Real-Time Trajectory Planning for UAVs in Environments with Moving Obstacles

António Ramalho
antonio.ramalho@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

April 2020

Abstract

In the present work a real-time trajectory planning algorithm for multi-rotor is developed. The algorithm is capable of avoiding both static and moving unexpected obstacles, in real time. The trajectory-planner is capable of generating optimal trajectories regarding operational costs, which are given by a combination of the estimated energy consumption and total trajectory time. The algorithm was integrated with a simplified TCAS system, showing the capability of the autonomous path planner to follow protocols originally used in manned aircraft. The solution is based on a modified RRT algorithm and a trajectory optimization algorithm. Multiple basic simulations were performed to evaluate the real-time capabilities of the algorithms. A simulation in the physics engine Gazebo was also analysed to validate the capability of a realistic multi-rotor to follow aggressive trajectories generated by the proposed planner.

Keywords: multi-rotor, trajectory planning, RRT, trajectory-optimization, TCAS

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.

Autonomous systems are however complex, requiring a vast set of capabilities. This work will be focused on only a portion of that: the generation of collision free trajectories in real-time and the integration of the trajectory-planning algorithms with existing collision avoidance protocols for manned aircraft.

1.1. Outline of the approach

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 (and for that the UAV dynamics must be considered) in partially unknown environments with moving obstacles. The algorithms should be able to quickly react to new detected obstacles (including moving obstacles) and safely avoiding 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 locally-optimal trajectory is computed if enough computation time is used. The trajectory will also be optimized while the UAV flies in order to improve the quality of the trajectories if they aren't locally-optimal to begin with.

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.

Further on the algorithms will be upgraded with other features, such as the capability of generating leveled trajectories at desired altitudes and follow desired climb rates.

The algorithms will finally be integrated with a TCAS system. This integration proves the capability of the proposed solution to respect this collision avoidance protocol (TCAS) originally designed for manned aircraft.

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 [1] 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 [2].

2.2. Real time path planning - Literature review

Online path planning in unknown environments for UAVs was accomplished before in [3], but it would ignore the vehicle dynamics, therefore the UAV would have to fly slowly in order to perform the computed trajectories. The most commonly used methods for real time path planning are, arguably, sample based methods. Basic RRTs for example allow a fast search for a trajectory, however the solution is very sub-optimal.

For accomplishing a proper path-planning it is desirable that the algorithm considers the kinodynamics of the problem. Attempts have been made to apply RRT*, an asymptotically optimal version of the RRT, for this problem. In [4] a kinodynamic RRT* was proposed, however, in this work, the times required to compute a trajectory were often greater than 10 seconds and sometimes greater than 100 seconds, making it unsuitable for using on online path-planning problems.

On-board path planning for small UAVs have 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 (offline however) 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 [7] and [8] an evolutionary algorithm is developed to allow online path planning in an unknown environment, this work considered static environments and it was never implemented in an aircraft.

An interesting approach for path-planning problems is trajectory optimization. It has been used for many years with diverse applications, for example to define missions of orbit transfer and also launching space vehicles, like it is referred in [9]. In recent years there have been a great amount of work regarding applying these algorithms to robotics. One of the most influential works was done by Matt Zucker et al. [10], in this work motion-planning for articulated robots was performed using optimization techniques. It discretized the trajectory as a series of configurations, and then the algorithm would minimize the distance to obstacles and maximize the smoothness of the trajectory, the trajectory time would, however, have to be predetermined. In prior work, by A. Richards et al. [11], motion planning for UAVs was addressed, however with clear limitations. More recently, in 2016, Helen Oleynikova et al. [12] developed an algorithm for UAV path planning in unknown static environments.

Generally trajectory optimization algorithms require a first iteration. This can be computed in many different ways, for example it is possible to use a straight line as a first trajectory, even if this one is not obstacle free [13]. In the present work however it is presented a first iteration given by a modified RRT algorithm which not only guarantees that the local-minimum is feasible as it also provides a trajectory close to satisfying the kinodynamic constraints of the problem.

Two different works [14, 15] are of particular interest for the current research.

In the work by Pavone et al. [15] the authors developed an algorithm which they claim to be the first successful real-time path planner for UAVs in unknown dynamic environments physically tested. This work is very recent (very end of 2018). The processing, however, was made on a ground station and communicated with the UAV. The authors claim that the algorithms can run in real-time in an on-board computer if their code is converted (code is in MATLAB) to C++. This work combines a series of interesting features such as the use of machine learning to allow an online performance of a kinodynamic, asymptotically optimal sample based planner, in this case the Fast Marching Tree star (FMT*).

In "*Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments*" ([14]) the authors explore the fact that trajectory optimization techniques allow the computed trajectory to be continuously improved while the robot executes it.

2.3. Traffic Collision Avoidance System

The Traffic Alert and Collision Avoidance System (TCAS) is an on-board conflict detection and resolution system which alerts pilots to the presence of nearby aircraft that pose a mid-air collision threat and issues conflict resolution advisories [16]. TCAS is a complex safety-critical system in the area of air traffic management. TCAS is able to operate independently of the ground-based air traffic control (ATC) system. All TCAS systems provide a specific degree of collision threat alerting and a traffic display by making use of radar beacon transponders.

TCAS systems provide different degrees of alerting regarding air-born collision. To do so these systems rely on the communication between the beacon transponders on-board of the aircrafts.

The TCAS system provides two types of alerts to the pilot: Traffic Advisories (TA) and Resolution Advisories (RA). A traffic advisory (TA) is a sound message that alerts the pilot for proximate cooperative traffic, with possible risk of collision. Resolution advisories (RAs) are sound messages stating actions for pilots to take in order to avoid collision by assuring a vertical separation between aircrafts (climb, descend, level...).

In case of resolution advisory the TCAS systems on board of the conflicting aircraft coordinate the computed resolutions to avoid collision (for example avoiding that both aircrafts start climbing towards a new collision point). The TCAS systems will then select complementary resolutions.

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 1 - 7) will now be presented to exemplify this concept.

The selection of part of the trajectory that should be optimized at a given time will now be described. If UAV is following the trajectory reference corresponding to the current time t_C, and the time used for an optimization increment is t_i, than the optimization is performed for the trajectory portion between $t_C + t_i$ and $t_C + t_i + t_n$ where t_n is a chosen parameter for a normal optimization scenario. The trajectory between t_C and $t_C + t_i$ is fixed (will not be changed) once the optimization increment result will only be available at $t_C + t_i$. Once the optimization increment is finished the current time t_C is updated and the process restarts (Fig 1 - 3).

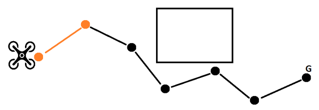

Figure 1: Part of the trajectory fixed (orange) in an initial trajectory computed by an RRT

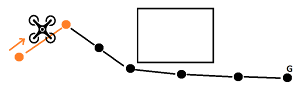

Figure 2: Trajectory is optimized while UAV flies

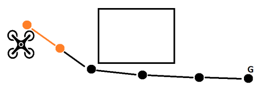

Figure 3: New part of the trajectory is fixed and process re-starts

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 4 - 5).

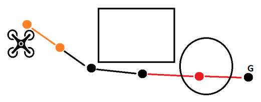

Figure 4: Unknown obstacle detected in the trajectory

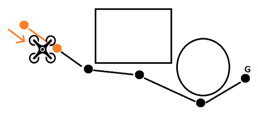

Figure 5: Trajectory optimizer adjusts the trajectory avoiding the obstacle

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 MAX_f) the RRT algorithm regrows a trajectory around the unfeasible portion (Fig 6 - 7). This might happen if the trajectory falls into an unfeasible local minima.

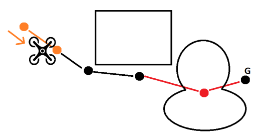

Figure 6: Unknown complex obstacle detected in the trajectory

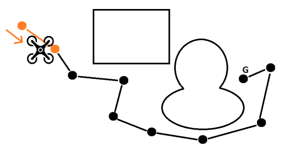

Figure 7: The maximum number of failures is reached and a trajectory is regrown around the obstacle

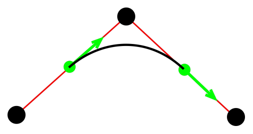

Figure 8: The green points and arrows represent the robot's position and speed respectively. The black points represent the vertices of the RRT.

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".

4. Proposed RRT algorithm

Rapidly exploring Random Trees (RRTs) where first introduced by Lavalle as a family of randomized planners ([17], chapter 5) 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. It is

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 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:

$$\alpha = \pi - 2 * arctan\left(\frac{R_{min}}{d/2}\right) \tag{1}$$

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 $\mathbf{e_1}$ represent one edge (position of vertex k minus position of vertex k-1) and $\mathbf{e_2}$ represent its consecutive edge (position of vertex k+1 minus position of vertex k). We can state that an edge $\mathbf{e_2}$ is acceptable after an edge $\mathbf{e_1}$ if and only if:

$$\frac{\mathbf{e_1} \cdot \mathbf{e_2}}{\|\mathbf{e_1}\| * \|\mathbf{e_2}\|} \geq cos(\alpha_{MAX}) \tag{2}$$

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 2 is not respected), such as in Figure 9.

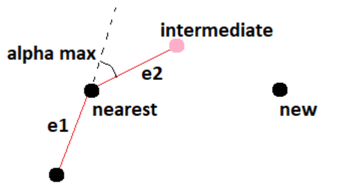

Figure 9: The new vertex is in such a position that it is not possible to expand the tree directly towards it, the tree is expanded than using a maximum curvature expansion.

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. 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 smooth 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 10:

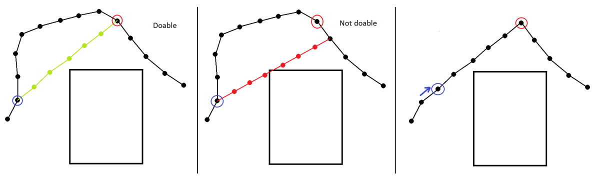

Figure 10: Consecutive steps in the enhancement algorithm.

Figures 11 and 12 show some examples of the computed trajectories before and after the enhancement step.

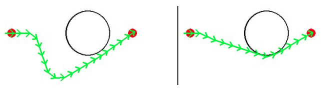

Figure 11: Example of the trajectory before (on the left) and after (on the right) the trajectory enhancement

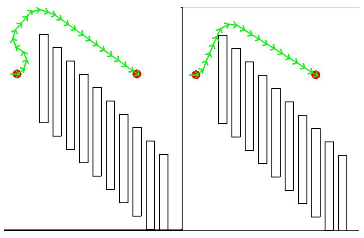

Figure 12: Example of the trajectory before (on the left) and after (on the right) the trajectory enhancement

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. $[\sigma_0, ..., \sigma_{N-1}]$ 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 $\sigma_i = [\mathbf{p_i}, \mathbf{v_i}]$, where $\mathbf{p_i}$ and $\mathbf{v_i}$ corresponds to the position and speed vector of the robot in a given moment. This means that $\sigma \in \mathbb{R}^4$ for a bi-dimensional environment and $\sigma \in \mathbb{R}^6$ 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. l, as j, will also be used as a subscript to refer to components of position/speed. Basically: $\sigma_i = [\mathbf{p_i}, \mathbf{v_i}] = [p_{i,x}, p_{i,y}, p_{i,z}, v_{i,x}, v_{i,y}, v_{i,z}]$ (in 3 dimensional space).

It will be assumed that the robot acceleration $\mathbf{a}$ 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):

$$x = [\Delta t, p_{0,x}, p_{0,y}, p_{0,z}, v_{0,x}, v_{0,y}, v_{0,z}, \dots$$

$$\dots, p_{N-1,x}, p_{N-1,y}, p_{N-1,z}, v_{N-1,x}, v_{N-1,y}, v_{N-1,z}]$$

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 and energy consumption. 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:

$$f(x) = k_T * f_T(x) + k_F * f_F(x) \tag{3}$$

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 multirotor it was used closed form expressions from the work by Marins et al. [18].

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 $\mathbf{p}_{i+1}$ should be given by:

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \frac{\mathbf{v}_{i+1} + \mathbf{v}_{i+1}}{2}\Delta t \tag{4}$$

To write Equation 4 in the form $c(x) = 0$ the right side of Equation 4 is subtracted in both sides of the equation.

5.4. Maximum speed and maximum acceleration

The maximum speed constraint can be written as:

$$\|\mathbf{v}_i\| < v_{MAX} \tag{5}$$

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

$$\|\mathbf{a_i}\| < a_{MAX} \Leftrightarrow \frac{\|\mathbf{v}_{i+1} - \mathbf{v}_i\|}{\Delta t} < a_{MAX} \tag{6}$$

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

$$\|\mathbf{v}_{i+1} - \mathbf{v}_i\| < a_{MAX}\Delta t \tag{7}$$

In order to create a constraint in the form $d(x) < 0$ the right side of the inequality stated in Eq. 7 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 $s(\mathcal{O}_k, \mathbf{p}_i)$ represent the distance from a point $\mathbf{p}_i$ to the closest point on the surface of a convex obstacle $\mathcal{O}_k$, *this distance is negative if the point $\mathbf{p}_i$ 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

$$s(\mathcal{O}_k, \mathbf{p}_i) > d_{safe} \tag{8}$$

This formulation for the obstacle clearance constraints was formulated based on [13]. 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 environment. The algorithm also supports moving spheres (and circles).

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.

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.

5.6. Problem formulation

The problem formulation can finally be stated:

$$\begin{array}{ll} min & k_T * f_T(x) + k_F * f_F(x) \\ s.t. & \mathbf{p}_{i+1} = \mathbf{p}_i + \frac{\mathbf{v}_{i+1}+\mathbf{v}_{i+1}}{2}\Delta t \\ & \|\mathbf{v_i}\| < v_{MAX} \\ & \|\mathbf{v}_{i+1} - \mathbf{v_i}\| < a_{MAX}\Delta t \\ & s(\mathcal{O}_k, \mathbf{p}_i) > d_{safe} \\ & s(\mathcal{O}_k, \mathbf{p}_{inter(i,m)}) > d_{safe} \end{array} \tag{9}$$

In Equation 9 $\mathbf{p}_{inter(i,m)}$ represents the mth intermediate collision checking point for the trajectory segment that joins two consecutive way-points (σ_i to σ_{i+1}).

5.7. Biasing the algorithm behaviour

By introducing terms in the cost function and changing its constants it is possible to control the algorithm behaviour. This property was used to allow the computation of leveled trajectories (track a desired altitude), avoidance trajectories (track a certain climb rate in part of the trajectory and for tuning the energy efficiency. For following a desired altitude the method the error between the desired altitude z_{ref} and the altitude at each state was considered as a cost, the modified cost function is:

$$f_{new}(x) = f(x) + k_h * \sum_{i=0}^{N-1} \|p_{i,z} - z_{ref}\| \tag{10}$$

Where $f(x)$ is the old cost function and k_h the constant that allows to tune the "strength" of the altitude suggestion.

The TCAS system, as it was discussed in Section 2.3, indicates climb rates for the pilot to follow, in order to avoid conflict. The term added to the cost function penalizes errors between the climbing speed and the desired climbing speed vz_{ref} **for the first half of the way-points**. If this penalization considers all the way-points than it won't change the trajectory because the average climbing speed is only defined by the start and goal altitudes and the trajectory time. The modified cost function is:

$$f_{new}(x) = f(x) + k_c * \sum_{i=0}^{round((N-1)/2)} \|v_{i,z} - vz_{ref}\| \tag{11}$$

Where $f(x)$ is the old cost function and k_c the constant that allows to tune the "strength" of the climb rate suggestion.

6. Simulation

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.

6.1. Real-time avoidance

The following figures (Fig. 13-21) 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.

The algorithm was also tested against moving intruders. When the moving intruders have a known trajectory the algorithm was no problem avoiding them. When the moving intruders move in smooth trajectories the algorithm is also able to avoid them, by assuming constant speed of those same intruders. The simulated scenario that presented the greatest challenge involves a zig-zagging intruder that moves in the opposite direction to the aircraft that follows the algorithm computed trajectory. Once the algorithm assumes a constant speed for the intruder, and the intruder speed is constantly changing direction, the algorithm fails in computing a collision free path in approximately half of the runs.

6.2. Simulating an unknown environment

To simulate an unknown environment the algorithm was used to compute the trajectories in a 2 dimensional environment, without knowing the position or number of obstacles in that environment. It was then assumed that the algorithm would acquire information on the obstacle when the aircraft was at a distance to the obstacle smaller than a certain detection range. The results of these simulations are now demonstrated:

To have some statistical data on the performance of the algorithm several simulations were run in random maps, Figure 22 shows one of these random maps.

run	regrow count	time (s)
1	0	34
2	0	32
3	0	36
4	1	62
5	1	46
6	0	32
7	3	-
8	0	39
9	0	47
10	1	49

Table 1: Results of the runs in random maps.

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 and the maximum speed chosen for the aircraft was 15 display units per second. Table 1 shows the time taken to reach the goal and the number of times the trajectory was fully or partially regrown using the RRT algorithm. 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. Physics simulation

To validate that a real multi-rotor is capable of following the computed trajectories it will be used the physics engine Gazebo [19] and the Robotic Operative System (ROS) [20]. A plugin for Gazebo, RotorS, was developed in the Autonomous Systems Lab of ETH Zurich university [21], this plugin allows to perform physical simulations of multi-rotors. The controller used was adapted from the one described in the work by Lee et al. [22]. It was chosen a simple map for validation of the algorithm. The map consisted of a gazebo model of a fast-food chain restaurant. The multi-rotor had to go from one of the sides of the building to the other. An altitude constraint was created force the multi-rotor not

Four runs were performed. 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 two of the runs the maximum acceleration allowed was $6m/s^2$ and for the other two $10m/s^2$.

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 evolution of the position error along the time for

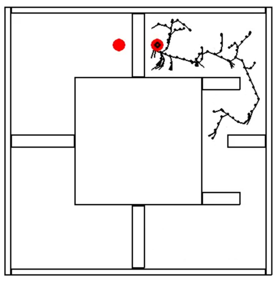
Figure 13: RRT is grown

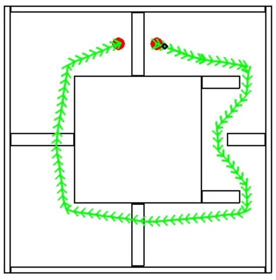
Figure 14: Initial trajectory is computed

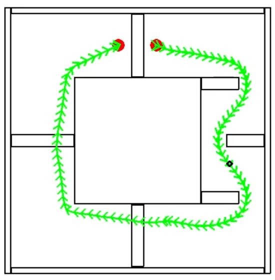
Figure 15: The algorithm optimizes part of the trajectory ahead of the aircraft.

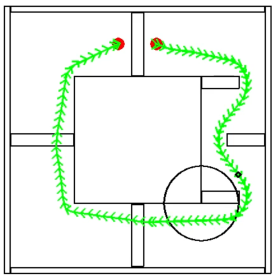
Figure 16: Obstacle is detected

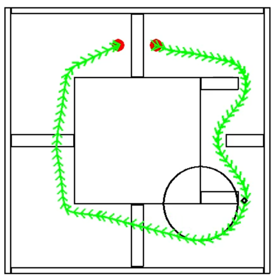
Figure 17: Trajectory optimizer adjusts the trajectory avoiding the obstacle

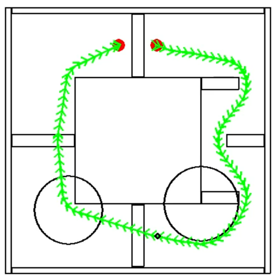
Figure 18: Obstacle is detected

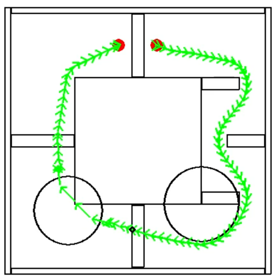
Figure 19: Trajectory optimizer gets trapped in an unfeasible local minima

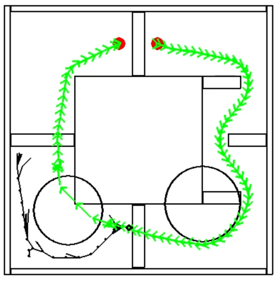
Figure 20: RRT algorithm regrows the critical part of the trajectory

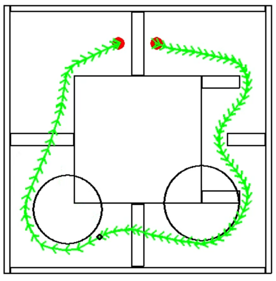
Figure 21: A feasible trajectory is found

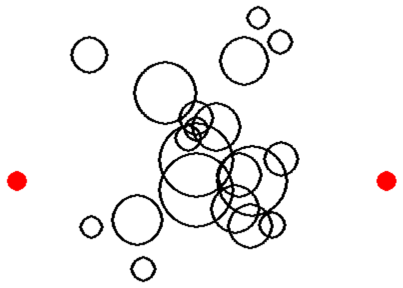
Figure 22: Random unknown map.

run number 4 is now presented in Figure 23.

It is possible to observe in Figure 23 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. As expected the error is greater when the maximum acceleration allowed is greater.

6.4. TCAS system testing

A simulation was performed to validate the correct function of the TCAS system and its proper integration with the remaining solution. This simulation was performed using Gazebo.

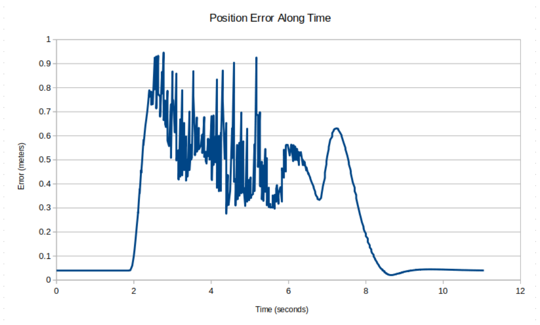

Figure 23: Position error along the simulation time (Max. acceleration = $10m/s^2$).

Several measures were stored along the time to allow an analysis of the simulation. Figure 24 presents the altitude of each of the aircrafts (vertical axis) corresponding to their horizontal position (horizontal axis) along the simulation. It is possible to see in Figure 24 that the conflict was cleared by the TCAS systems. The TCAS system corresponding to the aircraft that departed from the right (orange in the graphic) resolved the conflict by determining a certain climb rate. The TCAS system corresponding to the other aircraft (blue) determined a descend rate to solve the conflict. The determined resolutions were correctly followed until the conflict was cleared.

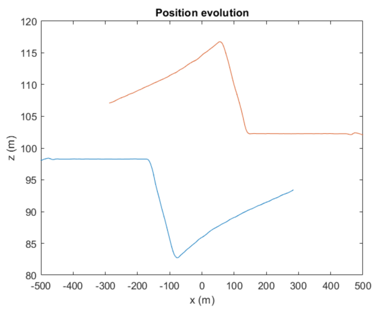

Figure 24: Trajectory of the aircraft (altitude in the vertical axis, horizontal coordinate, in the horizontal axis).

7. Conclusions

In this work it was developed a real-time trajectory-planning algorithm, using a modified RRT algorithm and a trajectory optimization algorithm. An enhancement step was also implemented, this step allows a quick improvement of the trajectory generated by the RRT. The trajectory optimization algorithm allows to generate locally-optimal trajectories, it considers a cost function that is a combination of energy consumption and trajectory time. The developed algorithm was further improved with an emergency protocol that stops the multi-rotor in critical scenarios. A simplified testing framework was developed to access the capabilities of the developed algorithms. The real-time path-planner showed to be capable of creating collision free trajectories in partially unknown environments, by quickly adjusting the trajectory when new obstacles are detected. To prove that the algorithm is fit for a variety of maps the planner was tested in random maps, where the obstacles are initially unknown and the information about their position is revealed only within a certain range. This algorithm is also capable of avoiding unexpected moving obstacles, considering also sensor delay.

Besides this dynamic capabilities, the developed trajectory planner allows to select the amount of computational time spent in an initial optimization. This gives the algorithm anytime capabilities, letting the user choose between computational time and trajectory optimally.

The algorithms at the time are only capable of considering spheres and cuboids as obstacles. This can be changed in the future for general convex obstacles, for example by using the GJK and the expanding polytope algorithms. The algorithms can also be extended to fixed-wing aircrafts by, for instance, limiting the minimum speed (stall speed) and the climb rate of the aircraft.

In order to prove that a real-multi-rotor is capable of following the computed trajectories, a simulation in Gazebo was performed. In this simulation the virtual UAV was capable of following the trajectory while keeping a position error smaller than one meter at all times, in an aggressive maneuver.

The developed algorithms were then upgraded in order to make them more suitable for non-segregated air-space. This integration includes the possibility of computing trajectories that promote the flight at a certain flight level and the integration with a simplified TCAS system. The interrogation rate was considered to be the same as in scenarios involving manned aircraft. The algorithm proved to be able to follow the resolutions provided by the TCAS system, this interesting feature shows how the algorithms can follow protocols designed for human pilots.

For future work, besides extending the algorithm for general obstacles and fixed wing aircrafts. It would be interesting to implement the algorithms in C++ and test if it is possible to use them for on-board planning.

Acknowledgements

The author would like to thank Professor Rodrigo Ventura and Professor Afzal Suleman for the guidance provided throughoutthis work. (Acknledgement for Boeing and NRC funds here.)

References

[1] Lozano-Perez, "Spatial planning: A configura-

tion space approach," *IEEE Transactions on Computers*, vol. C-32, pp. 108–120, Feb 1983.

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

[3] G. Grisetti, S. Grzonka, and W. Burgar, "A fully autonomous indoor quadrotor.," *IEEE TRANSACTIONS ON ROBOTICS*, p. 90100, 2012.

[4] J. van den Berg Dustin J. Webb., "Kinodynamic, rrt*: Optimal motion planning for systems with linear differential constraints," May 2012.

[5] J. Kok, L. F. Gonzalez, and N. Kelson, "Fpga implementation of an evolutionary algorithm 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] J. T. Betts, "A survey of numerical methods for trajectory optimization," August 1998.

[10] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," *IEEE International Conference on Robotics and Automation*, May 2009.

[11] 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.

[12] 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.

[13] 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.

[14] C. Park, J. Pan, and D. Manocha, "Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments.," *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling*, p. 207215, 2012.

[15] 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.

[16] C. Livadas, J. Lygeros, and N. A. Lynch, "High-level modeling and analysis of tcas," in *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No.99CB37054)*, pp. 115–125, Dec 1999.

[17] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.