Real-Time Onboard Path Planning for Quadrotors

Hélder Cristóvão
helder.cristovao@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

June 2021

Abstract

The work done in this thesis is focused on the development of a framework that empowers an UAV with the capability of going from a start to a destination while simultaneously avoiding static and dynamic obstacles during its course. The framework was developed with the intention of running onboard of an UAV the associated computational limitations.

The framework is composed of a real-time trajectory planning algorithm that is split into a two-step approach: an Offline/Pre-Flight Path Planning and an Online/Real Time Path Replanning.

In the first step techniques are implemented to generate trajectories in a known static environment. The proposed solution makes use of sampling-based motion planning algorithms (both the RRT-Connect and the Informed RRT* are used) to find an initial feasible path that's then feed into an optimization method which turns it into a feasible trajectory. This optimization method seeks to find smooth trajectories by minimizing the 4th derivative of position (Snap).

The second step consists of a real time avoidance module which allows the UAV to avoid dynamic obstacles in its course by means of a local generation of a transitioning trajectory around them. A pragmatic approach is used to tackle this problem that leverages the quick generation of trajectories provided by the aforementioned optimization method.

The algorithm was developed under the assumption that the UAV surroundings are modeled in the form of an Octomap.

To test and validate the capabilities of the developed framework, simple simulations were designed and Software-in-the-loop tests were carried out using PX4 and Gazebo as simulation tools.

Keywords: onboard, RRT-Connect, Informed RRT*, Replanning, smooth, Snap, Octomap, Software-in-the-Loop, PX4, Gazebo

1. Introduction

An Unmanned aerial vehicle (UAV) is a type of aircraft which can be classified according to its degree of automation as autonomous if able to fly preplanned routes with no human intervention or non-autonomous if remotely piloted. The recent ascension of UAV hardware as a mean of fulfilling the requirements of civilian, commercial, military and aerospace applications (due to the versatility they offer) coupled with the technological development in areas such as computation hardware and software and also Artificial Intelligence (AI) has led to a shift from non-autonomous to autonomous UAV use. While full automation has yet to be achieved, there are studies that shown the potential market for tasks performed by autonomous UAVs [16]. To achieve any degree of automation a number of competences are required: Self-Localization, the ability of an agent to determine its own position and orientation within a certain frame; Map-Building, capability of representing the map of the environment through information acquired by sensing equipment such as cameras or radar; Path Planning, for creating the necessary path configurations that allow hypothetical movement from a start point to a goal destination while satisfying the constraints of the UAV's own performance expectation and the environment; Sense and Avoid (S&A) technology, necessary for preventing collisions with dynamic obstacles. The path planning problem can be split into two different problems in regards to reachability [9] as Global Planning and Local Planning. The first deals with finding a preliminary (global) feasible and optimal path based on the *a priori* environmental information obtained. Global Planning cannot deal with real-time problems and is a type of static programming. Meanwhile, Local Planning makes use of newly acquired sensorial information to correct the initial path planning assumptions made (in the event of a possible collision with a detected static or dynamic obstacle). Unlike the former, it is applicable in real-time (a type of dynamic programming).

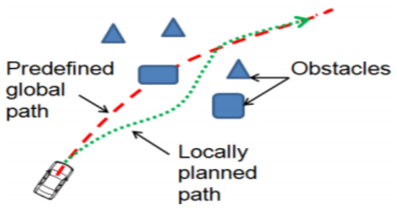

Figure 1: Local path (green) generated to correct collision in tracking of global path (red). Taken directly from []

On the subject of path planning, the main approaches followed include graph-based and biologically inspired algorithms. The first group includes several methods such as Rapid Exploring Random Tree method (RRT), directed graph-based method, A* search algorithms, artificial potential field method (APF) among others. However, they present difficulties addressing motion constraints. The second group includes the Ant Colony Optimization (ACO) method, Particle Swarm Optimization (PSO) and Genetic algorithm. These algorithms present some advantages regarding reduced complexity and dimensions. In order to tackle the above mentioned challenges, a framework was developed where a strategy of global planning (defined as Offline/Pre-Flight Planning) followed by a dynamic local planning (referred to as Online/Real-time Path Replanning) to deal, respectively, with static and dynamic obstacles (referred to as intruders). Both planning components follow a similar construction by employing RRT based methods to generate paths which are sub-sequentially fed to an optimization method which seeks to minimize the fourth derivative of position (snap) in order to transform the initial paths to flyable smooth trajectories. The framework starts by building an occupancy grid in the form of an octomap [1], a type of occupancy grid, from the initial information about the environment. The Offline/Pre-Flight component is then activated and it generates a first trajectory which is fed to the UAV. After initiating the flight, the Online/Real-time component is activated and seeks periodically for potential disturbances to the intended trajectory in the form of collision from dynamic obstacles (referred to as intruders). To simulate a real instance of detection, the intruders are assumed to be detected within a radius of influence. If there is a potential collision, a transitioning trajectory is generated between safe points in the original trajectory or a new trajectory is generated from an exit point in the original trajectory. The main contribution is the development of a framework targeted towards the capability of being employed onboard a small quadrotor using a computationally limited integrated processor such as the Raspberry Pi. The remaining part of the paper is organized as follows. Background relevant work is introduced in Section 2 followed by the proposed framework and implementation in Section 3. Experimental results along with interpretation are presented in Section 4. In Section 5, a brief summary is presented with remarks about future work.

2. Background

A general overview on quadrotor dynamics and its importance in the overall autonomous navigation problem is now considered.

2.1. Quadrotor Dynamics and Control

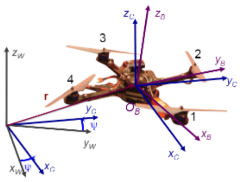

Figure 2: Quadrotor model body frame (subscript B) and earth fixed frame (subscript E). Taken directly from [13]

Using the information from the previous figure, the rigid body equations of a quadrotor can be written as:

Where (m) is the quadrotor mass, (J) is the quadrotor inertia matrix in with respect to B-frame, ω is the angular velocity with respect to the B-frame, $(e_3) = [0, 0, 1]^T$ is the third vector of the canonical basis of $\mathbb{R}^3$, $[T_z, T_\phi, T_\theta, T_\psi]$ are the inputs of the quadrotor representing the collective force, roll torque, pitch torque, yaw torque, respectively, (ω_i) is the speed of the ith motor in , (k) is the thrust factor, (d) is the drag factor, (l) is the distance between the center of the quadrotor and the center of a propeller, (R) is the rotation matrix needed to map the orientation of a vector from B-frame to W-frame. The rank of the input matrix $(rankF(q) = 4)$ is lower than the dimension of the configuration vector $(dim(q) = 6)$ the

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\xi} & \boldsymbol{\Theta} \end{bmatrix} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}$$

$$D = \begin{bmatrix} m \cdot \boldsymbol{I}_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & \boldsymbol{J} \end{bmatrix}$$

$$\boldsymbol{C}(q, \dot{q}) = \begin{bmatrix} 0_{3 \times 1} \\ \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} \end{bmatrix}$$

$$\boldsymbol{G}(q) = \begin{bmatrix} -g \cdot \boldsymbol{e}_3 \\ 0_{3 \times 1} \end{bmatrix}$$

$$\boldsymbol{F}(q) = \begin{bmatrix} \boldsymbol{R} \cdot e_3 & 0_{3 \times 3} \\ 0_{3 \times 1} & \boldsymbol{I}_{3 \times 3} \end{bmatrix}$$

$$u = \begin{bmatrix} T_z \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l & 0 & l & 0 \\ 0 & l & 0 & -l \\ d & -d & d & -d \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

$$f_i = k \cdot w_i^2$$

system is said to be underactuated and cannot follow an arbitrary trajectory. However, in [13] the authors prove that a quadrotor is a differential flat system meaning we can find a set of outputs (equal in number to the number of inputs) such that we can express all states and inputs in terms of those outputs and their derivatives. In a formal way this means that in a non-linear system such as the quadrotor system:

$$\dot{q} = f(q, u) \tag{1}$$

$$j = h(q) \tag{2}$$

where q represents the configuration vector and u, the control input, is differentially flat if it is possible to find outputs k:

$$k = \zeta(q, u, \dot{u}, ..., z^l) \tag{3}$$

such that,

$$q = q(k, \dot{k}, ..., k^l) \tag{4}$$

$$u = u(k, \dot{k}, ..., k^l) \tag{5}$$

q are the tracking outputs and k the flat outputs.
The authors of [] choose as flat outputs:

$$q = (x, y, z, \psi) \tag{6}$$

and proved they could be written as q and it is derivatives. This result is of particular importance since the method utilized in computing feasible trajectories in this work makes use of it as it will be detailed ahead.

2.2. Map Representation

Among the 3D world building approaches, octrees stand out as a viable option by providing an efficient way of storing and representing multi-resolution maps.

In this hierarchical data structure, each node stores the information of a cubic voxel in the occupancy grid and is subdivided into eight smaller voxels until a given minimum voxel size is reached (resolution of the octree). In this method if a certain measured volume within an environment is occupied then the node corresponding to that volume is set to occupied. Free volumes can be represented as free nodes and any uninitialized volume or subvolume is set as unknown. This allows for a reduction in the number of subvolumes of the environment that

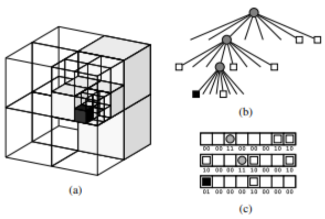

Figure 3: Example of an octree storing free (shaded white) and occupied (black) cells (a), the corresponding tree representation (b), and the corresponding bitstream for compact storage in a file (c). Taken directly from [1])

actually need to be explicitly represented. However, octree based approaches pose additional problems such as map update, overconfidence and compression.

2.3. Global Planning

In the context of the two-stepped approach outlined in this work, the planning methods are presented according to the global or local nature of their application.

RRT-based methods: Rapid Exploring Random Tree (RRT) is one of the most used methods in literature which works by trying to connect sampled nodes from the search space. A graph is initialized only with the initial state as a node and at each step a random sample from the search space is attempted to be connected to the nearest node on the graph and if the connection is successful (the path between is collision free), then the node is added to the graph and the process repeats itself until the goal state is added to the graph. Variants of this basic implementation such as the RRT* [7] which guarantees asymptotic optimallity by applying two different concepts which are called near neighbor search and rewiring tree operations and the Informed RRT* [5] where an heuristic informed search strategy is implemented in RRT* in order to shrink the planning problem to subsets after finding an initial solution have been featured in several literature works in path planning problems as in [3] where the authors developed a planning system for dynamic obstacle avoidance in a cluttered 3D environment by taking advantage of the fast convergence of the former method coupled with real-time feedback of the UAV and obstacles poses obtained from motion tracking cameras and showed the ability to handle multiple obstacles in real-time. While the Informed RRT* is considered to be a global method, it was successfully adapted to deal with the necessities of local planning. Another variant of the RRT was developed to be fully applicable to dynamic path planning in 2015 [2] called Real-Time RRT* (RT-RRT*) by means of an online rewiring of the path which can be summarized in the figure below. **Artificial Neural Network**: A neural network consists of a collection of methods that employ mathematical models to mimic the neural network of organisms, in other words the goal is to find underlying relationships between input data in a similar way to which our brain operates under outside stimulus. Neural networks have been applied to path planning problems such as in [11] where a method called DeepSarsa was used to tackle the problem. This method gains information and rewards from the environment and helps UAV to avoid dynamic obstacles as well as finds a path to a target based on a deep neural network. Tested in a ROS-Gazebo environment and it showed impressive results when compared to other algorithms.

2.4. Local Planning

Potential field (PF): Artificial Potential Field (APF) based algorithms have been applied in local planning as part of reactive obstacle avoidance. APF model the world by a force field with obstacles acting as repulsive poles and defined goals as attractive poles therefore there is an attractive force that increases towards the goal and repulsive forces that increase towards obstacles. In APF the robot is modeled as a charged particle moving through this world where these potential fields can ”drive” the robot away from static and dynamic obstacles. These algorithms are low in computational requirements and provide smooth paths while not offering guarantees of optimality or completeness since one of the main drawbacks is that they tend to become trapped in *local minima*, preventing the robot from reaching its goal. One recent example to mitigate this drawback was developed in [6] where an improved APF algorithm was proposed for autonomous obstacle avoidance planning of a quadrotor. The novelty introduced in this approach is that the algorithm can now calculate in real-time the different potential field at each instance. **Model Predictive Control (MPC)**: It consists of a nonlinear control technique that uses a dynamic model of a plant and past control history to optimize future states over a defined time period and subject to a cost function, therefore the trajectory optimization problem is solved through feedback control. For an UAV, the dynamic model corresponds to the UAV and the response to external and internal forces, while the cost function is generally a combination of variables like distance, velocity or acceleration to be minimized. A recent use of this method has been made in [10], where the dynamics of multiple obstacles are anticipated through the use of active set algorithms that only consider obstacles that affect the UAV, reducing the computational burden of the algorithm. The authors also employed orientable ellipsoids to model the obstacles in order to improve the smoothness of avoidance maneuvers. Results in two real-time implementations showed the algorithms capability to avoid dynamic obstacles.

3. Methodology

3.1. Proposed Solution

The developed framework consists of a two-phased planning:

- **Offline/Pre-Flight Path Planning** In the first phase, from the start position and with full knowledge of the current state of the environment, a global path planning strategy is employed to generate a feasible collision-free optimal (length) trajectory to lead the UAV to the chosen destination.

- **Online/Real-time Path Replanning** From the instance in which the UAV starts tracking the desired trajectory, the dynamic environment is checked periodically for possible collisions with moving obstacles. In case of a collision, replanning protocols are activated in order to generate a new collision-free trajectory towards the destination. At this point the optimality guarantee is sacrificed to ensure collision avoidance.

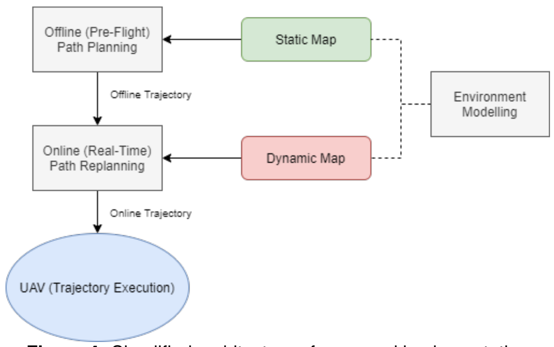
Offline (Pre-Flight) Path Planning
Static Map
Environment Modelling
Offline Trajectory
Online (Real-Time) Path Replanning
Dynamic Map
Online Trajectory
UAV (Trajectory Execution)

Figure 4: Simplified architecture of proposed implementation

3.2. Map Representation

The choice of environment representation in this work fell on octomap representation.

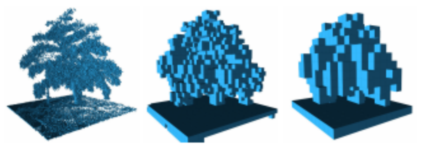

Figure 5: Multiple resolutions of the same map from a higher resolution on the left to a progressively lesser one on the right. Taken directly from [1]

An Octomap is an occupancy grid mapping approach based on octrees and probabilistic modelling to build 3D environments by dividing the space in cubic volumes of equal size called *voxels*. The main advantages of this method are the capability of modelling arbitrary environments (free, occupied and unknown areas), real-time environment updates (through the use of probabilistic occupancy estimation), multi-resolution and compactness of storage. In the scope of this thesis, we are restricted to partially known environments where the static obstacles location are known *a priori* and dynamic obstacles detection is simulated within the range of equipped sensors. There are two different situations in which octomap modelling is employed in the developed framework:

- **Static Map** contains the *a priori* knowledge of the environment before the quadrotor takes flight. Only the static obstacles are modelled;
- **Dynamic Map** is the designation of the Static Map when the quadrotor is moving. When an intruder is within a defined range of the quadrotor, the octomap is updated periodically according to the position of the intruder.

The octomap specific functions and data structures are available on the open source C++ Octomap library. The routine that allows the generation of the static map from a file containing the description of the environment starts by taking the boundaries of each obstacle in a description file, converting them to the appropriate measurements creating for each one the corresponding node and inserts them in the octree. Once the octree is completed, the map is saved under the corresponding octomap type and is ready to be used by the trajectory planning algorithms. The dynamic map update can be translated by the following pseudo code which shares the same functioning as the previous one. It is assumed that the quadrotor is equipped with the necessary sensors to accomplish the detection and is able to continuously track it as long as it stays within range of the sensors. The intruder is represented using a bounded box for simplicity. According to the size of the intruder a box is created around it with a certain dimension and given the boundaries of that box, we again create the necessary measurements and nodes, inserting them in the octree and removing them if the intruder moves to a different position until the intruder falls out of sight. This is done by a second algorithm allowing the update of the static map with the position of dynamic obstacles.

3.3. Offline/Pre-Flight Path Planning

A geometric path from start to goal states can be defined by a continuous function f:

$$f : [0, 1] \to C_{free} \tag{7}$$

subject to start and goal configurations $f(0) = q_{start}, f(1) = q_{goal}$, respectively. C_{free} represents the free space within the three dimensional configuration space defined by $C = \{x, y, z\} = \mathbb{R}^3$.

A trajectory consists of the path plus the additional information on how to transverse it with respect to time. Formally we can define a trajectory as a continuous function g:

$$g : [0, 1] \to C_{free} \tag{8}$$

$$[T_{start}, T_{goal}] \ni t \to \tau \in [0, 1] : q(t) = \gamma(\tau) \in C_{free} \tag{9}$$

The time parametrization (t) of each state s_i in our trajectory implies that one or more considerations need to be made such as:

- **Position**, $q(t)$;
- **Velocity** of a motion, $\frac{d}{dt}q(t)$;
- **Acceleration** of a motion, $\frac{d^2}{dt^2}q(t)$;
- **Jerk** of a motion, $\frac{d^3}{dt^3}q(t)$;
- **Snap** of a motion, $\frac{d^4}{dt^4}q(t)$;

The following scheme illustrates the individual subcomponents that make up this part of the framework and how they are interconnected.

Using a sampling based method (Informed RRT* [5]) , an optimal (length-wise) collision-free path is generated. From this path, a trajectory is formed by feeding the path waypoints to an algorithm based on a minimum-snap trajectory generation [15], in which trajectories are represented as piecewise polynomials and obtained through solving a quadratic programming(QP) problem. The generated polynomial trajectories are smooth and show continuous motion between the different UAV poses (preventing control input oscillations).

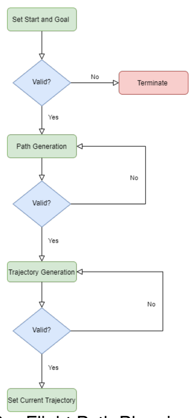

Figure 6: Offline Pre-Flight Path Planning proposed solution.

3.3.1 Informed RRT*

The informed RRT* exhibits (regarding general aspects) a similar behaviour to the basic RRT algorithm and represents an improvement in convergence to the RRT* by implementing a direct sampling of the ellipsoidal heuristic which reduces the size of the available sampling domain. The following pseudo-codes illustrate the functioning of this sampling method. The functions mentioned below are from the implementation in [5]. In a similar way to the basic RRT, a newly random configuration is sampled and we try to connect it to the nearest configuration in the graph. If the connection is successful, it is added to the vertex set V. We attempt to connect this new vertex q_{new} to other vertexes already in V but limited to a defined region in space which corresponds to the space within a ball of radius r_{RRT*}. However, not all connections result in new edges being added since this method avoids adding "redundant" edges by employing a rewiring of the stored tree by considering an additive cost function c_{min} that keeps track of the lowest possible path cost (in terms of length), only allowing vertexes to be added if they lead to a lower path cost. When the algorithm runs for the first time the solution set is empty Q_{soln} and as convention we take its cost as infinite. Therefore, sampling is performed as previously from an uniform distribution U. In the next iteration we have a value different than infinity for this cost and now the sampling algorithm starts sampling within the ellipsoidal domain, leading to a smaller sampling domain. This progressive reduction in the size of the sampling space greatly benefits the convergence of this method relative to the RRT*.

3.3.2 Optimization Method

Due to the concept of differential flatness, control inputs are represented as functions of some trajectory derivatives which is used by the authors of [15] where a method that allows the generation of high-quality minimum-snap piece-wise polynomial trajectories based on jointly optimizing polynomial path segments is developed. Through this method the need for computationally expensive kinodynamic (where kinematics and dynamic constraints restrict the available range of motion during sampling) path planning is eliminated. In [4], the previous work is further expanded to include non-linear optimization features. The authors turn the initial linear optimization problem as in [15] into a nonlinear optimization problem by modifying some optimization variables. The present work makes use of this latter extension of the method.

Formally, any polynomial P with n coefficients can be written as:

$$P(x) = \sum_{k=0}^{n-1} c_k x^k = c_{n-1}x^{n-1} + ... + c_1 x + c_0 \tag{10}$$

where $c_0, c_1, c_2..., c_{n-1}$ are constants and x is the indeterminate or variable. The value of $n-1$ defines the order of the polynomial.

A piece-wise polynomial trajectory consists of M continuous polynomial $P(x=t)$ segments where each individual polynomial is valid from $t=0$ until the segment duration $t = T_{s,i}$ ($i = 1, 2, ..., M$ denotes the corresponding segment). Each segment is composed of D polynomials where D is the number of dimensions of our configuration space. The quadratic cost function for each individual trajectory segment in a given dimension can be written as a combination of the polynomial and associated derivatives:

$$J_{i,d} = \int_0^{T_{s,i}} (c_4)_{i,d} P_{i,d}^{(4)}(t)^4 \, dt \tag{11}$$

Since we want to minimize the snap, all derivatives coefficients up to that derivative and with higher order are set to zero. The quadratic cost for the whole trajectory can now written as the sum of the cost for each segment in each dimension:

$$J_{total} = \sum_{i=1}^{M} \sum_{d=1}^{D} J_{i,d} \tag{12}$$

Specific derivative values (velocity, acceleration, jerk, snap and higher order derivatives) related to polynomial optimization are constraints and re formulated as follows:

$$A_{i,d} c_{i,d} = d_{i,d} \quad , \quad A_{i,d} = \begin{bmatrix} A(t=0) \\ A(t=T_{s,i}) \end{bmatrix}_{i,d} \quad , \tag{13}$$

$$d_{i,d} = \begin{bmatrix} d(t=0) \\ d(t=T_{s,i}) \end{bmatrix}_{i,d} \tag{14}$$

$A_{i,d}$ is a mapping matrix consisting of row vectors t and $\frac{d}{dt}t$ between the coefficients $c_{i,d}$ and derivatives $d_{i,d}$ at the start and endpoints of a polynomial segment. Derivative continuity between segments must be set through all the trajectory segments since a polynomial is infinitely derivable on its interval:

$$A_{T,i} p_i = A_{0,i+1} p_{i+1} \tag{15}$$

where the left side of the equation represents the constraints at the endpoint of segment i^{th} and the right side the same constraints at the beginning of the $i+1^{th}$ segment.

Given the total quadratic cost function J_{total} and all of the imposed constraints $A_{total} p_M = d_M$, the constrained quadratic optimization problem is transformed in [15] into an unconstrained quadratic optimization problem to improve computational efficiency. This done by incorporating the constraints in the objective function as penalization terms (the become "soft" constraints):

$$J_{total} = \begin{bmatrix} d_F \\ d_P \end{bmatrix}^T CA^{-T}QA^{-1}C^T \begin{bmatrix} d_F \\ d_P \end{bmatrix} \tag{16}$$

where d_F are unspecified/free derivatives and d_P specified derivatives. The authors also incorporate a new quadratic term that measures the cost of segment times in the initial cost function:

$$J_{new} = J_{total} + k_T \cdot (\sum_{i=1}^{M} T_{s,i})^2 \tag{17}$$

This new formulation aims to achieve the ideal trade-off between smoothness and trajectory duration. k_T is a parameter that controls this trade-off (higher values place a deeper importance in minimizing trajectory duration) and depends on the specific requirements of each individual case (it is always a good practice to keep a balance between both). Global constraints related to velocity or acceleration are formulated as soft constraints instead of hard constraints (they would lead to an unfeasible amount of this type of constraints):

$$J_{soft} = exp(\frac{x_{max,actual} - x_{max}}{x_{max} \cdot \epsilon} \cdot k_s) \tag{18}$$

x_{max} is the defined maximum value associated with the constraint, ϵ defines the acceptable deviation from the maximum and k_s is a constant that in a similar way to k_T defines the weight of the soft constraint on the overall cost. We can now write the full cost function to minimize in the approach followed in this work as:

$$J = J_{new} + J_{soft,velocity} + J_{soft,acceleration} \tag{19}$$

where a maximum velocity $v < v_{max}$ and maximum acceleration $a < a_{max}$ constraints were imposed. The use of this method was made possible through an implementation of both works available in [12].

3.4. Online/Real-time Path Replanning

The replanning component is activated immediately after the quadrotor takes flight. The **Collision Detection** routine happens at each time step k and checks for the presence of intruders, updates the configuration space C_i and forms a collision query between the affected states in the trajectory by the changes in the configuration space. The most imminent collision is treated with the highest priority and avoidance maneuvers in the form of a check and repair strategy are triggered.

3.4.1 Collision Detection

At any given instance in time t_i, the state of each intruder Obs_i is defined over the next k time steps by extending the current motion from the last known location:

$$Obs_{i+k} = [p_{obs}(t_i) + k \cdot v_{obs}(t_i), v_{obs}(t_i)] \tag{20}$$

This prediction implies that the velocity at that time $v_{obs}(t_i)$ remains the same for the defined time horizon,

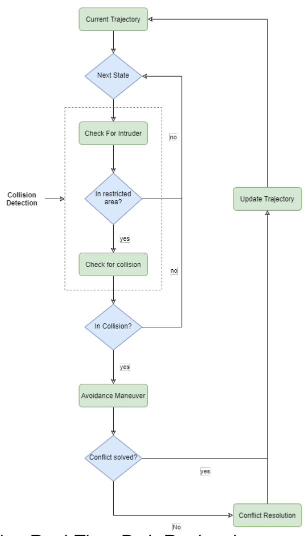

Figure 7: Online Real-Time Path Replanning proposed solution.

which must be as short as possible for this assumption to remain accurate.

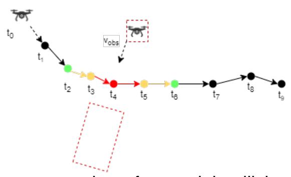

Figure 8: Visual representation of potential collision situation between an autonomous agent and an intruder (bounded by a red box). States (in red) in the agent's trajectory are considered to be in collision course with the intruder, while states near the collision are signaled in orange. Green states represent exit points and possible re-entrance points.

3.4.2 Avoidance Maneuvers

A similar trajectory generation method is employed. However, to decrease as much possible computation time, the non-optimal RRT-Connect coupled with a subsequent path shortening recursive method is employed. This method works like the basic RRT but now grows two trees, one from the start configuration and the other from the goal configuration and attempts to connect them. The shortening algorithm works by iteratively trying to reduce the size of the current path (while maintaining its validity) by attempting connections between randomly sampled points along path segments and not only the available vertexes from the original path.

In the event of a collision (illustrated in figure 8), two different avoidance maneuver approaches are considered:

- The first one consists of reusing part of the original trajectory not affected by the collision. To do this we first define an exit point and a re-entrance point in the current trajectory and then generate a collision-free trajectory between these points through the optimization method mentioned. Additional constraints need to be specified on the exit and re-entrance points according to the values of the derivatives in the original trajectory. At the exit point at we have to guarantee $[v = v_{exit}, a = a_{exit}, j = j_{exit}, sn =$

$sn_{exit}]$ and at re-entrance $[v = v_{re-entrance}, a = a_{re-entrance}, j = j_{re-entrance}, sn = sn_{re-entrance}]$.

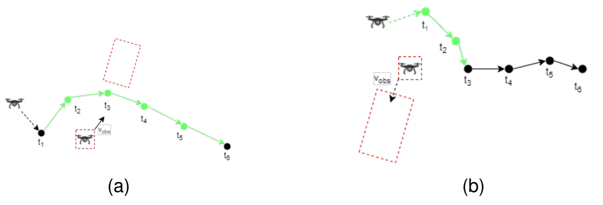

Figure 9: Progression of first type of avoidance maneuver. The transitioning trajectory in green and the feasible portion of the previous trajectory in black.

- In the event that the first method fails to provide a solution by exceeding a defined timeout, another attempt to correct the collision-bound trajectory is made. This time an exit point is chosen the same way as before but the trajectory from this point on is fully discarded and a new trajectory is generated originating at the exit point and ending at the same final state as defined in the original mission objective. As added constraints only the derivatives at the exit point need to be imposed in the newly generated trajectory. The resulting trajectory consists of this plus the remaining of the previous trajectory.

In both approaches, there is a guarantee of a collision free trajectory for an immediate time horizon, but subsequent corrections if a new collision situation develops.

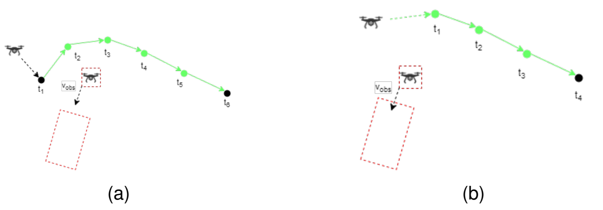

Figure 10: Progression of second type of avoidance maneuver. The transitioning trajectory in green and the feasible portion of the previous trajectory in black.

4. Results & discussion

To validate the developed framework, simulations were performed across various environments. They first set are non-physics simulations where the quadrotor is taken by its center of gravity. Subsequent robust, flexible and accurate computer physics simulations with environment and quadrotor dynamics modelling were constructed.

4.1. Non-Physics Simulation

The resolution of voxel grids is a critical factor for the performance of our proposed method mainly when performing collision checking between sampled points and the environment. The following graph illustrates the decrease in performance when resolution increases.

At $10\ cm$ and even $5\ cm$ we have a good compromise between resolution and a substantial reduction in computational effort compared to the highest resolution. The choice fell on the first.

The simulations were done on a $30 \times 30 \times 4m$ map and the maximum velocity and acceleration limits are set

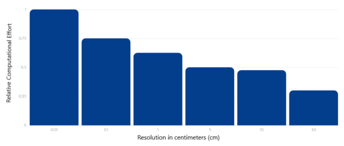

Figure 11: Collision Checking Simulation Results (averaged over 50 simulations).

as $9m/s$ and $9m/s^2$ respectively. Optimization specific parameters were set:

Max Iterations	25000
Objective Function Threshold (f$_{rel}$)	5%
Optimization Variables Threshold (x$_{rel}$)	10%
Time penalty (k$_T$)	2000
Initial Step Size	10%
Tolerance (ϵ)	5%
Sampling Interval (Δt)	0.1s

Table 1: Optimal parameter values for the simulations

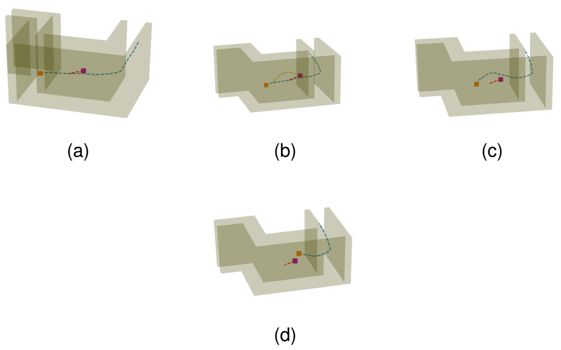

Figure 12: At the moment a potential collision is detected, the first avoidance maneuver approach is used and a transitioning trajectory is successfully created. The resulting trajectory allows the quadrotor to avoid the intruder and it continues its new trajectory towards the destination.

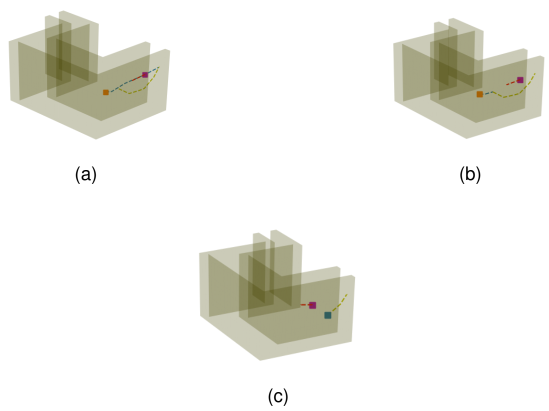

Figure 13: At another instance in the trajectory a new collision is detected. However, now there is no possible states before and after the collision to generate a transitioning trajectory and after failing to apply the first method, the second approach ensues by discarding the previous trajectory and generating a complete new trajectory.

The figures shown are only one instance of comparable simulations tested where the relative velocity between the quadrotor and the intruder was kept within a reasonable range. The maximum velocity of the intruder was of $4m/s$ and the avoidance success rate hovered at around 80% when considering the application of both avoidance approaches. When dealing with intruders with higher velocity values than this, the success rate drops significantly since the duration of the optimization algorithm is more or less fixed for a given number of path waypoints (figure 15).

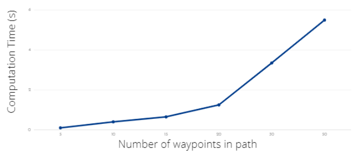

Figure 14: Evolution in computation time of optimization algorithm with the number of path waypoints. The results are averaged over 50 simulations for each.

4.2. Physics Simulation

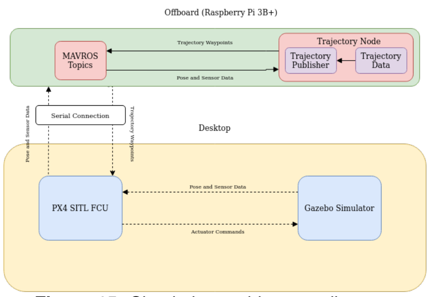

Figure 15: Simulation architecture diagram.

Software-In-The-Loop (SITL) simulations allow for an accurate depiction of a real-world testing without the need for actual hardware. Three major components are outlined: Gazebo simulator, PX4 SITL firmware and the MAVROS[14]/Trajectory nodes. The Trajectory Node consists of the developed algorithms as well as some necessary adjustments implemented running on a Raspberry Pi 3B+ acting as a companion computer. We have a desktop, where PX4 code is running in SITL mode (simulating the flight control unit, FCU, of the quadrotor) while communicating directly with Gazebo simulation environment [8] (simulating the environment dynamics). The companion computer and the desktop are connected via serial (using a USB-to-TTL cable). The Trajectory node publishes trajectory data to the MAVROS node which converts it into the appropriate MAVLink messages and streams it via serial connection to the simulated FCU on the desktop. The FCU receives this, decodes and turns them into the necessary command inputs (through the simulated controllers) which are then streamed via MAVLink to the simulated quadrotor in Gazebo. There is also an exchange of information regarding the quadrotor state back to the Trajectory Node. Despite having piece-wise polynomial trajectories of higher degrees, a controller must be used to drive the quadrotor current state to the desired state. PX4 firmware provides a feedforward cascade position and velocity controller that was used consisting of a proportional position controller (P) and a proportional-integral-derivative controller (PID). In the simulations, the quadrotor used was the already mentioned 3DR Iris which is one of the standard aircrafts already modelled in PX4. The simulations are divided according to the evolution of the framework as described in chapter 3, testing first without avoidance capabilities and then inserting them. They show the deviation error between the commanded position p_{com} that is fed to PX4 controller and the resulting position p_{res} that is exhibited by the quadrotor. The position error (which we want to keep as low and fluctuation free as possible) allows us to check on the potential real-world feasibility of both the trajectory generation method and the obstacle avoidance maneuvers followed. Formally it can be written as:

$$Error = |p_{com} - p_{res}| \tag{21}$$

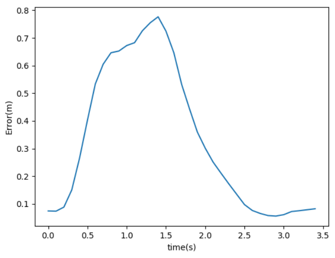

Figure 16: Position error evolution with time.

The first test corresponds to a simulation in which a simple maneuver of overcoming a wall between two points, one from each side. Both velocity and acceleration were set the highest values ($v = 9m/s$ and $a = 9m/s^2$) in which the trajectory generation algorithm can potentially operate according to the paper where the method was first introduced. After setting both values without first tuning the position and velocity controller from the default values, the quadrotor struggled to keep up with the position and velocity commands which lead to high position error translated in erratic behaviour such as hitting the wall or spinning out of control. After adjusting the controller parameters, the best results were achieved in figure 16 where position error reaches a maximum of $0.8m$, an acceptable error specially in spaced outdoor environments. There are only small oscillations and overall the control is smooth. This maximum error is verified when the quadrotor after ascending upwards straight to an altitude above the wall, it suddenly accelerates and this steep change leads to an overshoot in position. The tuned values were kept for the following simulations.

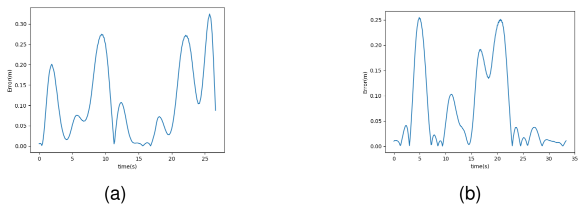

Figure 17: (a)-(c) Maze and Obstacle course gazebo environment, respectively. (b)-(d) Position error evolution with time.

The simulations continued in two distinct environments (a maze like environment and a spaced corridor with obstacles, respectively) and to achieve a better tracking performance, acceleration was limited to $6m/^2$ in order to prevent high overshooting. In both instances, the maximum position error (figure 17) is kept under $0.3m$ which is evidence of a good tracking. The overshooting verified in both instances tends to happen primarily due to changes in direction when turning around obstacles in the first environment and when cutting corners in the second. Nonetheless this is expected behaviour which could be further mitigated by lowering the maximum acceleration and also by tuning optimization parameters such as the time penalty to reduce abrupt changes in velocity.

To test avoidance maneuvers similar simulations to before were outlined. All of them were performed on the spaced corridor with obstacles environment under the same velocity and acceleration conditions.

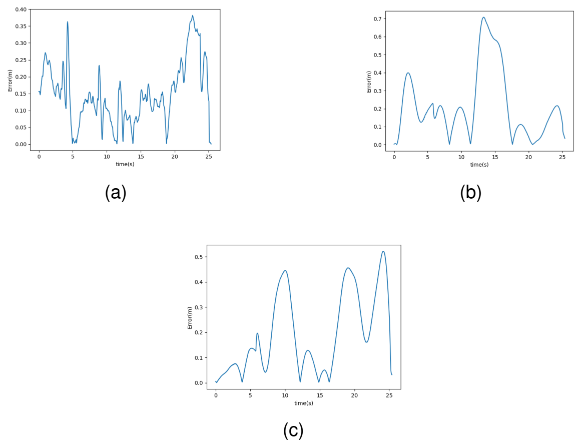

Figure 18: Position error evolution with time for three distinct simulations (a), (b) and (c) on the same environment.

The maximum peaks of position error (figure 18) in all three graphs correspond to situations where avoidance maneuvers were taken and they exhibit a similar behaviour as the situations mentioned before when turning corners or around obstacles. However, these peaks tend to be of a higher magnitude since it is common that the transitioning trajectories generated in avoidance maneuvers are more aggressive with more sudden changes in direction and therefore more steep control inputs. Nonetheless in all cases, when limiting the velocity of intruders to reasonable values (lower than $4m/s$), the maximum error was under $0.7m$ which falls within reasonable values for UAV applications.

5. Conclusions

We focused more on the concept of generating flyable smooth trajectories that require as least of a computational effort as possible. This lead to a two step approach to deal with the problem: an Offline/Pre-Flight path planning and Online/Real Time Path Replanning. To test the capabilities of the developed framework, several non-physics and physics simulations were carried out which showed the effectiveness of the approach. The next step of performing to further validate the results should be Hardware-In-The-Loop (HIL) simulations where actual flight controller unit are used. It would be beneficial to further expand this work to larger quadrotors and other multirotor aircraft. In order to improve tracking performance, a deeper dive in control theory could be done as well as integrating it into the current PX4 code. Integration of detection and tracking of dynamic obstacles capabilities in the real world testing of the framework could also be explored.

References

[1] *OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems*, volume 2, Jan. 2010.

[2] *RT-RRT*: a real-time path planning algorithm based on RRT**, Nov. 2015. doi::10.1145/2822013.2822036.

[3] *UAV Path Planning System Based on 3D Informed RRT* for Dynamic Obstacle Avoidance*, Dec. 2018. doi::10.1109/ROBIO.2018.8665162.

[4] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1872–1878, 2015. doi::10.1109/IROS.2015.7353622.

[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, sep 2014. doi::10.1109/iros.2014.6942976.

[6] Y. Guo, X. Liu, W. Zhang, L. Xuhang, and Y. Yue. Obstacle avoidance planning for quadrotor uav based on improved adaptive artificial potential field. pages 2598–2603, Nov. 2019. doi::10.1109/CAC48633.2019.8996746.

[7] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *CoRR*, 2011.

[8] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004. doi::10.1109/IROS.2004.1389727.

[9] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, USA, 1991.

[10] B. Lindqvist, S. S. Mansouri, A. Agha-mohammadi, and G. Nikolakopoulos. Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles. *IEEE Robotics and Automation Letters*, 5(4):6001–6008, 2020. doi::10.1109/LRA.2020.3010730.

[11] W. Luo, Q. Tang, C. Fu, and P. Eberhard. *Deep-Sarsa Based Multi-UAV Path Planning and Obstacle Avoidance in a Dynamic Environment*, pages 102–111. 06 2018. doi::10.1007/978-3-319-93818-9$_{1}0$.

[12] A. Markus, B. Michael, O. Helen, B. Rik, and P. Marija. mav_trajectory_generation. https://github.com/ethz-asl/mav_trajectory_generation, 2017.

[13] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011. doi::10.1109/ICRA.2011.5980409.

[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. volume 3, Jan. 2009.

[15] C. Richter, A. Bry, and N. Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Apr. 2016. doi::10.1007/978-3-319-28872-7_37.

[16] C. Weller. Drones could replace $127 billion worth of human labor. *https://www.businessinsider.com/drones-could-replace-127-billion-of-human-labor-2016-5*, May 2018.