendo uma estrutura com três blocos principais integrados - geração de caminhos, detecção de colisão e cálculo de custos - para encontrar um caminho adequado num ambiente em três dimensões que evite com segurança tanto a ameaça iminente como outras ameaças presentes no mesmo. Para gerar os caminhos, um método de *spline* cúbicas é utilizado para gerar um grupo inicial de caminhos candidatos que é posteriormente expandido para o espaço 3D, utilizando matrizes de rotação. A seguir, este grupo de caminhos candidatos é avaliado para detecção de colisão e quaisqueres caminhos em que haja a possibilidade de colidir com ameaças presentes no ambiente não são considerados. Finalmente, uma função de custo escolhe um caminho adequado como solução. Esta função de custo foi desenvolvida para ter em conta vários aspectos do problema (desde o ambiente à volta do caminho candidato até ao desempenho da aeronave) e os parâmetros escolhidos foram normalizados tendo em conta a sua distribuição em vários cenários. Os pesos destes parâmetros foram ajustados para assegurar uma função de custo equilibrada. Os resultados mostram que o trabalho desenvolvido tem potencial e pode encontrar soluções para a maioria das situações com tempos de execução promissores. No entanto, ainda é necessário realizar várias melhorias, modificações e testes antes de implementar o trabalho desenvolvido numa aeronave.

Palavras-Chave: evasão da colisão, obstaculos estáticos e dinâmicos, ambiente 3D, função de custo, computação *online*

Contents

List of Tables

List of Figures

Acronyms

APF Artificial Potential Field. 11, 12

CDR Conflict Detection and Resolution. 10

CfAR Centre for Aerospace Research. xiii, 1, 3, 4, 9, 12, 15, 28, 35, 68

CSV Comma-Separated Values. 36

FCL Flexible Collision Library. 6, 12, 13, 16, 24, 25, 57, 68

GPS Global Positioning System. 23

HITL Hardware in the Loop. 16, 68

MDP Markov Decision Processes. 12

MO Moving Obstacles. xiii, xiv, 27, 32, 33, 39, 40, 44–47, 49, 52, 54, 55, 57, 59–61, 64, 67, 69

OMPL Online Montion Planning Library. 16

POMDP Partially Observable Markov Decision Processes. 12

ROC Rate of Climb. 4, 23, 57

RRT Rapidly-exploring Random Tree. 10, 12, 15, 16

SAA Sense and Avoid. 1

SITL Software In The Loop. 12, 16, 68

TCAS Traffic Alert and Collision Avoidance System. 11

UAV Unmanned Aerial Vehicle. xi, xiii, xiv, 1–6, 9–13, 15–29, 31–33, 36, 37, 41–44, 46–53, 55, 57–59, 61–64, 68, 69

Chapter 1

Introduction

In this chapter, the developed is introduced starting with context and motivation behind it, namely the applications of UAVs in the present context and the future growth of this sector. Several terms that are used throughout this work are also clarified in this chapter. The different restrictions of the work developed and the assumptions made when approaching the overall problem are discussed in the problem statement section. Finally, the objectives, contributions and outlines of the current work are also presented.

1.1 Context and Motivation

A UAV (Unmanned Aerial Vehicle) is an aircraft with no on-board crew or passengers. These aircraft are controlled by onboard computers and thus have autonomous flight capabilities. They come in several sizes, weights, and configurations (fixed wing, multi-rotors, hybrid, etc.).

UAVs are used in several real life applications such as payload delivery, traffic monitoring, surveillance, agriculture, military applications, construction industry, recreational uses, etc. [1] [2]. Figure 1.1 illustrates the UAV market share in 2020, according to a report by Precedence Research [2], where it can be observed that the main current application for UAVs is the military sector, followed by the recreational and finally the commercial sector. The use of UAVs in this context requires not only offline path planning but also these aircraft should be equipped with systems capable of avoiding sudden obstacles not planned for in the initial path planning phase.

Moreover, the growth of the use of UAVs has also been relevant; according to the same report by Precedence Research [2] the UAV market was valued at US$ 14.3 billion in 2020 and is expected to hit around US$ 53 billion by 2030, growing at a CAGR (Compound Annual Growth Rate) of 14% over the forecast period 2021 to 2030. This growth is illustrated in Figure 1.2, and shows that this technology has future and its systems should be invested in and improved on. Furthermore, the interest in UAV technology is global. While currently it is mainly focused in North America, the forecast period (2021-2030) is expected to see the highest growth rate in the Asia-Pacific region, mainly China. Due to the lucrative opportunities the market has to offer, in recent years, industry giants like Google, Amazon, DHL, Uber, and Boeing have all made significant investments in the research and development in this sector.

The work developed in this thesis is a contribution to the current effort at CfAR to develop a Sense and Avoid (SAA) system. A SAA system is a system which is composed of two main components - the sense function and the avoid function. The sense function is responsible for identifying and tracking surrounding traffic whereas the avoid function is responsible for detecting possible conflicts (namely

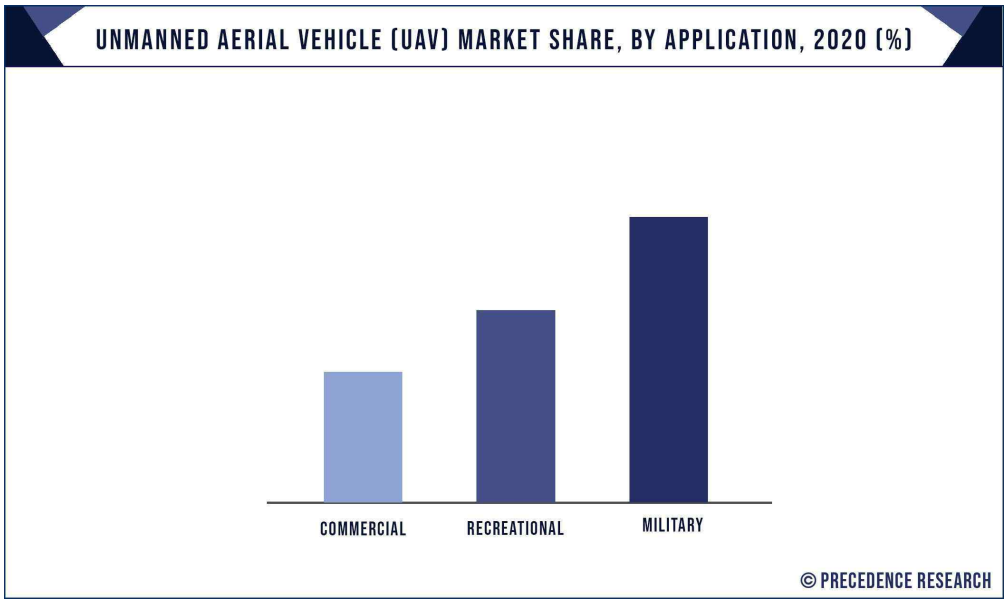


Figure 1.1: UAV Market Share by application 2020 [2].

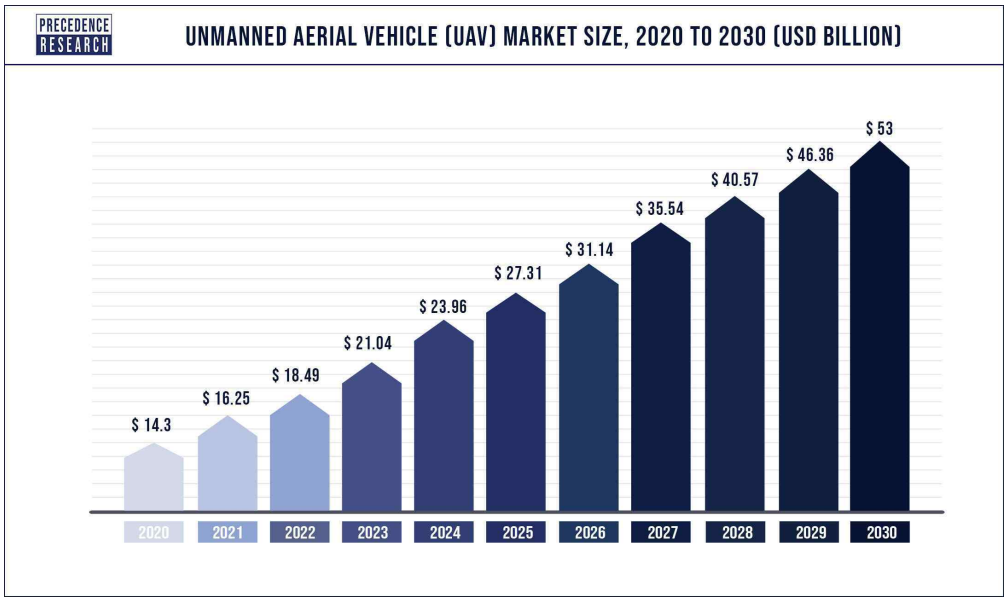


Figure 1.2: UAV market size growth from 2020 to 2030 (USD Billion) [2].

collision threats) and to instruct maneauvers to the UAV capable of safely avoiding these conflicts [3]. The work in this thesis is inserted into the second function - the avoid function - and a system that, given information about the environment where it is inserted, is capable of avoiding that threat is developed.

1.2 Definitions

Throughout this thesis several terms will be used that will now be clarified.

- **UAV** - Unmanned Aerial Vehicle, refers to the the aircraft that has a trajectory associated with it where the dynamic avoidance system is to be implemented;
- **Environment** - The 3D space where the UAV is inserted into, it contains all the information available of the objects present and specific constraints associated with it;

- **Threat** - Refers to any objects in the environment that the UAV should avoid collision with it;
- **Main Threat** - the threat that is in direct collision route with the trajectory of the aircraft, can be static or dynamic;
- **Path** - A path indicates how to move from point A to point B through an environment and is defined by a set of points;
- **Trajectory** - A trajectory is a path associated with a time stamp for each point that defines it;
- **Candidate Path** - A candidate path is one of the paths that are generated by the algorithm to avoid the incoming threats;
- **Static Threat** - A threat with no velocity.
- **Dynamic Threat** - A threat which is moving in the environment.
- **Offline vs. Online** - offline refers to work done, on the ground, before the flight, whereas online refers to something done during the flight. These phases have different constrictions, as an example since the online phase is performed in real-time it has more time-sensitive restrictions.

1.3 Problem Statement

One of the problems when developing and flying UAVs is avoiding obstacles that would collide with the already established trajectory that the UAV is following. These are due to unexpected changes in the environment, namely birds, other UAVs, humans or manned aircraft, or static obstacles that were not taken into account when the initial trajectory was planned (terrain, trees, buildings) [4]. In these cases the UAV has to be equipped with a real-time system capable of detecting the threat, calculating the evading trajectory and executing the evading trajectory. These threats can be both **static** (no velocity and fixed position) or **dynamic** (moving).

Regarding the requirements for the framework and algorithm developed, the system should be capable to, when provided with information of threats present in the environment, detect if these entail collision risk and if so, generate an avoidance path that avoids not only the collision threat but also other threats present in the environment. The avoidance path should start along the initial trajectory of the UAV and re-enter the initial calculated trajectory after the threat is avoided.

One of the biggest constraints when trying to solve this problem is the computational constraints associated with it. The solution must be simple enough that it can be incorporated in an on-board flight computer like the *Raspberry Pi 4 Model B* [1] used in the *Tarot* presented in Figure 1.3 and developed at CfAR; and fast enough so that the evading trajectory can be calculated and executed before the collision.

[1] `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/`, accessed 12/04/2022

Figure 1.3: *Tarot* quad-rotor developed at CfAR.

1.3.1 Assumptions

Some assumptions also have to be made in order to simplify the problem and make it easier and feasible to solve in the time frame available for the development of this work. There are assumptions made regarding four of aspects of the problem and the information available: the **threat**, the **UAV**, the **trajectories** and the **environment** of each.

It is assumed that the **threat**'s size and all the components (x, y and z) of the velocity and position are known with some uncertainty that will be further discussed in section 3.2. The acceleration is considered to be null (*i.e.* constant velocity) and the **trajectory** can be modelled as a straight line. The size of the threat is expected to be up to $3m$ (radius of the threat since the threat will be modelled as a sphere) and the velocity has no restrictions. Knowing all the components of the velocity also implies that the collision angle is known and it can be calculated from that initial information. It should be noted that the algorithm should be easily expanded to deal with bigger threats, but the size had to be constrained to a specific range in order to facilitate the initial development.

Regarding the **UAV**, its position, velocity, acceleration is also assumed to be known at all times. The size is not considered and the UAV is considered to be a point for simplification purposes. The only constraint regarding the capabilities of the UAV implemented is the maximum Rate of Climb (ROC). In the initial development the **trajectory** followed by the UAV is assumed to be straight with zero acceleration. This trajectory is assumed to be already set beforehand and it will not be handled by the system developed here. This work will mainly focus on a quad-rotor similar to the one developed at CfAR - *Tarot*, but one the goals of the algorithm and the framework developed is to be general enough that it can be easily adapted to other types of UAVs with other constraints and dynamics. Thus, the variables considered are based on the default values used by the PX4 autopilot software [2]

- **rate of climb**: $3\,m/s$,
- **cruise velocity**: $5\,m/s$.

It is also assumed that the UAV is equipped with a sensing system capable of detecting the threats with some precision and far enough from the UAV so as for the UAV to have time to calculate and execute the evading trajectory. So, the threats are assumed to be from $3m$ to $10m$ away from the UAV, but lower bound is variable and capped taking into account the threat's size.

Some other parameters of the UAV will also need to be known to the algorithm to calculate some aspects of the **cost function**, these are the drag coefficient (C_D), the effective Area (A_e), the mass (m),

[2] PX4 Autopilot, Parameter Reference, Table "Multicopter Position Control" `https://docs.px4.io/main/en/advanced_config/parameter_reference.html`, accessed 13/09/2022

the number of blades per propeller (N), the chord of the blades (c), the radius of the blades (R) and the number of rotors (n). The default values for these variables are summarized in Table 1.1. These values are based on the *CFProp 15x5R T-motor blades* and in the *Tarot*.

Table 1.1: Default values for the UAV parameters.

Parameter	Value
C_D	0.5
A_e	2.5 m^2
m	3 kg
N	2
c	0.035 m
R	0.1905 m
n	4

1.4 Project Overview and Objectives

The main goal of this work is to develop an algorithm capable of producing paths to avoid threats that would otherwise result in collision with the pre-determined trajectory of the UAV. The framework developed takes as input the current attributes of the **UAV** (the constants described in Table 1.1, and its position and velocity), information about the **environment** (number of threats, position, velocity, size and constants of the environment - air density, gravitational acceleration) and generates paths capable of avoiding the main threat (**path generation**). After these paths are generated, they are checked for **collision detection** and the paths that collide with any threat in the environment are disregarded. For the remaining paths, their quality is quantified using a **cost calculation** procedure by means of a cost function that takes into account several attributes of the path and characteristics of the UAV . Figure 1.4 illustrates the basic flow of the framework developed. The framework was mainly developed in Python 3.7.13 . This coding language was chosen due to its ease to quickly develop and test different solutions, despite not being the fastest language in terms of runtime.

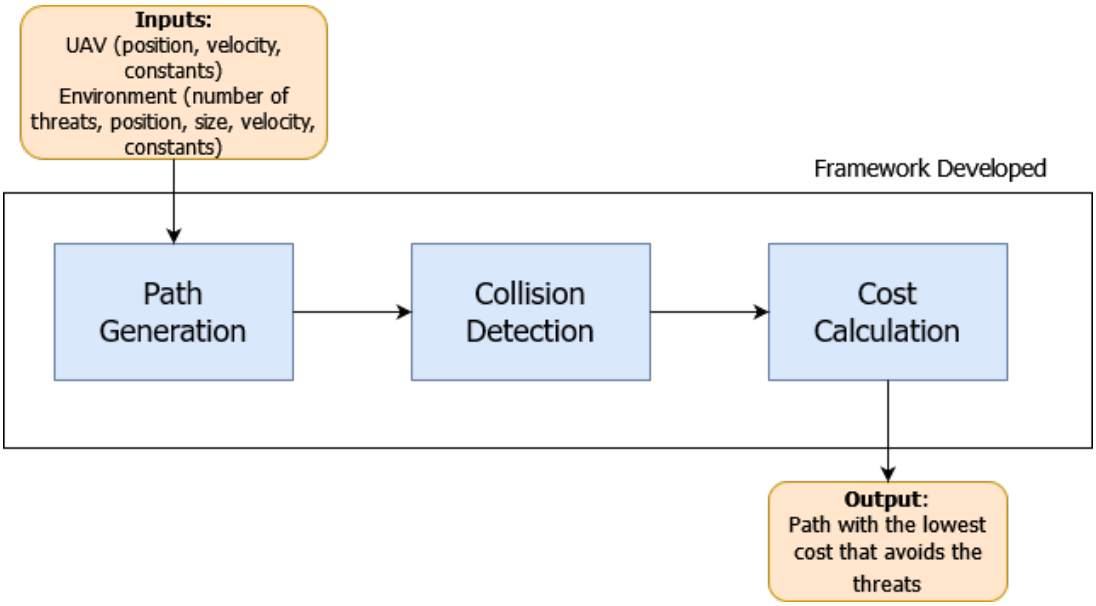


Figure 1.4: Flow of the framework developed.

1.4.1 Path Generation

With the inputs given, candidate paths are generated according to a geometric method based on the work of Chen, et al. (2020) [5]. In Chen's work the candidate paths are generated in a 2D environment using the principle of cubic spline second-order continuity. The framework developed on this thesis expands the mentioned method to the 3D space and adds another type of paths, still based on the initial group of candidate paths, that will be named **symmetric paths**. The candidate paths are also generated in relation to the UAV's reference frame, being initially generated in the horizontal plane of this reference frame and then rotated to cover the 3D space.

1.4.2 Collision Detection

The collision detection is done with the help of the Python library Flexible Collision Library (FCL) [6]. As mentioned before the threats are modelled as spheres and the UAV, that for path generation purposes is considered a point, is modelled as a sphere as well, with a defined radius that adds uncertainty considerations to the collision detection. Also based on the work of Chen, et al. (2020) [5], the collision detection is divided into two sections: **static security** and **dynamic security**. The first deals with the collision with static objects and the second with the collision with dynamic objects. Paths that are detected to collide with threats in the environment are considered unavailable and disregarded.

1.4.3 Cost Calculation

Finally, the last step is the cost calculation of the remaining candidate paths. The cost function is composed of different parameters that measure different characteristics of the paths to ensure a well balanced chosen path. These parameters take into account not only the geometry of the path, but also the surroundings of the candidate path (*i.e.* closeness to moving objects), and the UAV performance while flying the candidate path. To reach the final cost function, the distribution of these cost parameter's values across different environments was studied as well as the runtime to calculate them. After studying the parameters distribution it was possible to normalize them between 0 and 1 so they can be compared between each other. In order to choose the best parameters to represent the candidate paths the different parameters are compared and analyzed and the weights of each one were empirically chosen. In the end, the candidate path with the lowest cost is chosen.

1.5 Contributions

The main contributions of the work developed on this thesis are:

- Development and testing of a new and different type of a dynamic online avoidance algorithm with collision detection;
- Expansion of a geometric avoidance method to the 3D space;
- Development of a cost function to numerically measure the quality of a candidate path.

1.6 Thesis Outline

This thesis is organized in the following way:

- **Chapter 2**: an overview on the theoretical background needed to comprehend the topics present in this thesis and a summary of the state of the art in the area.
- **Chapter 3**: the decisions made when implementing the path generation algorithm are discussed as well as their implementation. The collision detection as well is also presented in this chapter.
- **Chapter 4**: the process for calculating, normalizing and tuning the parameters of the cost function are described and explained. A code architecture overview is also introduced.
- **Chapter 5**: based on the simulations presented in the previous chapters, the final results are synthesized and analyzed.
- **Chapter 6**: gives a summary of the work developed in this thesis, the main conclusions and suggestions for future work.

Chapter 2

Theoretical Background

In this chapter, an overview of the theoretical background is presented. Terms, concepts and definitions are discussed to ensure that a comprehensive understanding of the problem under study. A summary of the state of the art in this area as well as where the work developed fits within it is described. First, the different safety management approaches, concerning the path generation block of the framework, are discussed and compared. Second, a background on collision detection is presented. Next, the background on the cost calculation block of the framework is addressed. The previous solutions developed at CfAR in this area are also summarized and preliminary findings are discussed.

2.1 State of the Art

2.1.1 Path Generation

UAV and its development is a very popular research topic - from the conceptual design, to the avionics, flight computers, building and assembly. One of the aspects that makes UAVs so useful and sought after is the fact that there is no need for on-board crew. This ensures that the UAV has more freedom (it can access places that a regular manned aircraft might not be able to), it is easily deployable, can be mass produced, there is less risk involved (no human life at risk), among other advantages. In the case that the UAV is not remote controlled by a crew/individual on the ground there is a need for the aircraft to be able to fly on its own, this entails **path planning** to be implemented in the aircraft. Path planning is usually done offline and searches for the most suitable path to connect a starting point to a goal point. However, the environments where UAVs are inserted in can, sometimes, be unpredictable, and threats or other aircraft that were not initially considered on the offline path planning phase can appear. For this reason there is also a need for a mechanism to avoid unexpected threats to be implemented - the work developed on this thesis focus on developing/testing a framework with the goal of addressing this problem. Considering everything, even if there is no crew and no inherent human life at risk, the safety of the airspace still needs to ensured. This is where the **safety management** approaches, that will be further discussed, come in.

2.1.1.1 Safety Management Approaches

Safety management guarantees the safety and integrity of the aircraft and the airspace/environment. Zhang et al. (2018)[7] provides a summary of safety management approaches to UAV and enabling technologies (such as sensing, command and control communication), where different solutions are presented depending on the scale of safety. The different scales of safety defined are:

- **Large-Scale Safety**: refers to a global path-planning problem. The goal is to search out an optimal or near-optimal flight path from a starting point to a desired goal point and avoid any collision with any known obstacles. [8]
- **Middle-Scale Safety**: refers to short term flight conflicts due to the dynamic changes of the environment during flight such as the changing wind speed and direction. [9]
- **Small-Scale Safety**: refers to imminent collision avoidance problem that must be handled by the UAV itself.

For each scale of safety, a quick overview will be given to the solutions presented. However, since the work developed focus more on solving small-scale safety, a higher emphasis will be given to this type of safety and its solutions.

The distinction between **global path planning** and **local path planning** should also be made. Global path planning takes into account the whole picture and plans a path from the initial starting position until the goal position while avoiding the known obstacles in the environment. Local path planning (also referred as dynamic path planning) works with the information that is receiving while flying to avoid dynamic obstacles [5] (middle-scale and small-scale safety).

Large-Scale Safety

As mentioned before, large-scale safety mainly focuses in solving global path planning algorithms. This is usually done offline (*i.e.* before the flight, on the ground) since it requires heavy computational resources and time. As evaluation parameters path length, flight duration time, flight manoeuvre efforts, collision risks and the cooperative nature of the UAV are used. [10] [9]

The path planning algorithms can be divided into three classes: **Deterministic Algorithm**, **Probabilistic Algorithm**, **Heuristic Algorithm** [9].

Deterministic Algorithms have certain search mechanisms to find the best path and fixed cost equations - for a given scenario it will always generate the same result. A-Star Search Algorithm [11] [12], Potential Field Method [13], and Mathematical Programming Method (models the problem as a numerical optimization problem) [14] [15] are examples of this type of algorithm.

Probabilistic Algorithms are based on a randomized search process to achieve feasible or near-optimal solutions. Contrary to the deterministic algorithm, it may generate different solutions for the same problem. One of the most popular path planning algorithms - Rapidly-exploring Random Tree (RRT) [16], is a probabilistic algorithm.

Finally, an **Heuristic Algorithm** is a combination of the two previously mentioned algorithms, but represents the problem through an heuristic representation. Some of the main heuristic algorithms are Genetic Algorithm [17], Particle Swarm Algorithm [18] and Artificial Bee Colony Algorithm [19].

Middle-Scale Safety

A middle-scale safety problem focuses on addressing changes in the environment during flight such as changing wind speed and direction. An important part in this type of safety is to study how to predict the possible conflicts in a few hours or minutes and apply conflict resolution algorithms to these changes - this is called Conflict Detection and Resolution (CDR). CDR is composed of five modules: environment sensing, trajectories prediction, conflict detection, conflict resolution and manoeuvre strategies [9] . Conflict resolution and manoeuvre strategies are the most relevant modules for the work developed in

this thesis. For conflict resolution, the methods used are: **geometric methods**, **numerical optimization methods** and **multi-agent methods**.

Geometric methods utilizes geometric features and relations to find the best evasion manoeuvre in specific scenarios. An example of this type of methods is the work developed by Park et al. [20] which applies a simple geometric approach to solve collision avoidance between two UAVs.

In order to generate collision-free strategies, the **numerical optimization method** approach applies a kinematic model of the aircraft while subject to a variety of constraints. A cost evaluation module is also needed in order to choose the most economical path and optimize with respect to it. Some relevant works in this method are the works of Pallotino et al. [21] and Raghunathan et al. [15].

Lastly, **multi-agent methods** uses multi-agent framework to generate conflict resolution strategies in multi-UAVs scenarios. In this system each UAV is viewed as an agent , and via the use of a variety of utility functions, they are able to communicate with one another and negotiate over the best course of action. Rong et al. [22] and Wollkind et al. [23] are some of the relevant work in this area.

Small-Scale Safety (Collision Avoidance Approaches)

In the case of an imminent collision, the UAV must be able to handle it on its own even with lack of future information on the obstacles presented. In these cases, the reactive collision avoidance system, which is inserted into the small-scale safety management, is the last line of defence. The collision avoidance system performs evasive manoeuvres on these threats, although these manoeuvres may not be the most efficient due to the urgency of the situation. When developing small-scale safety algorithms there are two main constraints: **UAV physical performance** (velocity, accelaration, etc.) and the **runtime of the algorithm** (how much time it takes to run the algorithm). The solutions for small-scale safety can be divided into **coordinated algorithms** and **non-coordinated algorithms**.

Coordinated algorithm are used when the UAVs/aircraft in collision route share flight intention with each other - so, there is opportunity to coordinate between the different aircraft; or the UAV has information or can get information on the obstacle. In this type of algorithm, **rule-based methods**, **Artificial Potential Field (APF) methods** and **geometric methods** are included.

Traffic Alert and Collision Avoidance System (TCAS) is the most popular example of **rule-based methods** and is used on commercial aircraft. The idea behind rule-based methods is to use predefined rules to avoid collisions. This method is simple; however, it is not capable to adapt to dynamic environments, despite being computer efficient.

Another example of a coordinated algorithm is the APF. The APF treats each aircraft (or threat) as a charged particle in a potential field - the aircraft's goal point is considered an attractive force and the threats repulsive forces [24] . It was first introduced by Khatib et al. (1986) [25] for mobile robots and has been expanded to UAVs by other authors [26] [27] [28]. This method is well known and used in collision avoidance due to its efficiency, robustness, and flexibility [7] - it is based on simple mathematics equations so the computational time needed is low and thus can be used for real time application [26] [29]. However, APF main problem is local minima in some situations (one obstacle, two obstacles, goal not reachable, dynamic or moving obstacles) [26]. The proposed ways to solve these problems are either modify the APF method itself or combine it with other algorithms [7].

The last type of coordinated algorithm that will be discussed is the **geometric method**. Geometric methods, although complex, are highly efficient, but it can be difficult to implement to adapt to complex environments. Most of the geometric methods also try to avoid deviation from the original path [7]. These methods use geometric relations between the obstacles and the UAV to calculate a new path so as the UAV can avoid these conflicts. One of the advantages of these geometric based approaches is that

they require less processing power and thus can be more easily implemented on on-board computers [30]. These geometric-based methods can be categorized into four groups: those that use geometry information, such as the motion or location of the vehicle, to produce angle changes; those that use velocity variation; those that combine these methods; and those that also take other types of information into account (global and probabilistic) [30]. Geometric methods can handle both static and dynamic obstacles . [31]

Finally, on non-coordinated algorithms, these try to solve collision avoidance in a non-coordinated environment. A non-coordinate environment is an environment where it is difficult to know the future movement and state of the threats/environment - which makes this problem harder to solve [7]. Temizer et al. (2010) developed a random probability model that incorporates Partially Observable Markov Decision Processes (POMDP) and Markov Decision Processes (MDP) to provide an optimized collision avoidance method while accounting for sensor and aircraft dynamics uncertainty [32]. Another solution implemented was combining some of the methods discussed previously, such as RRT and APF with reachable sets in order to avoid dynamic obstacles [33] [34].

Initially, the research on path planning focused mainly on 2D path planning since these algorithms were being developed for mobile robots that usually did not have the third degree of freedom (altitude). However, with UAVs there is a need to account for altitude changes - this increases computational time and also adds an extra level of complexity to the problem. The work developed on this thesis will develop a path generation algorithm that works in the 3D space.

The algorithm developed in this work is an improvement/extension on the work developed by Chen et al. (2020) [5] and falls into the geometric methods for small-scale safety category. The methodology of Chen's solution and the improvements made will be further discussed in Section 3.1 .

2.1.2 Collision Detection

An important topic in the work developed is the ability to check for collisions between the candidate path and the threats in the environment. The FCL [6] is a library for performing three types of proximity queries on a pair of geometric models . This library, originally deployed on C++, allows for Collision Detection (detect if two models overlaps), Distance computation (distance between the closest pair of points of the models), Tolerance verification (wheter two models are closer or farther than a tolerance distance), Continuous collision detection (collision detection with moving models), and contact information when collisions are detected (i.e. what are the coordinates for the collision point). FCL also supports different types of objects shapes (box, sphere, cylinder, etc.) which makes it a very complete library and tool to use. [1]

In this work Python-FCL[2] , an unofficial Python interface for the FCL, was used. This package supports most of the features of the original C++ library.

This library was also chosen due to its use in previous work developed at CfAR and its integration with OctoMap [35] - a library used for environment representation that could be used in the future work for Software In The Loop (SITL) tests and implementations. It was also chosen since it allows for collision detection between both static objects and dynamic objects, which is an important aspect of collision detection.

[1] FCL (GitHub). `https://github.com/flexible-collision-library/fcl` accessed 19/09/2022.
[2] Python-FCL (GitHub). `https://github.com/BerkeleyAutomation/python-fcl` accessed 19/09/2022.

2.1.3 Cost Calculation

The final block on the framework is the cost calculation. The cost calculation is done by resorting to developing a cost function. The goal of the cost function is to numerically describe how good or bad a candidate path is - i.e. evaluate the performance of the UAV while flying the candidate path. In general terms, a cost function can have **fixed costs** and **variable costs**. In this case, all of the parameters of the cost function will be associated with variable costs.

One of the most important factors when defining a cost function is choosing which parameters to use that will, in this case, accurately describe the candidate path and the UAV performance. Fu et al. (2018) [36] proposes three different different cost functions that measure the smoothness of the path, the feasibility of the path, and the energy consumption of the UAV while flying the path. De Filippis et al. (2011) [37] develops an algorithm for path planning that then tries to minimize a cost function reflecting path length and collision risk. Phung et al. (2021) [38] also formulates a cost function that takes into account the danger of the candidate path, the feasibility of the path, the smoothness of the path (that they measure by taking into account the turning rate and altitude changes of the path) and the altitude cost. Chen et al. (2020) [5] uses 4 factors to set their cost function: static security, dynamic security, smoothness and consistency. The work developed in this thesis, groups static security (if the path collides with a static threat) and dynamic security (if the path collides with a dynamic threat) into the collision detection aspect, and uses the FCL as mentioned above, to do so. This was also done to save computation time - if a candidate path is known to be involved in a collision the cost function will not be calculated for it.

It can be concluded that the most important factors when choosing parameters for a cost function are: **collision risk, aircraft energy consumption, shape of the path (smooth), altitude changes and path length.** So, the different parameters should measure a mix of these factors. The parameters that will be studied to be implemented in the final cost function are **smoothness** (measures the shape of the path) , **smoothness with penalization**(measures the shape of the path and changes in altitude), **closeness to moving objects** (measures collision risks), **energy** (measures energy consumption), **time** (measures energy consumption and path length) and **hovering power** (measures energy consumption). The final goal is not to use all of these parameters, but to choose the ones that better (accurately and quickly) represent a candidate path while also taking into account the factors mentioned above. The implementation of these different parameters will be further discussed in Section 4.1.

2.2 Theoretical Concepts

2.2.1 Types of Normalization

Normalization aims to transform parameters to be on a similar scale. This will be used to normalize the different parameters for the cost function so that they can be comparable and the weights can be tuned accordingly. There are different normalization methods that can be used, the ones studied are: **min-max normalization**, **feature clipping**, **log scale normalization** and **z-score normalization** [3]. A quick introduction as well as the main advantages and disadvantages of each one will be discussed.

Min-max normalization or **scaling to range** means converting values from their standard range (their minimum and maximum values) into a standardized range, in this case it would be between 0 and 1. To do this, the following formula is used, where x is a data point and x_{min} and x_{max} are the lower and upper limits used to scale to range.

[3]Developers Google - Normalization. `https://developers.google.com/machine-learning/data-prep/transform/normalization`, accessed 12/09/2022

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} \, . \tag{2.1}$$

This type of normalization is a good option when both the lower and upper bounds are known with a few or no outliers and the data is approximately uniformly distributed across the range. Data that is very concentrated in a certain range and that then tampers out is not ideal since a big quantity of the data would be squeezed into a small part of the scale.

Feature clipping is used when the data contains extreme outliers. This method caps all values above a certain threshold to a lower fixed value that would be considered the upper range of the data (or vice-versa, a values below a certain threshold would be capped into the lower range of the data). This can be done on its own or even before or after another normalization method. The values used to cap the data can be user selected or based on the standard deviation (σ), for example clipping the data to $\pm N\sigma$, where N is a value chosen by the user.

Log scaling computes the logarithm of the values to compress a wide range to a narrow range. Assuming that x is a data point

$$x_{new} = log(x) \, , \tag{2.2}$$

in this situation x_{new} values would then need to be scaled to a 0 to 1 range to ensure that the requirements set for the cost function were fulfilled. Log scaling is useful when a handful of values have many points while most other values have few points.

Z-Score is a variation of scaling to range that represents the number of standard deviations away from the mean and ensures that the distribution of the data has mean of 0 and a standard deviation of 1. Z-Score is helpful when there are a few outliers, but not so many that clipping is needed. The new data point using z-score can be calculated using the following equation, where μ is the mean and σ the standard deviation

$$x_{new} = \frac{x - \mu}{\sigma} \, . \tag{2.3}$$

2.2.2 Correlation

Correlation is a concept used in statistics to describe the strength of the linear relationship between two variables. To measure the correlation between two variables the correlation coefficient is used. The correlation coefficient of two variables (A and B) for their N observations, measures their linear dependence. This coefficient, the Pearson correlation coefficient, is defined as [4]

$$\rho(A, B) = \frac{1}{N-1} \sum_{i+1}^{N} \left(\frac{A_i - \mu_A}{\sigma_A} \right) \left(\frac{B_i - \mu_B}{\sigma_B} \right) \, , \tag{2.4}$$

μ and σ are the mean and standard deviations. The correlation coefficient matrix R is defined as [4]

$$R = \begin{pmatrix} 1 & \rho(A, B) \\ \rho(B, A) & 1 \end{pmatrix} \, . \tag{2.5}$$

$\rho(A, B)$ ranges from -1 and 1. -1 representing a direct, negative correlation, 0 representing no correlation, and 1 representing a direct, positive correlation. A positive correlation means that, as one variable moves the other variable also moves in the same direction, the closer the correlation coefficient is to 1, the more in sync the variables move. A negative correlation implies that the two variables move in

[4] MathWorks - Correlation Coefficient `https://www.mathworks.com/help/matlab/ref/corrcoef.html?s_tid=doc_ta`, accessed 05/10/2022

opposite direction. Figures 2.1 and 2.2 show scatter plots for which there is a perfect linear relationship and a perfect negative linear relationship, respectively. Figure 2.3 shows a scatter plot for which there in no linear relationship, as so $\rho(x, y) = 0$. These 3 cases represent the extreme cases of the Pearson's correlation coefficient, with real data, the correlation coefficient is not expected to be exactly -1,0 or 1; Figure 2.4 shows a scatter plot with real data where the correlation coefficient is $\rho(x, y) = 0.97$. [39]

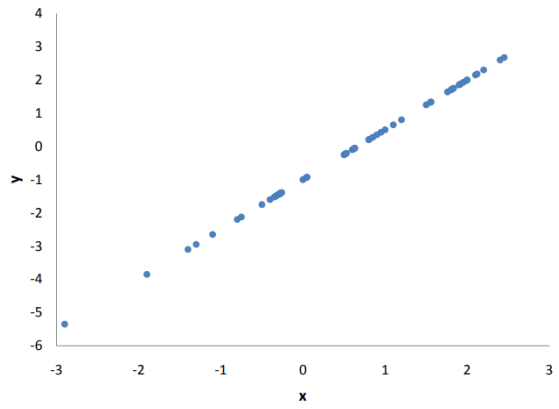


Figure 2.1: Scatter plot for which there is a perfect linear relationship,$\rho(x, y) = 1$. [39]

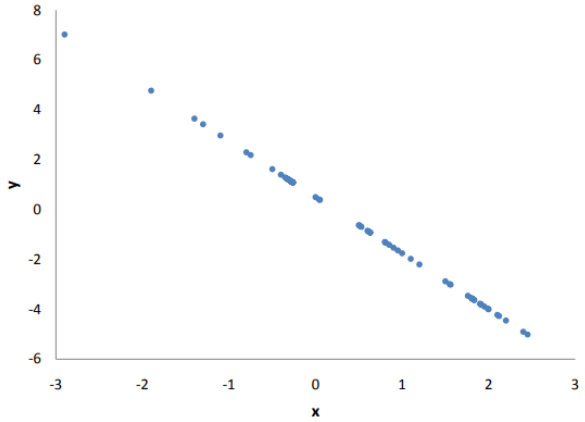


Figure 2.2: Scatter plot for which there is a perfect negative linear relationship,$\rho(x, y) = -1$.[39]

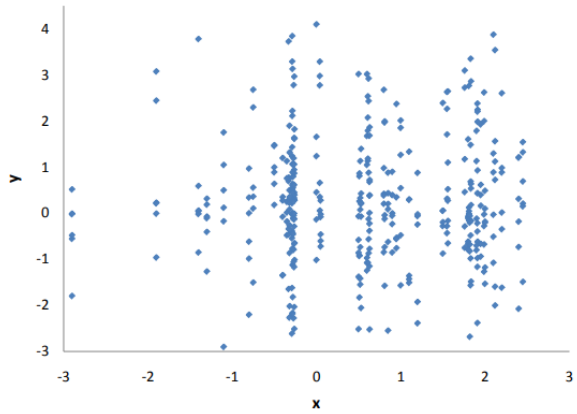


Figure 2.3: Scatter plot for which there is no linear relationship, $\rho(x, y) = 0$. [39]

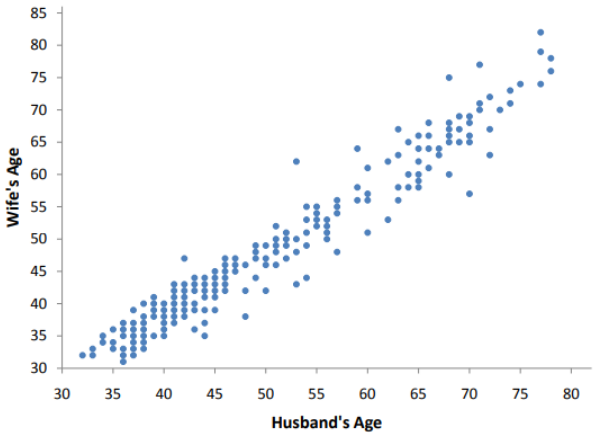


Figure 2.4: Scatter plot with real data for which $\rho(x, y) = 0.97$ [39]

2.3 Previous work

The recent work developed at CfAR on avoidance was developed by Cristóvão [40] and Tavares [41]. Cristóvão [40] developed a framework capable of empowering a UAV from going from a start to a goal point while avoiding static and dynamic obstacles during its course. This framework utilizes a two step approach: there is an offline step that includes Pre-Flight Path Planning and an Online phase with Real Time Path Planning. The offline phases uses a sampling-based motion planning algorithm that is then optimized by minimizing the jerk (fourth derivative of the position) of the UAV. Cristóvão [40] compared two algorithms RRT* and Informed RRT* and reached the conclusion that the Informed RRT* produced better and more optimal results. In the online phase, when dealing with dynamic obstacles, the framework generates a transitioning trajectory around them. Three methods were compared: Informed

RRT*, RRT-Connect and RRT-Connect with a path shortening strategy - the RRT-Connect with the shortening strategy had the best results and therefore was the chosen method. However, this framework had some issues, that were described and addressed by Tavares [41]. The main conclusion was that the online algorithm does not generalize well to an outdoor environment where the static obstacles are mainly at ground level and the dynamic obstacles are in collision route with the UAV. In these situations, the algorithm behaved unexpectedly, creating unfeasible trajectories and the time to generate these trajectories was too high.

Some of these issues were addressed by Tavares [41]. The framework was improved to be more user friendly and it was extensively documented, as well as other small improvements to make it more suitable for real time applications. Moreover, a new online avoidance algorithm was developed and tested . The main goals of this algorithm was for it to be fast (low runtime) and minimization of the part of the original trajectory that is replaced. Thus, Tavares approach was to develop an algorithm that strategically places points around the collision point to generate an avoidance manoeuvre. The entry and exit points of this manoeuvre were calculated with trained Machine Learning models that depended on the velocity and acceleration of the UAV and the avoidance altitude. Simulations were then done to test the efficacy of this new method. With these simulations, some issues were identified - in the first block of tests the shape of manoeuvre was not as expected especially for larger collision angles and uncertainties - the approach failed with collision angles bigger than 153°. The second block of tests raised problems of the conjunction of this method with the sensing method since the space to react is small and most of the corrections to the original trajectory were unfeasible.

It was considered that, given the many changes that had to be done and corrections to this work, that a new approach would be taken for developing an online avoidance algorithm. One of the issues of Tavares [41] work was that it was not fast enough to be implemented in an online scenario with the restrictions that the sensing algorithm provided - for this reason a geometric method was selected as the basis of the new approach. As stated before, one of the advantages of geometric methods is that they are simpler and faster to run. It should be mentioned that the framework developed by Cristóvão and Tavares has state of the art tools that are already implemented, like the Online Montion Planning Library (OMPL) library as well as integration with the FCL; however, the work developed in this thesis started before the previous work had been completed, so an integration between the two was not possible. Nevertheless, this integration could be done in the future in order to build a more robust framework, that would then be easier to test in SITL, Hardware in the Loop (HITL) and eventually flight test.

Chapter 3

Methodology - Path Generation & Collision Detection

In this chapter, the selected method for path generation is discussed as well as its expansion and implementation. An overview on the collision detection method used is also provided.

3.1 Generation of Candidate Paths

The first step in developing the collision avoidance system is the **path generation**. As discussed in the previous chapter, there are a lot of solutions that can be used to generate a group of candidate paths, but the work developed in the path generation stems from the work developed by Chen et al. (2020) [5]. The path generation method adopted (cubic spline method) is a geometric based method that was chosen due to its simple and easy to understand implementation and potential for low computational time. One of the improvements that can be made to the path generation algorithm is its expansion to the 3D space that will be further discussed in Section 3.1.3.

3.1.1 Initial group of candidate paths

The initial group of candidate paths is generated assuming that the UAV is flying a pre-defined path when a threat (static or dynamic) is detected. It is assumed that the position of the threat (in case of a static threat) or the collision point of the dynamic threat and its movement (velocity and its direction) is known, as well as the size. Thus, the UAV needs to avoid these sudden threats, that are assumed to be circular. The corresponding cubic spline curve equation is established as follows [5]

$$y = a(x - x_{start})^3 + b(x - x_{start})^2 + c(x - x_{start}) + y_{start},\ x \in (x_{start}, x_{end}) \tag{3.1}$$

$$c = 0 \tag{3.2}$$

$$a = \frac{c\Delta x + 2(y_{start} - y_{mid})}{\Delta x^3} \tag{3.3}$$

$$b = \frac{3(y_{mid} - y_{start}) - 2c\Delta x}{\Delta x^2}\ . \tag{3.4}$$

y_{start}, y_{end}, x_{start} and x_{end} are the starting position ($O(x_{start}, y_{start})$) and the ending point position ($A(x_{end}, y_{end})$) of the candidate path. x_{mid} is the central abscissa of the threat and $\Delta x = x_{mid} - x_{start}$. The parameters $a, b,$ and c are the cubic spline parameters and since $c = 0$ the shape of the candidate

path is controlled by the variables a and b. For these parameters, the variable that is yet to be defined and that will influence them is the variable y_{mid}. So, when N y_{mid} values are set, N different sets of a and b parameters will be calculated and consequently N candidate paths will be generated. For simplification purposes, y_{mid} is defined as [5]

$$y_{mid} = y_{start} + \omega \, . \tag{3.5}$$

The values of ω are controlled by a step $\Delta\omega$. The values of ω were defined with trial and error and were set as

$$\omega = [-3.2 \times R_{threat}, 3.2 \times R_{threat}], \quad \Delta\omega = \frac{2\omega_{max}}{l} \, , \tag{3.6}$$

and l is a variable that can be changed that controls how many candidate paths are generated. With these values there are certain values of y_{mid} that would be placed within the inside of the threat, these paths are disregarded since they would result in a collision. Finally, the end point is defined as [5]

$$(x_{end}, y_{end}) = (1.5\Delta x + x_{start}, y_{start}) \, . \tag{3.7}$$

Figures 3.1 and 3.2 show examples of the candidate paths generated with this implementation. In these Figures the distance between the threat and the UAV is the same but the size of the threat (R_{threat}) is changed.

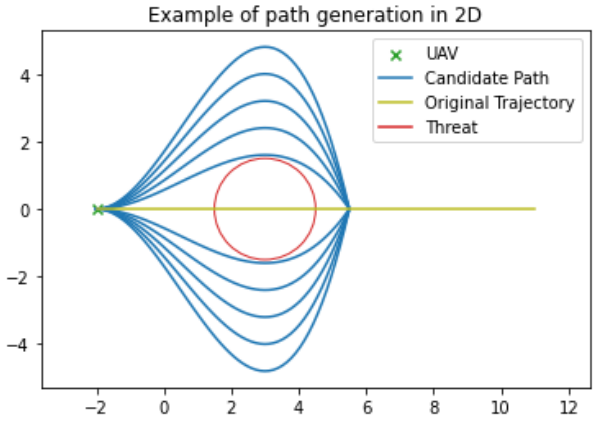


Figure 3.1: Paths generated for $R_{threat} = 1.5$, $l = 12$, using Chen's [5] geometric method.

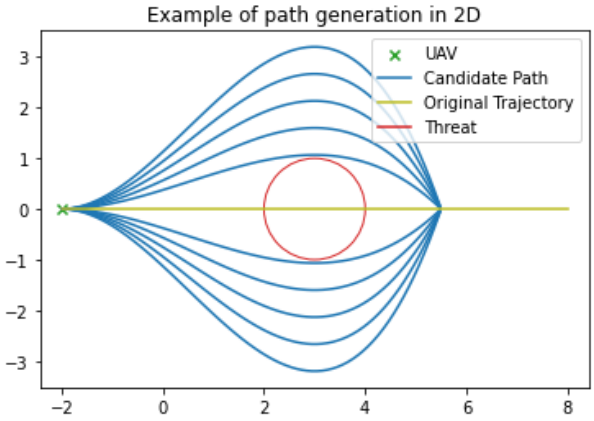


Figure 3.2: Paths generated for $R_{threat} = 1$, $l = 12$, using Chen's [5] geometric method.

It should be taken into account that the paths can only be generated if the UAV is far away enough from the threat. With trial and error, it was found that the minimum distance that should separate the centre of the threat and the UAV is 2.5 $\cdot R_{threat}$. Figure 3.3 shows the limit situation where the UAV is at the limit distance from the centre of the threat and the paths generated in this situation.

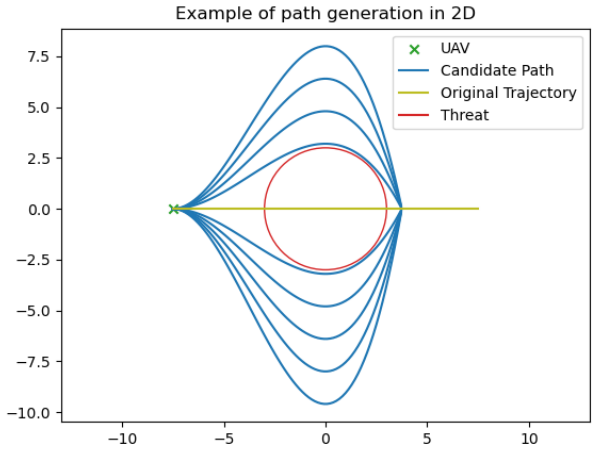


Figure 3.3: Limit situation of the separation between the UAV and the threat $R_{threat} = 3$, $l = 12$, using Chen's [5] geometric method for path generation.

In the situations discussed in this section, the threat and the UAV have the same ordinate. In the case that this does not happen, as previously mentioned, it is assumed that the threat is still centred in the original trajectory of the UAV, and so the candidate paths can still be used as long as they are rotated accordingly (this will be handled by the expansion to 3D space - Section 3.1.3)

3.1.2 Symmetric Candidate Paths

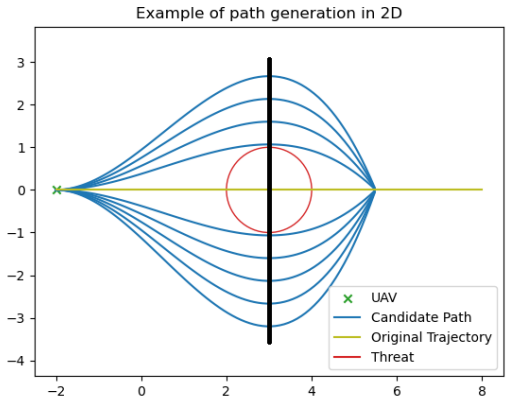
As it can be seen in the Figures (Figures 3.1 and 3.2), the candidates paths generated have a different shape before and after crossing the threat - after crossing the threat the aircraft returns to the original trajectory in a more abrupt way than the one that it leaves the original trajectory with. These type of paths will be named **simple candidate paths**. A symmetry between the beginning of the path and the ending could improve the smoothness of the path and thus making it easier for the UAV fly. So, after the initial group of simple candidate paths is generated, to generate the symmetric group of candidate paths the highest absolute point of the y coordinate is found and the symmetric candidate paths will be mirrored along this y axis in the respective x coordinate. Figures 3.4a and 3.4b illustrate this process.

One of the drawbacks that these group of candidate paths can have is that the ending point of the paths is further away than with the simple group of candidate paths, which will increase the time and energy it takes for the UAV to fly these paths.

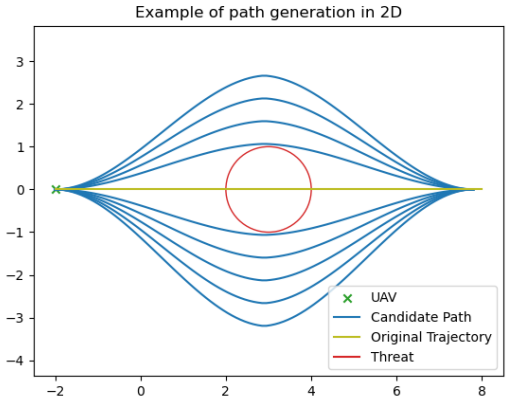
It should also be mentioned that both types of candidate paths (simple and symmetric) will be coded into the framework and the type of paths that is generated can be user-selected (the user can choose between only simple, symmetric, or both).

3.1.3 Expansion to 3D space

Having the equations for candidate paths defined, expanding these to cover more of the 3D space is crucial. This is done in order to have more options to evade the main threat and to reduce the probability of colliding with other threats that may be in the environment, thus making the algorithm more versatile and adaptable. For this, the initial group of candidate paths will be generated in the UAV reference frame and the candidate paths will be rotated across the 3D space.



(a) Simple candidate paths using Chen's [5] geometric method. Black line marks the axis where y has the highest absolute value.



(b) Correspondent symmetric candidate paths.

Figure 3.4: Paths generated for $R_{threat} = 1$, $l = 12$.

3.1.3.1 UAV reference frame

Throughout this work, the reference frame considered is a reference frame like the one in the image below (Figure 3.5).The x axis is represented in red, the y axis in green, and the z axis in blue.

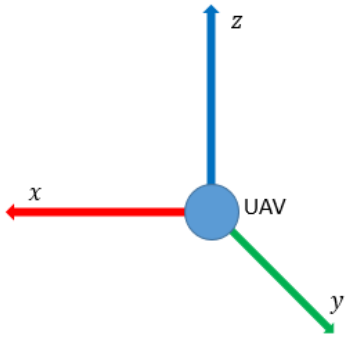


Figure 3.5: UAV reference frame used.

As it was previously explained, to expand the generation of candidate paths to the 3D space, first a plane needs to be chosen to generate the initial group of candidate paths before rotating them to cover the 3D space. To do this, the reference frame of the UAV is used and the xy plane in the UAV reference frame is the one chosen for this task. For the purposes of simplifications, as explained above, the threat is considered to be centred in the trajectory of the UAV and so the xy plane of the UAV has to be a plane that slices the threat in half, with the x axis in the direction of the trajectory and the origin of the reference frame is the current position of the the UAV. Figure 3.6 shows an example of a reference frame with this restrictions.

It should be noted that, even if the threat is not centred in the trajectory of the UAV, the algorithm can consider a threat that is and that encompasses the first one.

In future work, the information for the UAV reference frame should be taken as an input. For now, only the x axis and the origin of the reference frame is considered and a "sample" reference frame is created with this information. This is done by, with the x axis vector coordinates, generating a vector - y axis, that is perpendicular to the x axis that also goes through the origin and then using the cross

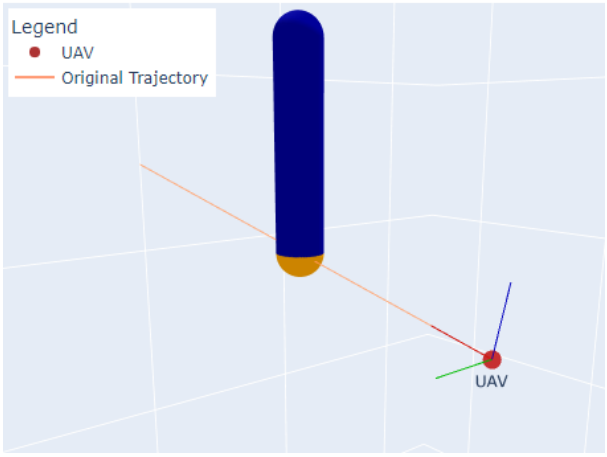


Figure 3.6: Example of reference frame in the 3D environment. x axis is aligned with the original trajectory and the origin of the reference frame is the UAV current position. The collision point is represented by the orange sphere, collision path by the blue cylinder.

product between the two to get the remaining vector - z axis , that is perpendicular to both. With the UAV reference frame defined, the plane to generate the candidate paths can also be defined.

To handle conversions from the global reference frame to the UAV reference frame, the `pytransform3d` library is used [42] - a python library for transformations in three dimensions. The velocity of the UAV is given in the global reference frame. The coordinates of the candidate paths are originally generated in the UAV reference frame but are then transformed to the global reference frame with the use of this library.

3.1.3.2 Rotation of the candidate paths

As mentioned in the previous section, the original plane where the initial group of candidate paths is generated is the xy plane of the reference frame of the UAV . The initial group of candidate paths are the candidate paths that would be generated in 2D, but that are now written in the UAV reference frame. An example representation of this plane can be seen in Figure 3.7 in pink.

Secondly, the rotation axis is chosen. The rotation axis is the line that connects the evading point and the centre of the threat. This can be seen in Figure 3.8 where the rotation axis is represented in yellow.

Thirdly, by rotating the original plane along the defined rotation axis according to a certain angle θ, target planes will be obtained where new candidate paths will be generated. The goal is to cover the 3D space in the most efficient way, so all the target planes need to be equally spaced out from each other. Since the original group of candidate paths are generated both at the left and right of the threat, θ only needs to be defined between [0, 180 °]. To equally space out the planes according to the number of target planes to be generated n, equation (3.8) is defined that describes θ_i, the rotation angle for the i-th target plane. An example representation of the target plane can be seen in Figure 3.9.

$$\theta_i = \frac{180°}{n+1} \cdot i, \; i = 1, ..., n \tag{3.8}$$

With all of this information, a rotation matrix that rotates points from the original plane to the target plane based on the angle θ_i can be defined. These rotation matrix are obtained using the `pytransform3d` library function `active_matrix_from_angle`.

So, with the initial group of candidate paths with the coordinates in the UAV reference frame, these points are rotated to a target plane separated by θ_i by means of a rotation matrix. These points, that

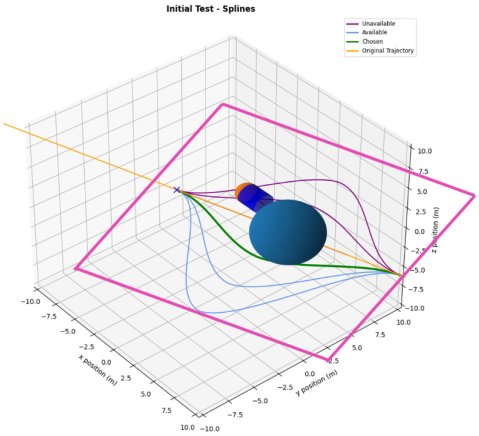


Figure 3.7: Initial group of candidate paths generation plane (pink).

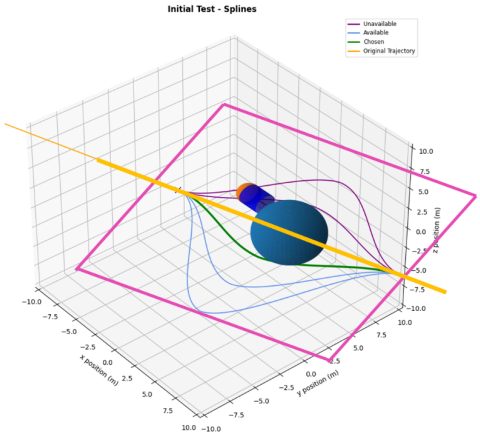


Figure 3.8: Rotation axis (yellow) used to rotate the initial group of candidate paths.

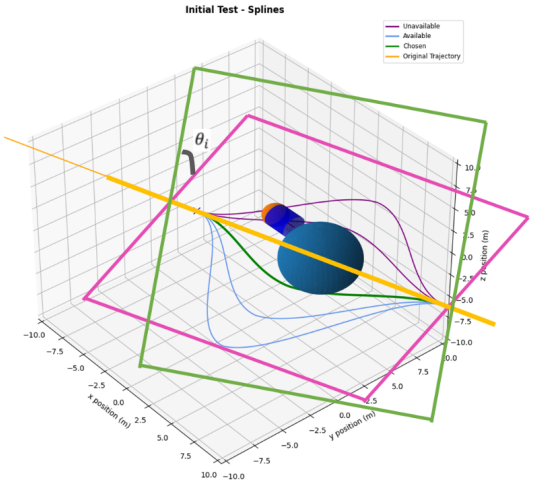


Figure 3.9: Target plane (green) and rotation angle (θ_i) example for rotating the initial group of candidate paths (in the pink plane) to 3D space.

are still written in the UAV reference frame, are then transformed to the global reference frame. With the candidate path in the global reference frame, the angle of elevation α (the angle between the plane where the candidate path is and the ground plane) is calculated that will then be used for cost calculation purposes (see Section 4.1.3).

It should also be mentioned that this approach is reasonable because the threat to avoid is assumed to be spherical, so candidate paths that would avoid it in the original plane will still not collide with it in the target plane.

When generating new target planes where to project new candidate paths, computational limits have to be taken into account. Thus, the impact of the number of target planes generated is important. In environments with few threats (1 to 3) it was found that a reasonable amount of target planes is $n = 3$. With more threats, more target planes can be added.

The total number of candidate paths generated is calculated as follows

$$L = l \cdot (n + 1) \,. \tag{3.9}$$

3.2 Uncertainty

Uncertainty has to be considered in order to ensure the safety and robustness of the algorithm and its implementation. There are several factors to contribute to it and these have to be carefully considered. Two types of uncertainty are considered: zero-order (associated with the position) and first-order uncertainty (associated with the velocity). The main factors that contribute to these are **sensor's uncertainty** and **tracking error**.

Sensor's uncertainty comes from the uncertainty associated with the sensors that are used onboard of the UAV. These come from namely the Global Positioning System (GPS). There may also be some uncertainty associated with the speedometer (measures the velocity), the gyroscope (measures the orientation) or the accelerometer (measures the acceleration), among other sensors.

Tracking error is associated with the original path that is generated for the UAV. It should not be assumed that the aircraft is perfectly following the path originally programmed. Some small delays may have occurred that made the UAV being slightly ahead or behind schedule.

To mitigate the effects of the uncertainty, a collision radius is introduced in the collision detection step of the framework. With this collision radius, instead of the UAV being considered a point it is considered a box with that given collision radius and the collision detection is done with that in mind, as it will be further explained in Section 3.4. The default value used for the collision radius is `collision_radius` = 2 m.

3.3 Velocity

To be able to calculate the cost function to choose the best path out of the candidate paths, the velocity in each waypoint has to be known, as it will be discussed in Section 4. The velocity has 3 components: v_x, v_y, v_z, and to calculate it the cruise velocity of the UAV in question is considered to be constant throughout the path. For simplification, it is also assumed that the trajectory between each waypoint can be modelled as a straight line.

To calculate the velocity in the $i-th$ waypoint v_i, first the direction vector between two consecutive waypoints (in the UAV reference frame) is calculated

$$d_{v_i} = p_{i+1} - p_i; \tag{3.10}$$

this vector is then normalized and multiplied by the cruise velocity to obtain the velocity in each component

$$v_i = \frac{d_{v_i}}{||d_{v_i}||} \cdot v_{cruise} = (v_{ix}, v_{iy}, v_{iz}). \tag{3.11}$$

Depending on the waypoints, some of the velocities may be unviable for the UAV, for example if v_{iz} exceeds the maximum ROC, the UAV would not be able to fly that path. To combat this, and since the limits of the UAV are known, v_{iz} is made equal to the max ROC and the remaining components v_{ix} and v_{iy} are calculated to ensure the direction calculated previously d_{v_i} is maintained. The new velocity in case the ROC is exceed is calculated as follows.

$$v_i^{new} = \frac{max_{ROC}}{v_{iz}} \cdot v_i \tag{3.12}$$

Figure 3.10 shows the results of this method for a small sample of points. The dots represent the

waypoints p_i of the UAV and the line the velocity vector v_i in that waypoint. In each waypoint it is shown the absolute velocity of the UAV $||v_i||$ followed by the values of each component $[v_{ix}, v_{iy}, v_{iz}]$.

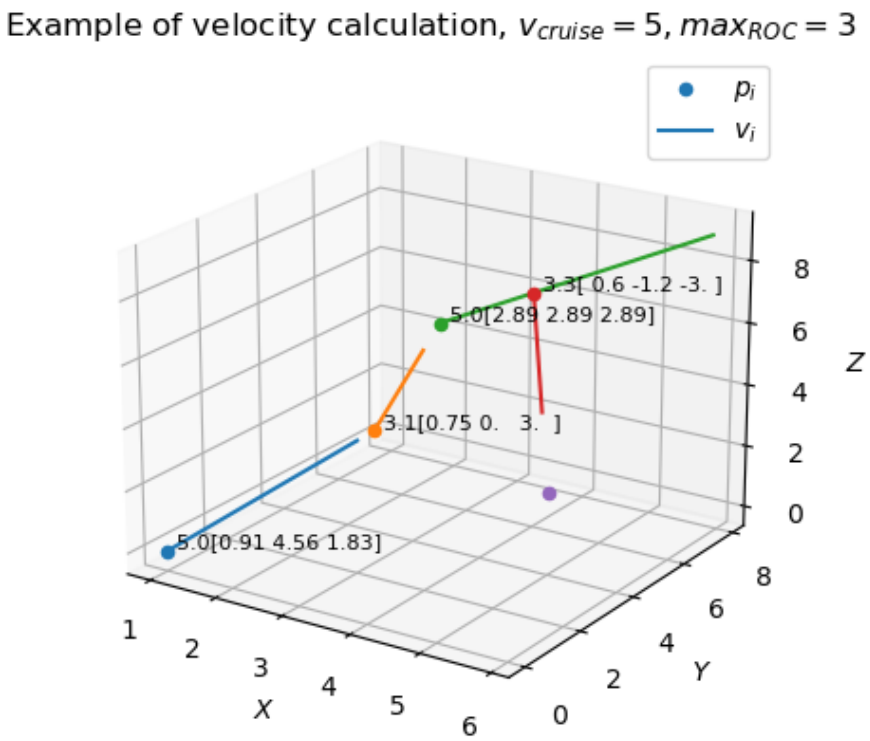


Figure 3.10: Example of the velocity calculation method working. The dots are the waypoints p_i, and the lines the velocity vectors v_i.

In future work, the velocity of the UAV in each waypoint should be given by a model that has the dynamics of the aircraft in mind, and thus gives more accurate values for the velocity in each waypoint. For now, the method described above is used as a simple approximation.

3.4 Collision Detection

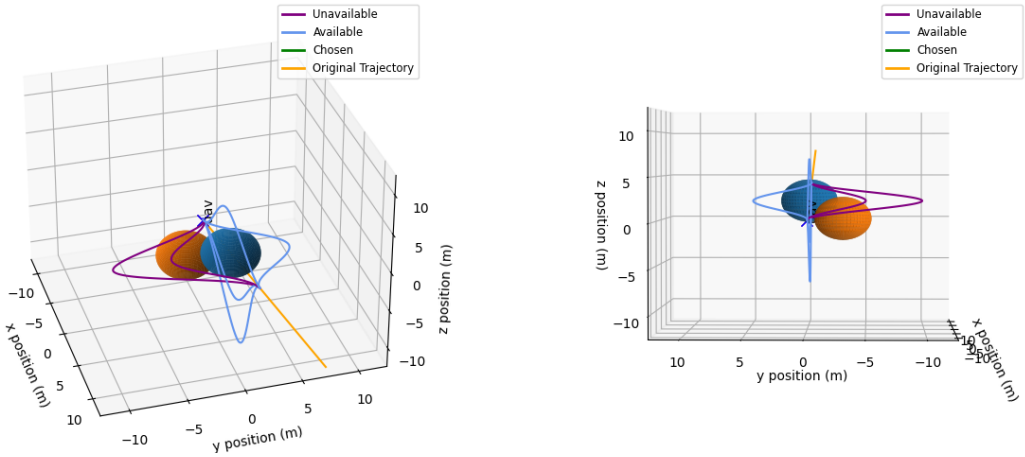
In this section the collision detection implementation will be discussed. After all the paths are generated, each one is first classified as either ”Available” or ”Unavailable” taking into account its **static security** and **dynamic security**. The static security describes whether there was a collision with a static (no velocity) object, whereas the dynamic security describes whether there was a collision with a dynamic (moving) object. Both the static and dynamic security were inspired by the work developed by Chen et al. (2020)[5].

3.4.1 Static Security

Static security is implemented on the basis of 0/1, *i.e.* either a path is statically secure or not. If there is not a known collision between the UAV and static threat the path is considered to be statically secure and will be checked for dynamic security. This implementation is done using the FCL, and assumes that the position and the size of threats are known or at least a good estimation of these variables is available. The FCL [6] is a library for performing three types of proximity queries on a pair of geometric models [1], in this work the Continuous Collision Request function is used.

[1] FCL (GitHub). `https://github.com/flexible-collision-library/fcl` accessed 19/09/2022.

For the static collision detection, first, a collision object on the form of a box, more specifically a cube, with a side equal to the collision radius defined in the configuration files to account for uncertainty is created. For each segment (line between two consecutive waypoints) of the candidate path, the collision box is considered to start in the coordinates of the first waypoint of the segment and moving to the second waypoint of the segment in a straight line. The collision object for the static threat is already defined when the threat is created and, as mentioned previously, is modelled as a sphere. Finally, a continuous collide request between the UAV moving from the first waypoint of the segment to the second and the threat is made, and once there is a collision, the candidate path is marked as unavailable by setting the collision parameter of the candidate to `True`. If there is not a collision detected, the collision parameter remains set as `False` and the collision detection moves on to the dynamic security check. Figures 3.11a and 3.11b illustrates an example of collision detection with static obstacles. Paths that are not available (that collide) are highlighted in purple.



(a) General View.

(b) Front view.

Figure 3.11: Static security detection example. Blue and orange spheres are threats.

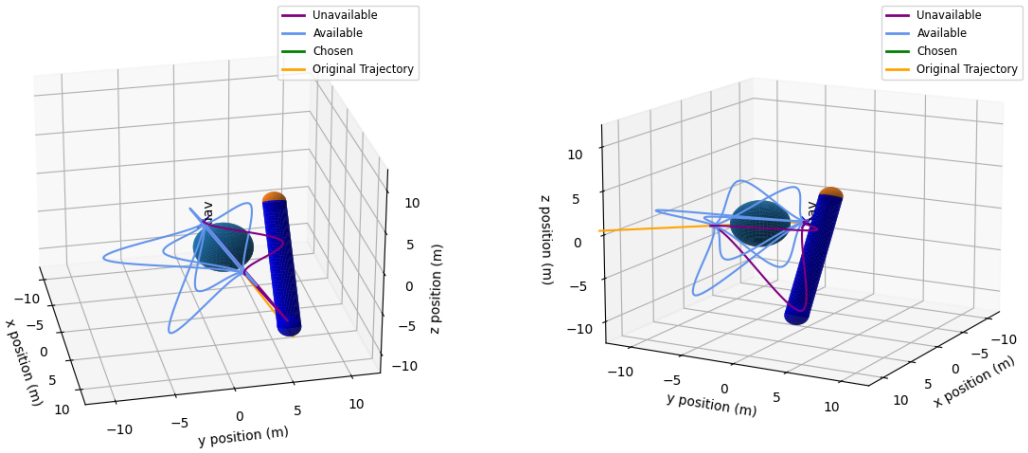
3.4.2 Dynamic Security

After a path is deemed to be statically secure, it is then checked for its dynamic security. Similarly to the static security, the dynamic security is also implemented on 0 or 1 basis, either the path is dynamically secure or not.

Dynamic security takes into account the moving objects/threats present in the environment and if any of them have the possibility to collide with the UAV. For this, the position and the velocity of threats needs to known or at least estimated. An estimation of the time that the UAV will take to evade the main threat has also to be known in order to calculate how much movement is expected from the moving threats/objects while the UAV is evading the main threat, the velocity estimation discussed in Section 3.3 is used to calculate this time. Again, uncertainty in the size, position and velocity of the threats has to be taken into account in order to make the algorithm more robust.

For the collision detection with dynamic threats, firstly the end point of the threat given its trajectory is calculated. This is done assuming a straight line trajectory from the threat for simplification purposes. Each candidate path is associated with the aircraft , and the maximum time that the threat is possibly going to be interfering with the aircraft that is flying is defined as the maximum time that any of the candidate paths would take to fly by the aircraft. So, the end point of the trajectory of the threat is defined based on this maximum time, and on the velocity that the the threat is moving. The dynamic security was implemented with the FCL[6]. Again, the UAV is modelled as a box (cube) with sides equal to the

collision radius variable defined in the configurations to take into account uncertainty. Again, for each segment of the aircraft's trajectory, there is a continuous collision request made between the segment and the moving threat. Similarly as with the static security, once a collision between the aircraft and the threat is detected the candidate path is marked as unavailable and its cost will not be calculated. Figures 3.12a and 3.12b shows collision detection with moving obstacles. The moving threat is represented by the orange threat, and its path while the UAV is flying is represented by the dark blue cylinder. The unavailable paths (paths with collision) are coloured purple.



(a) General view.

(b) Side view.

Figure 3.12: Dynamic security detection example. Blue cylinder is the collision path, orange spheres is the dynamic threat initial position, blue sphere is the main (static) threat.

Chapter 4

Methodology - Cost Function

This chapter discusses the process of defining a cost function to evaluate the candidate paths generated on the previous chapter. First, the six different cost function parameters are defined and described (Smoothness, Smoothness with penalization, Energy, Hovering Power, Time and Closeness to MO). Next, thousands of candidate paths are generated and the value of each cost function parameter is calculated for that given candidate path. The results are analyzed in order to choose the most relevant parameters and normalize them so they can be comparable. Finally, the weights for the cost function are chosen and tuned, based on empirical data and correlation analysis between the different parameters.

4.1 Cost Function Parameters

As mentioned in section 2.1.3, the goal of the cost function is to numerically describe how good or a bad a candidate path is. To do this, 6 parameters are studied (energy, smoothness, smoothness with rotation penalization, closeness to moving objects, time and hovering power). These 6 parameters will be defined in this subsection and the process to reach those definitions will be explained and justified.

4.1.1 Energy Consumption

Choosing an energy efficient path is an important factor to consider, since the energy is one of the constraints regarding the flight of a UAV. Thus, choosing an energy model to describe the energy that would be spent in each path is a crucial part of the cost function. To do this, it is important to understand the different factors that affect the energy consumption of the UAV. This can be illustrated in Figure 4.1. The dotted lines represent an indirect relationship between the two concepts and the full lines a direct relationship. Weather (temperature, air density) is critical for energy consumption as it affects the travel speed of the UAV, and the temperature in the atmosphere can affect the energy capacity of batteries used in UAVs - i.e. battery performance could be negatively impacted by cold temperatures until the batteries warm up [43]. Flying with the wind could, for example, reduce energy consumption, and the opposite, flying against the wind, would increase this consumption.The weight and payload carried by the UAV are also a critical factor, since a heavier aircraft or heavier payload will need more energy to fly.

Zhang et al., (2021) [44] compares 12 different energy consumption models for delivery drones, with various approaches taken to the problem. These approaches differ in the thrust assumption for flight, the travel components considered (horizontal, hover, vertical), if it takes into account different factors (wind, avionics, empty return), the type of model (theoretical model, regression model or both) and if the model was field tested. They come to the conclusion that even using the same or similar parameters for 5 selected models, the results greatly differ, proving that there is not a unified approach to this problem.

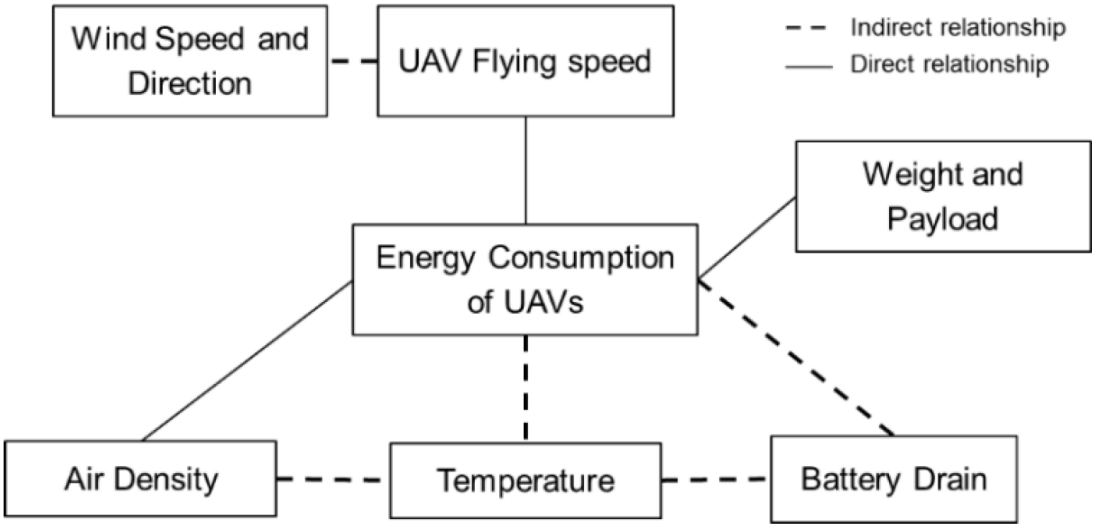


Figure 4.1: Factors that affect energy consumption of UAVs. [43]

However, some of the models present similar behaviour in the results, for example in the shape of the curves - this can be seen in Figure 4.2.

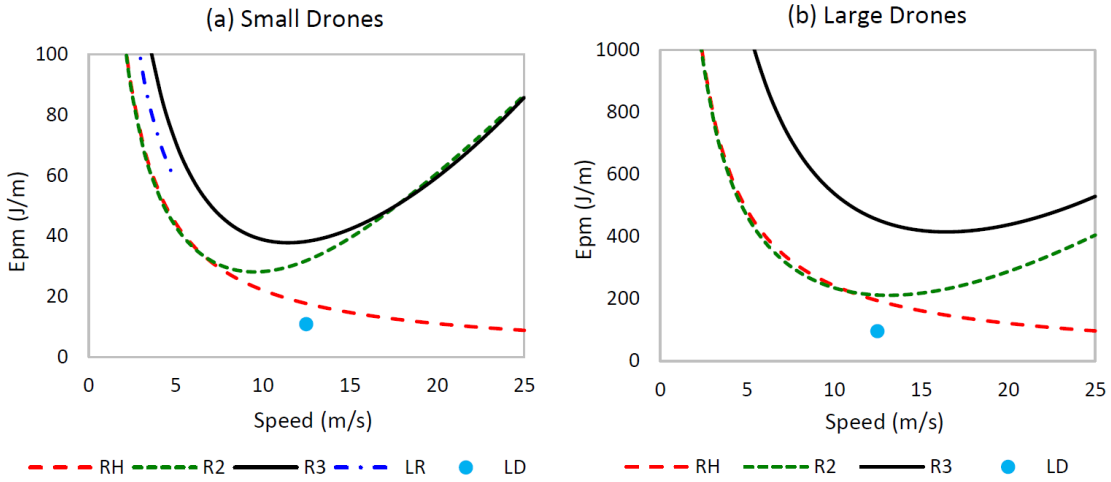


Figure 4.2: Energy consumption rate versus airspeed for small and large drones.[44]

The models studied by Zhang et. al [44], among others, require several aerodynamic components to be calculated. Since one of the goals of the dynamic avoidance algorithm is to be as a general as possible, an ideal model would take into account the least amount possible of aerodynamic components of the UAV to simplify the problem. This is a valid approach since the main goal is to compare between different candidate paths, so a specific number for the energy consumption for the given candidate path is not needed, but how this energy consumption compares to the other candidate paths is more relevant.

The model chosen was one developed by Marins et al. (2018) [45]. This model was chosen since it had also been previously used by Ramalho (2020) [46] in previous work developed at CfAR and it takes into account the factors that affect energy consumption described by Thibbotuwawa (2020) [43].

In this model, the power related to the acceleration/deceleration can be calculated as the variation of

kinetic energy.

$$E_1 = ||E_M(i+1) - E_M(i)|| \tag{4.1}$$

$$E_M(i) = \frac{1}{2}m||v_i||^2 + mgp_{i,z} \tag{4.2}$$

In this set of equations, E_1 is the consumed energy, $E_M(i)$ the mechanical energy at state i, m is the mass of the UAV, g is the gravitational acceleration, v_i the speed of the UAV at state i, and $p_{i,z}$ the position of the UAV along the z axis.

It is assumed that the work done by the drag forces is proportional to the distance between waypoint, times the average speed squared.

$$E_D(i) = \frac{1}{2}C_D\rho A_e||p_{i+1} - p_i||\frac{||v_{i+1} + v_i||^2}{2}\ , \tag{4.3}$$

and

$$E_2 = \sum_{i=0}^{n} E_D(i)\ , \tag{4.4}$$

where C_D is the drag coefficient, ρ the air density, A_{ef} the effective area of the aircraft, p_i the position at state i and v_i the velocity at state i.

The total energy throughout the path is the sum of the two components

$$E_T = E_1 + E_2\ . \tag{4.5}$$

This model takes into account all of the direct relationships that affect the energy consumption of UAVs as described in Figure 4.1. The air density (ρ) is used when calculating the energy due to drag forces (4.3), the weight of the aircraft is considered when calculating the variation of kinematic energy (4.2), and the UAV flying speed is taken into account when calculating both types of energy. It was not possible to reproduce the curves of Figure 4.2 to test if this model is similar to the ones studied by Zhang, nonetheless, since it had been previously implemented and used the factors described by the same work, it was considered a good fit for this work.

4.1.2 Smoothness

Another factor that is used to evaluate the quality of the paths generated is the smoothness of the paths generated, this allows the UAV to fly stably and without sudden changes of movement. Several methods to calculate the smoothness of the path were studied, these are:

- curvature,
- standard deviation,
- second derivative.

Each of these options will be further explained below.

4.1.2.1 Curvature

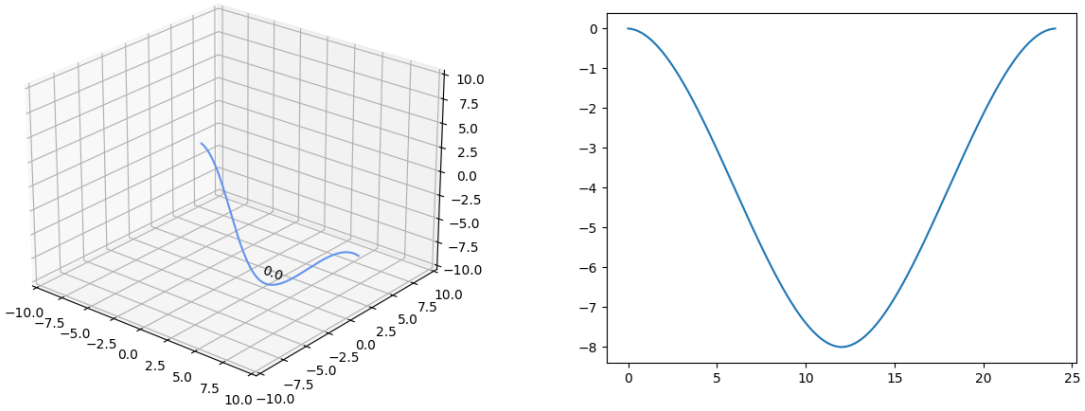
Chen et al. [5] defines the smoothness as:

$$S(i) = \int_{x_{start}}^{x_{end}} \kappa_i^2\, dx \tag{4.6}$$

where κ_i is the curvature of the i-th candidate path. The curvature of the candidate path is defined as

$$\kappa = \frac{|x''y' - x'y''|}{((x')^2 + (y')^2)^{3/2}} \tag{4.7}$$

x', x'', y' and y'' are obtained using the numpy function `np.gradient` and are calculated from the coordinates (x and y) of the shape of the candidate path. The **shape of a candidate path** is the 2D representation of the path in the plane where it is defined. It is calculated with the method `define_shape` that rotates the candidate path from the plane where it is defined to the xy plane, the resulting path is the shape of the candidate path. Figure 4.3a shows one candidate path in the 3D space and Figure 4.3b the respective shape.



(a) Plot of the candidate path in 3D.

(b) Shape of the same candidate path.

Figure 4.3: Example of a candidate path and its shape.

The less curvature a path has, the more smooth the path will be which means that the smoothness cost will be lower.

4.1.2.2 Standard Deviation

Another method studied to calculate the smoothness of a path was the standard deviation of the y coordinate of the shape of the candidate path. The idea behind this approach is, since the standard deviation is a measure of the spread of a distribution, that the higher the standard deviation is, the higher the difference between the y coordinates will be and the less smooth the path will be. Thus, the smoothness with this method is defined as

$$S(i) = \sqrt{\frac{\sum |y - \mu_y|^2}{N}}\,. \tag{4.8}$$

Where N is the number of points and μ_y the mean of the y coordinate values. This is calculated with the `numpy` function `numpy.std` that calculates standard deviation along a specified axis.

4.1.2.3 Second Derivative

The last method studied to define the smoothness of a candidate path is taking the second derivative of the shape of said path. The second derivative measures the instantaneous rate of change of the first derivative. Since the first derivative tells us the slope of a tangent line to the curve at any instant, the higher the rate that the first derivative changes, the more curve the path will be - which means less smoothness. Thus, the smoothness can also be defined by taking into account the second derivative as

$$S(i) = \frac{\sum_{n=0}^{N} |y''(x_n)|}{N} , \tag{4.9}$$

where N is the number of points that defined the candidate path, $y(x_n)$ is the y value of the shape of the candidate path when $x = x_n$, and x_n corresponds to x coordinate of the n-th waypoint that defines the shape of the candidate path.

4.1.2.4 Comparison and Selection

All of these 3 methods, in a way or another, measure the smoothness of the candidate path.

The standard deviation only takes into account the y coordinate, which means that, unless the shape of candidate path is always aligned with the x axis (which happens as the way `define_shape` is defined right now) the standard deviation will not be accurate. So, this method is not future proof and expandable to all kinds of candidate paths in case the `define_shape` method needs to be changed.

Although there are no significant problems with the second derivative, curvature was the approach chosen since it has been used successfully in other research [5]. Additionally, it is also the one that more accurately depicts the concept of smoothness (inherently the more curvature a path has, the less smooth it is). Nonetheless, if needed, the smoothness method can be changed through the configuration file. It should be noted that if this is done, the normalization process (that will be discussed in Section 4.2.2) needs to be re-calibrated.

4.1.3 Smoothness with Rotation Penalization

Since the smoothness only evaluates the shape of the curve and thus does not distinguish paths with the same shape in the 3D space, a penalization term for situations where the path is rotated from the horizontal plane and thus the UAV would have to climb/descend is added. So, the smoothness with rotation penalization can be defined as

$$S(i)_{rp} = S(i)(1 + k_p|\sin\alpha_i|) , \tag{4.10}$$

where α_i is the angle of elevation of the candidate path, the angle between the candidate path plane and the ground plane. This means that paths that are rotated 90 ºwill have the maximum possible penalization ($S(i)_{rp} = S(i) + k_p S(i)$) and have a higher cost and the ones that are in the horizontal plane will have the least penalization ($S(i)_{rp} = S(i)$) and a lower cost.

4.1.4 Closeness to Moving Objects

Another component that can be added to the cost function to measure the safety of the created candidate paths is the closeness to moving objects. Moving objects are more unpredictable than static ones and so there should be more care used when handling them. To measure the safety of a candidate path in relation to a moving object the closeness of the path to moving objects will be measured.

Since each path is represented by the set of points that defines it and since the threat path can be defined in the same way, to measure the closeness between the two paths, the distance between each waypoint and each threat path waypoint will be calculated. This can be seen in Figure 4.4, where for the waypoint *a* the distance to the different threat waypoints is calculated (lines 1,2,3 and 4). Then, the smallest one, *i.e.* the closest neighbour, is associated with the waypoint of the trajectory of the UAV; this is repeated for the remaining waypoints of the trajectory (*b,c,d,e* in the figure). Thus, each UAV waypoint will have a distance to the moving threat associated with it. This process is done with the help of the

SciPy library [47] with the function `scipy.spatial.KDTree` that finds the k closest neighbour between a point (the candidate path waypoint) and a group of points (the threat waypoints). In this context $k = 1$. A constant r equal to the radius of the threat is subtracted to the distance between the waypoint and the closest neighbour in order to get the minimum possible distance between the UAV and the moving threat. This distance r is not illustrated in the remaining figures of this section in order to simplify the illustration of the problem.

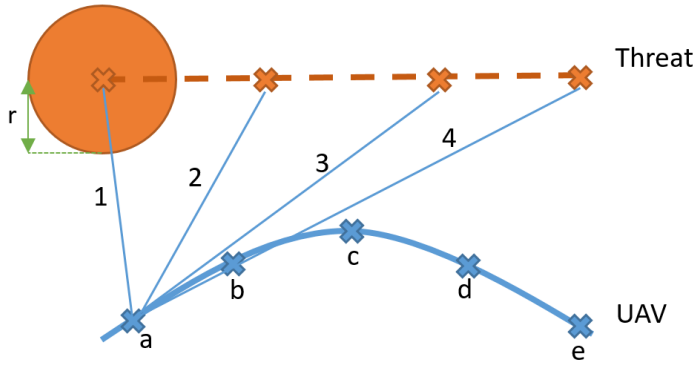


Figure 4.4: Calculation method used for computing distance for Closeness to MO. Orange line and circle represent the threat, blue line represents a candidate path. r is the radius of the threat.

With this information there are now two ways that were considered to measure this metric. The first one is to take the **average of the distances** ($\bar{c}$) between the UAV waypoints and the moving threat waypoints. The second one is to take into account only the **minimum distance** (c_{min}) from the UAV waypoint to the threat. Figures 4.5 and 4.6 illustrate two possible situations where each option will be studied. Let us assume that using the first method (average of the distances) the result would be the same in both situations. However, it is clear that the second situation is more dangerous given that the UAV comes in close proximity to the threat and this calculation method does not reflect that. Using the second method, this proximity would be taken into account. Thus, to calculate the closeness to moving objects the **minimum distance** between the UAV trajectory and the moving threat has to be taken into account.

Figure 4.5: Possible situation 1 when calculating Closeness to MO .

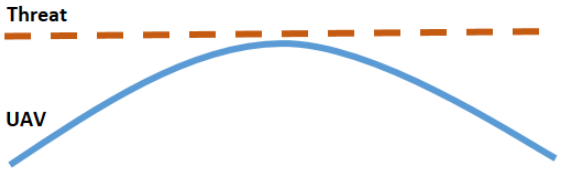


Figure 4.6: Possible situation 2 when calculating Closeness to MO .

However, it is also important to not consider only one distance when calculating the closeness to moving objects, since it will not accurately describe the bigger picture. Figures 4.7 and 4.8 illustrate this problem. Although in both figures the minimum distance c_{min} is the same, in the second figure the path is clearly more dangerous since the UAV spends more time closer to the moving threat, and this fact can be described by the average distance $\bar{c}$. Thus, both methods should be considered: the minimum distance to the moving threat and the average of the distances to the moving threat.

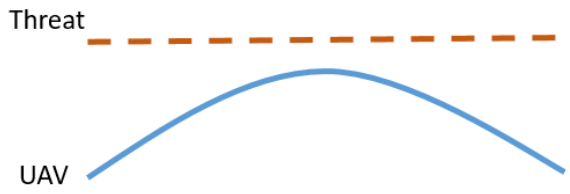


Figure 4.7: Possible situation 3 when calculating Closeness to MO .

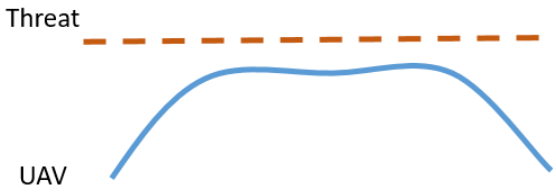


Figure 4.8: Possible situation 4 when calculating Closeness to MO .

Therefore, the closeness to moving objects can be described as

$$f_c(i) = (k_1\bar{c} + k_2 c_{min})^{-1} \,, \tag{4.11}$$

the expression is inverted since a lower distance means more danger which in consequence should mean more cost. The constants k_1 and k_2 can be tuned to give more weight to either $\bar{c}$ or c_{min}. For simplifications purposes these weights will be tuned to the same values $k_1 = k_2 = 0.5$, however they can be changed by the user to prioritize one factor or the other.

In the situation that there is more than one moving object in the environment, the cost function for the closeness with moving objects is calculated individually for all the moving objects present in the environment. Then, the final value for the cost function is the average between the maximum value of $f_c(i)$ and the average of all the $f_c(i)$ values .

$$M(i) = 0.5max(f_c(i)) + 0.5\overline{f_c(i)} \,. \tag{4.12}$$

4.1.5 Time

The time that it takes for the UAV to fly the path can be taken into account when calculating the cost function for a certain path. Knowing the velocity in each waypoint (as it was discussed in Section 3.3) and the coordinates for each waypoint makes this calculation trivial. The distance between each set of waypoints is calculated by subtracting the coordinates between the $i+1$ and i set of coordinates and calculating the norm for the result; this assumes that the path flown between each set of waypoints is a straight line. The time between each set of waypoints is then calculated by dividing the distance by the velocity, and the sum of all of these times is the final time that the UAV takes to fly the path.

4.1.6 Hovering Power

Using the hovering power to calculate the cost function of a candidate path allows to distinguish between paths that climb/descend based on the amount and velocity that they do so. There are three possible methods for calculating the hovering power depending on which phase of flight the UAV is at, which is determined in each segment between two waypoints and then added for the entire path.

1. Hovering Flight ,
2. Vertical Climb Flight ,
3. Vertical Descent Flight .

Assuming that the UAV that is flying is a quadrotor, according to [48], assuming no wings are located below the rotors, the total power required for **hovering** is

$$P = P_{i,R} + P_0 \,, \tag{4.13}$$

where $P_{i,R}$ is the real induced power and P_0 the rotor's profile power.

The ideal induced power is calculated as

$$P_i = T v_{i,h} = T\sqrt{\frac{T}{2\rho A}} \,, \tag{4.14}$$

where $v_{i,h}$ is the induced velocity at the rotor in hover, T the rotor thrust, ρ the air density at hover altitude and A the rotor disk area. However, this calculation is for the **ideal** induced power, which does not take into account drag and other inefficiencies; to calculate the **real** induced power, a multiplicative factor k_i is included. This factor is approximated to $k_i = 1.15$. Thus, the real induced power can be calculated as

$$P_{i,R} = T k_i v_{i,h} = k_i W\sqrt{\frac{W}{2\rho A}} = k_i \frac{W^{3/2}}{\sqrt{2\rho A}} \tag{4.15}$$

The rotor's profile power, P_0 in equation (4.13), considers rectangular blades and C_d (average blade drag coefficient), independent of Mach and Reynold's numbers, and will be calculated as follows

$$P_0 = \rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) \,. \tag{4.16}$$

The maximum value that the blades' tip velocity (V_{tip}) can be is determined as $V_{tip} = 0.8M$; otherwise, the blades would enter the transonic regime.

So, for **hovering flight**, the total power equation can be written as

$$P = P_{i,R} + P_0 = k_i \frac{W^{3/2}}{\sqrt{2\rho A}} + \rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) \,. \tag{4.17}$$

For **vertical climb flight**, assuming the absence of wings under the rotors, the total power required is given by

$$P = P_0 + P_c + P_i = T(V_y + k_i v_{i,c}) + \rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) \tag{4.18}$$

where P_c is the power to climb, V_y the rate of climb and $v_{i,c}$ the induced velocity in climb, written as

$$v_{i,c} = -\frac{1}{2}V_y + \frac{1}{2}\sqrt{{V_y}^2 + \frac{2T}{\rho A}} \,. \tag{4.19}$$

By rearranging the initial equation, and assuming that the drag generated during climb is negligible with respect to the aircraft's weight ($T = W$), the vertical climb flight power is given by

$$P = W\left(V_y - \frac{k_i}{2}V_y + \frac{k_i}{2}\sqrt{{V_y}^2 + \frac{2W}{\rho A}}\right) + \rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) \,. \tag{4.20}$$

For the power when in **vertical descent flight**, the momentum theory can be invalid given that a more complicated recirculating flow pattern can exist at the rotor. Thus for the interval $-2v_{i,h} \leq V_y \leq 0$, empirical relations will be used and for $V_y/v_{i,h} \leq -2$ the momentum theory will be employed. The power for vertical descent flight can be calculated similarly as in equation (4.18), whereas instead of considering the $v_{i,c}$ the $v_{i,d}$ - induced velocity for the descent condition, is written as

$$v_{i,d} = -\frac{V_y}{2} - \frac{1}{2}\sqrt{{V_y}^2 - \frac{2DL}{\rho}} \tag{4.21}$$

and thus the power required for **vertical descent flight** when $V_y/v_{i,h} \leq -2$ is given by

$$P = W\left(V_y - \frac{k_i}{2}(V_y + \sqrt{{V_y}^2 - \frac{2DL}{\rho}}\right) + \rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) . \tag{4.22}$$

As for $-2v_{i,h} \leq V_y \leq 0$, $v_{i,d}$ is approximated by the quartic equation

$$v_{i,d} = v_{i,h}\left(k_i + K_1\left(\frac{V_y}{v_{i,h}}\right) + K_2\left(\frac{V_y}{v_{i,h}}\right)^2 + K_3\left(\frac{V_y}{v_{i,h}}\right)^3 + K_4\left(\frac{V_y}{v_{i,h}}\right)^4\right) \tag{4.23}$$

where $K_1 = -1.125$, $K_2 = -1.372$, $K_3 = -1.718$, $K_4 = -0.655$. $v_{i,h}$, the induced velocity at the rotor in hover, is given by

$$v_{i,h} = \sqrt{\frac{W}{2\rho A}} . \tag{4.24}$$

The rotor solidity σ used in equations (4.17), (4.20), (4.22) is defined by the are of the rotor disk that is actually occupied by blade area. Thus, given that the are of rotor blades is given by $N \cdot c \cdot R$ and the area of the rotor disk πR^2, rotor solidity is defined as

$$\sigma = \frac{Nc}{\pi R} , \tag{4.25}$$

where N is the rotor's number of blades, c is the chord, and R is the blade's radius. The *Tarot* quadrotor at CfAR, which uses *CFProp 15x5R T-motor* blades with the corresponding values of $N = 2$, $c = 0.035m$, and $R = 0.1905m$, served as the basis for the values used. This gives a rotor solidity value of $\sigma = 0.117$ which is within the range mentioned in [48]: $0.07 < \sigma < 0.12$.

The rotor disk area A is calculated as follows

$$A = n\pi R^2 \tag{4.26}$$

where n is the number of rotors and R is the rotor's blade radius.

The disk loading is calculated by dividing the total weight of the aircraft by the rotor area

$$DL = \frac{W}{A} . \tag{4.27}$$

The drag coefficient C_d and the air density ρ are imported from the configuration file and are set by the user prior to the running of the algorithm.

4.2 Final Cost Function

The final cost function should fulfill a number of requirements, namely :

- **fast** to compute ;
- **represents well** how good a path is, *i.e.* a more adequate path will have a cost closer to 0 ;
- **clearly distinguishes** between a good and a bad path and **penalizes** certain aspects of a path accordingly ;
- **well defined**, between 0 and 1 .

As studied before, several components can be used to evaluate a path, the ones implemented are the following :

- **Smoothness** ;
- **Smoothness with a rotation penalization** ;
- **Time** to complete the trajectory ;
- **Hovering Power** used during the path ;
- **Energy** used to fly the path ;
- **Closeness** to moving objects .

To do this, **16200** environments are created from which **1 062 285**candidate paths were generated . Each environment is created with different parameters, Table 4.1 summarizes the parameters that are modified between each environment and how they vary. The UAV is considered to be always at the origin of the referential ($p_i = (0, 0, 0)$) with a cruise velocity of $5m/s$ ($v_{cruise} = 5m/s$), and the elevation, the size of the threats, and its distance in relation with the the UAV varies. Only the elevation is being changed because, according to the algorithm, it is irrelevant if the obstacle is to the left or right of the UAV, what is relevant is the distance to it and how much the UAV has to climb or descend. The range of the values of the parameters present in table 4.1 were chosen by trial and error, varying each parameter between a feasible (initially larger) range and confirming that there were not any cases where the algorithm would not be suitable to use. For simplification purposes, there are only two objects considered in the environment - a static one (the main threat) and a moving one; however, the algorithm can handle more objects in the environment.

Table 4.1: Parameter variation

Parameter	n	Range min	Range max	Interval
Rotation of the path (θ_i)	13	0 °	180 °	15 °
Distance UAV-Main Threat (D_M)	5	3 m	10 m	1.75 m
Elevation Main Threat (E_M)	9	-7 m	7 m	1.75 m
Main Threat Size (S_M)	5	1 m	3 m	0.5 m
Distance UAV-Moving Threat (D_m)	3	3 m	10 m	3.5 m
Elevation Moving Threat (E_m)	9	-10 m	10 m	3.33 m
Moving Threat Size (S_m)	3	1 m	2 m	0.5 m
Type of path	2	-	-	Symmetric or Simple

For each path generated, all the different types of cost are calculated. This is done to study the variation of each type of path, its maximum and minimum values, and how each parameter affects it when generating new paths. The time taken to calculate a specific type of cost for each path is also registered.

To create the environments, an excel file is generated with all of the possible combinations of the parameters, where in each row a single combination of the different parameters is written that corresponds to one single environment. Then, a script reads each row and generates the configuration/environment files corresponding to that combination, these are the files that will be further described in Section 4.5.3.

4.2.1 Results

The algorithm was run with all of the different environment files, the results being saved to a Comma-Separated Values (CSV) file. This file with the results for each candidate path was then analyzed with

MATLAB [49] to better comprehend the behaviour of the different parameters and so the data can be pre-processed before the parameter tuning is made. For each candidate path, the value of each cost parameter and the time it took to compute is saved, as well as the environment variables for which the candidate path was generated.

4.2.1.1 Parameters Distribution

The figures below show the distribution of the different parameters. It can be observed that all of the different parameters range between different values and have different behaviours in its distribution, which confirms the need for data normalization. It should also be mentioned that some outliers were removed from the data plots (data points that occur less than 100 times) to better show the true range of the different parameters. On the x axis the range of values of the parameters is represented, on the left y axis the frequency of each of these values is represented using a histogram, on the right y axis, in orange, the cumulative percentage of values that are less than the value in the x axis is shown.

It can be observed that both the smoothness (Figure 4.9), the smoothness with penalization (Figure 4.10) and the closeness (Figure 4.11) exhibit similar behaviour - with a greater frequency of values in the lower range, that then diminishes. Similarities can also be observed between the energy (Figure 4.12) and the time (Figure 4.14), that exhibit a distribution more similar to a standard normal distribution. The hovering power (Figure 4.13) exhibits a more atypical behaviour, with all of the data points being concentrated within a certain range and with similar frequencies throughout. This is due to the fact that the hovering power mostly depends on the vertical velocity, which in term depends on the cruise velocity since it is assumed that the UAV is flying at cruise velocity when possible, which was not changed when the algorithm was executed. So, despite having an acceptable average runtime as it will be pointed out in the next subsection, the distribution of its values is not statistically relevant enough to be considered to measure the cost of a path, as it would not allow to distinguish well between different paths, one of the requirements of the final cost function. Thus, **the hovering power will not be used as a parameter for the final cost function**.

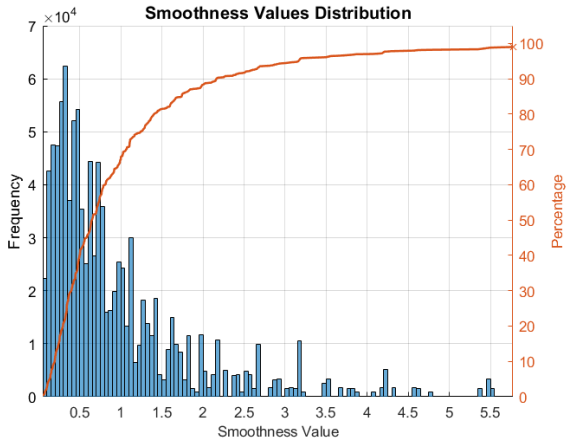


Figure 4.9: Smoothness values Distribution (blue) and cumulative percentage (orange).

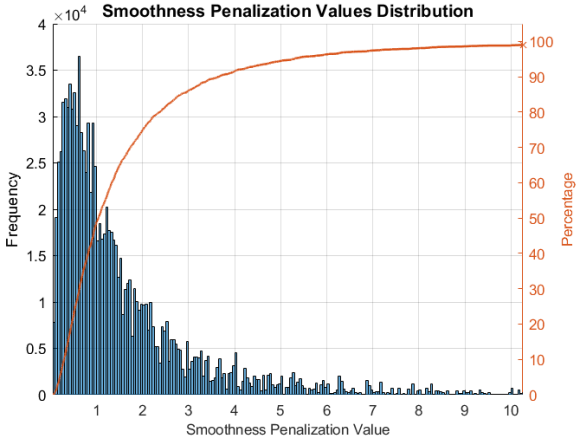


Figure 4.10: Smoothness with penalization values Distribution (blue) and cumulative percentage (orange).

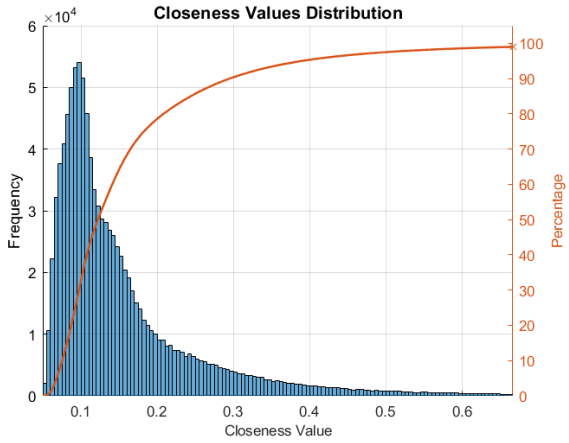


Figure 4.11: Closeness Distribution (blue) and cumulative percentage (orange).

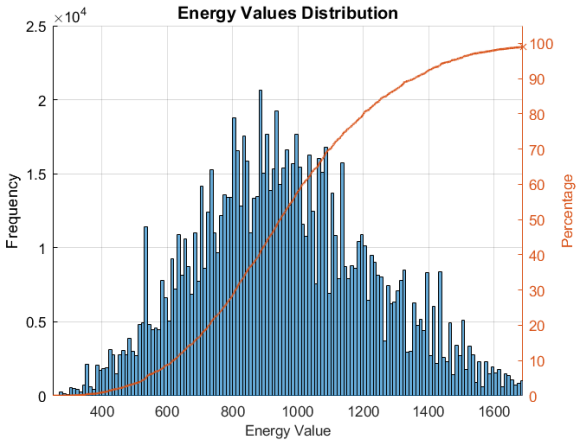


Figure 4.12: Energy values Distribution (blue) and cumulative percentage (orange).

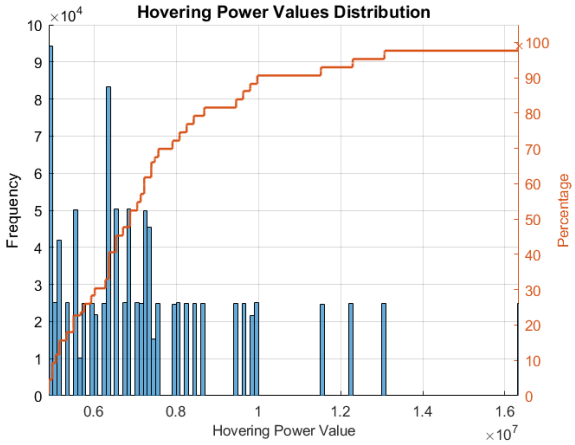


Figure 4.13: Hovering power values Distribution (blue) and cumulative percentage (orange).

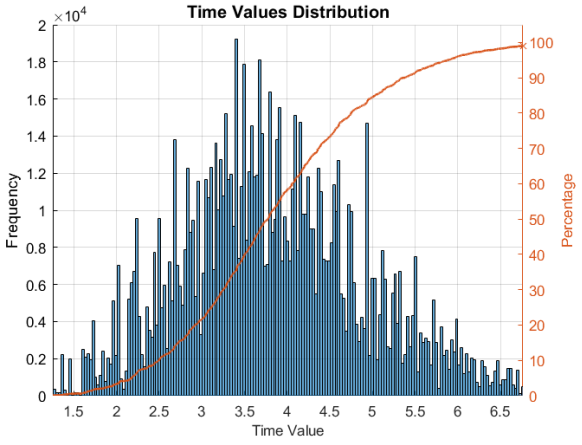


Figure 4.14: Time values Distribution (blue) and cumulative percentage (orange).

4.2.1.2 Runtime results

Figures 4.15, 4.16, 4.17, 4.18,4.19 and 4.20 show the runtime results of the different parameters for each path. On the x axis the runtime for each path in microseconds (μs) is shown, with the y axis being the frequency. For all the parameters, it can be observed that they follow a somewhat normal distribution with a higher frequency of values in a certain range that then tapers out.

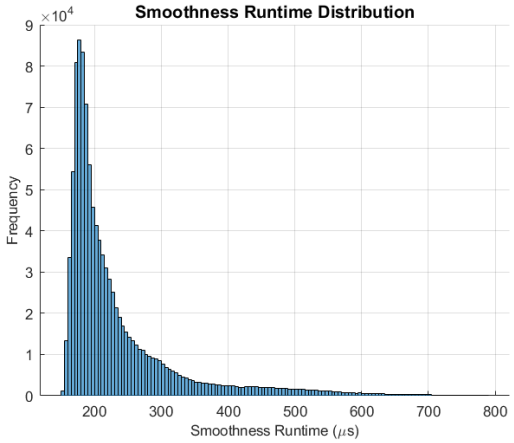


Figure 4.15: Smoothness Runtime distribution.

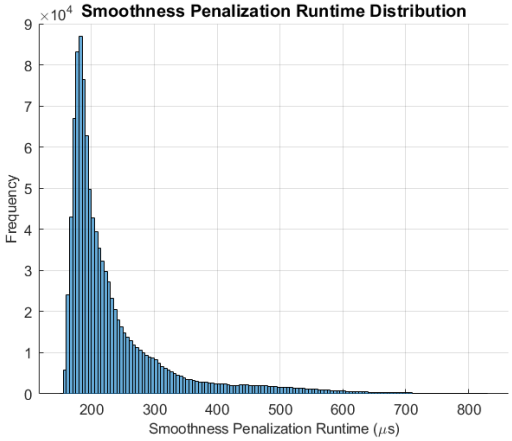


Figure 4.16: Smoothness with penalization Runtime distribution.

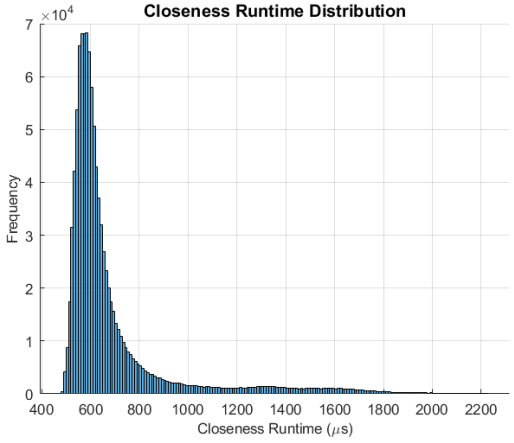


Figure 4.17: Closeness Runtime distribution.

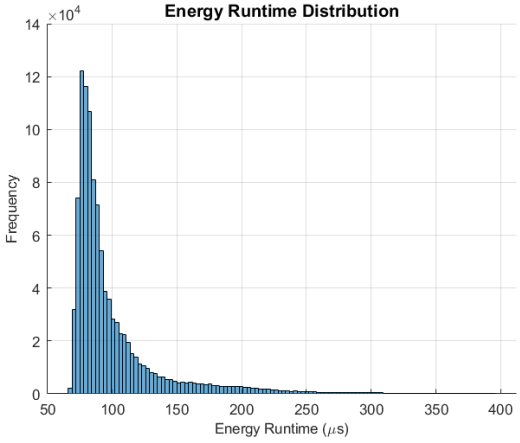


Figure 4.18: Energy Runtime distribution.

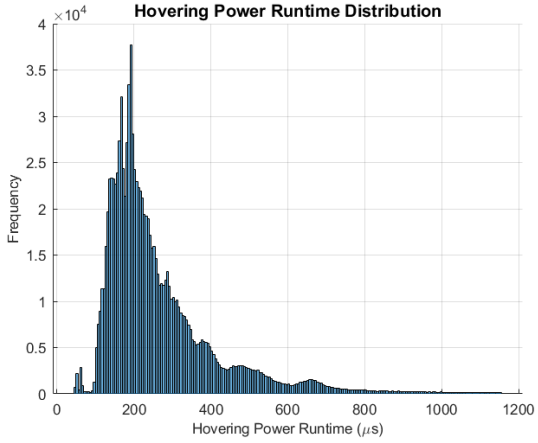


Figure 4.19: Hovering Power Runtime distribution.

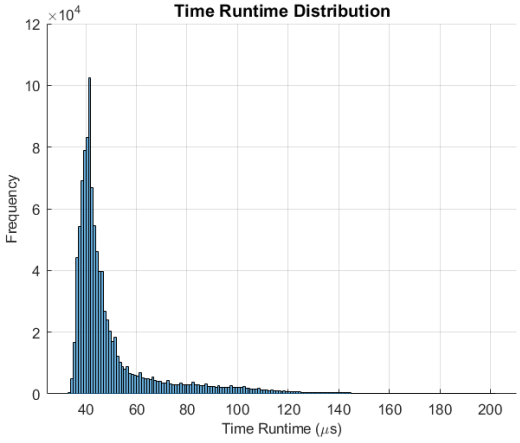


Figure 4.20: Time Runtime distribution.

Figure 4.21 shows the comparison between the runtime distribution of the different cost function parameters. The vertical lines represent the average of each parameter. Using the average as a metric, the closeness with MO is the parameter that takes the most time per path (702.25 μs), followed by the

hovering power (274.78 μs), smoothness with penalization (241.52 μs), smoothness (238.00 μs), energy (103.23 μs) and time (52.488 μs). The closeness with MO takes the most time since it uses all the waypoints of both paths (from the candidate path and the path of the MO) to calculate the closeness to the MO - this could be one of the areas that could be improved in future work (see Section 6.1).Smoothness and smoothness with penalization have very similar values (238μs vs. 241.52 μs) since the smoothness with penalization only adds a small extra step to the calculation of this parameter. Energy and Hovering Power have reasonable runtime since they are calculated with simple equations that are optimized with the use of matrix multiplication.

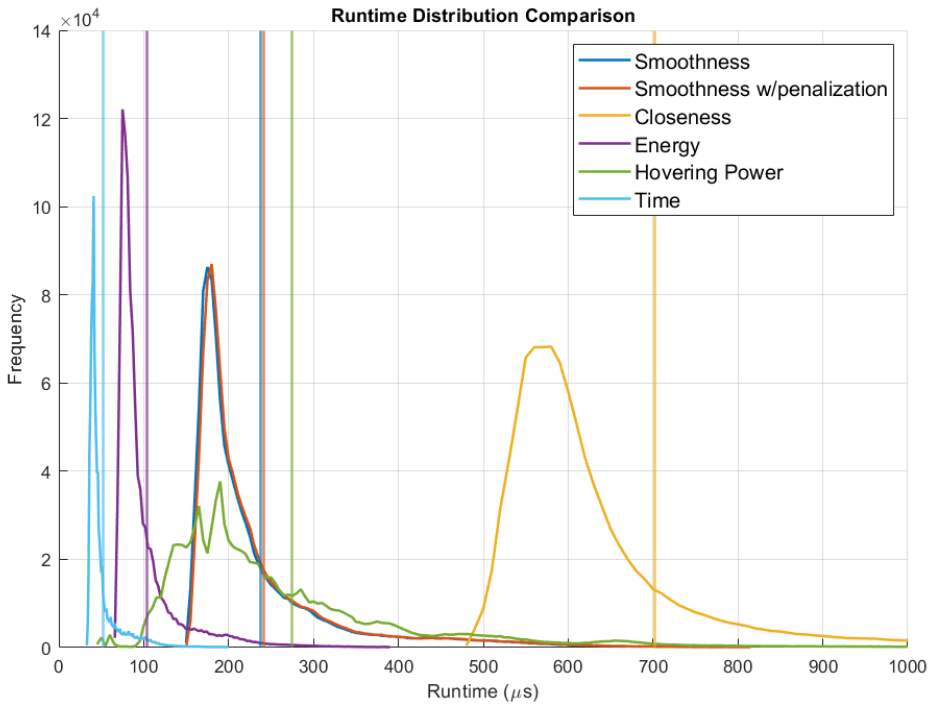


Figure 4.21: Combined runtime graph for the different cost function parameters.

4.2.2 Normalization

4.2.2.1 Time & Energy

Since both the energy and time distributions (Figures 4.12 and 4.14) present a normal distribution across a well-defined range, these parameters will be normalized using **feature clipping** first to remove the outliers and then they will be scaled using the **min-max normalization** to a 0 to 1 range. Due to the big quantity of data points on the dataset and the existence of outliers, the upper cap for feature clipping that will be chosen will be the data point that represents the 99.5% of the data points in cumulative percentage. Table 4.2 shows the minimum and maximum values found in the original dataset (without the removal of the outliers with a count of less than 100 as it was done when plotting the graphs) for the given parameter and the corresponding value for the 99.5% cumulative percentage.

Table 4.2: Upper and lower bounds for feature clipping of the parameters time and energy.

Parameter	Minimum (lower cap)	99.5% (upper cap)	Maximum
Energy (E)	249.59	1759.0	2028.5
Time (T)	1.2530	7.0814	8.9620

It should also be noted that the **time** data points were calculated assuming a cruise velocity of $v_{cruise} = 5m/s$, so before clipping and scaling to range the time data points, the minimum and maximum caps should be scaled as well, since an UAV with higher cruise velocity will take less time to fly the same path. Thus, the minimum and maximum caps must be different for each UAV and should be adjusted. To solve this issue, a offline step is added where these limits are calculated for the given UAV. The steps to implement this are:

1. The candidate path for which the lower and upper cap where calculated are saved as objects using the Python library `pickle` [1] (this is only done once).
2. Offline, these paths are again loaded and the time values are calculated for the new UAV flying the given path.
3. These new values (C_L^t and C_U^t) are saved and will be then used as the new lower and upper caps when normalizing the time parameter.

The candidate paths saved are **path ID 23, environment #1440 for the lower cap** and **path ID 39, environment #11088 for the upper cap**. The environment for the candidate path for the upper cap is summarized in Table 4.3.

Table 4.3: Environment for the upper cap of the energy.

$D_M(m)$	$E_M(m)$	$S_M(m)$	$D_m(m)$	$E_m(m)$	$S_m(m)$
8.25	-1.75	3	3	-10	1

The values for the environment for which the candidate path for the lower cap is generated can be seen in Table 4.4.

Table 4.4: Environment for the lower cap of the energy.

$D_M(m)$	$E_M(m)$	$S_M(m)$	$D_m(m)$	$E_m(m)$	$S_m(m)$
3	0	1	3	-10	1

With the new C_U^t and C_L^t values calculated in an offline phase, the time parameter can be normalized as such

$$x_{new}^t = \begin{cases} 0, & \text{if } x^t < C_L^t \\ 1, & \text{if } x^t > C_U^t \\ \frac{x^t - C_L^t}{C_U^t - C_U^t}, & \text{otherwise} \end{cases} \tag{4.28}$$

Figure 4.22 presents the distribution of values for the time ($T(i)$) after the normalization was implemented for the default values of the UAV. Figure 4.23 presents the distribution of values after normalization was implemented for alternative values of the parameters of the UAV - Table 4.5 presents the values used

[1] Pickle Library, `https://docs.python.org/3/library/pickle.html`, accessed 15/08/2022

in these last simulations. These simulations were run with a smaller dataset than the one used for the original distribution figures for time saving purposes.

Table 4.5: Parameter values for the UAV2.

Parameter	Value
C_D	0.8
A_e	10 m^2
m	10 kg
v_{cruise}	15 m/s
max_{ROC}	8 m/s

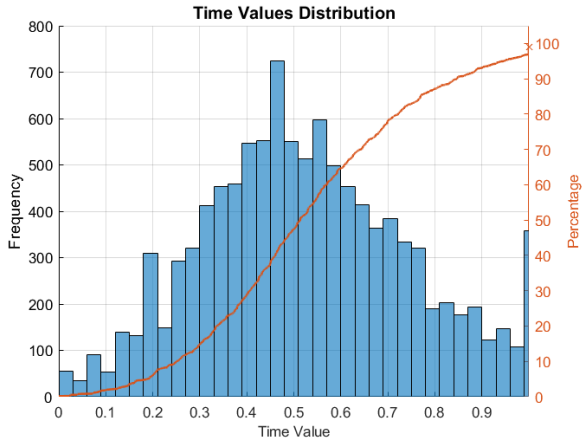


Figure 4.22: Time parameter normalized (UAV1).

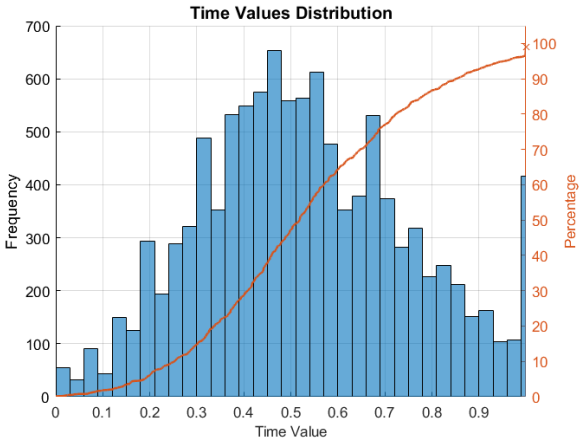


Figure 4.23: Time parameter normalized (UAV2).

Looking at these two images, one of the issues is that there is a high number of candidate paths with $T(i) = 1$ - an explanation for this phenomenon is that the upper cap of the time parameter is too low. To solve this problem, with trial and error, a multiplying factor was introduced to the upper cap in order to make the distribution of values more balanced out. The new distribution with the new multiplying factor ($C^t_{U_{new}} = 1.15 \times C^t_U$) is plotted in Figures 4.24 and 4.25. It can observed that there is no longer a jump in frequency at $T(i) = 1$ and that the distribution continues to be balanced out.

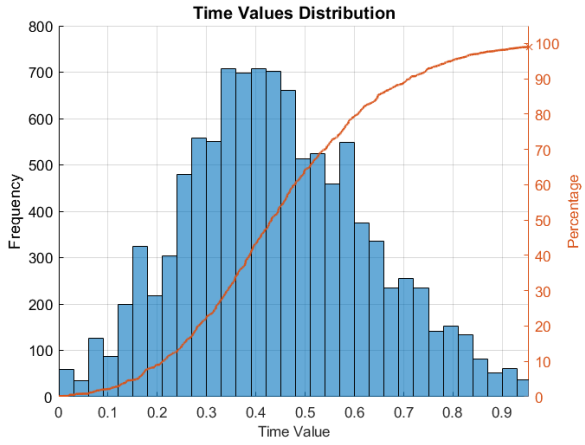


Figure 4.24: Time parameter normalized (UAV1) with 1.15 factor.

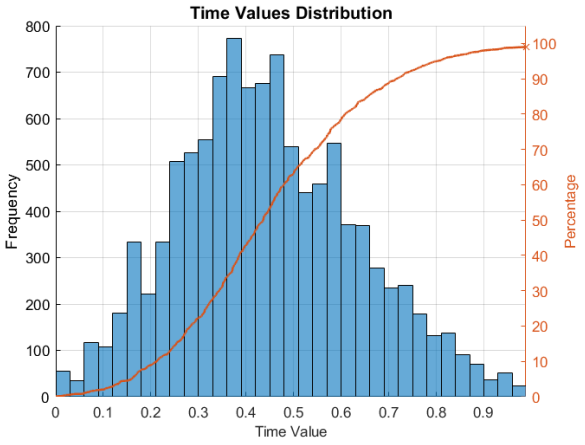


Figure 4.25: Time parameter normalized (UAV2) with 1.15 factor.

For the **energy**, a similar logic can be implemented. The energy has two components: the mechanical energy and the energy due to drag forces. According to Equation (4.2) the mechanical energy depends on the mass (m), absolute velocity square ($||v_i||^2$) - which mainly depends on the cruise velocity (v_{cruise}), on the gravitational acceleration (g) and on the vertical position ($p_{i,z}$). And the drag forces energy, according to Equation (4.3) depends on the drag coefficient (C_D), on the air density (ρ), on the effective area (A_{ef}), on the absolute change in position ($||p_{i+1} - p_i||$) and on the absolute change in velocity squared ($||v_{i+1} + v_i||^2$) - which also mainly depends on the cruise velocity (v_{cruise}). The gravitational acceleration and the air density are constants and the variation in the vertical position and position is already considered to be taken into account by varying the elevation and position of the threats. However, the other variables ($m, C_D, v_{cruise}, A_{ef}$) are UAV dependent and so new lower/upper bounds need to be calculated for different UAVs.

The same method used for calculating the upper and lower caps for the time is implemented for the energy. For the same candidate paths generated for the previously mentioned environments, the energy for each UAV is calculated in an offline phase and saved.

Having the upper and lower caps for the normalization for any given UAV using the upper and lower caps calculated offline, it is now possible to write the normalization equations. Again, the normalization methods used are: **feature clipping** followed by **min-max normalization** to a 0 to 1 range. The new upper and lower caps are referred to as C_U^e and C_L^e. Thus, the energy normalized can be written as:

$$x_{new}^e = \begin{cases} 0, & \text{if } x^e < C_L^e \\ 1, & \text{if } x^e > C_U^e \\ \frac{x^e - C_L^e}{C_U^e - C_L^e}, & \text{otherwise} \end{cases} . \tag{4.29}$$

Figure 4.26 presents the distribution of values after the normalization was implemented for the default values of the UAV. Figure 4.27 presents the distribution of values after normalization was implemented for alternative UAV. These new simulations were again run with a smaller dataset than the one used for the original distribution figures for time saving purposes. This shows that, the distribution of the values is similar with a different UAV and that, although there is a higher frequency of values in the extremes end of the values (0 and 1), this frequency is still reasonable and it can be concluded that the lower and upper minimum and maximum values were correctly implemented.

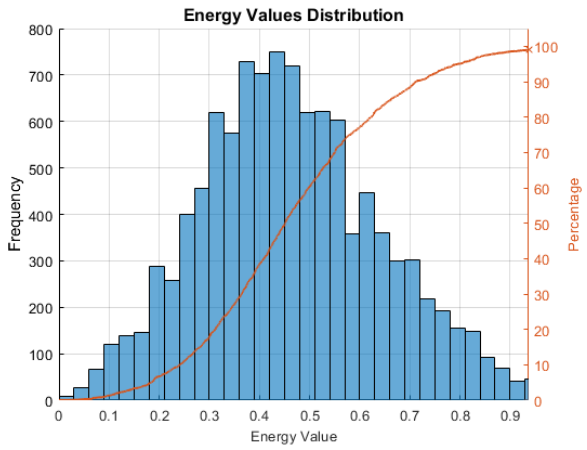


Figure 4.26: Energy parameter normalized (UAV1).

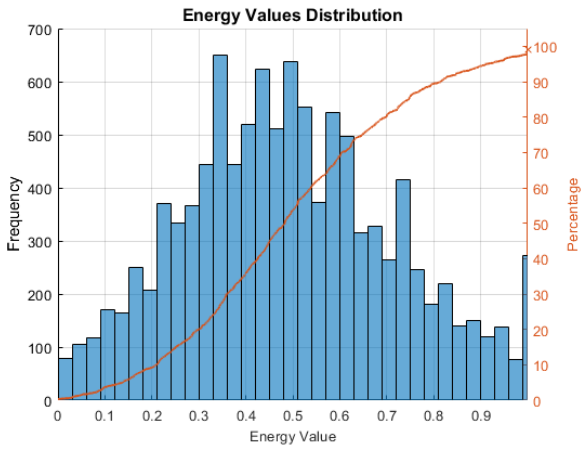


Figure 4.27: Energy parameter normalized (UAV2).

4.2.2.2 Smoothness, Smoothness with penalization & Closeness to Moving Objects

All of the three remaining parameters (Smoothness, Smoothness with penalization and Closeness to MO) exhibit similar behaviour regarding its distribution, with a higher frequency of values focused on the lower range of the distribution that then tappers off - this can be seen in figure 4.9, 4.10 and 4.11. It should be also noted that, contrary to the time and energy, these parameters do not depend on the specifications of the UAV, so there is no need for them to be adjusted depending on the aircraft that is flying the candidate path. Both the smoothness and the smoothness with penalization only depend on the geometry of the path, whereas the closeness with MO depends on the position of the path and on the environment (which threats are in the environment and its position).

With these data from the original dataset results, MATLAB[49] was used to try different normalization methods. **Log scaling** is one of the types that could be used with this type of distribution, since there is handful of values that have many points and the other values have few points. The results of applying log scaling to the data for each parameter can be seen in Figures 4.28, 4.30 and 4.32 for smoothness, smoothness with penalization and closeness with MO respectively. These histograms were then analyzed in order to define the limits for **feature clipping** and for **min-max normalization** since the expected range of each parameter should be between 0 and 1. Table 4.6 shows the limits chosen - these limits were chosen based on analyzing the range of the distribution after log scaling and with trial and error for small adjustments. Figures 4.29, 4.31 and 4.33 shows the results with all of the normalization methods implemented for a smaller dataset - these simulations were run with a smaller dataset than the one used for the original distribution figures for time saving purposes. It can be seen that the results are more well distributed than in the original distribution - this allows for a better distinction between god and bad paths and more separation between paths that may be similar.

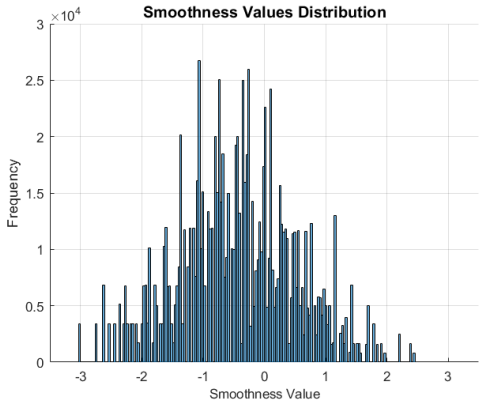


Figure 4.28: Smoothness distribution histogram after log scaling.

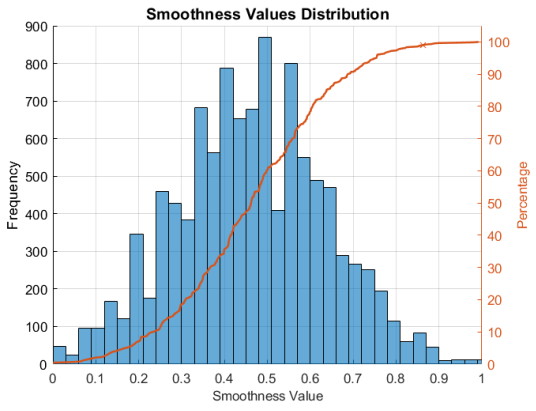


Figure 4.29: Smoothness distribution histogram after log scaling, feature clipping and min-max normalization.

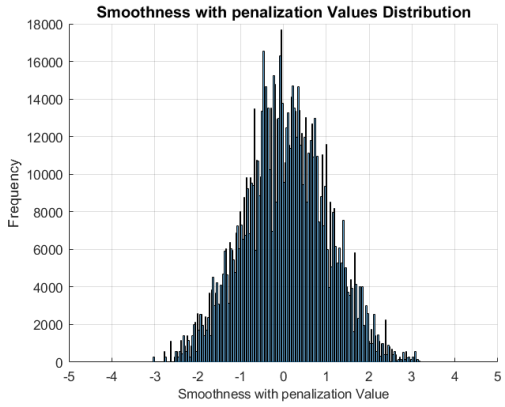


Figure 4.30: Smoothness with penalization distribution histogram after log scaling.

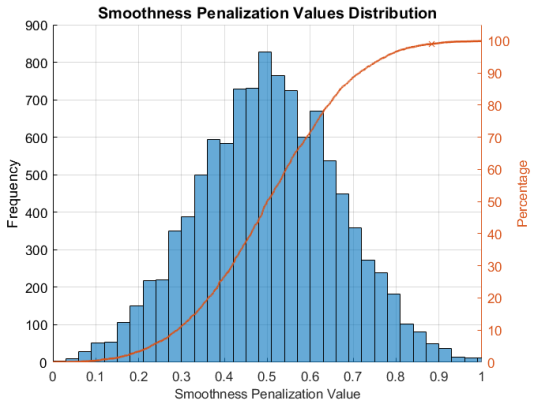


Figure 4.31: Smoothness with penalization distribution histogram after log scaling, feature clipping and min-max normalization.

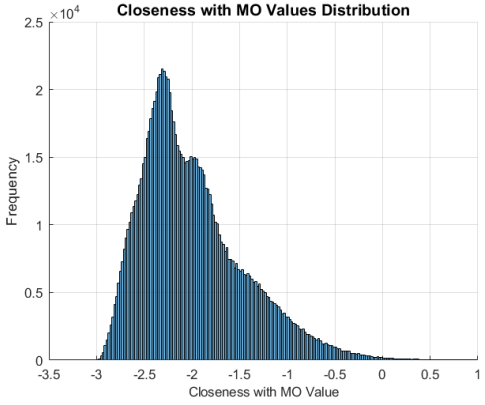


Figure 4.32: Closeness with MO histogram after log scaling.

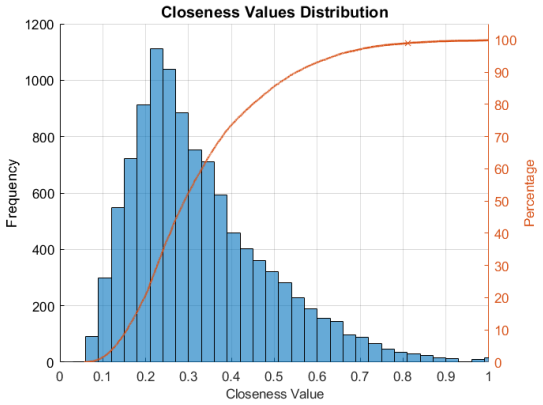


Figure 4.33: Closeness with MO histogram after log scaling, feature clipping and min-max normalization.

Table 4.6: Limits for min max normalization and feature clipping. These limits are also the ones used for feature clipping.

Parameter	x_{min}	x_{max}
Smoothness (S)	-3	2.5
Smoothness w/penalization (S_{rp})	-3	3
Closeness w/MO(M)	-3	0.001

Thus, using equation (2.1) and substituting the x_{max} and x_{min} values for the values present in the table above, the set of equations to normalize and feature clip the data after log scaling for the smoothness can be written as

$$x^s_{new} = \begin{cases} 0, & \text{if } log(x^s) < -3 \\ 1, & \text{if } log(x^s) > 2.5 \\ \frac{log(x)+3}{5.5}, & \text{otherwise .} \end{cases} \tag{4.30}$$

For the smoothness with penalization, the process is similar, but with different limit values. Thus,

$$x_{new}^{sp} = \begin{cases} 0, & \text{if } log(x^{sp}) < -3 \\ 1, & \text{if } log(x^{sp}) > 3 \\ \frac{log(x)+3}{6}, & \text{otherwise .} \end{cases} \tag{4.31}$$

And finally, for the closeness with MO

$$x_{new}^{m} = \begin{cases} 0, & \text{if } log(x^{m}) < -3 \\ 1, & \text{if } log(x^{m}) > 0.001 \\ \frac{x+3}{3.001}, & \text{otherwise .} \end{cases} \tag{4.32}$$

4.3 Results analysis

Understanding how each parameters interacts and varies with each of the different **aspects of the candidate path** is also relevant. By understanding these relations, the final cost function parameters can be chosen so as they do not overlap and are all measuring different aspects of the quality of the path.

The different aspects of the path that are considered are:

- **Geometry of the path** - if the shape of the candidate path influences the calculation of the parameter;
- **Variance in altitude** - if a change in altitude will affect the calculation of the parameter;
- **Imminent danger** - if moving objects and its proximity will affect the calculation of the parameter;
- **Performance while flying the path** - if the performance of the UAV while flying the candidate path is measured by this parameter.

Analyzing Table 4.7, smoothness and smoothness with penalization depend on the shape of the candidate path, but whereas smoothness does not, smoothness with penalization introduces a penalization factor that takes into account the variance in altitude. Time and energy also both measure the performance of the UAV while flying the candidate path and closeness with moving objects is the only one that takes into account threats outside of the main threat.

Table 4.7: Properties of the path - analysis.

Parameter	Geometry of the path	Variance in altitude	Imminent danger	Performance
Smoothness (S)	X			
Smoothness w/penalization(S_{rp})	X	X		
Time (T)				X
Energy (E)				X
Closeness w/MO (M)			X	

Since smoothness with penalization and smoothness both take into account the geometry of the path, only one of them is needed as a final cost function parameter. Smoothness with penalization also considers the variance in altitude and so, the choice between the two is trivial. Moreover, one of the novelties of the path generation algorithm is its expansion to the 3D space and the inclusion of a penalization factor that is related with the change in altitude (the variable that was added when

expanding to 3D space) is relevant to the work developed. It should also be noted that the average runtime between the two parameters is similar, with only a difference of approximately $3.5\mu s$ between the two which is not a difference in performance that is big enough to give preference to the smoothness over the smoothness with penalization.

Closeness with MO is the only cost function parameter that measures the imminent danger associated with the path. For this reason it should be included in the cost function. However, it should also be noted that it has one major drawback - its average runtime of $702.25\mu s$. This is something that it will be needed to be addressed in the future. Nevertheless, there is a need for this cost function parameter in the framework developed. Security is one of the most important factors when flying UAVs, and making sure that the final candidate chosen is as safe as possible is very important. One of the solutions to overcome this problem could be to generate a smaller pool of initial candidate paths, so as the total runtime to choose the best candidate path is also smaller. This has also a drawback, if less candidate paths are being generated, the likelihood of a very good candidate path being chosen is also smaller. Nonetheless, the usage of closeness with MO is crucial and a threshold for the final value of the cost function should be implemented to ensure the quality of the chosen candidate path.

The final choice that needs to be done is between the time and energy parameters. Although both parameters measure the performance of the UAV while flying the path, they measure it in different ways and may or may not be strongly correlated. To test this hypothesis, for a smaller pool of the 16200 environments used when studying the cost function parameters and its distribution, different paths will be generated and the time and energy outputs will be recorded (these outputs are already normalized). The following subsection discusses and analyzes the results.

4.3.1 Correlation Analysis between Time & Energy

Figure 4.34 shows one of the plots used to analyze the correlation between the two parameters. The x axis represents the ID of the path generated, the y axis the value of the parameter (the magnitude of each variable), the blue dots represent the energy values and the orange ones the time values. It can be seen that there is a strong relation between energy and time, with both parameters following a similar pattern with the different path IDs. It should also be mentioned that the pattern visualized is due to the fact that the dataset used is not in a random order, and the variables in the environment files that are used to generate the candidate paths are being increasingly incremented as the environment ID (and consequently the path ID) increments.

To better study the correlation between these two parameters, Figure 4.35 was generated with MATLAB. The data used was the data pertained to 10648 paths that were generated from a smaller number of environments than the original dataset used in Section 4.2 - this was done to save computational time, since, from other plots, it is already established that the same relations between the different parameters are maintained with this smaller dataset. This figure shows the correlation graph between energy and time; the x axis represents the energy values and the y axis the time values. As described in Section 2.2.2, the behaviour observed in this plot (the linear relationship between the two variables) is typical of the behaviour presented between two variables that are strongly correlated. The fitted line that describes the relation between the energy (E) and time(T) is proof of that and can be written as

$$T = 1.0013E + 0.0060\,, \tag{4.33}$$

this equation, that was calculated with the least squares method, shows the clear linear relation between the two variables, with a slope very close to 1 ($m = 1.0013$) and the line intercepting the origin very close to 0 as well ($b = 0.0060$) The correlation coefficient is $corr(E, T) = 0.9120$. As mentioned in Section

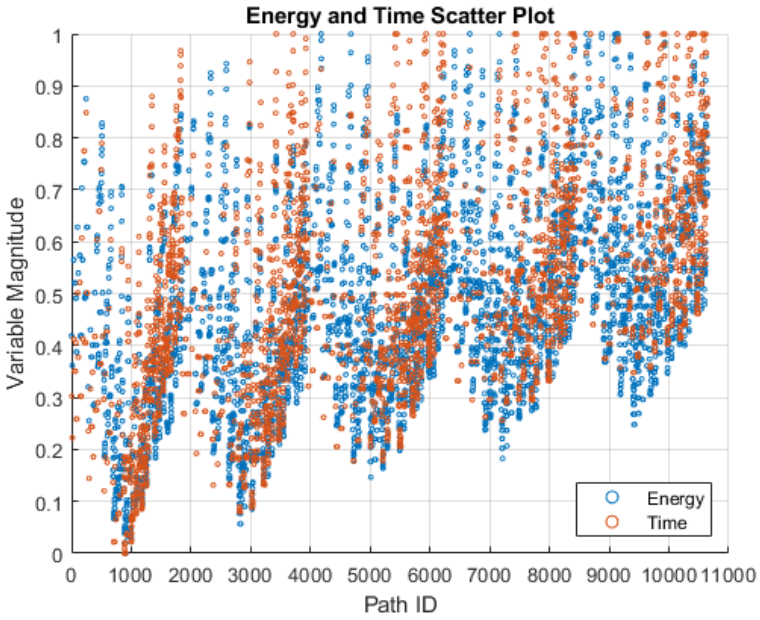


Figure 4.34: Energy and Time scatter plot with path ID.

2.2.2, the closer a correlation coefficient is to 1, the more positively correlated these two variables are (as an example, $corr(T, T) = 1$).

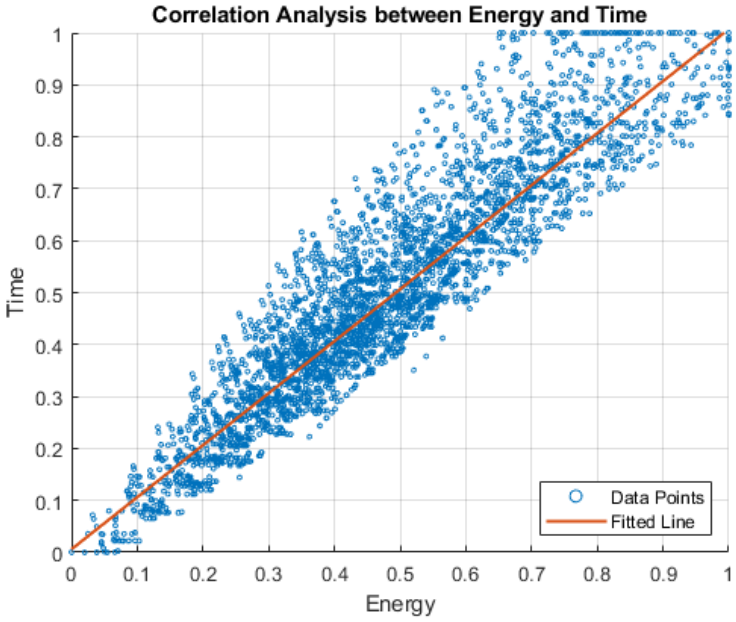


Figure 4.35: Energy and Time Correlation Analysis.

It can be concluded that there is a strong correlation between time and energy. For this reason, it does not make sense to use them both as cost function parameters since they would be measuring very similar attributes of the path. From a theoretical stand point these results can be easily justified - if a UAV is taking more time to fly a certain path it will inherently spend more energy to do so and vice versa. Nonetheless, the fact that the correlation analysis backs up this, is a testament to the validity of the time and energy calculations. It should also be noted that the time parameter has a clear advantage that the energy does not have - the runtime per path ($52.5\mu s$ vs. $103.23\mu s$). Considering all of this, the logical

conclusion is to only use one of them as a cost function parameter, and since the time parameter takes less time to compute it should be the one chosen.

4.4 Implementation

To recap, the final three cost function parameters chosen were:

- Smoothness with penalization,
- Time,
- Closeness with Moving Objects.

The final step of the cost function implementation is choosing the weights to tune the cost function to. The cost function is defined as

$$C(i) = k_S S(i)_{rp} + k_T T(i) + k_C M(i) \tag{4.34}$$

where $C(i)$ is the final value of the cost function for the path i, $S(i)_{rp}$ the smoothness with penalization value for the path i, $T(i)$ the time cost value for the path i, $M(i)$ the closeness with moving objects value for the path i, and k_S, k_T, k_C the weights for the smoothness with penalization, time and closeness with moving objects, respectively.

4.4.1 Weight Tuning

It should be noted that all of the candidate paths that will be evaluated by the cost function are already considered adequate candidate paths that have no collisions and that, baring any unexpected changes, would be safe for the UAV to fly. So, the goal of the weight tuning of the cost function is to choose the most desirable path between the available solutions. What is considered the most adequate path or which factors have more importance can be interpreted differently by each user. In this work, a qualitative approach was taken; however, this can be argued and changed by future users. Additionally, a quantitative approach can be explored in future work, with more set and defined constraints.

To tune the weights for the cost function, firstly the importance of each parameter, from a theoretical point of view, should be established. **Closeness to MO** is the most important parameter - the security of the aircraft is always the top priority and this parameter is the one that best measures this, so k_C should be the highest weight. The second highest weight should be the **smoothness with rotation penalization** - this parameter measures the ”ease” that the UAV has while flying the given path, an easier path to fly is a better path to fly since the UAV will waste less resources doing so. This parameter also takes into account altitude changes, which influences the effort that it takes for the aircraft to fly the path. Finally, k_T should be the lowest - although **time** is an important factor, it is not a top priority; moreover, any loss of time sustained while flying a slower path, can then be made up when the aircraft re-enters its original trajectory by increasing velocity. In conclusion, $k_C > k_S > k_T$. It should also be noted that $k_S + k_T + k_C = 1$ is a restriction of the problem, since each parameter was normalized between 0 and 1 and one of the requirements of the final cost function is for it to be defined between 0 and 1 as well.

In order for the closeness to MO to not be too dominant, a limit was introduced and $k_C \leq 0.70$. A minimum limit was also established, and $k_T \geq 0.10$ for the opposite reasons. Table 4.8 summarizes the 7 different weight combinations that will be tested.

Table 4.8: Weight combinations.

#	k_C	k_S	k_T
1	0.70	0.20	0.10
2	0.60	0.30	0.10
3	0.60	0.25	0.15
4	0.50	0.40	0.10
5	0.50	0.30	0.20
6	0.50	0.35	0.15
7	0.40	0.35	0.25

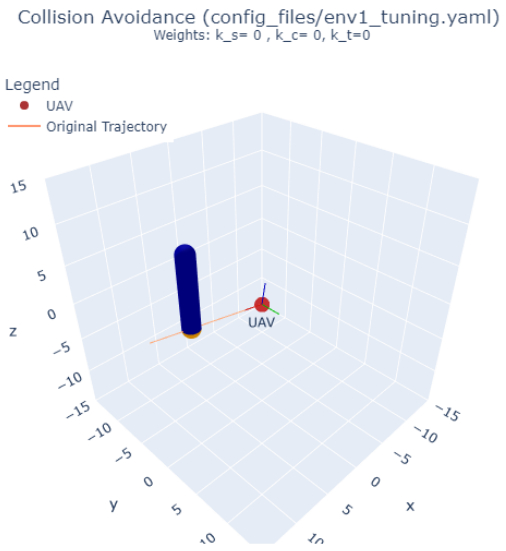
For these 7 combinations, several types of environments will be used; for these environments there are a number of different variables that can be changed, such as:

- Type of the main threat (dynamic or static);
- Size of the main threat;
- Direction that the main threat is moving;
- Number of other threats in the environment (1 to 3);
- Type of other threats (static or dynamic);
- Size of the other threats;
- Direction that the other threats are moving;
- Position of the threats in relation to the UAV;

It is not feasible to test all of the possible combinations, so 4 environments were crafted that will cover a combination of these different variables.

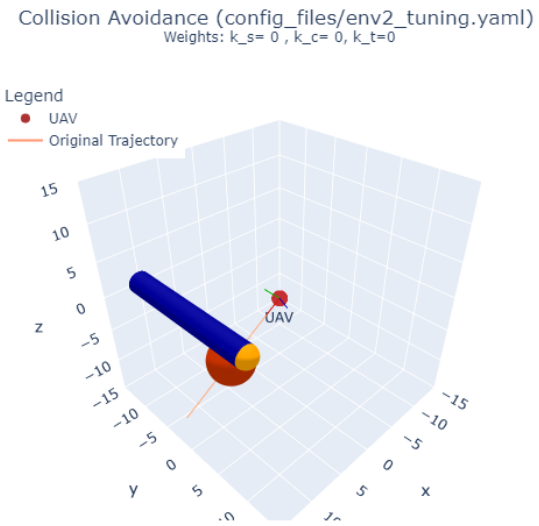
Environment 1

- Main threat size = 1;
- Dynamic ;
- No other threats ;
- UAV 10.2 m from the main threat.



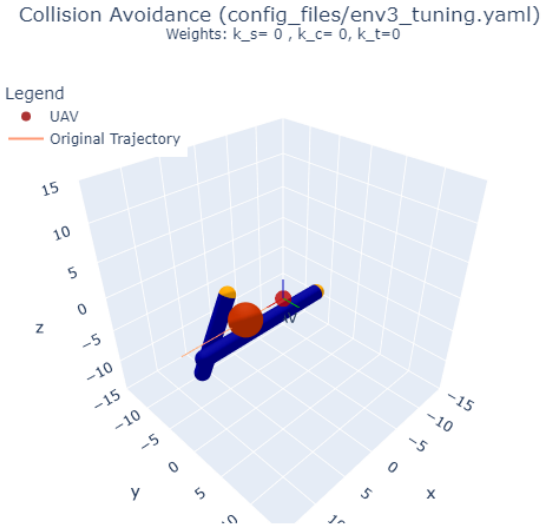
Environment 2

- Main threat size = 3;
- Static ;
- One other threat, dynamic ;
- UAV 9.4 m from the main threat.



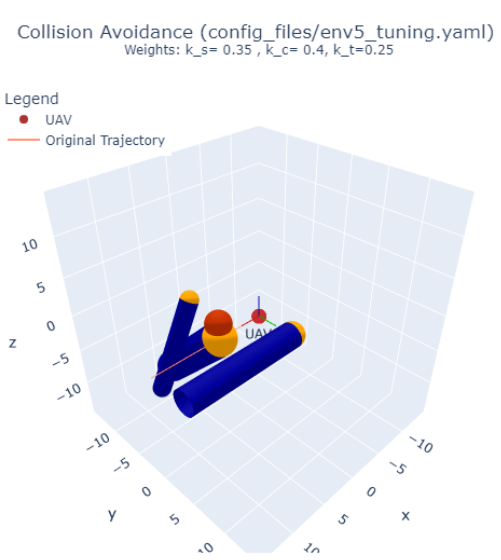
Environment 3

- Main threat size = 2;
- Static ;
- Two other threats, dynamic ;
- UAV 6 m from the main threat.



Environment 4

- Main threat size = 2;
- Dynamic ;
- Three other threat, 2 dynamic, 1 static ;
- UAV 6 m from the main threat.



So, for each weigh combination, the results will be generated for the 4 environments. Each result will be manually checked and qualitatively classified. The chosen path will be compared to the other available paths in aspects such as:

- Overall distance from the threats in the environment;
- How (and if) the collision angle affects the chosen path;
- Path length and curvature;
- Changes in altitude;

Environment 1: As an example, Figure 4.36 shows a zoomed in detail on the solution found when using the weight combination #1 on the environment (this is also the same solution reached with combinations #2 and #3, although with different final cost values). In this situation there are more adequate paths like the one directly on the left of it. This second path is shorter and less curve than the chosen one, while still safely avoiding the threat in the environment. The cost function chooses the other path due to the high importance that is being given to the closeness to MO ($k_C \geq 0.60$) - it can be extrapolated that with $k_C \geq 0.60$ the penalization from the distance to moving threats is too high and that it overshadows the other cost function parameters. The remaining weight combinations choose a path that, with the criteria defined, is more adequate. With these criteria, combinations #1 to #3 are disregarded.

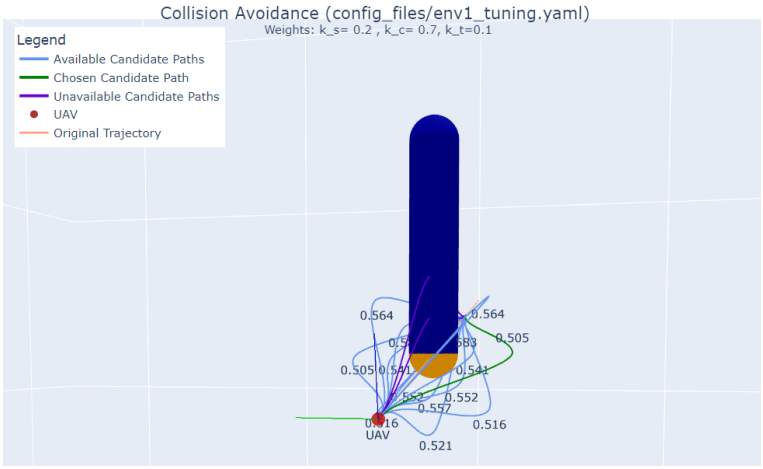
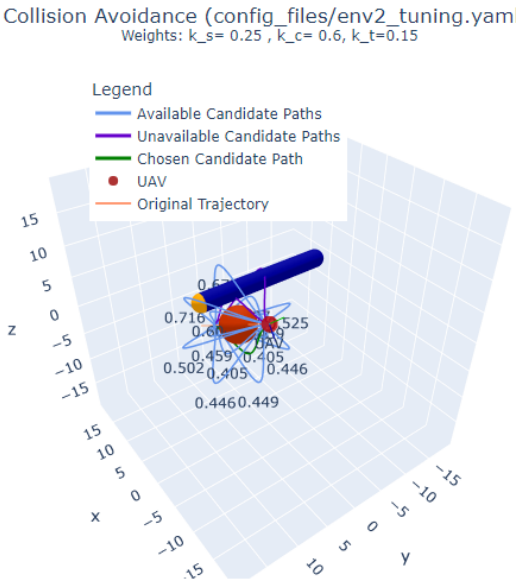
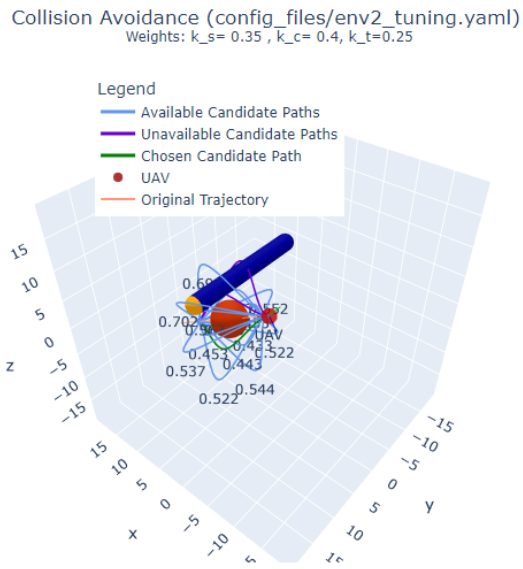


Figure 4.36: Zoomed in detail on the solution reached with Combination #1 on Environment 1.

Environment 2: On environment 2 the main threat is static with a moving object that is flying above the static one, in parallel with the y axis. With the weight combinations used, two different chosen paths are found. The first one, illustrated in Figure 4.37a is the one chosen by combinations #1, #2, #3, #4, whereas the remaining combinations choose the path shown in Figure 4.37b. Due to the direction that the MO is moving, it can be argued that all of the candidate paths in the UAV reference frame or under it are considered safe enough - so the two solutions meet this criteria. When moving from combination #4 to #5, the weight given to closeness with MO is the same, but the importance given to smoothness with penalization decreases (from $k_S = 0.40$ to $k_S = 0.30$) and the importance given to time increases (from $k_T = 0.10$ to $k_T = 0.20$) - which means that the time calculations consider that the path chosen by the second set of combinations is quicker to fly. This is due to the fact that the first path entails a higher rate of descent, and since there is a limit imposed on the rate of the descent (to avoid issues such as Vortex Ring State [50]), the other candidate path is considered to be faster. So, an higher weight to the time is also an important factor and thus $k_T \geq 0.20$. With the limits found when analyzing environment 1 and 2, the remaining combinations that will be considered are combinations #5 and #7.



(a) Chosen path for combinations #1, #2, #3, #4.



(b) Chosen path for combinations #5, #6, #7.

Figure 4.37: Solutions found for the different weight combinations for Environment 2

Environment 3: For Environment 3, the same path is chosen for all of the weight combinations, as it can be seen in Figure 4.38. This is expected since the solution is more straightforward, than the other environments. The available paths on the right of the UAV are similar to the the others in terms of path length and curvature (time and smoothness) but pose an higher danger due to its close proximity to the threat diagonally moving. The chosen path, in green, also does not have any change in altitude and its one of the shortest path, making it an adequate solution for all of the 3 remaining combinations (#5, and #7).

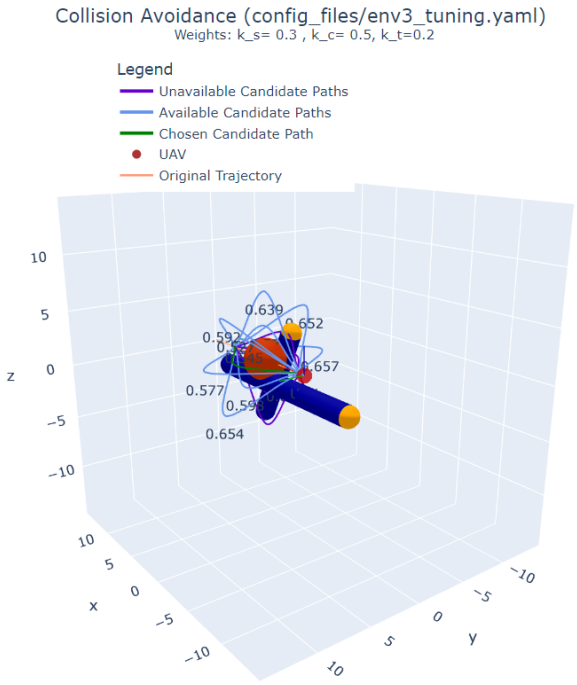
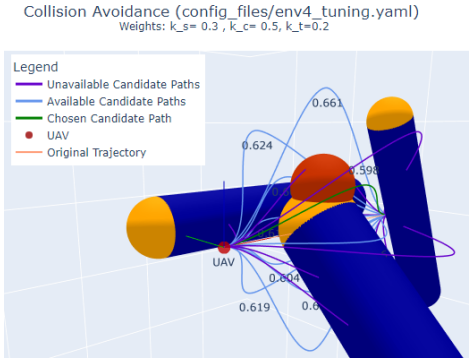
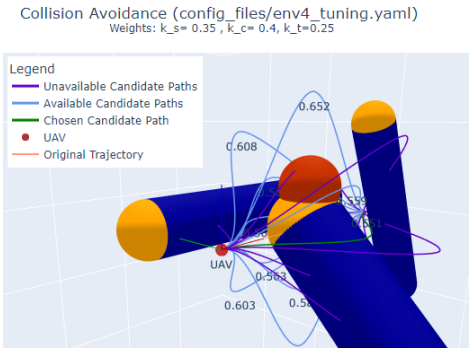


Figure 4.38: Results for the Environment 3 with Combination #5.

Environment 4: Finally, for Environment 4, the two remaining weight combinations find two distinct solutions - combination #5 finds the solution presented on Figure 4.39a and combination #7 finds the solution presented on Figure 4.39a. Both of the solutions are valid; however, due to the collision angle and the lower importance given to closeness with MO, the path chosen by combination #7 is closer to the MO (reminder that this is still a safe path, since for collision detection uncertainty is taken into account). In this case, giving more importance to closeness with MO was prioritized and so combination #5 was chosen as the most adequate one. Combination #5 is also the one that has the most balanced weights (*i.e.* there is not a huge disparity between each weight (like 0.50 and 0.15 in combination #6) or there are not any weights that are too close (like 0.40 and 0.35 on combination #7)).



(a) Zoomed in detail on the chosen path for combination #5.



(b) Zoomed in detail on the chosen path for combination #7.

Figure 4.39: Solutions found for the different weight combinations for Environment 4

Thus, **the final cost function can be written as**:

$$C(i) = 0.30S(i)_{rp} + 0.20T(i) + 0.50M(i) \tag{4.35}$$

4.5 Code Architecture

In this section the architecture of the code will be explained. First, the focus is given to the overall architecture of the code. Secondly, the main classes of the program will be discussed, which variables are associated with them and some of the main methods will be explained in more detail. Next the configuration files used to set different variables and load the environment and the objects for the algorithm to calculate and choose the paths will be discussed.

4.5.1 Overall review

The code can be divided into 4 sections: Initialization, Path Generation, Collision Detection and Cost Calculation. A flowchart that describes the code can be seen in Figure 4.40.

The first step is the **Initialization** - in this step the configuration files are read and the environment is created. This includes creating the instances of the threats that are present as objects of the type `threat`, initializing the UAV as an object of the type `UAV` and defining the equation of the original trajectory of the UAV and the reference frame of the aircraft. This step also includes the calculation of the upper and lower limits that are used to normalize the cost function parameter time (see Section 4.2.2.1). Ideally most of this section would be done offline, before the UAV flies and the values and variables would be stored and used when needed. Of course that the initialization of the threats cannot be done offline since they are suppose to be sudden and thus, in a realistic scenario, are initialized just before the next section - path generation.

In the **path generation** section, the candidate paths are created. Firstly, the distance to the main threat that is in collision route with the UAV is calculated. If this distance is less than the minimum security distance defined ($2.5 \times R_{threat}$) it is not possible to calculate candidate paths and the code is exited (see Section 3.1.1). Otherwise, the code moves to the next step - generating the candidate paths in the UAV reference frame. These paths only have x and y coordinates on the UAV reference frame associated with them (the z coordinate is always equal to 0), and thus need to be expanded to three dimensions. This is done by rotating the candidate paths by a rotation axis in order to cover the 3D space and then, with a transformation matrix, transform the coordinates into the global reference frame (see Section 3.1.3). When the candidate paths are defined, the velocity for each waypoint is calculated which allows for the calculation of the maximum time - the maximum time that the UAV takes to fly any of the given candidate paths. This variable is used in the collision detection section.

The next section, **collision detection**, will check if there is any collision between the candidate paths and the threats present in the environment. This collision detection is performed individually for each candidate path so. For each candidate path, each threat is checked for collision individually as well. Firstly, it is checked if the threat is static or dynamic. For a static threat, the static security will be checked and for a dynamic threat the dynamic security will be checked (see Sections 3.4.1 and 3.4.2). In both of these securities, if there is a collision the path is deemed to be unavailable and its cost will not be calculated in the future. If there is no collision, the path is marked as available. Once the security for all of the candidate paths and all of the threats is checked, the code flows to the next section.

The final section, **cost calculation**, is where the cost function is calculated for each candidate path in order to choose the best one. So, for each available candidate path, each cost parameter (smoothness with penalization, closeness to MO and time) is calculated and normalized. These cost parameters are then added with the weights that are chosen when weight tuning (equation (4.35)) . Finally, the candidate path with the lowest cost is chosen. The available path status is maintained for the paths that were not selected but for which there were no collisions.

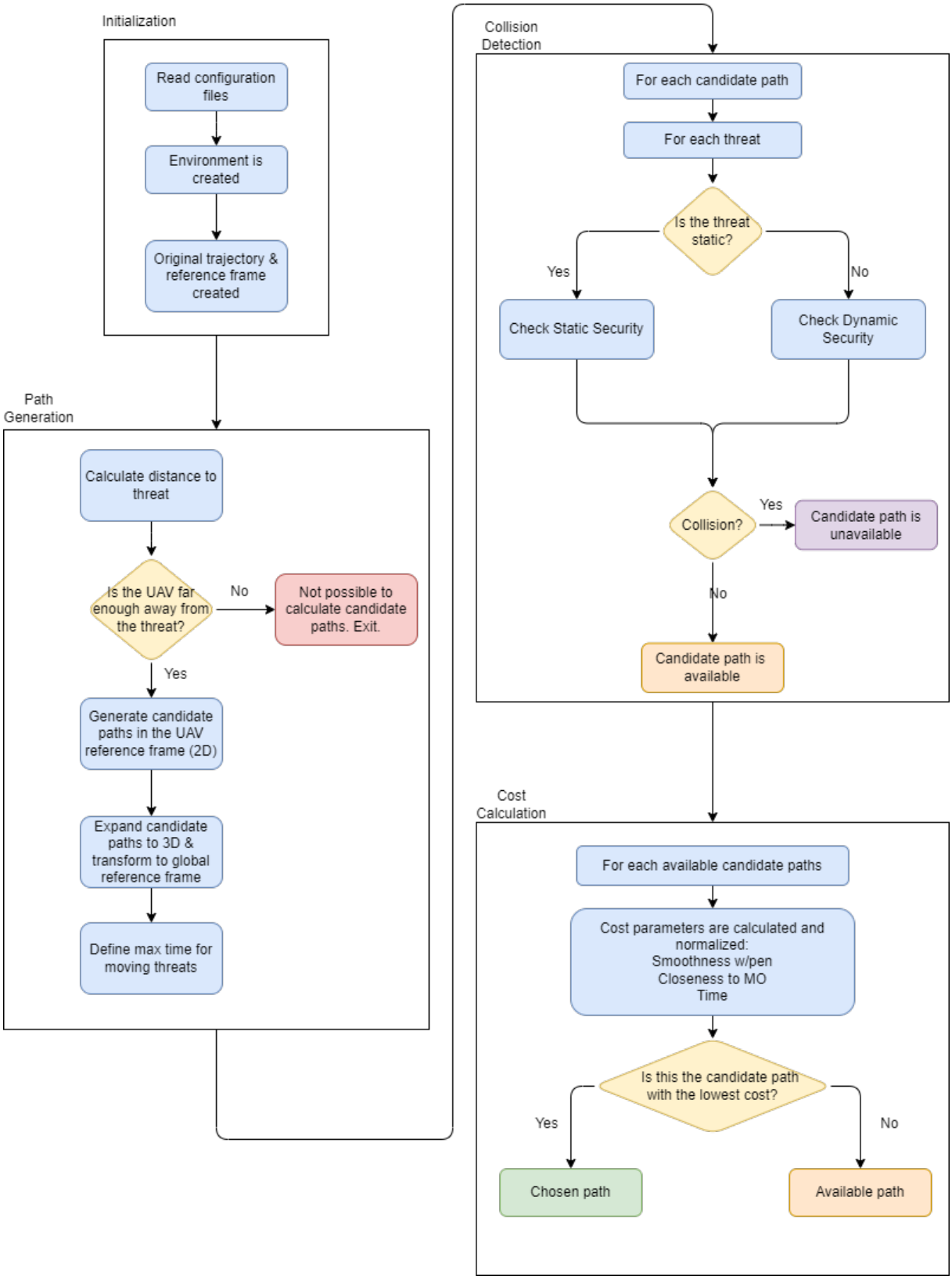


Figure 4.40: Code flowchart.

4.5.2 Main Classes

There are 3 main classes used in the implementation of the algorithm developed: `UAV`, `threat` and `candidate_path`. A fourth class `reference_system` defines the UAV reference frame and handles the transformations between itself and the global frame and target planes.

4.5.2.1 UAV

The `UAV` class represents an UAV that is present in the environment and has a known position (x, y, z: `x,y,z`) in meters (m), a known velocity (v_x, v_y, v_z: `vel_x, vel_y, vel_z`) in meters per second (m/s), an effective area (A_{ef}: `effective_area`) in squared meters (m^2), a drag coefficient (C_D: `drag_coefficient`), a mass (m: `mass`) in kilograms (kg), the number of blades (`N`), the chord of the blades (`c`) in m, the radius of the blades (`R`) in meters, the number of rotors (`n`), the max ROC (`max_ROC`) in m/s, the current time (t: `t`) in s, and the `caps` dictionary that includes the lower and upper limits for the normalization of the time and energy parameters. Since the hovering power and the energy were not chosen as a cost function parameter, some of these variables are not used in the final version of the code (`R,N,n`, etc.), nonetheless, they are still available in case the cost function parameters need to be changed. Most of the information to initialize an object of this class is set by reading a configuration file, that will be discussed in Section 4.5.3.

Regarding some of the methods in this class, these include methods to calculate certain variables (rotor solidity, rotor disk area, disk loading, absolute velocity), methods to calculate certain relationships with other objects in the environment (calculate distance to a point or the direction of the aircraft) and a method to plot the UAV.

4.5.2.2 Threat

The `threat` class represents a threat that is present in the environment that the UAV has to avoid. These threats are modelled as a sphere in 3D with a radius of `size` in meters (m). The `threat` class has the position of the threat (`x, y, z`, in meters m) and the velocity (`vel_x, vel_y, vel_z`, in meters per second m/s) at a given time `t` and the maximum time `max_time` that the threat will be interfering with the UAV. The `threat` also has a `static` property, a boolean, that is set to `True` if all the components of the velocity are 0 and `False` otherwise, as well as a FCL collision object (`collision_object`).

Regarding the methods present in this class, besides updating the position according to the velocity and the current time, there are also methods to plot the threat and the collision paths. The collision paths - the space that the threat will occupy given its velocity and the expected time for the UAV to avoid the main threat - are plotted as cylinders with a radius of the size of the threat, the direction of the velocity vector of the threat and a height of the absolute velocity of the threat times the expected time referenced before (`max_time`). In Figure 4.41 two plotted threats can be seen, one static that is the threat that the UAV will try to avoid in blue, and one dynamic in orange where the collision path is represented by the dark blue cylinder.

4.5.2.3 Candidate Path

The `candidate_path` class represents one of the paths generated by the algorithm to avoid the main threat. The parameters needed to initialize this class are the position array in the 3 axis (x, y, z) and an instance of the class of the UAV that is supposed to fly that path. Other parameters include the value of the different cost parameters (smoothness, smoothness with penalization, energy, closeness to MO, hovering power), the total cost, the runtime that it takes to calculate each of these parameters,

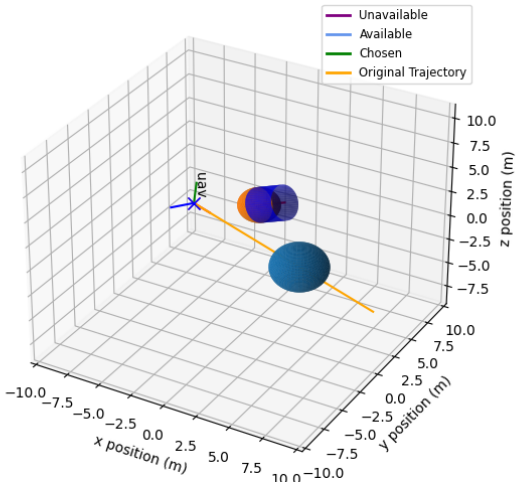


Figure 4.41: Two example threats: blue - static, orange - dynamic.

the velocity vector in each waypoint, the total time that the UAV takes to fly the path, the shape of the path and the angle of elevation. It also includes two boolean variables that define if either the path is in collision or if the path is the chosen one. If the collision boolean is set to False, the path is deemed as available.

Regarding the methods, this class has methods to plot the path according to its availability (purple - unavailable, light blue - available, green - chosen), to define the shape (for smoothness calculation purposes - see Section 4.1.2), to define the velocities in each waypoint (see Section 3.3) and to calculate the total path time (see Section 4.1.5).

4.5.3 Configuration Files

The work uses 3 different configuration files: one general one (named `config.yaml`), one that describes the environment, and one that defines the UAV. All of the configuration file are of the type YAML. A YAML file was chosen since it is both easily readable for a human as well as a machine, making it a popular option for configuration files. Also, there are already libraries in Python that allow for the easy parsing of the data from this type of file, namely `pyyaml` [2] that was used in this work. When parsing from the configuration file it is capable of knowing if the variable is a string, boolean, integer or float.

The main configuration file is named `config.yaml` and has information about the collision radius used for uncertainty when collision detecting (see Section 3.2), the size of the plotted graph (`map: size`), the type of method used to calculated the smoothness (see Section 4.1.2), the weights used for the final cost function, the number of target planes used when expanding to 3D (see Section 3.1.3), the air density and gravitational acceleration values and the location of the other two configuration files.

There are also two other types of configuration file: **UAV configuration file** and **environment configuration file**, both YAML files as well. The **UAV configuration file** contains the information needed to initialize an instance of the class `UAV` as mentioned in 4.5.2.1. This way, it is possible to run and compare how the algorithm performs with different types of UAVs in the same environment. The other configuration file, the **environment configuration file**, contains information about the objects present in the environment, its type (mainly the objects are threats - 4.5.2.2), position (`x,y,z`; floats, in meters m), velocity (`vel_x, vel_y, vel_z`; floats, in m/s) and size (float, in m). So, it is possible to test the performance of the algorithm in different environments and compare the results.

[2] PyYAML library, `https://pypi.org/project/PyYAML/`,accessed 12/06/2022

Chapter 5

Evaluation - Results

In this chapter, the framework developed is evaluated and validated on simulated scenarios, a parametric study is performed and the results are compared with previous cases reported in the literature.

5.1 Runtime analysis

The time it takes for the code to execute (runtime) is one of the most important factors when dealing with online collision avoidance - the fastest a solution is found, the better it is. In this section, firstly the results will be presented - both for the runtime of each cost function parameter, as well as for the total runtime. Next, since these results are obtained in a desktop computer, the runtime times will be extrapolated to an onboard flight computer. With this information, the results will be discussed a potential changes that would need to be made to the framework due to the runtime will be suggested.

5.1.1 Results

The runtime is analyzed for 3 different types for environments, that mainly vary in the amount of threats considered in the environment. Thus, the 3 types of environments are:

- Environments with 1 static threat (blue) - 54 different environments, run three times;
- Environments with 2 total threats: 1 static threat and 1 dynamic threat (orange) - 288 different environments, run once ;
- Environments with 3 total threats: 1 static threat and 2 dynamic threats (blue) - 72 different environments, run twice;

Figures 5.1, 5.2, 5.3 and 5.4 show the violin plots that describe the runtime distribution for the three parameters and the total time. The time for the three parameters is expressed in microseconds (μs) whereas the time for the total time is expressed in (s). The parameters runtime corresponds to the time it took to calculate that cost parameter for one candidate path, the total time runtime is the runtime of the total online phase (Path Generation, Collision Detection, Cost Calculation). For these results the values $l = 4$ (variable that controls the number of initially generated candidate paths in the UAV reference frame) and $n = 3$ (variable that controls the number of target planes generated) were used, which leads to, using equation (3.9), to $L = 16$ candidate paths per environment to be evaluated. Table 5.1 presents the minimum, maximum and mean values for each runtime for the different parameters and the total time. Since for the environment with 1 static threat there is no moving obstacles, there are not runtime values for the closeness with MO associated with it.

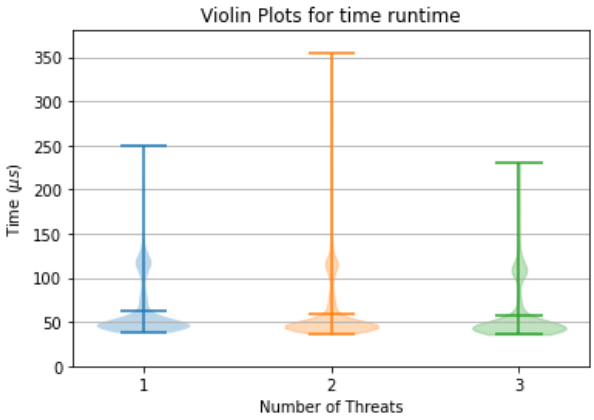


Figure 5.1: Time runtime.

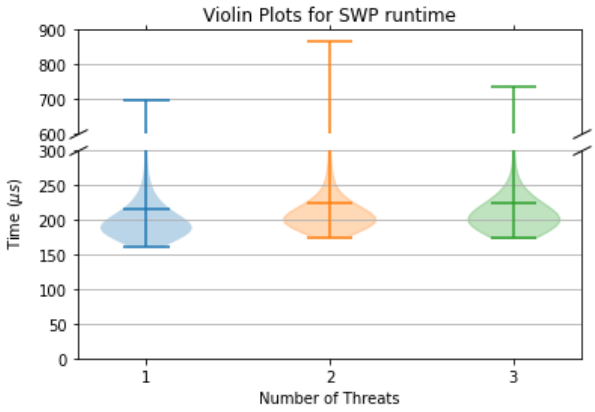


Figure 5.2: Smoothness with penalization runtime.

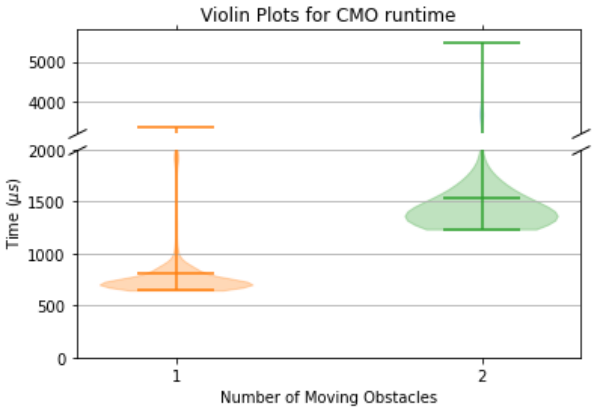


Figure 5.3: Closeness with MO runtime.

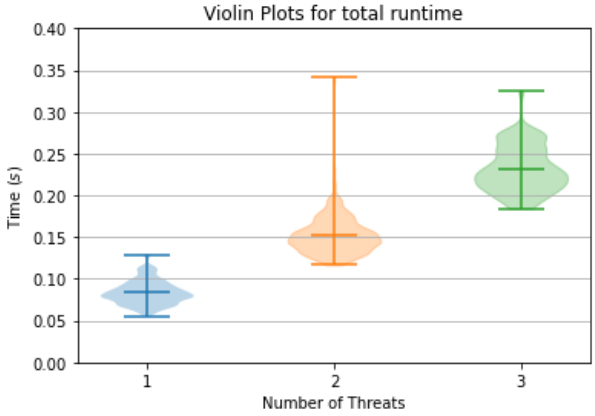


Figure 5.4: Total time runtime .

Table 5.1: Minimum, Maximum and Mean time values for the parameters calculated in different number of threats in the environment. N is the number of threats in the environment: 1 static and $N-1$ dynamic. SWP: Smoothness with Penalization, CMO: Closeness with Moving Obstacles.

Parameter	Min			Max			Mean		
	N=1	N=2	N=3	N=1	N=2	N=3	N=1	N=2	N=3
Time (μs)	37.50	37.20	35.80	249.8	355.0	230.80	62.90	59.54	56.897
SWP (μs)	159.7	173.9	173.5	698.9	867.2	735.6	214.8	222.8	224.0
CMO (μs)	-	646.0	1226.2	-	3347.8	5473.3	-	802.64	1537.4
Total (s)	0.05547	0.1165	0.1842	0.1292	0.3414	0.3247	0.0837	0.1527	0.2319

5.1.2 Desktop results vs. Onboard flight results

The results from the previous subsection were obtained in a computer with the following specifications:

- **Processor:** Intel Xeon CPU E5-1620 v3 @ 3.50 GHz
- **Installed memory (RAM):** 32.0 GB
- **System type:** 64-bit OS, Windows 10

Assuming the use of a Raspberry Pi Model 4B as a flight computer, its specifications are: [1]

[1] `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/`, accessed 15/10/2022

- **Processor:** Quad core Cortex-A72 (ARM v8) @ 1.5GHz
- **Installed memory (RAM):** 1GB to 8GB
- **System type:** Raspberry Pi OS

According to the PassMark Software CPU bench-marking tool [2] , from the desktop CPU to the Raspberry Pi CPU **the loss of performance expected is of 83 %**. The exact CPU specified in the Raspberry Pi specification page was not available in the CPU benchmark website, however, a similar one, the ARM Cortex-A72 6-Core @ 1.5GHz, was used as a comparison. Since the CPU is the main responsible for performance, the comparison between the CPU's will be the one used to extrapolate computational times to the flight computer.

With this loss of performance (83 %), the total time expected for the different number of threats in the environment is presented in Table 5.2 for the total time taken.

Table 5.2: Total time expected performance for the Raspberry Pi Model 4 as a flight computer.

Parameter	Min			Max			Mean		
	N=1	N=2	N=3	N=1	N=2	N=3	N=1	N=2	N=3
Total (s)	0.1015	0.2132	0.3371	0.2364	0.6248	0.5942	0.1532	0.2794	0.4244

5.1.3 Discussion

Looking at the violin plots that describe the runtime distribution of the different cost function parameters (Figures 5.1, 5.2 and 5.3) and of the total time (Figure 5.4), it can be stated that most results are concentrated within a certain range, with some outliers on the upper scale. This difference of computation time within the same parameter was to be expected. The fastest cost function parameter to compute was time (mean of $62.90\mu s$, $59.54\mu s$ and $56.897\mu s$ per path for $N = 1, 2, 3$ respectively), followed by smoothness with penalization (mean of $214.8\mu s$, $222.8\mu s$ and $224.0\mu s$ per path for $N = 1, 2, 3$ respectively) and finally Closeness with MO (mean of $802.64\mu s$ and $1537.4\mu s$ per path for $N = 2, 3$ respectively).

These results also confirm that, along with the number of candidate paths generated (equation (3.9)), the number of threats in the environment is the other factor that greatly influences the total runtime. As expected, when moving from 1 to 2 MOs in the environment, the mean average time approximately doubles (from $802.64\mu s$ to $1537.4\mu s$) - this shows that there needs to be an effort to either optimize the calculation of the Closeness with MOs parameter or to limit the number of MOs considered when calculating the Closeness with MO value. More in-depth discussion on how to implement these improvements will be discussed in section 6.1, but a simple solution would be to either not consider MOs that are more than a certain distance from the UAV when calculating the cost for the Closeness with MO or to only include the 3 closest MOs when calculating this parameter. Since both the smoothness with penalization and time only depend on the path, the number of threats in the environment does not influence its calculations only the number of candidate paths to be evaluated, as it can be seen in Figure 5.2 and 5.1.

In Tavares [41] work the upper time limit that it took for a solution to be found was around $100ms = 0.1s$. Although the restrictions of the problem were different (Tavares work only dealt with one dynamic object in collision route with the the UAV), a comparison can still be made between the two works. The most similar situation that can be compared are the results for $N = 2$: an Environment with 2 total threats

[2]PassMark Software CPU bench-marking tool. Intel Xeon E5-1620 v3 @ 3.50GHz vs ARM Cortex-A72 6 Core 1512 MHz `https://www.cpubenchmark.net/compare/2409vs4679/Intel-Xeon-E5-1620-v3-vs-ARM-Cortex-A72-6-Core-1512-MHz`, accessed 21/10/2022

(1 static and 1 dynamic). In this situation the solution developed takes a minimum of $0.1165s$, maximum of $0.3414s$ and a mean of $0.1527s$ for desktop results and a minimum of $0.2132s$, maximum of $0.6248s$ and a mean of $0.2794s$ for the estimated time for the Raspberry Pi Model 4B. The results presented by Tavares pertain to desktop simulations with the code being written in C++. It is known that Python can be significantly slower than C++ (around 29x slower, according to [51]), so, the work developed here as the potential to be as fast or faster than Tavares' work, if implemented in C++. However, it should be stated that it is not certain, and only an implementation of this work in C++ could compare, with more certainty, the computational times of the two works.

Finally, the effect of the calculation time has to be discussed. With a mean calculation time of $0.1532s$ for $N = 1$, $0.2794s$ for $N = 2$ and $0.4244s$ for $N = 3$, and a cruise velocity of $v_{cruise} = 5m/s$, the UAV would move on average $0.766m$ for $N = 1$, $1.397m$ for $N = 2$, and $2.122m$ for $N = 3$. This shift in position has to be taken into account when generating the candidate path, and the starting point should be adjusted accordingly. One solution could be to shift the starting point $3.75m$ (for $v_{cruise} = 5m/s$, which would leave $0.75s$ for computation time) from the position received when a collision is detected. This shift has to be cruise velocity dependent and adds another restriction to the problem.

5.2 Performance in different Scenarios

To evaluate the performance of the framework developed, 3 scenarios will be created and the solutions found presented and discussed. One of the issues outlined by Tavares [41] in her framework was that her work could not handle collision angles greater than 153 °, so, a scenario with this specification will be created. This scenario, **scenario 1**, will test increasing collision angles to test the solutions presented by the framework in these situations and its limitations. **Scenario 2** will have two threats moving in parallel with the UAV's original trajectory with the main threat also moving in order to reduce the candidate paths availability and test the framework in a more difficult scenario. **Scenario 3** will test the framework's behaviour by having two dynamic obstacles in very close proximity to most of the candidate paths.

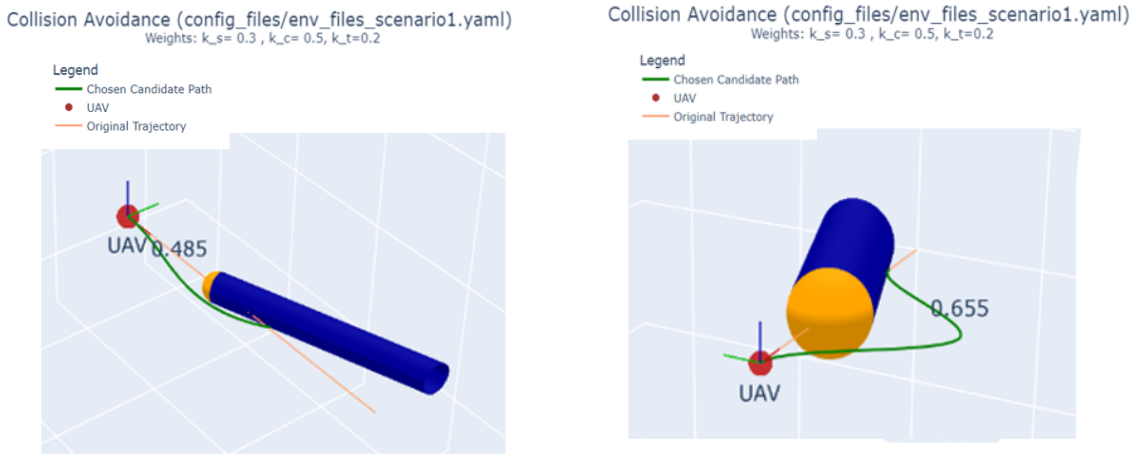
5.2.1 Results & Discussion

Scenario 1

In the first scenario, the effect of the collision angle (the angle from which the main threat is approaching the collision point from) is studied. Figures 5.5a and 5.5b show its behaviour with a collision angle of 160 ° and different threat sizes. In both situations the threat is successfully avoided; however, the candidate path chosen when dealing with the bigger threat is wider than the one chosen with the smaller one since that, with a bigger threat, the less wide candidate path comes in closer proximity to its collision path and so the other one is prioritized. Nonetheless, the framework can handle collision angles bigger than the 153 ° limit of Tavares' work considering reasonably sized threats.

The collision angle was then increased to 174 °. In this variation the framework is no longer capable of finding a solution since all of the candidate paths are unavailable, as it can be seen in Figure 5.6. This happens since that, with a greater collision angle, the collision path of the threat is now also in collision with the original trajectory of the UAV, and since all of the end points of the candidate paths are the same and the candidate path has to be, by definition, in original trajectory, it is not possible to find a suitable solution. However, analyzing the figure below, in real-time, when the UAV is approaching the end point (considering every candidate path), the main threat would be at the collision point (in orange), and so, in reality, there would not be a collision and the path should be suitable. This shows that considering the entirety of the threat's collision paths to increase collision security may have been an over-correction,

and a collision detection method that also considers the threat's trajectory (path with time associated) may be necessary - more detailed discussion on this topic will be presented in the Future Work section - Section 6.1. It is also not possible to define a concrete limit to the collision angle for which the framework can find a solution. Nonetheless, it can be stated that the collision angle limit for smaller threats will be bigger than for bigger threats.



(a) $R_{threat} = 1.$ (b) $R_{threat} = 3.$

Figure 5.5: Solutions found for a collision angle of 160 ° (Scenario 1).

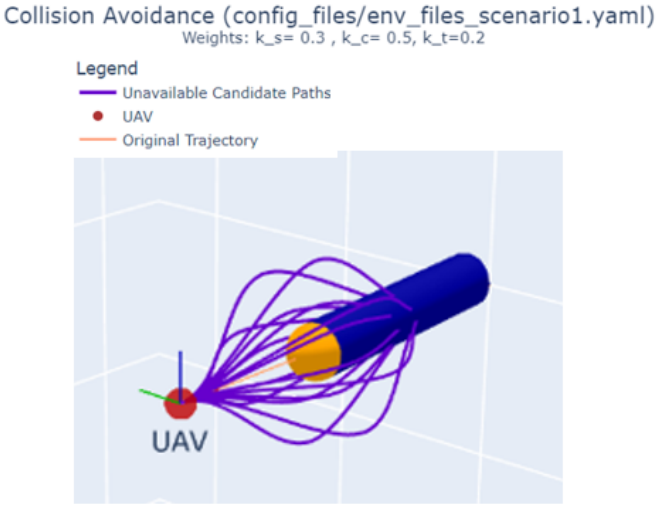
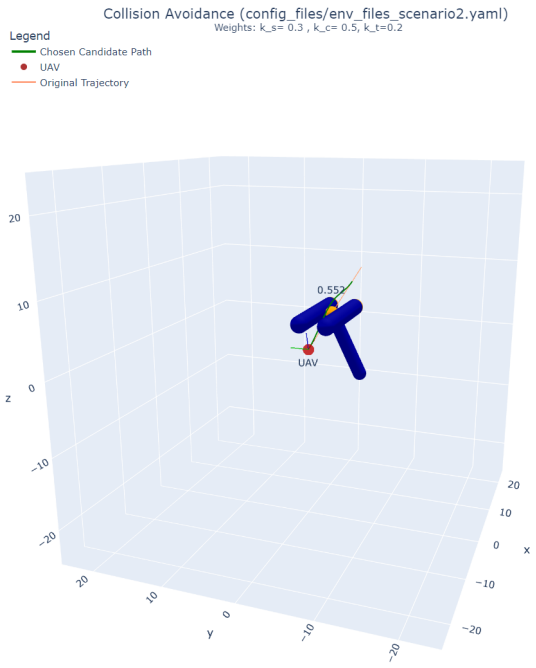


Figure 5.6: $R_{threat} = 1.5$. No solution found for a collision angle of 174 ° (Scenario 1).

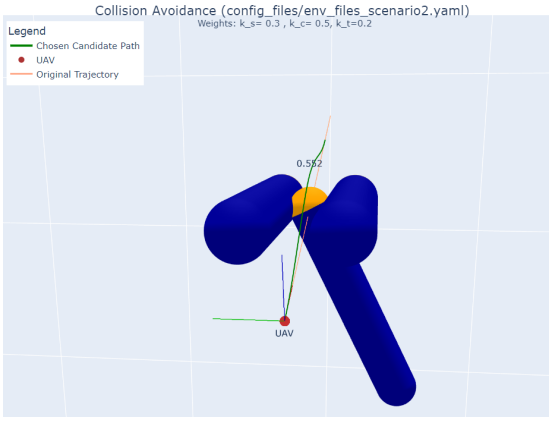
Scenario 2

In the second scenario, the UAV is climbing when a moving threat approaches from below with two other threats moving in parallel with the UAV's original trajectory. Figures 5.7a and 5.7b show the behaviour of the framework in this scenario. Here, the candidate path chosen avoids the threat by climbing over the collision point - this path is chosen over the similar path that, in contrast, descends first and then climbs due to the proximity to the collision path that the UAV would have if chosen. All of the the other candidate paths would collide with the two threats moving in parallel. Analyzing the chosen candidate path, it is evident that there should be some consideration about the collision path of the main threat after the collision point. If there were unexpected variations on the velocity of the

main threat, it could have reached the collision point before expected and may have collided when the UAV was climbing over the collision point. Thus, adding an extra layer of security by either extending the collision path to after the predicted collision point or to also consider the collision angle as a cost function parameter can be an improvement. This improvement can also be argued for scenario 1 when discussing the collision angle of 160 ° and $R_{threat} = 1$ - if the main threat were to be faster than predicted, it could have collided with the chosen candidate path since both the chosen candidate path and the main threat are moving in the same plane ($z = 0$). In this scenario 1, a candidate path that would climb/descend over the collision point may have been safer.



(a) Overall view.

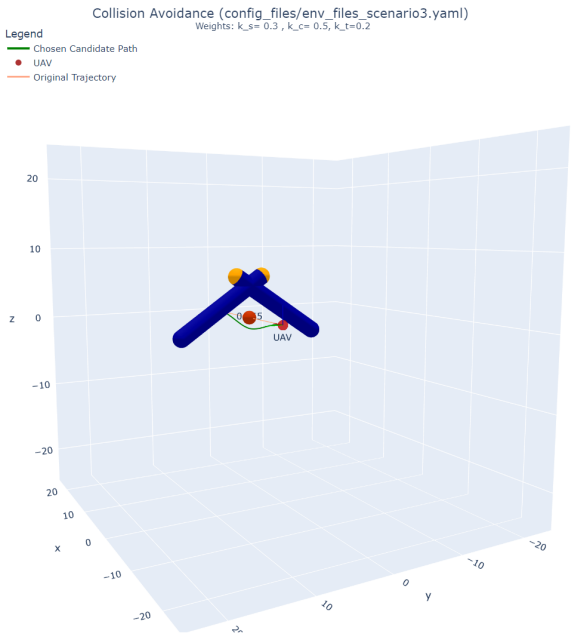


(b) Zoomed in.

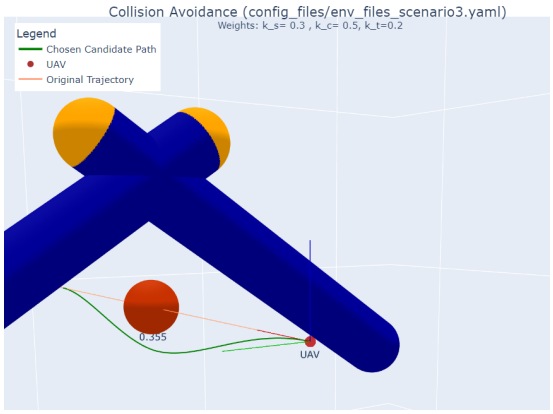
Figure 5.7: Solutions found for Scenario 2.

Scenario 3

In this scenario, there are two threats moving diagonally above the collision point . These threats and its collision paths either collide or are in close proximity to most of the candidate paths. Firstly, the main threat is considered to be static, as it can be seen in Figures 5.8a and 5.8b. In this situation, the solution found shows the prioritization of the Closeness to MO, as the candidate path that descends first and then climbs is chosen due to its distance from the collision paths, despite the higher energy that it takes. This is evidenced when the main threat is changed and now considered dynamic, as it can be seen in Figures 5.9a and 5.9b, the final cost of the chosen path of the first version of scenario is 0.355 and 0.405 in the second version. Here, since it is now not possible to choose the path previously chosen, the chosen one avoids the main threat by going to the left.

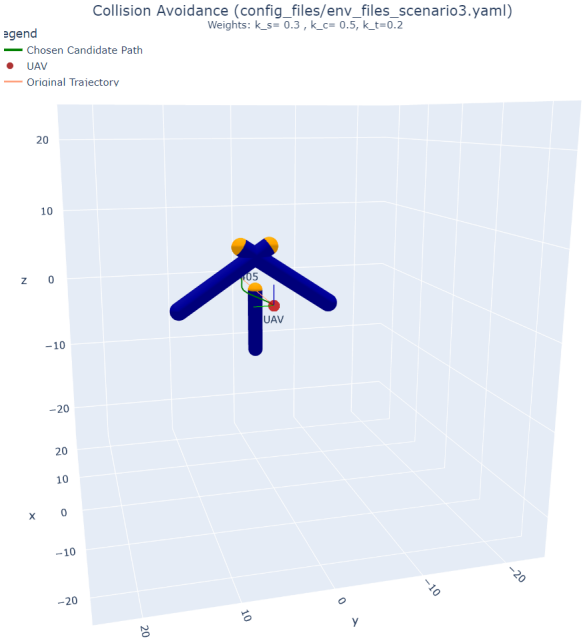


(a) Overall view.

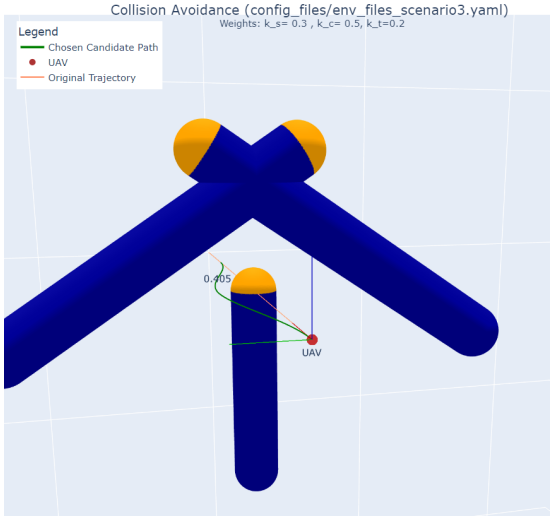


(b) Zoomed in.

Figure 5.8: Solutions found for Scenario 3.



(a) Overall view.



(b) Zoomed in.

Figure 5.9: Solutions found for Scenario 3.

In summary, the framework developed can find solutions for most situations. However, some improvements and adjustments should be made to increase the robustness of the work developed. These adjustments mainly pertain to the collision detection block, by either expanding the collision path considered beyond the collision point or by changing how the collision detection is made - i.e. not considering all of the collision path for collision detection. In addition, the collision angle should also be taken into

account when choosing a candidate path, as it was evidenced by scenario 1 and 2. Furthermore, the biggest impediment to finding a solution is that a threat intersects the candidate paths ending points - one the common attributes between all of the candidate paths. This is what happens when dealing with collision angles close to 180 ° as it was explored in Scenario 1. The work may also fail in a situation where there are more than one collision point, since, by avoiding the first collision point, there may not be enough space and time to calculate and avoid the second.

Chapter 6

Conclusion

In this thesis, a framework was developed that, given adequate information about either a dynamic or static threat that is in collision course with the UAV and other threats present in a 3D environment, generates two types of candidate paths (based on a cubic spline method) capable of safely avoiding the main threat. A candidate path is chosen based on a cost function that was developed to take into account several aspects of the environment and UAV performance parameters. Six parameters were studied and compared in order to find the three most suitable ones (time, SWP and CMO). Next, these parameters were normalized by an analysis of their distribution and weights in the cost function and were tuned using a qualitative approach. The framework is also equipped with collision detection for both static and dynamic obstacles, where the uncertainties associated with the movement of the UAV are considered.

The framework finds a solution most times but fails to find one when the end point of the candidate paths is in collision path with any threat present in the environment (which happens for collision angles $> 160\,°$, but the collision angle limit depends on the size of the threat). In these situations, the UAV is only given a message to stop. Nonetheless, the results obtained are promising, with adequate runtimes that could be greatly improved by porting the framework to C++ or by code optimization, specially for the CMO calculation. Moreover, several modifications still need to be made to make the work reliable, robust and secure. These modifications would still need to be followed by software and hardware-in-the-Loop testing before a potential flight test.

6.1 Future Work

One of the biggest factors in online dynamic avoidance algorithms is the runtime. To improve the runtime of the algorithm, one of the factors that can be improved is the calculation of the parameter closeness MO since it is the one that takes, on average, the most time to calculate. There are a few ways that this could be improved. One of the reasons that it takes a lot of time is that it uses all of the waypoints of the candidate path and of the threat path and calculates the distance between them. Using less waypoints for each path would reduce the runtime of this parameter. Another method could also be used to improve the time efficiency of this parameter - either by using a different library or by taking into account the relative position between the two objects throughout time and calculating the closeness to MO in each time step instead.

Another step that could be taken to improve the runtime of the algorithm is to improve the collision detection process. By checking if, for each threat, there is a collision with any of the candidate paths (instead of the other way around as it is currently implemented) would lead to less collision detection

queries. In addition, for the dynamic security, the collision is being checked for all of the path of the dynamic threat. In case there is accurate information about the trajectory (path with time information) of both objects, the collision detection could be performed with this information, i.e. check if there are collision in each time step, instead of checking for the complete time interval as it stands - there are FCL functions for this purpose. This would also improve the selection of the candidate paths, since paths that in real-time would not collide with an obstacle (but the totality of its path does), would be considered secure.

It should also be mentioned that Python is not known to be the fastest coding language; however some of its strong suits are its simplicity and the high number of libraries available which makes it easier to quickly develop a prototype (one of the goals of this work). A more computer-efficient language, like C++, could be used to obtain better results in terms of runtime. This would also make it easier to integrate with the framework developed by Tavares [41] and Cristóvão [40]. Some of the important libraries used in this work have C++ counterparts, like FCL, so this translation from Python to C++ is not unreasonable. An integration with the work previously developed at CfAR would also make it easier to perform SITL, HITL and eventually flight tests, since some of the libraries and framework to perform these tests are already implemented.

Another improvement that should be added is taking into account the UAV restrictions when generating paths. As it stands, this algorithm does not check for that and assumes that the UAV can fly any of the candidate paths. An extra step after the initial group of candidate paths are generated that checks for the feasibility of the candidate path could be added (for example, checking if the turn rates needed are possible). In addition to this, some of the variables used in this work are currently being estimated with simple methods (like the velocity and the reference frame). Ideally the velocity calculations would be adapted to each type of aircraft and take into account its restrictions and the reference frame should also be taken as an input instead of creating one based only on the direction of the original trajectory. Studying the optimal number of waypoints needed to generate a path is also a topic that can be addressed.

Another restriction of the work is that the original trajectory of the UAV is assumed to be straight. In some situations, this assumption may be unreasonable and the aircraft may be performing a curved trajectory. In that case the end point of the candidate paths generated would not coincide with the original trajectory, and so a correction would need to be done to account for that.

In situations where none of the generated candidate paths are available, two options arise: activate an emergency loop that stops the UAV - in case of a quad-rotor it can easily just hover and wait for the threats to pass - or generate a new batch of candidate paths to try to avoid the threat. As it stands only the first option is implemented - if all of the candidate paths are unavailable, no solution is found and the aircraft is instructed to stop. There are several ways to generate new paths using the same path generation equations. For example, the end point of the candidate paths can be changed, the size of the threat can be considered bigger so the trajectories are wider or the number of target planes when expanding the initial group of candidate paths can be increased. The generation of new paths should be also implemented in the case that a threshold for the cost of chosen path is defined (i.e. a path is only deemed as chosen if its cost is lower than a certain amount). The decision on the value to choose for the threshold would also need to be studied.

Another route that the work developed on this thesis can take is using the cost calculation for optimization purposes and find the overall candidate path with the least cost by using either uninformed searching methods like breadth-first search or depth-first search or informed search methods like greedy search. Although this can be an interesting approach, it would increase the computation time. Moreover, in online avoidance scenarios, the most optimized path is not needed in most cases, just a good enough path to avoid the imminent danger. Nonetheless, if it were to be efficiently implemented, it could be a

valuable added feature.

Additionally, the weight tuning section could also be expanded by taking a more quantitative approach to choose the weights. Although the approach taken here is still valid, an approach more set in numerical data could be proven useful and give another insight into the work developed.

Finally, the improvements discussed in Section 5.1.3 to take into account the expected loss of time while calculating the chosen candidate path could also be implemented, such as, shifting the starting point of the candidate paths. Additionally, considering limiting the number of MOs used for Closeness with MO calculation is another implementation that would improve the runtime - this can be done by either not considering MOs that are more than a certain distance away from the UAV and/or to consider only the x closest MOs for cost calculation purposes since they may not have a real influence on the candidate paths and greatly add to the runtime.

Bibliography

[1] Magdi S. Mahmoud, Mojeed O. Oyedeji, and Yuanqing Xia. “Chapter 10 - Path planning in autonomous aerial vehicles”. In: *Advanced Distributed Consensus for Multiagent Systems*. Academic Press, 2021, pp. 331–362. DOI: 10.1016/B978-0-12-821186-1.00018-0.

[2] Precedence Research. *Unmanned Aerial Vehicle (UAV) Market (By Class: Tactical UAVs, Small UAVs, and Strategic UAVs; By Technology: Fully-autonomous, Semi-autonomous, and Remotely Operated; By System: UAV Payloads, UAV Airframe, UAV Avionics, UAV Software, and UAV Propulsion; By Application: Commercial, Military, and Recreational) - Global Industry Analysis, Size, Share, Growth, Trends Analysis, Regional Outlook and Forecasts, 2021 - 2030*. 2020. URL: https://www.precedenceresearch.com/unmanned-aerial-vehicle-market (visited on 09/12/2022).

[3] Martina Orefice, Vittorio Di Vito, and Giulia Torrano. “Sense and Avoid: Systems and Methods”. In: *Encyclopedia of Aerospace Engineering* (Dec. 2015). DOI: 10.1002/9780470686652.eae1149.

[4] Eulalia Balestrieri, Pasquale Daponte, Luca De Vito, Francesco Picariello, and Ioan Tudosa. “Sensors and Measurements for UAV Safety: An Overview”. In: *Sensors* 21.24 (2021). DOI: 10.3390/s21248253.

[5] Xia Chen, Miaoyan Zhao, and Liyuan Yin. “Dynamic Path Planning of the UAV Avoiding Static and Moving Obstacles”. In: *Journal of Intelligent & Robotic Systems* 99.3 (Sept. 2020), pp. 909–931. DOI: 10.1007/s10846-020-01151-x.

[6] Jia Pan, Sachin Chitta, and Dinesh Manocha. “FCL: A general purpose library for collision and proximity queries”. In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3859–3866. DOI: 10.1109/ICRA.2012.6225337.

[7] Xuejun Zhang, Yanshuang Du, Bo Gu, Guoqiang Xu, and Yongxiang Xia. “Survey of Safety Management Approaches to Unmanned Aerial Vehicles and Enabling Technologies”. In: *Journal of Communications and Information Networks* 3.4 (Dec. 2018), pp. 1–14. DOI: 10.1007/s41650-018-0038-x.

[8] Antonios Tsourdos, Brian White, and Madhavan Shanmugavel. “4. Collision Avoidance”. In: *Cooperative path planning of Unmanned Aerial Vehicles*. American Institute of Aeronautics and Astronautics, 2011. DOI: 10.1002/9780470974636.

[9] B. M. Albaker and N. A. Rahim. “Unmanned aircraft collision detection and resolution: Concept and survey”. In: *2010 5th IEEE Conference on Industrial Electronics and Applications*. 2010, pp. 248–253. DOI: 10.1109/ICIEA.2010.5516808.

[10] Changwen Zheng, Lei Li, Fanjiang Xu, Fuchun Sun, and Mingyue Ding. “Evolutionary Route Planner for Unmanned Air Vehicles”. In: *Robotics, IEEE Transactions on* 21 (Sept. 2005), pp. 609–620. DOI: 10.1109/TRO.2005.844684.

[11] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[12] Fan Hsun Tseng, Tsung Ta Liang, Cho Hsuan Lee, Li Der Chou, and Han Chieh Chao. “A Star Search Algorithm for Civil UAV Path Planning with 3G Communication”. In: *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. 2014, pp. 942–945. DOI: 10.1109/IIH-MSP.2014.236.

[13] Jen-Hui Chuang and N. Ahuja. “An analytically tractable potential field model of free space and its application in obstacle avoidance”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.5 (1998), pp. 729–736. DOI: 10.1109/3477.718522.

[14] Michael B. Bragg Scot E. Campbell and Natasha A. Neogi. “Fuel-Optimal Trajectory Generation for Persistent Contrail Mitigation”. In: *Journal of Guidance, Control, and Dynamics* 36.6 (2013). DOI: 10.2514/1.55969.

[15] Arvind U. Raghunathan, Vipin Gopal, Dharmashankar Subramanian, Lorenz T. Biegler and Tariq Samad. “Dynamic Optimization Strategies for Three-Dimensional Conflict Resolution of Multiple Aircraft”. In: *Journal of Guidance, Control, and Dynamics* 27.4 (2004). DOI: 10.2514/1.11168.

[16] Mangal Kothari and Ian Postlethwaite. “A Probabilistically Robust Path Planning Algorithm for UAVs Using Rapidly-Exploring Random Trees”. In: *Journal of Intelligent & Robotic Systems* 71.2 (Aug. 2013), pp. 231–253. DOI: 10.1007/s10846-012-9776-4.

[17] I.K. Nikolos, K.P.Valavanis, N.C. Tsourveloudis, A.N. Kostaras. “Evolutionary algorithm based offline/online path planner for UAV navigation”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 33.6 (2003), pp. 898–912. DOI: 10.1109/TSMCB.2002.804370.

[18] Yangguang Fu, Mingyue Ding, Chengping Zhou, and Hanping Hu, “Route Planning for Unmanned Aerial Vehicle (UAV) on the Sea Using Hybrid Differential Evolution and Quantum-Behaved Particle Swarm Optimization”. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 43.6 (2013), pp. 1451–1465. DOI: 10.1109/TSMC.2013.2248146.

[19] Chunfang Xu, Haibin Duan, and Fang Liu. “Chaotic artificial bee colony approach to Uninhabited Combat Air Vehicle (UCAV) path planning”. In: *Aerospace Science and Technology* 14.8 (2010), pp. 535–541. DOI: 10.1016/j.ast.2010.04.008.

[20] Jung-Woo Park, Hyon-Dong Oh, and Min-Jea Tahk. “UAV collision avoidance based on geometric approach”. In: *2008 SICE Annual Conference*. 2008, pp. 2122–2126. DOI: 10.1109/SICE.2008.4655013.

[21] L. Pallottino, E.M. Feron, and A. Bicchi. “Conflict resolution problems for air traffic management systems solved with mixed integer programming”. In: *IEEE Transactions on Intelligent Transportation Systems* 3.1 (2002), pp. 3–11. DOI: 10.1109/6979.994791.

[22] Jie Rong, Shijian Geng, J. Valasek, and T.R. Ioerger. “Air traffic conflict negotiation and resolution using an onboard multi-agent system”. In: *Proceedings. The 21st Digital Avionics Systems Conference*. Vol. 2. 2002, 7B2–7B2. DOI: 10.1109/DASC.2002.1052919.

[23] S. Wollkind, J. Valasek, and T.R. Ioerger. “Automated Conflict Resolution for Air Traffic Management Using Cooperative Multiagent Negotiation”. In: *Session: GNC-15: Airborne Separation Assurance Algorithms* (2012). DOI: 10.2514/6.2004-4992.

[24] Lidia Rocha and Kelen Vivaldini. “Analysis and Contributions of Classical Techniques for Path Planning”. In: *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE)*. 2021, pp. 54–59. DOI: 10.1109/LARS/SBR/WRE54079.2021.9605425.

[25] Oussama Khatib. “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots”. In: *The International Journal of Robotics Research* 5.1 (1986), pp. 90–98. DOI: 10.1177/027836498600500106.

[26] Alfian Ma’Arif, Wahyu Rahmaniar, Marco Antonio Márquez Vera, Aninditya Anggari Nuryono, Rania Majdoubi, and Abdullah Çakan. “Artificial Potential Field Algorithm for Obstacle Avoidance in UAV Quadrotor for Dynamic Environment”. In: *2021 IEEE International Conference on Communication, Networks and Satellite (COMNETSAT)*. 2021, pp. 184–189. DOI: 10.1109/COMNETSAT53002.2021.9530803.

[27] Thi Thoa Mac, Cosmin Copot, Andres Hernandez, and Robin De Keyser. “Improved potential field method for unknown obstacle avoidance using UAV in indoor environment”. In: *2016 IEEE 14th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*. 2016, pp. 345–350. DOI: 10.1109/SAMI.2016.7423032.

[28] Herath Mpc Jayaweera and Samer Hanoun. “Real-time Obstacle Avoidance for Unmanned Aerial Vehicles (UAVs)”. In: *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2021, pp. 2622–2627. DOI: 10.1109/SMC52423.2021.9659197.

[29] Pengwei Wang, Song Gao, Liang Li, Binbin Sun,and Shuo Cheng. “Obstacle Avoidance Path Planning Design for Autonomous Driving Vehicles Based on an Improved Artificial Potential Field Algorithm”. In: *Energies* 12.12 (2019). DOI: 10.3390/en12122342.

[30] Zijie Lin. “3D Fast Geometric Collision Avoidance Algorithm (FGA) and Decision-Making Approach Based on the Balance of Safety and Cost for UAS”. PhD thesis. Faculty of the Graduate School of the University of Maryland, College Park, 2021. DOI: https://doi.org/10.13016/kshr-d9nl.

[31] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. “Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches”. In: *IEEE Access* 8 (2020), pp. 105139–105155. DOI: 10.1109/ACCESS.2020.3000064.

[32] Selim Temizer, Mykel J. Kochenderfer, Leslie P. Kaelbling, Tomas Lozano-Perez and James K. Kuchar. “Collision Avoidance for Unmanned Aircraft using Markov Decision Processes.” In: *AIAA Guidance, Navigation, and Control Conference 2 - 5 August 2010* (2010). DOI: https://doi.org/10.2514/6.2010-8040.

[33] Yucong Lin and Srikanth Saripalli. “Sampling-Based Path Planning for UAV Collision Avoidance”. In: *IEEE Transactions on Intelligent Transportation Systems* 18.11 (2017), pp. 3179–3192. DOI: 10.1109/TITS.2017.2673778.

[34] Hao-Tien Chiang, Nick Malone, Kendra Lesser, Meeko Oishi, and Lydia Tapia. “Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2347–2354. DOI: 10.1109/ICRA.2015.7139511.

[35] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss and Wolfram Burgard. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013). Software available at https://octomap.github.io. DOI: 10.1007/s10514-012-9321-0.

[36] Zhangjie Fu, Jingnan Yu, Guowu Xie, Yiming Chen, and Yuanhang Mao. “A Heuristic Evolutionary Algorithm of UAV Path Planning”. In: *Wireless Communications and Mobile Computing* 2018 (Sept. 2018), p. 2851964. DOI: 10.1155/2018/2851964.

[37] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. “A Minimum Risk Approach for Path Planning of UAVs”. In: *Journal of Intelligent & Robotic Systems* 61.1 (Jan. 2011), pp. 203–219. DOI: 10.1007/s10846-010-9493-9.

[38] Manh Duong Phung and Quang Phuc Ha. “Safety-enhanced UAV path planning with spherical vector-based particle swarm optimization”. In: *Applied Soft Computing* 107 (2021), p. 107376. DOI: 10.1016/j.asoc.2021.107376.

[39] David Lane. “Values of the Pearson Correlation”. In: *Introduction to statistics*. Open Textbook Library, 2003.

[40] Hélder Cristóvão. “Real-Time Onboard Path Planning for Quadrotors”. MA thesis. Instituto Superior Técnico, 2021.

[41] Mariana Tavares. “Real-Time Onboard Trajectory Planning for Quadrotors in Open Environments with Moving Obstacles”. MA thesis. Instituto Superior Técnico, 2022.

[42] Alexander Fabisch. “pytransform3d: 3D Transformations for Python”. In: *Journal of Open Source Software* 4.33 (2019), p. 1159. DOI: 10.21105/joss.01159.

[43] Amila Thibbotuwawa, Grzegorz Bocewicz, Peter Nielsen, and Zbigniew Banaszak. “Unmanned Aerial Vehicle Routing Problems: A Literature Review”. In: *Applied Sciences* 10.13 (2020). DOI: 10.3390/app10134504.

[44] Juan Zhang, James F. Campbell, Donald C. Sweeney II, and Andrea C. Hupman. “Energy consumption models for delivery drones: A comparison and assessment”. In: *Transportation Research Part D: Transport and Environment* 90 (2021), p. 102668. DOI: https://doi.org/10.1016/j.trd.2020.102668.

[45] João L. Marins, Tauã M. Cabreira, Kristofer S. Kappel, and Paulo Roberto Ferreira. “A Closed-Form Energy Model for Multi-rotors Based on the Dynamic of the Movement”. In: *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2018, pp. 256–261. DOI: 10.1109/SBESC.2018.00047.

[46] António Ramalho. “Real-Time Trajectory Planning for UAVs in Environments with Moving Obstacles”. MA thesis. Instituto Superior Técnico, 2020.

[47] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[48] Afzal Suleman. *Lecture 5b: VTOL Design Point*. https://fenix.tecnico.ulisboa.pt/disciplinas/PEAer/2021-2022/1-semestre/lectures. Accessed: 2022–17-08. 2022.

[49] MATLAB. *version 9.9.0.1592791 (R2020b) Update 5*. Natick, Massachusetts: The MathWorks Inc., 2020.

[50] Joel McQuaid. “Early On-Set Prediction Of Vortex-Ring State Of Quadrotors”. In: (Oct. 2021). DOI: 10.32920/16811278.v1.

[51] David Lion, Adrian Chiu, Michael Stumm and Ding Yuan. “Investigating Managed Language Runtime Performance: Why JavaScript and Python are 8x and 29x slower than C++, yet Java and Go can be Faster?” In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA: USENIX Association, July 2022, pp. 835–852. ISBN: 978-1-939133-29-40.