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

Carolina Pereira Pinheiro
carolina.p.pinheiro@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

December 2022

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 work 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, 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

1. Introduction

An UAV (Unmanned Aerial Vehicle) is an aircraft with no on-board crew or passengers, that are controlled by onboard computers and thus have autonomous flight capabilities. They exist in several sizes, weights, and configurations (fixed wing, multi-rotors, hybrid, etc.) and are used in several real life applications such as payload delivery, traffic monitoring, surveillance, agriculture, military applications, construction industry, recreational uses, etc. [1]. The use of UAVs in this context requires not only offline path planning but also for them to 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 should be highlighted; according to a report by Precedence Research [1] the UAV market was valued at US$ 14.3 billion in 2020 and is expected to hit around US$ 53 billion by 2030.

The work developed is inserted into the continuous effort at CfAR (Centre for Aerospace Research) to develop a SAA (Sense and Avoid) system. A SAA system is a system 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 collision threats) and to instruct maneauvers capable of safely avoiding these conflicts [2]. The work 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.

1.1. Problem Statement

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 but with known trajectory).

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 that trajectory after the threat is avoided.

One of the biggest constraints when trying to solve this problem is the computational time available. The solution must be simple enough that it can be incorporated in an on-board flight computer like the *Raspberry Pi 4 Model B*.

It is also assumed that adequate information about the threats (position, size, velocity, etc.) and UAV(position, velocity, physical variables like C_D, A_{ef} etc.) is available by either the sensing system or the UAV's sensors.

1.2. Project Overview and Objectives

The framework developed takes as input the current attributes of the **UAV**, information about the **environment** (number of threats, position, velocity, size, etc.) 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. The framework was mainly developed in Python 3.7.13 .Python was chosen due to its ease to quickly develop and test different solutions, despite not being the fastest language in terms of runtime. The main contributions of this work is the development and testing of a new and different type of online avoidance algorithm, that could hopefully improve where the previous work lacked. The cost calculation section is also a good contribution that could be implemented independently from this work with other path generations approaches.

2. Theoretical Background

2.1. Path Generation

When 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** implementation. 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. This is where the **safety management** approaches are inserted.

Safety management guarantees the safety and integrity of the aircraft and the airspace/environment. Zhang et al. (2018)[3] provides a summary of safety management approaches to UAVs, where different methods are presented depending on the scale of safety. The different scales of safety defined are **Large**, **Middle** and **Small** and they are dependent on the available time frame to find a solution (Large-Scale pertains to a global path-planning problem and Small-Scale to an imminent collision avoidance problem). 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 [4] (small-scale safety). Since this work is mainly focused on small-scale safety, this will be the type that will be further discussed.

In 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, acceleration, 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**. An higher emphasis will be given to coordinated algorithms, specifically geometric method, since the method chosen in this work is of this type.

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** (like TCAS - Traffic Collision and Avoidance System), **APF (Artificial Potential Field) methods** and **geometric methods** are included.

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 [3]. 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 is that they require less processing power and thus can be more easily implemented on on-board computers [5]. 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) [5]. Geometric methods can handle both static and dynamic obstacles .

The algorithm developed in this work is an improvement/extension on the work developed by Chen et al. (2020) [4] and falls into the geometric methods for small-scale safety category.

2.2. Collision Detection

The FCL [6] is a library, originally developed in C++, for performing proximity queries on a pair of geometric models . FCL supports both different types of collision queries/distance computations as well as different types of objects shapes (box, sphere, cylinder, etc.) which makes it a very complete library and tool to use.

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

2.3. Cost Calculation

The final block on the framework is the cost calculation. The goal of the cost function is to numerically evaluate the quality of the candidate paths.

Analyzing different articles and cost functions previously developed [4][7] [8] , it can be concluded that the most important factors when choosing cost function parameters are: **collision risk, aircraft energy consumption, shape of the path, 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.

3. Methodology

3.1. Path Generation

The first step in developing the collision avoidance system is the **path generation**. The path generation method adopted (cubic spline method[4]) 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 was made to the path generation algorithm is its expansion to the 3D space.

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 [4]:

$$y = a(x - x_{start})^3 + b(x - x_{start})^2 + \\ + c(x - x_{start}) + y_{start},\ x \in (x_{start}, x_{end}) \tag{1}$$

$$c = 0 \tag{2}$$

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

$$b = \frac{3(y_{mid} - y_{start}) - 2c\Delta x}{\Delta x^2}\ . \tag{4}$$

y_{start}, y_{end}, x_{start} and x_{end} are the starting position and the ending point position 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. Since $c = 0$, y_{mid} is the variable that controls the a and b variables. 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.

y_{mid} is defined as [4]

$$y_{mid} = y_{start} + \omega\ . \tag{5}$$

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

$$\omega = [-3.2 \times R_{threat}, 3.2 \times R_{threat}],\quad \Delta\omega = \frac{2\omega_{max}}{l}\ , \tag{6}$$

and l is a variable that can be changed that controls how many candidate paths are generated. Finally, the end point is defined as [4]

$$(x_{end}, y_{end}) = (1.5 \times \Delta x + x_{start}, y_{start})\ . \tag{7}$$

Figure 1 shows an example of the candidate paths generated with this implementation.

It should be taken into account that the paths can only be generated if the UAV is far away enough

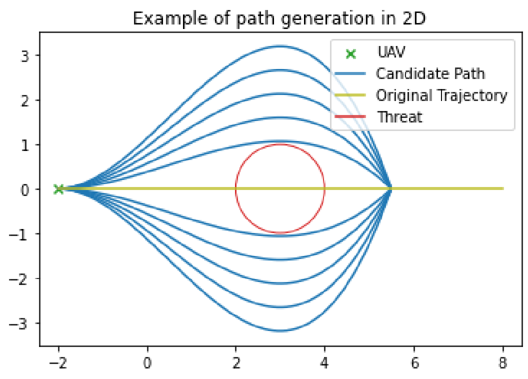


Figure 1: Paths generated for $R_{threat} = 1, l = 12$

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 $\times R_{threat}$.

3.1.2 Symmetric Candidate Paths

The candidates paths generated above 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. Figure 2 shows the symmetric paths correspondent to the simple candidate paths of Figure 1.

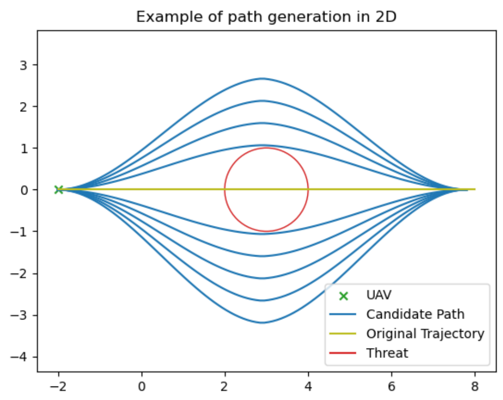


Figure 2: Symmetric candidate paths generated for $R_{threat} = 1, l = 12$.

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. So, 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.

Reference frame

For the expansion to the 3D space, first a plane needs to be chosen to generate the initial group of candidate paths before rotating them. 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, 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. 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.

To handle conversions from the global reference frame to the UAV reference frame, the `pytransform3d` library is used [9] - 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.

Rotation of candidate paths

The original plane where the initial group of candidate paths are generated can be seen in Figure 3 in pink. Then, the rotation axis - the line that connects the evading point and the centre of the threat - is chosen (yellow in Figure 3). Next, 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 (8) is defined that describes θ_i, the rotation angle for the i-th target plane. An example target plane is represented in green in Figure 3.

$$\theta_i = \frac{180^\circ}{n+1} \times i,\ i = 1, ..., n \tag{8}$$

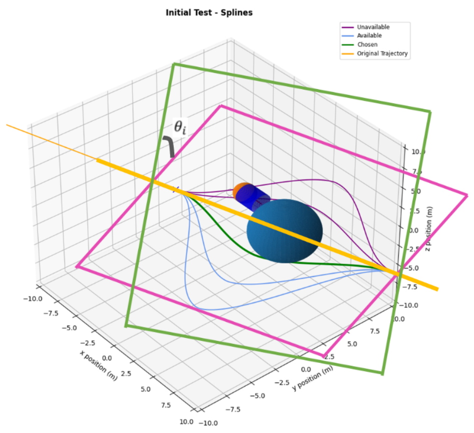


Figure 3: Expansion to 3D space process. Pink - plane of the initial group of candidate paths. Yellow - rotation axis. Green - target plane. θ_i - rotation angle.

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 using the `pytransform3d` library. The final group of candidate paths are written in the global reference system. 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.

3.2. Collision Detection

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 object, whereas the dynamic security describes whether there was a collision with a dynamic object.

Static security

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 estimated. 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 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 if there is a collision, the candidate path is marked as unavailable.

Dynamic Security

Dynamic security takes into account the moving objects/threats present in the environment. 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. 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 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 threat’s velocity. The dynamic security was implemented with the FCL [6]. The collision query is done similarly to the static security.

3.3. Cost Function

Next, a cost function is defined to evaluate the candidate paths generated. Firstly, the six different cost function parameters are defined and described (Smoothness, Smoothness with penalization (SWP), Energy, Hovering Power, Time and Closeness to MO (CMO)). Secondly, 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 normalized so they can be comparable. Finally, the weights for the cost function are chosen and tuned.

3.3.1 Parameters

Energy

The model chosen was developed by Marins et al. (2018) [10]. This model had been previously used in work developed at CfAR and it takes into account the factors that affect energy consumption described by Thibbotuwawa (2020) [11].

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

$$E_1 = ||E_M(i+1) - E_M(i)|| \tag{9}$$

$$E_M(i) = \frac{1}{2}m||v_i||^2 + mgp_{i,z} \quad (10)$$

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.

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

$$E_D(i) = \frac{1}{2}C_D\rho A_e||p_{i+1} - p_i||\frac{||v_{i+1} + v_i||^2}{2} \ , \quad (11)$$

and

$$E_2 = \sum_{i=0}^{n} E_D(i) \ , \quad (12)$$

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 ($E(i) = E_1 + E_2$).

Smoothness

Another factor that is used to evaluate the quality of the paths generated is the ability of the UAV to fly stably and without sudden changes of movement. The smoothness will be evaluated by the curvature of the candidate path, defined as

$$\kappa = \frac{|x''y' - x'y''|}{((x')^2 + (y')^2)^{3/2}} \quad (13)$$

and so, smoothness is written as [4]

$$S(i) = \int_{x_{start}}^{x_{end}} \kappa_i^2 \, dx \quad (14)$$

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

Smoothness with Penalization (SWP)

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. SWP is defined as

$$S(i)_{rp} = S(i)(1 + k_p|\sin\alpha_i|) \ , \quad (15)$$

where α_i is the angle of elevation of the candidate path, the angle between the candidate path plane and the ground plane.

Closeness to Moving Objects (CMO)

The closeness to moving objects (CMO) takes into account both the minimum distance between the waypoints of the two paths (the threat path, and the candidate path) (c_{min}) as well as the average distance between the two paths ($\overline{c}$). For each candidate path and MO pair, the CMO is defined as

$$f_c(i) = (0.5\overline{c} + 0.5c_{min})^{-1} \ , \quad (16)$$

the expression is inverted since a lower distance means more danger which in consequence should mean more cost.

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

$$M(i) = 0.5max(f_c(i)) + 0.5\overline{f_c(i)} \quad (17)$$

Time

The time (T) that it takes for the UAV to fly the path is another useful parameter. Knowing the velocity (v) in each waypoint and the coordinates for each waypoint (so the distance d between waypoints can be calculated) makes this calculation trivial, as $T = \frac{d}{v}$.

Hovering Power

Hovering power allows to distinguish between paths that climb/descend based on the amount and velocity that they do so. Three different methods are used depending on which phase of flight the UAV is at (Hovering, Vertical Climb and Vertical Descent Flight), which is determined in each segment between two waypoints and then added for the entire path.

For hovering flight, hovering power is calculated as [12]

$$P = P_{i,R} + P_0 = k_i\frac{W^{3/2}}{\sqrt{2\rho A}} + \rho A {V_{tip}}^3\left(\frac{\sigma C_d}{8}\right) \ . \quad (18)$$

$P_{i,R}$ is the real induced power, P_0 the rotor's profile power, W the weight, A the rotor disk area, V_{tip} the velocity at the tip of the blade ($V_{tip} = 0.8M$; otherwise, the blades would enter the transonic regime), C_d is the average blade drag coefficient.

For vertical climb flight, the power is defined [12]

$$P = W\left(V_y - \frac{k_i}{2}V_y + \frac{k_i}{2}\sqrt{{V_y}^2 + \frac{2W}{\rho A}}\right) + \quad (19)$$
$$+\rho A {V_{tip}}^3\left(\frac{\sigma C_d}{8}\right)$$

For vertical descent flight, if $V_y/v_{i,h} \leq -2$ [12]

$$P = W\left(V_y - \frac{k_i}{2}(V_y + \sqrt{{V_y}^2 - \frac{2DL}{\rho}}\right) + \quad (20)$$
$$+\rho A {V_{tip}}^3 \left(\frac{\sigma C_d}{8}\right) .$$

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

$$v_{i,d} = v_{i,h}(k_i - 1.125\left(\frac{V_y}{v_{i,h}}\right) - 1.372\left(\frac{V_y}{v_{i,h}}\right)^2 + \quad (21)$$
$$-1.718\left(\frac{V_y}{v_{i,h}}\right)^3 - 0.655\left(\frac{V_y}{v_{i,h}}\right)^4)$$

$v_{i,h} = \sqrt{\frac{W}{2\rho A}}$ is the induced velocity at the rotor in hover.

The rotor solidity, $\sigma = \frac{Nc}{\pi R}$, where N is the rotor's number of blades, c is the chord, and R is the blade's radius. The rotor disk area $A = n\pi R^2$, 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 $DL = \frac{W}{A}$. The drag coefficient C_d and the air density ρ are imported from the configuration file and are set by the user .

3.3.2 Normalization

To normalize the parameters, **16200** environments were created in different scenarios, in order to analyze the distribution of each cost function parameter. From the parameter distribution, hovering power is disregarded as a cost function parameter due to its atypical behaviour, with data points being concentrated within a certain range and with similar frequencies throughout (which means it won't be good to distinguish between paths). The remaining cost function parameters are normalized: time & energy depend on characteristics of the UAV so their min and max values pre min-max normalization and feature clipping are calculated by measuring its values in the lowest and highest time-consuming/energy-consuming candidate path. The three remaining parameters (smoothness, SWP, CMO) are normalized with log scaling, min-max normalization and feature clipping. Log scaling is used due to the parameters distribution - since they have a handful of values that have many points and other values that have few points.

3.3.3 Parameter Selection

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 don't 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 is influential, **variance in altitude** - if a change in altitude is influential; **imminent danger** - if moving objects and its proximity is influential; **performance while flying the path** - if the performance of the UAV while flying the candidate path is measured by this parameter.

Analyzing Table 1, smoothness and SWP depend on the shape of the candidate path, but whereas smoothness does not, SWP 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 CMO is the only one that takes into account threats other than the main threat.

Since SWP and smoothness both take into account the geometry of the path, only one of them is needed as a final cost function parameter, and due to the additional information it provides, SWP is chosen. CMO is the only parameter that measures the imminent danger associated with the path. For this reason it must be included, despite the higher computational time. The final choice is between the time and energy parameters, since they both measure the performance. An analysis was made and it was concluded that time and energy have a correlation coefficient of $\rho = 0.9120$ and thus, since time is quicker to compute, it was chosen over energy. The final 3 cost function parameters chosen are **SWP, CMO and time.**

3.3.4 Weight Tuning

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. So, different weight combinations are tried in a variety of scenarios in order to choose an adequate combination. It is assumed that CMO is the most important parameter, followed by SWP and time. After evaluating all of the scenarios, the final cost function is defined as

$$C(i) = 0.30S(i)_{rp} + 0.20T(i) + 0.50M(i) \quad (22)$$

Table 1: 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	

4. Results & discussion

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. The framework developed will be tested in different scenarios in order to comprehend its strengths, vulnerabilities and future improvements.

4.1. Runtime

The runtime is analyzed for 3 different types for environments, that vary in the amount of threats considered in the environment ($N = 1$: 1 static threat; $N = 2$: 1 static threat and 1 dynamic threat; $N = 3$: 1 static threat, 2 dynamic threats). CMO is only calculated for $N = 2$ and $N = 3$. Figure 4 shows the total runtime distribution. Table 2 shows the runtime for each parameter and the total time. Looking at both the table and the figure it can be stated that, as expected most results are concentrated within a certain range, with some outliers on the upper scale. The fastest cost function parameter to compute was time, followed by SWP and finally CMO. These results also confirm that, along with the number of candidate paths generated, the number of moving 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 - this shows that there needs to be an effort to either optimize the calculation of the CMO parameter or to limit the number of MOs considered when calculating the CMO value. To better understand the computational limitations, these times are then extrapolated to the time that would take in a Raspberry Pi CPU. According to the PassMark Software CPU bench-marking tool [13], from the desktop CPU to the Raspberry Pi CPU **the loss of performance expected is of 83 %**. The last line of table 2 shows these results. These times can be bettered by either a code optimization (mainly in the CMO calculation process) or by porting the work to a faster language, like C++.

Finally, the effect of the calculation time has to be discussed. While the avoidance manoeuvre is being computed, the UAV is still flying and thus, changing position. This shift in position has to be taken into account when generating the candidate path, and the starting point should be adjusted accordingly. This shift has to be cruise velocity dependent and adds another restriction to the problem.

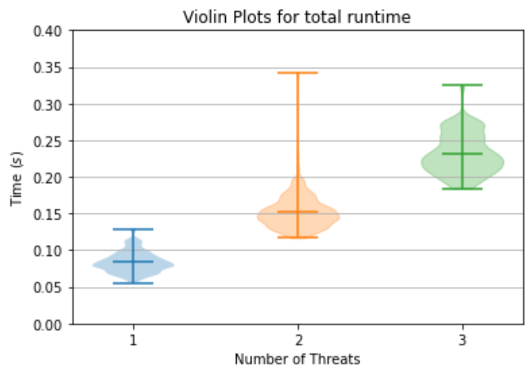


Figure 4: Total time runtime .

4.2. Performance

To evaluate the performance of the framework developed, 3 scenarios will be created. **Scenario 1** will test increasing collision angles, **Scenario 2** will have two threats moving in parallel with the UAV's original trajectory, **Scenario 3** will test the framework's behaviour by having two dynamic obstacles in very close proximity to most of the candidate paths.

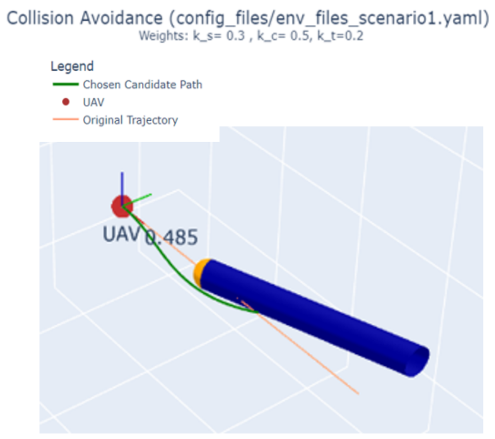
Scenario 1

Figure 5 shows the behaviour with a collision angle (the angle from which the main threat is approaching the collision point from) of 160 °. The threat is successfully avoided; however, for a bigger threat a wider path would have to be chosen. With increasing collision angles, eventually all of the candidate paths will be unavailable. This occurs since all of the candidate paths have the same end point, and so, a big enough threat for a big enough collision angle would have its collision path overlapping the end points. 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. 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

Table 2: 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, RBPi - Raspberry Pi.

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
Total RBPi (s)	0.1015	0.2132	0.3371	0.2364	0.6248	0.5942	0.1532	0.2794	0.4244

bigger threats.



Collision Avoidance (config_files/env_files_scenario1.yaml)
Weights: k_s= 0.3 , k_c= 0.5, k_t=0.2
Legend
Chosen Candidate Path
UAV
Original Trajectory
UAV
0.485

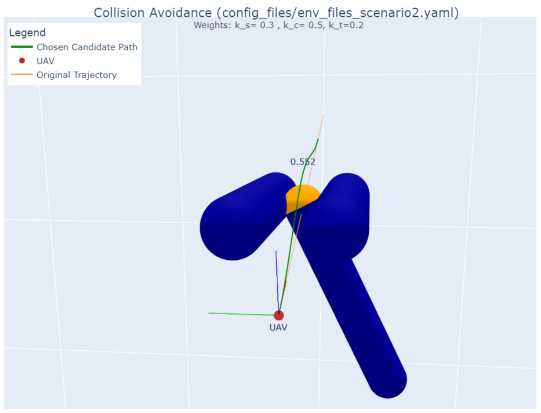
Figure 5: Solution found for a collision angle of 160 ° (Scenario 1). $R_{threat} = 1$.

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. Figure 6 shows the results. The candidate path chosen avoids the threat by climbing over the collision point. Analyzing it, 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 it. 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. Nonetheless, the chosen candidate path successfully avoids the incoming threats.

Scenario 3

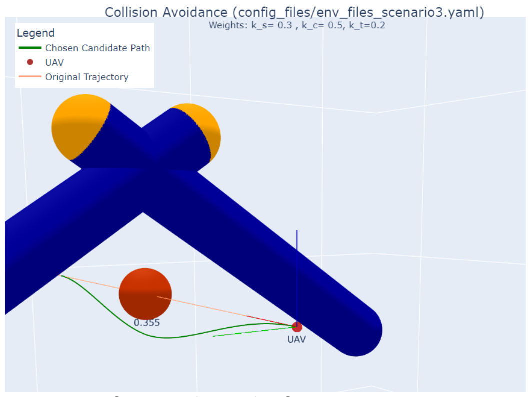
In the last scenario, there are two threats moving diagonally above the collision point . These threats and its collision paths either collide or are in close proximity to most of the candidate paths. Firstly, the main threat is considered to be static,(Figure 7). In this situation, the solution found shows the prioritization of the CMO, as the candidate path that descends first and then climbs is chosen due



Collision Avoidance (config_files/env_files_scenario2.yaml)
Weights: k_s= 0.3 , k_c= 0.5, k_t=0.2
Legend
Chosen Candidate Path
UAV
Original Trajectory
0.552
UAV

Figure 6: Solution found for Scenario 2 - Zoomed in.

to its distance from the collision paths, despite the higher energy it takes. This is evidenced when the main threat is changed and now considered dynamic (Figure 8), the final cost of the chosen path of the first version is 0.355 and 0.405 in the second version. In the second scenario, since it is now not possible to choose the path previously chosen, the chosen one avoids the main threat by going to the left.



Collision Avoidance (config_files/env_files_scenario3.yaml)
Weights: k_s= 0.3 , k_c= 0.5, k_t=0.2
Legend
Chosen Candidate Path
UAV
Original Trajectory
0.355
UAV

Figure 7: Solution found for Scenario 3-1, zoomed in.

In summary, the framework developed can find solutions for most situations, however some improvements and adjustments should be made to improve the robustness and security of the work developed. These adjustments mainly pertain to the collision detection block, by either expanding the collision path considered beyond the collision

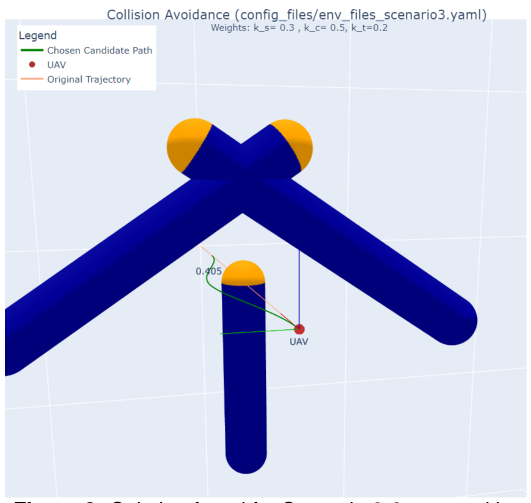

Collision Avoidance (config_files/env_files_scenario3.yaml)
Weights: k_s= 0.3 , k_c= 0.5, k_t=0.2
Legend
Chosen Candidate Path
UAV
Original Trajectory
0.405
UAV

Figure 8: Solution found for Scenario 3-2, zoomed in.

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.

5. Conclusions

In this work, a framework was developed in Python 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 route 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 in 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.

References

[1] Precedence Research. Unmanned aerial vehicle (uav) market - global industry analysis, size, share, growth, trends analysis, regional outlook and forecasts, 2021 - 2030.

[2] Martina Orefice, Vittorio Di Vito, and Giulia Torrano. *Sense and Avoid: Systems and Methods*. 12 2015.

[3] Xuejun Zhang, Yanshuang Du, Bo Gu, Guoqiang Xu, and Yongxiang Xia. Survey of safety management approaches to unmanned aerial vehicles and enabling technologies. *Journal of Communications and Information Networks*, 3(4):1–14, 12 2018.

[4] Xia Chen, Miaoyan Zhao, and Liyuan Yin. Dynamic path planning of the uav avoiding static and moving obstacles. *Journal of Intelligent & Robotic Systems*, 99(3):909–931, 09 2020.

[5] 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.

[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*, pages 3859–3866, 2012.

[7] Luca De Filippis, Giorgio Guglieri, and Fulvia Quagliotti. A minimum risk approach for path planning of uavs. *Journal of Intelligent & Robotic Systems*, 61(1):203–219, 01 2011.

[8] Manh Duong Phung and Quang Phuc Ha. Safety-enhanced uav path planning with spherical vector-based particle swarm optimization. *Applied Soft Computing*, 107:107376, 2021.

[9] Alexander Fabisch. pytransform3d: 3d transformations for python. *Journal of Open Source Software*, 4(33):1159, 2019.

[10] 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)*, pages 256–261, 2018.

[11] Amila Thibbotuwawa, Grzegorz Bocewicz, Peter Nielsen, and Zbigniew Banaszak. Unmanned aerial vehicle routing problems: A literature review. *Applied Sciences*, 10(13), 2020.

[12] Afzal Suleman. Lecture 5b: Vtol design point. `https://fenix.tecnico.ulisboa.pt/disciplinas/PEAer/2021-2022/1-semestre/lectures`, 2022. Accessed: 2022–17-08.

[13] PassMark Software. Intel Xeon E5-1620 v3 @ 3.50GHz vs ARM Cortex-A72 6 Core 1512 MHz.