Online Planner for Multi-Rotors based on Modified RRT and Trajectory-Optimization

António Ramalho, Rodrigo Ventura, Afzal Suleman
antonio.ramalho@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

January 2020

Abstract

In this work it is presented an online trajectory-planner for multi-rotors. The planner is capable of generating trajectories in partially unknown environments with moving obstacles. The developed algorithms are capable of quickly avoid unexpected obstacles, both static and moving. The trajectory planner is based on a modified RRT and a trajectory-optimization algorithm. The cost function used is a combination of trajectory time and energy/fuel consumption in order to allow the generation of optimal trajectories in terms of mission costs. Several simulations are performed and the real-time avoidance capability of the proposed solution is validated. Physics simulations are also performed using Gazebo to show the possibility of using these algorithms for agressive multi-rotor maneuvering.
Keywords: online-planning, multi-rotor, RRT, trajectory-optimization, avoidance

1. Introduction

With the constant development of Unmanned Air Vehicles (UAV) there has been an interest in using their potential for diverse applications. This potential could be further explored if autonomous UAV operation was possible, without the need for a human pilot. There is the challenge of making UAVs safe and capable, in order for them to be allowed to perform autonomously the desired applications. For example, there has been an effort to integrate these vehicles in the non-segregated airspace [1]. There is, however, a gap between the state of the art and these desired capabilities.

Autonomous UAV operation require a diversity of capabilities. To enable the absence of a human pilot, the system requires some capabilities that are in part provided by humans, namely environment perception, motion planning and trajectory execution. In the terrain automobile industry, there has been many advances in the past years. This evolution was partially pushed by technological advancements in areas such as information technology, data analysis, computer vision, etc. [2].

For UAVs this problem is usually simpler. The problem is simpler for UAVs once the environment is not nearly as populated as for the autonomous cars. A simple approach is then desirable to solve this problem. Not only because the problem is not as complex as the one for ground vehicles but also due to the computational limitations related to the limited payload of aircraft.

Autonomous systems are however complex, requiring a vast set of capabilities. This work will be focused in only a portion of that: the generation of collision free trajectories in real-time.

1.1. Literature review

In robotics, path planning in unknown environments is a subject studied for many years. In path planning problems for UAVs most of the existing solutions, to deal with unknown obstacles, require a dedicated ground station [3] [4] . This is due to the low computational power available on on-board computers in small UAVs. On-board path planning for small UAVs has been proposed in [5], using a field programmable gate array (FPGA) chip. In this work, a solution was created in which genetic algorithms are used to compute a path plan based on a provided environment and set of start and goal configuration. In [6], an online path planning algorithm for cooperative aircraft is developed and implemented in relatively powerful on-board computers. In two works by Ioannis K. Nikolos et al. [7] [8], an evolutionary algorithm is developed which allows online path planning in unknown environments. This work, however, considered static environments and it was never implemented in a vehicle.

Recently, in the end of 2018, Marco Pavone et al. [9] developed an online path planning algorithm that proved to be able to compute trajectories in real-time in partially unknown environments with moving obstacles. The authors claim it was the first

experimented algorithm, for multi-rotors, with such capabilities. The algorithms were, however, ran in a ground station. In another interesting work online path planning was accomplished with the environment being acquired by a depth camera [10].

The existing work regarding real-time path planning for multi-rotors in partially unknown environments with moving obstacles is not abundant and several ways of approaching the problem can be explored.

1.2. Objectives

In this work it is proposed a real-time path-planning algorithm for UAVs. The main goal is to make an algorithm capable of planning aggressive trajectories in partially unknown environments with moving obstacles. The algorithms should be able to quickly react to new detected obstacles (including moving obstacles) and safely avoid them by quickly adjusting the trajectory without having the need to stop the UAV for such.

In the field of trajectory planning it is also desirable to compute optimal trajectories. Optimality will be defined in terms of mission costs (a combination between mission time and fuel/energy consumption). The algorithm will have anytime capabilities: it is possible to quickly generate a sub-optimal trajectory and then optimize it for a given period of time. An optimal trajectory is computed if enough computation time is used.

To enable path planning in real time a simplified dynamic model for multi-rotors will be used. In order to validate that the computed trajectories are suitable for aggressive multi-rotor maneuvering a simulation will be performed and the position error of the multi-rotor, relatively to the provided references, will be stored through the simulation and analysed afterwards.

2. Background

2.1. Trajectory-planning

A path for a robot can be described as a series of configurations that the robot should assume to go from a start to a goal configuration. The concept of configuration is described in [11] as a set of independent parameters that describe the position of every point in a body.

The terms path and trajectory are often used interchangeably. Formally, however, these terms are distinct. Path refers to, as mentioned before, all the consecutive configurations that a robot has to assume in order to go from a start to a goal configuration. In a trajectory, however, there has to be a timing law associated to the configurations that define the trajectory, for example having an associated speed and acceleration or time instant to each configuration in the trajectory [12].

2.2. IPOPT

In the present work, the used solver is an implementation of an interior point with a filter line-search method called IPOPT [13]. The algorithm is complex and uses a primal-dual barrier method, originally developed by Andreas Wchter in [14], inspired in previous work by Fiacco et al. [15].

In sequential quadratic programming (SQP) there is the need to identify the active bound constraints. Barrier problems avoid this problem by replacing the constraints for logarithmic terms in the cost function. Thus being said, barrier methods solve optimization problems of the type:

$$\begin{array}{lll} \text{min:} & f(x) & \\ \text{s.t.:} & c(x) = 0 & \\ & x_i > 0 & , i \in \mathcal{I} \end{array} \tag{1}$$

where $\mathcal{I}$ defines the set of indexes of bounded variables. By converting the problem into the following one, stated in Equation 2, called a *barrier problem*:

$$\begin{array}{ll} \text{min:} & \phi_\mu(x) = f(x) - \mu \sum_{i \in \mathcal{I}} ln(x_i) \\ \text{s.t.:} & c(x) = 0 \end{array} \tag{2}$$

where μ is the barrier parameter, which is greater than 0. This formulation imposes that $x_i > 0$ for $i \in \mathcal{I}$ because, as x_i tends to 0, the value of $\phi_\mu(x)$, the barrier function, tends to infinite. The formulation in Equation 2 matches the problem defined in Equation 1 for $\mu \to 0$. For $\mu > 0$, it becomes clear that the solution never lies in the boundary of the problem, instead it always lies in the interior of the desired design variable region. For this reason, the barrier methods are also called interior point methods. In these methods, multiple sub-problems (barrier problems) are solved for decreasing values of μ.

IPOPT allows solving general dual-barrier problems of the type:

$$\begin{array}{lll} \text{min:} & f(x) & \\ \text{s.t.:} & c(x) = 0 & \\ & x_{il} < x_i < x_{iu} & , i \in \mathcal{I} \\ & d_l < d(x) < d_u & \end{array} \tag{3}$$

where x_{il} and x_{iu} are lower and upper boundaries for x_i. d_l and $d_j(x)$ are lower and upper boundaries for the inequality functions $d(x)$. In order to convert the typical barrier problem stated in 1 into the more general form shown in 3, it is required to, instead of using the variables $v_i = \frac{\mu}{x}$, create the variables $v_{il} = \frac{\mu}{x_i - x_{il}}$ and $v_{iu} = \frac{\mu}{x_{iu} - x_i}$. Doing this, the equalities forcing these new definitions of v_{il} and v_{iu} become $v_{il} * (x - x_{il}) = \mu$ and $v_{iu} * (x_{iu} - x_i) = \mu$ respectively.

In order to explain how IPOPT deals with inequalities defined by general functions $d(x)$, it is defined an isolated inequality:

$$d_{jl} < d_j(x) < d_{ju} \tag{4}$$

where $d_j(x)$ represents the jth inequality function and d_{jl} and d_{ju} represent the respective lower and upper bounds. These inequalities are changed to a set of equalities and inequalities shown in Equation 5 by introducing slack variables s_j:

$$\begin{aligned} d_j(x) - s_j &= 0 \\ d_{jl} < s_j &< d_{ju} \end{aligned} \tag{5}$$

The equalities are treated then as regular equalities. The inequalities $d_{jl} < s_j < d_{ju}$ are then imposed in the same way as the inequalities $x_{il} < x_i < x_{iu}$.

2.3. Trajectory Optimization

Optimization techniques can be used to compute trajectories. However, it is required to state an optimization problem in such a way that its solution can be used to compute a trajectory. In [16], an overview of trajectory optimization techniques and applications is performed. Besides giving an overview in numerical methods for optimization, in [16] a series of traditional formulations of trajectory planning problems as optimization problems is given. Usually, in these problems, the system dynamics is imposed using equality constraints such as:

$$\dot{\sigma} = f(\sigma, u) \tag{6}$$

where σ represents the system state and u the control inputs. Nevertheless, these equalities must be formulated in a discrete-time domain. To achieve this, the derivatives $\dot{\sigma}$ are usually taken using finite differentiation.

Trajectory optimization has been applied also to multi-rotors. In [9], optimization is only used to compute splines joining way-points in a previously computed trajectory. On other works, such as [17, 10], trajectory optimization is used to completely compute trajectories from naive first iterations. The approach taken to describe the trajectory varies. In [17], the trajectories are discretized into a series of way-points, unlike in [10] where trajectories are described as a series of high order splines, which parameters are optimized.

To sum up, in order to perform trajectory optimization it is required to choose a finite set of parameters to describe the trajectory, such as a set of way-points or a sequence of control inputs, and then formulate the problem in such a way that the system dynamic and actuation limits of the problem are respected. Finally, a cost function must be chosen to minimize.

2.4. Multi-rotor-rotor dynamics

It is now introduced the concepts of inertial frame and body frame. The inertial frame is, as the name suggests, a frame that moves in a constant speed (or static), where the z_w axis is aligned with the vertical, pointing upwards. The body frame is fixed to the multi-rotor and the z_b is perpendicular to the rotor plane and points up when the multi-rotor is hovering. A representation of these frames is shown in Fig 1.

As stated in [18], the UAV state can be written as the position of its center of mass (x, y and z), the speed of the center of mass ($\dot{x}$, $\dot{y}$ and $\dot{z}$), the roll pitch and yaw angles (ϕ,θ and ψ) and the angular velocities around the body axis x_b, y_b and z_b (p, q and r respectively).

$$\sigma = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^T \tag{7}$$

The inputs are considered to be the resulting force along the z_b component of the body frame of the vehicle (u_4) and the moments along x_b, y_b and z_b of the vehicle (u_1, u_2 and u_3). This thrust and moments are assumed to be given linearly from the square of each rotor speed (ω_1^2, ... , ω_4^2). It is also assumed that the rotor speeds can be directly controlled. Let $u = [u_1, u_2, u_3, u_4]^T$ and $\omega^2 = [\omega_1^2$, ... , $\omega_4^2]^T$. For multi-rotors with more rotors, the control input u will be the same (thrust along the axis perpendicular to the rotors plane z_b and moments applied to the center of mass), once a series of rotors that provide thrust in the same direction are only capable of generating force along that same direction. To map from the squared rotation speed of n rotors ω^2 into control inputs u, as described before, it is used a matrix usually called allocation matrix A $4 \times n$, such as in Equation 8.

$$u = A[\omega_1^2, ..., \omega_n^2]^T \tag{8}$$

It is clear that for more than 4 rotors the matrix does not have an inverse. For that reason, when it is desired to map u into $[\omega_1^2, ..., \omega_n^2]$, with $n > 4$, it is necessary to add some extra constraints to the system.

2.5. Quad-rotor differential flatness

In [18], it is proven that a quad-rotor is a differentially flat system. This implies that it is possible to compute the control inputs and the UAV configuration from a trajectory defined in the flat output space [19].

The flat outputs chosen in this work were the vehicles center of mass coordinates in the inertial frame (x_w, y_w and z_w) and the yaw angle (ψ). The trajectories will be computed for the center of mass position in the inertial frame and the yaw angle (ψ) will be considered as constant and equal to 0, there is however the freedom to control the yaw angle

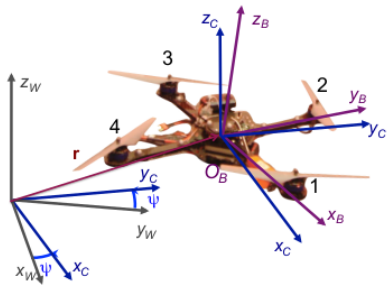

Figure 1: Inertial frame (subscript W) and body frame (subscript B) of a multi-rotor. Taken directly from [18]

which might be useful for, for example, directing a camera fixed to the UAV.

$$w = [x, y, z, \psi]^T \tag{9}$$

The authors provide expressions for computing the quad-rotor state σ and inputs u from the flat output variables and their derivatives up to the fourth derivative. This means that it is possible for the quad-rotor to follow the computed trajectory in the flat output space (position and yaw) as long as the inputs are not saturated.

2.6. Approximated dynamics

In this work, like in some others [9, 17], a first approximation is made, considering that the acceleration of the vehicle can be directly controlled (ignoring attitude information). In such approximated system, the UAV can be described as a point with a mass, and its state considers only the position of the center of mass and the speed. This approximation, however, leads to generated trajectories that can not be followed by the real quad-rotor. Discontinuities on the acceleration direction would translate into discontinuities in the vehicle attitude (the z_b axis of the body frame must be aligned with the acceleration vector at all times). If it is desired to make use of the differential flatness propriety, computing directly the control inputs from the flat output variables (from a trajectory in 3-dimensional space with freedom to control the yaw angle), it is required to have information on the flat output variables up to the 4^{th} derivative (snap and 4^{th} derivative of yaw angle with respect to time). This can be achieved by smoothing the trajectory, as it is made in [9].

In the present work, the computed trajectories only provide information up to the 2^{nd} derivative of the position. It is then desirable to have a suitable controller for the case, once the inputs can not be directly taken from the computed trajectory. The chosen controller is based on the work by Lee et al. [20]. Further on this work, this approximation is validated using experimental data.

3. Proposed Solution

The proposed solution consists in an incremental optimization approach. The core idea consists in continuously optimizing the trajectory while the UAV executes it. This however requires some additional logistics:

- It is required to define at each time which part of the trajectory should be optimized
- It is required to have a first iteration for the trajectory optimizer
- It is required to have a tool that prevents the optimizer to get trapped in unfeasible local minima

3.1. Example

A series of figures (Fig 2 - 8) will now be presented to exemplify this concept.

In most of the flight part of the trajectory is optimized while the aircraft flies. The part of the trajectory that is executed while the optimization process runs must be fixed. This process is exemplified in Figures 2 - 4.

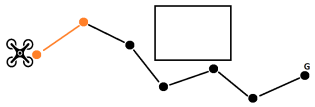

Figure 2: Part of the trajectory fixed (orange) in an initial trajectory computed by an RRT

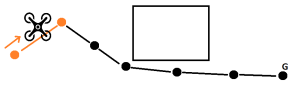

Figure 3: Trajectory is optimized while UAV flies

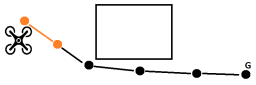

Figure 4: New part of the trajectory is fixed and process re-starts

In the case that there is some non-feasible portion in the trajectory (a part of the trajectory exceeds the maximum allowed speed or acceleration or there are collisions with obstacles) the trajectory is only optimized around the unfeasible portion (Fig 5 - 6).

If the optimizer can't make the trajectory feasible in a chosen, limited, number of increments (in this work it will be called the maximum number of failures ()*maximumFailCount*) the RRT algorithm

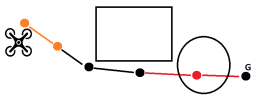

Figure 5: Unknown obstacle detected in the trajectory

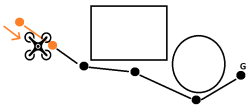

Figure 6: Trajectory optimizer adjusts the trajectory avoiding the obstacle

regrows a trajectory around the unfeasible portion (Fig 7 - 8). This might happen if the trajectory falls into an unfeasible local minima.

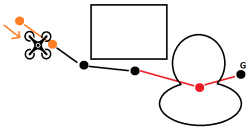

Figure 7: Unknown complex obstacle detected in the trajectory

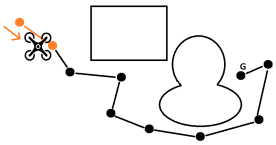

Figure 8: The maximum number of failures is reached and a trajectory is regrown around the obstacle

Finally, in the case that the non-feasible part of the trajectory corresponds to a time interval that is very close to the current time t_C a different approach should be taken. In this scenario the optimization increment might not be performed fast enough to avoid a collision. If this is the case the UAV enters an "emergency mode" which consists in stopping to follow the previous trajectory and recomputing the trajectory from scratch.

Concerning the first iteration given to the optimizer it will be used a modified RRT. This RRT will also be used to regrow critical parts of the trajectory whenever it gets trapped in unfeasible local minima. The trajectory is computed from the UAV to the goal in the beginning of the mission and every time the robot discards the old trajectory when entering the "emergency mode".

This planner requires access to two methods: one for getting the UAV position, one for getting the environment map. The behaiviour of the planner can be explainned using a state machine, represented in Fig. 9.

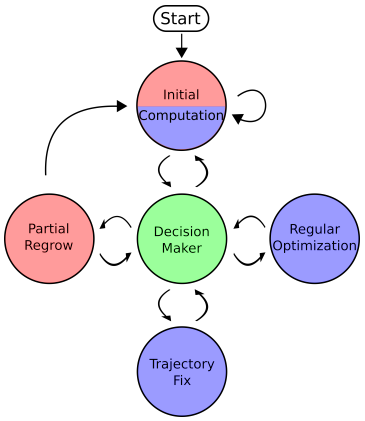

Figure 9: State machine describing the real-time trajectory planner. Red, blue and green states use methods from the **RRT**, **Optimizer** and **Feasibility Tester** scripts respectively.

3.2. Initial Computation

When the algorithm starts (and in some other situations) it enters the *Initial Computation* state. When this state is entered the UAV position is used as the starting state and if there is a trajectory, it is discarded. Then, a trajectory is computed using an RRT. If the RRT fails to compute a trajectory in a given time (*initialRRTTimeOut*) this state is restarted. If, on the other hand, the RRT can successfully compute a trajectory, a defined portion of that same trajectory is optimized for a defined time (*initialOptimizationTime*). This is where the anytime capability of the algorithm is evident, the *initialOptimizationTime* parameter controls the equilibrium between computational time and trajectory quality.

3.3. Regular Optimization

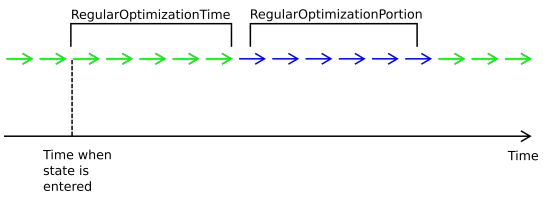

Figure 10: Portion of trajectory that is optimized in the *Regular Optimization* state is represented as a series of blue arrows. The first and last blue arrows are treated as a local start and a local goal (these waypoints are fixed during the optimization).

.

In this state a portion of the trajectory ahead of the UAV is optimized for a period of time defined by the parameter *regularOptimizationTime*. When this state is entered the algorithm chooses

a portion of the trajectory between the point *regularOptimizationTime* ahead of the current time and another point ahead defined by the parameter *regularOptimizationPortion*. The portion of the trajectory which is optimized is shown in Figure 10. This choice assures that when the optimization is complete and the trajectory is updated, the UAV is not executing the updated part of the trajectory.

3.4. Trajectory Fix

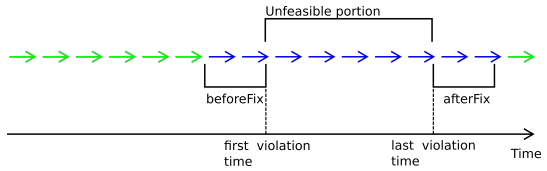

Figure 11: Portion of trajectory (blue) that is optimized in the state *Trajectory Fix*. The first and last blue arrows are treated as a local start and a local goal (these way-points are fixed during the optimization.

The state *Trajectory Fix* is called when there is an unfeasible portion of the trajectory. It optimizes a portion of the trajectory for a certain time (*trajectoryFixTime*) part of the trajectory is controlled by the parameters *beforeFix* and *afterFix*. The trajectory is optimized between the time of the first violation minus *beforeFix* and the time of the last violation plus *afterFix*. Figure 11 illustrates how this portion is defined.

3.5. Partial Regrow

This state is called when the planner is trapped in an unfeasible local minima. Unlike the other states, where the trajectory is segmented based on time, this method separates the trajectory based on distance. It regrows the trajectory between the state that is at a distance *beforeRegrow* before the first violation index until *afterRegrow* after the last violation index, as shown in Figure 12. The time-out associated with this state is called *regrowTimeOut* and, if it is not possible to compute a trajectory within this time, the algorithm jumps to the *Initial Computation* state.

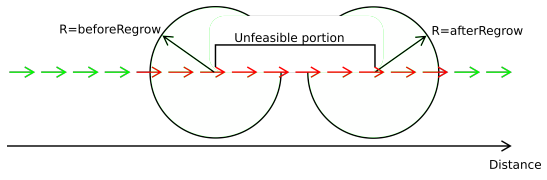

Figure 12: Portion of trajectory (red) that regrown in state *Partial Regrow*. The first and last red arrows are treated as a local start and a local goal for the RRT algorithm.

3.6. Decision Maker

The process behind the *Decision Maker* defines the strategy of this algorithm. Its working principles will now be explained.

This state doesn't have an associated timeout. Instead, when the algorithm enters this state it is quickly set to another state. Basically, this state is responsible for determining the future state of the algorithm. A parameter called *failCount* is set to 0 after the initial computation. In this state, it is checked if there are any violations of the constraints in the trajectory (bad kinematics, maximum speed exceeded, maximum acceleration exceeded or too close to an obstacle). If there isn't any violation, then *failCount* is set to 0 and the algorithm jumps to the *Regular Optimization* state. If there is a failure, it is checked if the first violation time is closer than a defined *criticalClearence*. If it is then the UAV is commanded to stop, *failCount* is set to zero and the algorithm jumps to the *Initial Computation* state. This represents an emergency stop of the UAV, to deal with critical situations. If the first violation time is after the defined *criticalClearence* than the *failCount* is incremented. If the *failCount* reaches a defined maximum (*maximumFailCount*), the *failCount* is set to 0 and the algorithm jumps to the *Partial Regrow* state. Otherwise the algorithm jumps to the *Trajectory Fix* state.

4. Proposed RRT algorithm

Rapidly exploring Random Trees (RRTs) where first introduced by Lavalle as a family of randomized planners [21] in 1998. These algorithms are sample based planners. An RRT algorithm grows a tree in the configuration space by randomly sampling configurations and then adding them to the tree. In path planning applications when the algorithm starts the tree contains only the start configuration, the tree is then grown until it contains also the goal configuration. There is a vast set of RRT variations in the literature, an overview won't, however, be presented in this work.

An RRT which computes paths with a maximum curvature limit is presented. This algorithm was also empowered with the capability to plan in time-dependent environments. In this work, unlike it is done in most of the literature, the sequence of waypoints that describe the trajectory is not defined by the vertices of the tree. **The waypoints are represented by the edges in the trajectory**. The middle point of the edge represents the position, while the alignment of the edge represents the direction of the speed.

4.1. Acceptable angle between edges

It will now be defined the maximum allowed angle between consecutive edges as a function of the robot minimum curvature radius. Let d be the edge

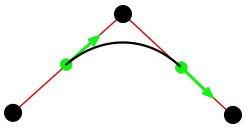

Figure 13: The green points and arrows represent the robot's position and speed respectively. The black points represent the vertices of the RRT.

length, R_{min} be the robot minimum curvature radius and α be the angle (maximum angle) between consecutive edges. The angle α will be given in rad as:

$$\alpha = \pi - 2 * arctan\left(\frac{R_{min}}{d/2}\right) \tag{10}$$

It is obvious that the angle α can only take values between 0 and π, in this range the function $cos(\alpha)$ is always decreasing, therefore limiting a maximum angle between two edges is the same as limiting a minimum $cos(\alpha)$. Let now $\vec{e_1}$ represent one edge (position of vertex k minus position of vertex k-1) and $\vec{e_2}$ represent its consecutive edge (position of vertex k+1 minus position of vertex k). We can state that an edge $\vec{e_2}$ is acceptable after an edge $\vec{e_1}$ if and only if:

$$\frac{\vec{e_1} \cdot \vec{e_2}}{|\vec{e_1}| * |\vec{e_2}|} \geq cos(\alpha_{MAX}) \tag{11}$$

4.2. Maximum curvature expansion

Sometimes a new vertex is created in such a position that it is not possible to create an edge from the previous vertex that is aligned with the new vertex (when the condition in equation 11 is not respected), such as in Figure 14.

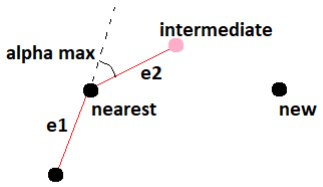

Figure 14: The new vertex is in such a position that it is not possible to expand the tree directly towards it, the tree is expanded than using a maximum curvature expansion.

In those cases an expansion is made in which the angle between the newly added edge and the edge that leads to that same edge is the maximum allowed angle α_{MAX}. In this work this sort of expansion of the tree is called maximum curvature expansion.

For creating paths that respect the minimum curvature radius constraint, the presented algorithm expands the tree towards the random configurations using successive maximum curvature expansions, until it is possible to expand the tree directly towards the random configuration.

4.3. Moving obstacles

To allow this RRT to plan trajectories in environments with moving obstacles a time instant must be associated with each vertex on the tree. For that, each time a vertex is added to the three a time instant is associated with that same vertex. This time-instant is simply the sum of the time-instant associated with the vertex to which the new vertex is connected, plus the distance between these vertexes, divided by the speed of the multi-rotor.

4.4. Enhancement step

As it is known trajectories computed by the RRTs might be very sub-optimal. Optimal RRT related algorithms require often great computational times like it was mentioned before. An enhancement step is now proposed. This enhancement step allows the solution computed by the modified RRT to be enhanced in a very significant way without requiring much computational time.

The enhancement step consists in trying, from each vertex in the RRT, to reach directly another vertex, as further in the trajectory as possible. A scheme is now presented in order to demonstrate this concept in Figure 15:

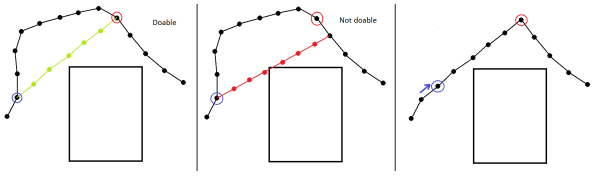

Figure 15: Consecutive steps in the enhancement algorithm.

Figures 16 and 17 show some examples of the computed trajectories before and after the enhancement step.

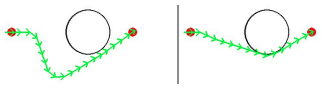

Figure 16: Example of the trajectory before (on the left) and after (on the right) the trajectory enhancement

The algorithms for enhancing the trajectory were performed after growing the RRT 100 times in each environment. Let environment 1 and 2 refer to the environments shown in Fig. 16 and 17 respectively.

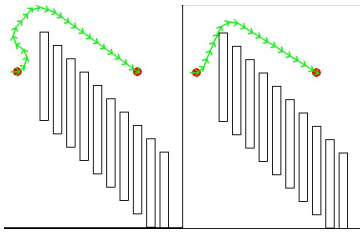

Figure 17: Example of the trajectory before (on the left) and after (on the right) the trajectory enhancement

Let pseudo-sub-optimality (pso) refer to the relation between the trajectory length and the minimum possible length if there were no curvature constraints. The pso is computed using the expression:

$$pso = \frac{length - optimalLength}{optimalLength} \times 100\%$$

Where *length* is the length of the trajectory and *optimalLength* the length if there was no curvature constraints. The results are presented in Tables 1 and 2:

Env. 1	time (s)	length	pso
First trajectory	0.028	474	38.6%
Enhancement	0.026	376	9.64%
Final	0.054	376	9.64%

Table 1: Results of the application of the RRT algorithm and enhancement step to environment 1

Env. 2	time (s)	length	pso
First trajectory	0.004	435	50%
Trajectory enhancement	0.013	308	6.21%
Final	0.017	308	6.21%

Table 2: Results of the application of the RRT algorithm and enhancement step to environment 2

It is possible to observe from Tables 1 and 2 that the enhanced trajectories have a length significantly closer to the optimal length.

5. Trajectory Optimization

It is important to state all the approximations and assumptions made for this problem.

- The multi-rotor is approximated to a point
- The attitude is, at this point, ignored
- It can move in any direction
- It has limited acceleration
- It has limited speed
- The multi-rotor state σ is defined by the position and speed of it's center of mass.

5.1. Nomenclature

It will be now introduced some nomenclature that will be used in this section. $\vec{\sigma} = [\sigma_0, ..., \sigma_{N-1}]$ is the vector with all the N robot states along the trajectory, excluding the start and the goal states. σ_s and σ_g are the start and goal states respectively and these will not be subjected to optimization. Also $\sigma_i = [\vec{p_i}, \vec{v_i}]$, where $\vec{p_i}$ and $\vec{v_i}$ corresponds to the position and speed vector of the robot in a given moment. This means that $\sigma_i \in \mathbb{R}^4$ for a bi-dimensional environment and $\sigma_i \in \mathbb{R}^6$ for three-dimensional environment (e.g. UAV). Let also $p_{i,j}$ represent the jth component of the robot position in the ith state and $v_{i,j}$ represent the j component of the robot speed in the ith state. For example $p_{i,z}$ represents the z component of the robot position in the ith state and $v_{i,y}$ the presents the y component of the robot speed in the ith state. Basically: $\sigma_i = [\vec{p_i}, \vec{v_i}] = [p_{i,x}, p_{i,y}, p_{i,z}, v_{i,x}, v_{i,y}, v_{i,z}]$ (in 3 dimensional space).

It will be assumed that the robot acceleration $\vec{a}$ between two consecutive states is constant. Each state σ_i represents the state of the robot at the time $t = t_s + (i+1) * \Delta t$ where t_s is the start time. Therefore, the total trajectory time is given simply by $t_{total} = (N+1) * \Delta t$ (one time step from start to σ_0, N-1 time steps between the N states and one time step between σ_{N-1} and the goal).

Considering this, the variables subjected to optimization will be $x = [\Delta t, \sigma_0, ..., \sigma_{N-1}]$. If it is written in the form of a vector of scalars (in 3 dimensional space):

$$x = [\Delta t, p_{0,x}, p_{0,y}, p_{0,z}, v_{0,x}, v_{0,y}, v_{0,z}, ...$$
$$..., p_{N-1,x}, p_{N-1,y}, p_{N-1,z}, v_{N-1,x}, v_{N-1,y}, v_{N-1,z}]$$

This will be the design variables of the problem, it is essential for the good comprehension of the following topics. The formulation of the optimization problem will be described in the following sections.

5.2. Cost function

The cost function chosen was a linear combination between the trajectory time $f_T(x)$ and energy consumption $f_F(x)$. The scalar weights k_T and k_F allow to tune the relevance given to each of these costs. Such cost functions are suitable, for example, for minimizing the mission related costs. The cost function is then:

$$f(x) = k_T * f_T(x) + k_F * f_F(x) \qquad (12)$$

The time component is the total trajectory time, which is given as $(N+1) * \Delta t$ where N represents the number of states subjected to optimization.

To model the energy consumption for the multi-rotor it was used closed form expressions from the work by Marins et al. [22].

5.3. Kinematics

The algorithm must respect the kinematics of the problem. It is assumed that the acceleration between to consecutive states σ_i and σ_{i+1} is constant. It is also assumed that the time elapsed for the robot to move from one state to another is Δt. With these assumptions, the position $\vec{p}_{i+1}$ should be given by:

$$\vec{p}_{i+1} = \vec{p}_i + \frac{\vec{v}_{i+1} + \vec{v}_{i+1}}{2}\Delta t \tag{13}$$

To write Equation 13 in the form $c(x) = 0$ the right side of Equation 13 is subtracted in both sides of the equation.

5.4. Maximum speed and maximum acceleration

The maximum speed constraint can be written as:

$$|\vec{v}_i| < v_{MAX} \tag{14}$$

Where v_{MAX} is the maximum allowed speed. The maximum acceleration constraint can be intuitively stated as:

$$|\vec{a_i}| < a_{MAX} \Leftrightarrow \frac{|\vec{v}_{i+1} - \vec{v}_i|}{\Delta t} < a_{MAX} \tag{15}$$

Where a_{MAX} is the maximum allowed acceleration. To keep the derivatives simple, the form chosen for this inequity to be written was:

$$|\vec{v}_{i+1} - \vec{v}_i| < a_{MAX}\Delta t \tag{16}$$

In order to create a constraint in the form $d(x) < 0$ the right side of the inequality stated in Eq. 16 is subtracted to both sides of the inequation.

5.5. Obstacle clearance

It is now required to define the signed distance of a point to a convex obstacle. Let this signed distance $s(\mathcal{O}_k, \vec{p}_i)$ represent the distance from a point $\vec{p}_i$ to the closest point on the surface of a convex obstacle $\mathcal{O}_k$, *this distance is negative if the point* $\vec{p}_i$ *is inside the obstacle* $\mathcal{O}_\mathcal{K}$.

The inequity that assures that the UAV is at least d_{safe} away from any obstacle is

$$s(\mathcal{O}_k, \vec{p}_i) > d_{safe} \tag{17}$$

This formulation for the obstacle clearance constraints was formulated based on [23]. The signed distance is however, at the moment of writing, only available for spheres and cuboids in 3 dimensional environments and circles and rectangles in 2 dimensional environments. The algorithm also supports moving spheres (and circles). It can be expanded for general convex obstacles using, for example, the GJK and the expanding polytope algorithms [24] [25]. It is now necessary to limit the maximum time-step between waypoints to avoid the trajectory to "jumb over" obstacles, like it is shown in Figure 18

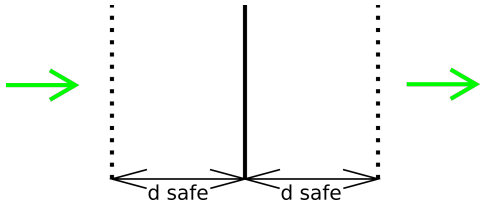

Figure 18: Trajectory segment between two waypoints (green arrows) crosses an obstacle (black line) without any of the waypoints being closer than d_{safe} to the obstacle.

To enable the usage of bigger trajectory segments, which reduce the number of way-points and increases the algorithm performance, new constraints were imposed. These new constraints assure that a number of intermediate points (equally spaced points in each trajectory segment between way-points) are not in collision with any obstacle.

To validate the performance improvement, the algorithms were used to optimize a trajectory from a first iteration (shown in Figure 19) to a local minima. This optimization was repeated replacing $M-1$ out of M waypoints by intermediate collision checking points, as shown in Fig. 20.

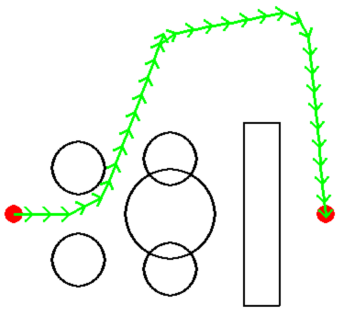

Figure 19: Initial trajectory, to be optimized

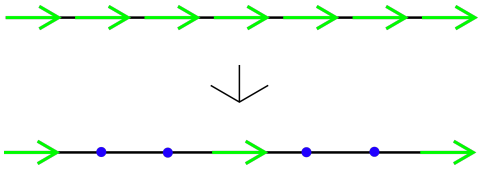

Figure 20: Trajectory before (top) and after (bottom) exchanging waypoints (green arrows) for intermediate collision checking points (blue circles). In this illustration the number of intermediate points per trajectory segment chosen was 2 (M=3).

After multiple simulations it is possible to observe that the computational time is smaller if some way-points are replaced for these intermediate collision checking points. The computational time, however, increases if the number of replacements is further increased. These results are shown in Table 3.

$M-1$	mean (s)	variance (s^2)
0	0.91429	0.00141
1	0.70004	0.00007
2	0.44371	0.00046
3	0.50951	0.00020
5	0.85712	0.00011

Table 3: Computational time until local-minima is reached, for different numbers of intermediate points per trajectory segment $M-1$, using 10 runs for each.

5.6. Problem formulation

The problem formulation can finally be stated in a simplified intuitive way:

$$\begin{aligned} min \quad & k_T * f_T(x) + k_F * f_F(x) \\ s.t. \quad & \vec{p}_{i+1} = \vec{p}_i + \frac{\vec{v}_{i+1}+\vec{v}_{i+1}}{2}\Delta t \\ & |\vec{v_i}| < v_{MAX} \\ & |\vec{v}_{i+1} - \vec{v_i}| < a_{MAX}\Delta t \\ & s(\mathcal{O}_k, \vec{p}_i) > d_{safe} \\ & s(\mathcal{O}_k, \vec{p}_{inter(i,m)}) > d_{safe} \end{aligned} \tag{18}$$

In Equation 18 $\vec{p}_{inter(i,m)}$ represents the mth intermediate collision checking point for the trajectory segment that joins two consecutive way-points (σ_i to σ_{i+1}). $s(\mathcal{O}_k, \vec{p}_i)$ represents the signed distance between a point a point $\vec{p}_i$ and an obstacle $\mathcal{O}_k$.

6. Simulation

In this chapter several simulations, and their respective results, are presented. It will be explained how a simplified simulation framework was developed to evaluate the performance of the algorithm in real time. Then several results are presented showing the capability of the algorithm to adapt to changes in the environment and deal with other aircraft in real-time.

To validate the usage of the algorithms for real-vehicles, realistic physics simulations were performed using Gazebo. The used plugin and controller are briefly explained. A simulation is analysed to evaluate the capability of the simulated multi-rotor to follow the desired trajectories.

6.1. Real-time avoidance

A simplified simulation was performed to validate the real time performance of the algorithms. A simplified model of a UAV was created, this model consisted of a point with a given mass to which the control inputs were applied as forces. A simple controller was developed to transform the reference acceleration, speed and position into force inputs for the model. The Figures 34-45 show the results for a real time testing of the algorithms in a complex indoor environments with unexpected obstacles appearing in the map as the model follows the trajectory.

Real time avoidance was also performed using moving obstacles. For avoiding non-cooperative intruders the algorithm assumes that these intruders travel at constant speed. Once the trajectories of these intruders might be unpredictable, the safety distance that the algorithm considers for these intruders is 3 times the one considered for static obstacles and cooperative intruders. The algorithm was also empowered to be able to simulate sensor delay.

Several simulations were performed to validate the capability of avoiding moving intruders. For the first test the aircraft flies through a map of known obstacles. It was possible in this simulation to manually insert moving obstacles. This moving obstacles would start in a desired position at the top of the map and they would move (at constant speed) towards the bottom of the map. These moving obstacles are represented as circles. An avoidance of an unexpected moving obstacle is shown in Figures 21 and 22.

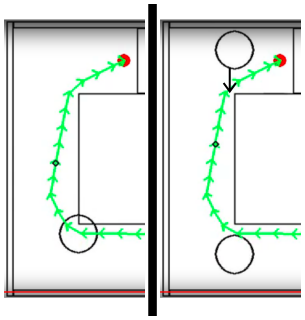

Figure 21: New obstacle appears, moving from the top to the bottom of the figure. (time=1m13s)

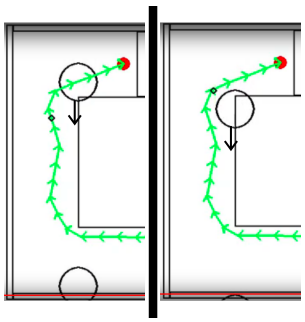

Figure 22: Optimizer adjusts the trajectory and the obstacle is avoided. (time=1m15s)

There is also the need of simulating intruders that do not move with constant speed. For that the trajectory of an aircraft moving in a cluttered map was stored. It was then possible to manually insert an intruder that would follow this trajectory, unknown to the algorithm. The algorithm only knows the position and speed of the intruder with one second of delay, to simulate sensor processing time. Two runs are now shown, in the first one (Figures 46-48) the intruder is launched later than in the second (Figures 49-51):

In order to test the limits of the algorithm a 2D simulation was run with an intruder traveling in a zig-zag trajectory towards the aircraft using the produced algorithm. This represents a very challenging situation since the intruder trajectory is constantly changing direction, making it difficult for the algorithm to determine if the aircraft should go above or below the intruder. This simulation was run 10 times and the algorithm failed 4 times (collision happened). One successful simulation is presented in Figures 23-29.

Figure 23: Intruder is detected. As the intruder descends the algorithm doesn't detect a trajectory conflict

Figure 24: The trajectory is adjusted as the intruder starts to climb

Figure 25: Intruder starts to increase the climb rate. Trajectory is further adjusted.

Figure 26: Intruder starts to climb with such a rate that it becomes (apparently) impossible to go above it. Trajectory is adjusted to go under the intruder.

Figure 27: Intruder starts to descend

Figure 28: Algorithm (one second later) is informed that intruder is descending and tries to avoid it from above

Figure 29: Avoidance is successful

6.2. Simulating an unknown environment

To simulate an unknown environment the algorithm was used to compute the trajectories in an environment, without knowing the position or number of obstacles in that environment. It was then assumed that the algorithm would acquire information on an obstacle when the aircraft was at a distance to that obstacle smaller than a certain detection range. The evolution of one of these simulations is now shown in Figures 52-57.

To have some statistical data on the performance of the algorithm several simulations were run in random maps, Figures 30 show one of these random maps.

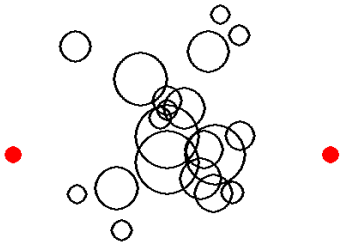

Figure 30: Random unknown map.

Each map has 20 randomly generated circles, with a radius between 10 and 40 (display units). The distance from start to goal was 370 display units. Table 4 shows the time taken to reach the goal and the number of times the trajectory was fully or partially regrown. The algorithm wasn't able to compute a trajectory once, in that case the aircraft stopped at a distance smaller than the safe distance, relatively to an obstacle. For that reason the RRT algorithm could not output any safe trajectory.

6.3. Multi-Aircraft

In a realistic scenario several aircrafts share the same air-space. In a future where urban transportation is also accomplished using aircrafts the algorithms must prove to enable safe circulation.

attempt	number of regrows	time
1	0	34 (s)
2	0	32 (s)
3	0	36 (s)
4	2	62 (s)
5	1	46 (s)
6	0	32 (s)
7	3	-
8	0	39 (s)
9	0	47 (s)
10	1	49 (s)

Table 4: Results of the runs in random maps.

Once this work aims for a decentralized approach it is expected that algorithms running independently in several aircrafts are able to provide collision free paths, without a centralized entity taking a decision. The aircraft, in this simulation, share their flight plan.

The experiment was conducted 10 times in a scenario where two aircrafts must coordinate the passage through a narrow gap. In all of the trials there were no collisions. The results are shown on Fig. 58 - 60.

The experiment was repeated when one of the aircraft ignores the presence of the other. This resembles a scenario where one of the aircraft has priority over the other and, therefor, flies unconcerned. This experiment was also successful in the 10 trials.

6.4. Physics simulation

The previous algorithms were developed taking as assumptions that the UAV could be described as a body subjected to forces as inputs. The resulting trajectories are a sequence of constant acceleration segments. Such a formulation of the problem leads to the existence of discontinuities in the acceleration. A physical multi-rotor can not, however, provide discontinuities in the acceleration, even if it is considered that the rotor velocities can change instantaneously. For this reason many works use polynomials with higher degrees, in order to provide continuity of the acceleration and jerk (3rd derivative of the position). This is associated to the fact that the multi-rotor can only produce thrust in the axis perpendicular to the rotors plane, meaning that discontinuities in the acceleration and jerk would correspond to discontinuities in the multi-rotor attitude and angular rate respectively. In [10] and [26] the authors use 11th and 9th order polynomials for computing the trajectories, respectively. In [9] the authors use a spline that assures continuity up to the 4th derivative of the position for smoothing the trajectory.

It is clear that there is a need to verify the capability of a multi-rotor with realistic dynamics to follow the computed trajectories, once the simplified model used to validate the real-time capability of the algorithms doesn't rigorously approximate a real UAV.

For simulating it will be used the Robotics Operative System (ROS) [27] and the physics engine Gazebo [28]. A plugin for Gazebo, RotorS, was developed in the Autonomous Systems Lab of ETH Zurich university [29]. This plugin includes multi-rotor models and example controllers, that are used in the current work. This open source simulator also has the advantage of enabling the simulation of a variety of sensors and allow a posterior straight forward implementation in real multi-rotors.

The controller for the simulated aircraft was adapted from the one described in the work by Lee et al. [20]. Changes were made to allow the controller to predict the references while the trajectory planner does not update them.

It was chosen a simple map for validation of the algorithm. The map consisted of a gazebo model of a fast-food chain restaurant. Fig. 31 represents the simulation scenario. The algorithm computes a trajectory from one of the sides of the building to the other.

Figure 31: Map visualization on the Gazebo simulator environment

Four runs were performed. The results of four runs are now presented. The runs differ in maximum acceleration allowed and the side of the building taken to arrive to the goal position. In all the runs the minimum distance allowed between the UAV and an obstacle was set to 3 meters and the maximum speed allowed for the UAV to travel was 15 m/s (54 km/h). For run 1 and 2 the maximum allowed acceleration was $6m/s^2$ and for run 3 and 4 $10m/s^2$.

The trajectory taken by the vehicle in one of the runs (run 3) is presented in Fig. 32.

The norm of the difference between the aircraft position and the reference position provided by the algorithm was stored over time for every time an update was received on the aircraft position. The

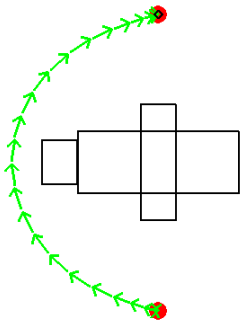

Figure 32: Trajectory taken in run 3 (Max. acceleration = $10m/s^2$).

evolution of the position error along the time is now presented for run 3 in Fig. 33.

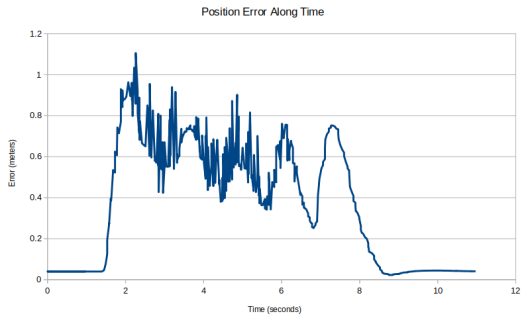

Figure 33: Position error along the simulation time in run 3 (Max. acceleration = $10m/s^2$).

It is possible to observe in Fig. 33 that the position error is low and stabilized while the UAV is hovering, before and after executing the trajectory. The error doesn't tend to zero because the controller expression doesn't contain an integral term. During the trajectory execution the position error rises but it is never greater than one meter except for brief moments in run 3. As expected measured error was greater when the maximum acceleration allowed is greater (for runs 3 and 4).

7. Conclusions

In this work a real-time path planning algorithm was developed. The algorithm is based on a modified RRT and a trajectory-optimization algorithm. The modified RRT allows to compute curvature constrained trajectories. The trajectory-optimization algorithm is based on a simplified dynamics model for the UAV. This dynamics is used to formulate a trajectory-optimization problem that can be solved in real time. This optimization problem is designed in such a way that it aims to minimize an estimate on the mission cost. This estimate is a combination between trajectory-time and energy/fuel consumption. Aiming for reducing the number of design variables, collision-free constraints are assured in intermediate points between way-points, to minimize the number of way-points necessary to describe the trajectory. Results show that this technique improves significantly the computational efficiency of the algorithm.

Both the RRT and the trajectory optimization algorithms were combined into a real-time trajectory-planner. An additional feature was which quickly stops the multi-rotor when a close collision time is predicted. Several simulations using a simplified environment are performed that validate the capability of the algorithm to generate collision free trajectories in partially unknown environments. It was also proven the capability of avoiding moving obstacles, which trajectories are predicted assuming a constant obstacle speed.

The algorithms were also used to plan the trajectories of two agents using a distributed approach. The agents were capable of coordinating among them collision free trajectories.

Finally, it was proven a vehicle with realistic dynamics is capable of following aggressive trajectories generated by the proposed planner. The validation was achieved using the Gazebo simulator and an existing controller.

For future work, the authors propose that the algorithms are expanded for dealing with general convex obstacles. It would also be interesting to expand the algorithms for fixed wing aircraft by, for example, limiting the stall speed and the climb rate.

Acknowledgements

(Acknowledgement for Boeing and NRC funds here.)

References

[1] M. Correa, J. C. Jr., M. A. Rossi, and J. R. A. Jr., "Improving the resilience of uav in nonsegregated airspace using multiagent paradigm," in *Second Brazilian Conference on Critical Embedded Systems*, 2012.

[2] K. Bimbraw, "Autonomous cars: Past, present and future," *12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, July 2015.

[3] P. Angelov, *Sense and Avoid in UAS*. Wiley, 1^{st} ed., 2012. ISBN:978-0-470-97975-4.

[4] S. Ramasamy, R. Sabatini, A. Gardi, and J. Liu, "Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid," *Aerospace Science and Technology*, 2016.

[5] J. Kok, L. F. Gonzalez, and N. Kelson, "Fpga implementation of an evolutionary al-

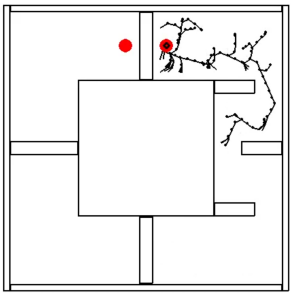

Figure 34: RRT is grown (t=0s)

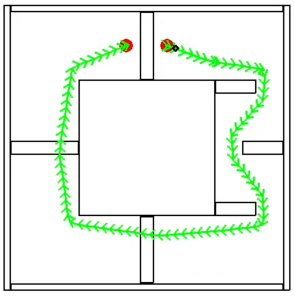

Figure 35: Initial trajectory is computed (t=3s)

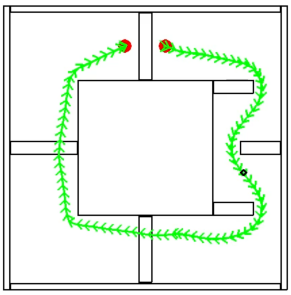

Figure 36: The algorithm optimizes part of the trajectory ahead of the aircraft.

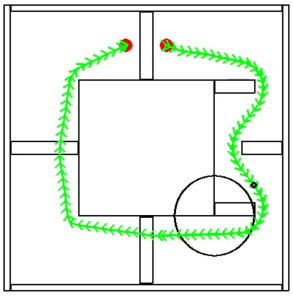

Figure 37: Obstacle is detected (t=25s)

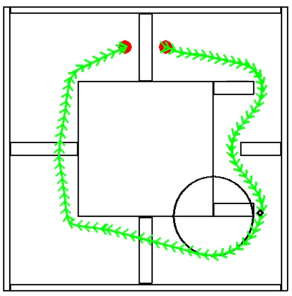

Figure 38: Trajectory optimizer adjusts the trajectory avoiding the obstacle (t=27s)

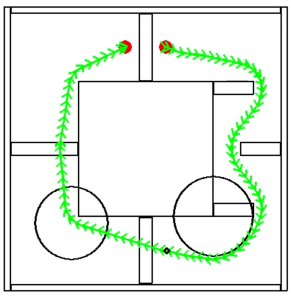

Figure 39: Obstacle is detected (t=41s)

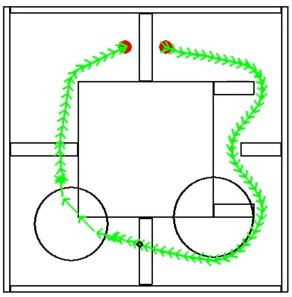

Figure 40: Trajectory optimizer gets trapped in an unfeasible local minima (t=46s)

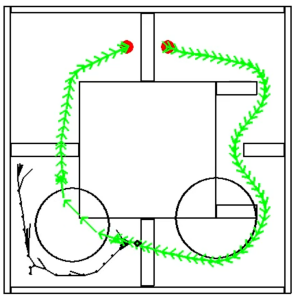

Figure 41: RRT algorithm regrows the critical part of the trajectory (t=47s)

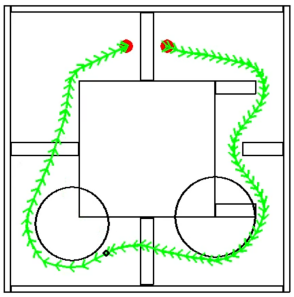

Figure 42: A feasible trajectory is found (t=48s)

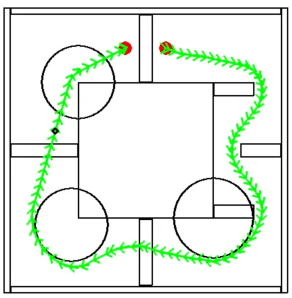

Figure 43: Obstacle is detected critically close, trajectory optimizer wouldn't have time to react in real time (t=1m11s)

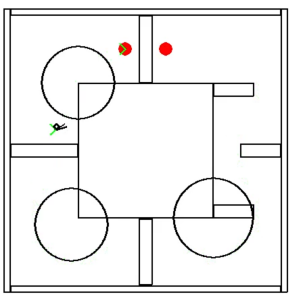

Figure 44: All references are turned off to prevent the UAV from colliding. Previous trajectory is discarded (t=1m11s)

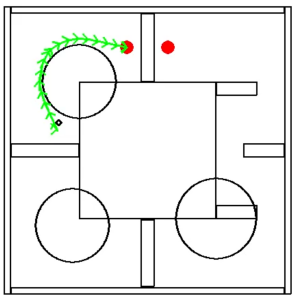

Figure 45: RRT algorithm regrows a new trajectory from the UAV position to the goal. (t=1m1ss)

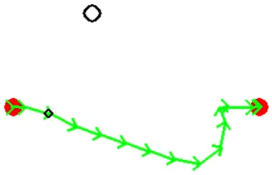

Figure 46: Trajectory while intruder flies to the right

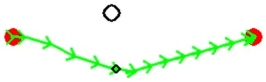

Figure 47: Trajectory changes as intruder changes direction

Figure 48: Intruder is avoided

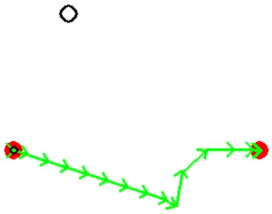

Figure 49: Initially algorithm plans to go below the intruder

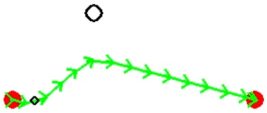

Figure 50: Algorithm plans to go above the intruder

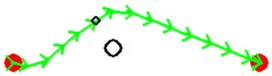

Figure 51: Intruder is avoided

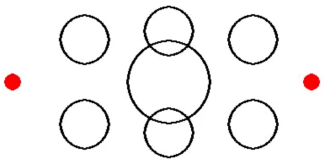

Figure 52: Complete map

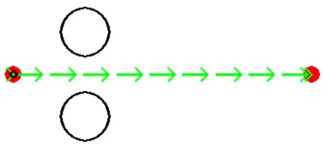

Figure 53: Initially only two obstacles are visible, none of them interferes with the trajectory

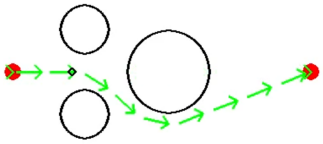

Figure 54: An obstacle is detected and the trajectory is adjusted.

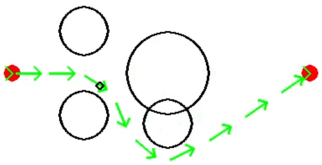

Figure 55: New obstacle is detected and the trajectory adjusted

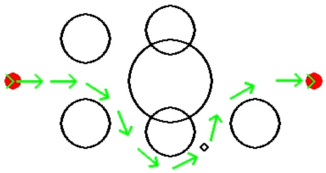

Figure 56: Another obstacle is detected and the trajectory is adjusted

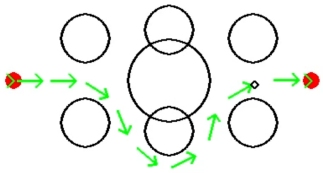

Figure 57: The final obstacle from the map is detected, it doesn't, however interfere with the trajectory.

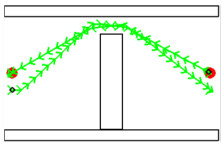

Figure 58: Initial trajectories, computed simultaneously, in collision rout.

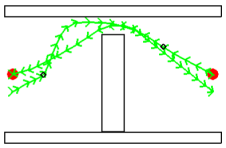

Figure 59: Collision free trajectories are generated.

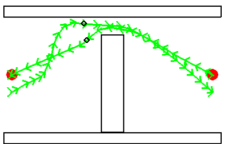

Figure 60: The aircraft avoid each other (scenario 1).

gorithm for autonomous unmanned aerial vehicle on-board path planning," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 17*, pp. 272 – 281, April 2013.

[6] J. TISDALE, Z. KIM, and J. K. HEDRICK, "Autonomous uav path planning and estimation," *IEEE Robotics & Automation Magazine*, pp. 35–42, June 2009.

[7] I. K. Nikolos, K. P. Valavanis, I. Senior Member, N. C. Tsourveloudis, and A. N. Kostaras, "Evolutionary algorithm based offline/online path planner for uav navigation," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART B: CYBERNETICS, VOL. 33, NO. 6*, pp. 818–912, December 2003.

[8] X. Zhang, J. Chen, B. Xin, and H. Fang, "Online path planning for uav using an improved differential evolution algorithm," in *Proceedings of the 18th World Congress The International Federation of Automatic Control Milano (Italy)*, pp. 6349–6354, August 2011.

[9] M. Pavone and R. E. Allen, "A real-time framework for kinodynamic planning in dynamic environments with application to quadcopter obstacle avoidance.," *Robotics and Autonomous Systems*, p. 174193, 2018.

[10] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online uav replanning," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[11] Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. C-32, pp. 108–120, Feb 1983.

[12] B. Siciliano, L. Sciavicco, V. Luigi, and G. Oriolo, *Trajectory Planning*, ch. 4. 01 2011.

[13] A. Wchter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming, mathematical programming," 2006.

[14] A. Wchter, *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, CARNEGIE MELLON UNIVERSITY, 2002.

[15] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: sequential unconstrained minimization techniques*, vol. 4. Siam, 1990.

[16] J. T. Betts, "A survey of numerical methods for trajectory optimization," August 1998.

[17] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," *Proceedings of the American Control Conference*, pp. 1936–1941, May 2002.

[18] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525, May 2011.

[19] J. von Löwis and J. Rudolph, "Real-time trajectory generation for flat systems with constraints," in *Nonlinear and Adaptive Control* (A. Zinober and D. Owens, eds.), (Berlin, Heidelberg), pp. 385–394, Springer Berlin Heidelberg, 2003.

[20] T. Lee, M. Leok, , and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," *49th IEEE Conference on Decision and Control*, pp. 5420–5425, December 2010.

[21] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., 1998.

[22] J. L. Marins, T. M. Cabreira, K. S. Kappel, and P. R. 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)*, pp. 256–261, Nov 2018.

[23] J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, J. Schulman, Y. Duan, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking.," *The International Journal of Robotics Research*, p. 12511270, 2014.

[24] G. E, J. D, and K. S, "A fast procedure for computing the distance between complex objects in three dimensional space.," *IEEE Journal of Robotics and Automation*, p. 193203, 1988.

[25] V. den Bergen, "Proximity queries and penetration depth computation on 3d game objects.," *Proceedings of the game developers conference (GDC)*, p. 2023, 2007.

[26] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2013.

[27] "Ros (robot operating system), documentation. accessed in 8th july, 2019.."

[28] 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)*, vol. 3, pp. 2149–2154 vol.3, Sep. 2004.

[29] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Cham: Springer International Publishing, 2016.