Planning agile motions for quadrotors in constrained environments

Alexandre Boeuf, Juan Cortés, Rachid Alami and Thierry Siméon

Abstract— **Planning physically realistic and easily controllable motions of flying robots requires considering dynamics. This paper presents a local trajectory planner, based on a simplified dynamic model of quadrotors, which fits the requirements to be integrated into a global motion planning approach. It relies on a closed-form solution to compute curves in the kinodynamic state space that tend to minimize the flying time. These curves have suitable continuity properties and guarantee respect of physical limits of the system (i.e. bounds for the time-derivatives of the pose coordinates). The paper explains how this local planner can be used within different motion planning approaches that enable the treatment of difficult problems in constrained environments.**

I. INTRODUCTION

When planning motions for Unmanned Aerial Vehicles (UAVs) such as quadrotors, it is important to consider the dynamic model of the system since not every geometrically valid path corresponds to a feasible motion. For example, because of its dynamic behavior, flying upside down, even for a relatively short period of time, is hardly manageable for a fixed-pitch quadrotor. Aiming to avoid the difficulties and the high computational cost involving kinodynamic motion planning [1], the problem is usually treated in two stages. The first stage applies a basic path planning approach, disregarding dynamic considerations. For this, sampling-based path planning algorithms [2], such as the Rapidly-exploring Random Tree (RRT) or the Probabilistic Roadmap (PRM), can be used to produce a collision free path for the center of mass of the robot. Indeed, since the robot orientation depends on dynamic aspects, collisions cannot be tested for the robot itself but for its smallest bounding-sphere. In a second stage, this path, usually consisting of a sequence of straight-line segments in $\mathbb{R}^3$, has to be transformed into a dynamic trajectory. Trajectory generation methods, such as [3], [4], [5], can be applied in this stage to each portion of the path. The overall trajectory then consists of a sequence of movements from one hovering position to another, which leads to severe sub-optimality in terms of execution time. However, such an unsuitable trajectory can be subsequently optimized using several types of algorithms.

The aforementioned *decoupled approach* is computationally efficient, and can be successfully applied to solve many motion planning problems for UAVs (see for example [6], [7], [8]). However, several classes of problems cannot be treated using this approach, because the robot orientation

All authors are with CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France and Univ de Toulouse, LAAS, F-31400 Toulouse, France (e-mails: aboeuf@laas.fr, jcortes@laas.fr, alami@laas.fr, nic@laas.fr)

This work has been partially supported by the European Community under Contract ICT 287617 "ARCAS".

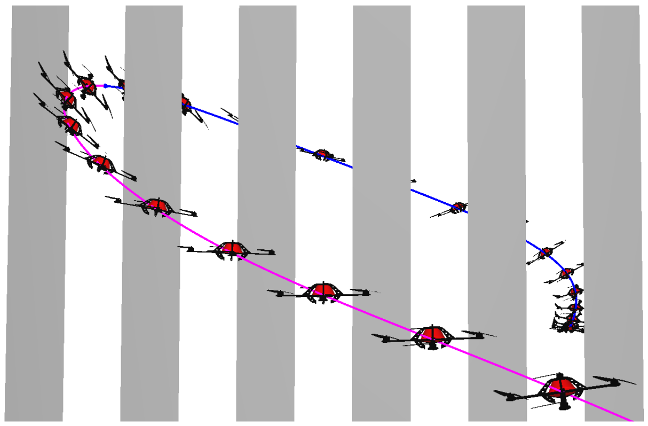

Fig. 1. Solution to the *slot problem* obtained with a basic RRT algorithm using the proposed local planner as a steering method.

cannot be properly considered at the geometric stage. One of such problems, which we refer to as the *slot problem*, is illustrated in Fig. 1. In this problem, the robot has to go through a narrow slot-shaped passage whose width is smaller that the diameter of the smallest bounding-sphere of the robot. Since a collision-free path does not exist for this sphere, the decoupled approach will fail to find a solution. Another type of problem that cannot be treated using a decoupled approach is the transportation of a large object (at medium or high velocity). When carrying such an object, it is not suitable anymore to consider the minimum bounding-sphere of the system because it does not fit its actual shape, thus leading to invalidity of far too many possible solutions. In these situations, collisions have to be tested for the actual shape of the system, and hence, non-hovering states have to be sampled and linked by collision-free flyable trajectories. According to [4], such trajectories must be smooth in the space of the flat outputs of the system with bounds on their derivatives. Therefore, motion planning in this context requires an efficient trajectory generation method able to interpolate two kinodynamic states (position, velocity and acceleration). Unfortunately, available trajectory generation methods are either computationally expensive, or unlikely for the present application, since they do not guarantee the respect of limits in velocity and acceleration, or because they require flying time as an input parameter [9].

This paper proposes a local motion (trajectory) planner that considers a simplified dynamic model of the UAV. It uses fourth-order splines to generate flyable trajectories of minimal time with respect to the proposed closed form solution. It can be used as part of an optimization method in a decoupled approach in order to generate high velocity trajectories in cluttered environments, or applied directly as a steering method within a sampling-based motion planner. Next sections describe the method and present results of its application to several motion planning problems in constrained environments, including the *slot problem*. These problems are also illustrated by the accompanying video.

II. LOCAL PLANNER

A. Overview

Thanks to the differential flatness of the quadrotor dynamics [4], any smooth trajectory in the space of flat outputs can be followed provided that derivatives are correctly bounded. Note that real physical limits of a quadrotor are its total thrust and angular accelerations (which are related to linear acceleration and snap respectively). However, as a simplification, we directly consider limitations on derivatives of the flat outputs. We also choose to limit linear velocity for safety reasons in prospect of future experimental validation.

This section proposes a local motion planner for quadrotors that, given a set of bounds for the derivatives of the flat outputs, produces such trajectories while trying to minimize the flying time. First, the dynamic model is presented together with some basic concepts. The closed-form of a specific fourth-order spline used to solve the problem independently for each output is then proposed. Finally, the way outputs are synchronized to form the final trajectory is explained.

B. Simplified dynamic model of a quadrotor, configuration space, geometric and kinodynamic state spaces

A quadrotor can be modeled as a free-flying object in $\mathbb{R}^3$. It has six degrees of freedom: three for its position ($\mathbf{OG} = \mathbf{r} \in \mathbb{R}^3$) and three for its orientation (pitch, roll and yaw angles: $[\theta, \phi, \psi]$). Thus, its configuration space is $C = \mathbb{R}^3 \times SO(3)$. As illustrated in Fig. 2, a quadrotor of mass m is assumed to be submitted to three forces applied to its center of mass G: his weight $\mathbf{W} = -mg.\mathbf{e_z}$, a fluid friction force $\mathbf{f} = -f.\mathbf{v}$ with f a fluid friction coefficient and $\mathbf{v} = \dot{\mathbf{r}}$ (linear velocity of G), and a thrust force $\mathbf{T}$, which is the resulting force of the action of the four rotors. The direction of the thrust vector is directly related to pitch and yaw angles. Furthermore, Newton's second law of motion shows that $\mathbf{T}$ is a function of $\mathbf{v}$ and $\mathbf{a} = \ddot{\mathbf{r}}$ (linear acceleration of G):

$$m.\mathbf{a} = -mg.\mathbf{e_z} - f.\mathbf{v} + \mathbf{T} \tag{1}$$

Let $\mathbf{x}^G = [\mathbf{r}, \psi] \in \mathbb{R}^3 \times SO(2) = \mathcal{X}_G$ be the geometric state of the system (the flat outputs) and $\mathbf{x}^K = [\mathbf{x}^G, \dot{\mathbf{x}}^G, \ddot{\mathbf{x}}^G] \in \mathcal{X}_G^3 = \mathcal{X}_K$ be the kinodynamic state, $\mathcal{X}_G$ and $\mathcal{X}_K$ being the the geometric and the kinodynamic state spaces, respectively. Using (1), attitude angles $[\theta, \phi]$ can be computed from $\mathbf{v}$ and $\mathbf{a}$. Hence, intermediate configurations $q \in C$ can be extracted from a trajectory in $\mathcal{X}_K$ with arbitrary resolution for collision deception.

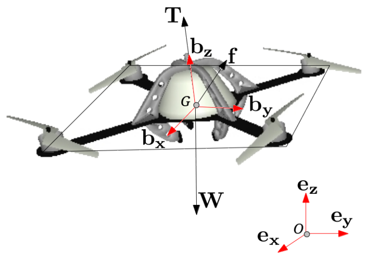

Fig. 2. Forces acting on the quadrotor in our simplified dynamic model.

C. Closed-form solution of the trajectory generation problem

This subsection addresses the generation of smooth trajectories in the kinodynamic state space with inequality constraints on the norm of the derivatives. A trajectory between two states $\mathbf{x}_0^K,\ \mathbf{x}_F^K \in \mathcal{X}_K$ is a time parametrized function $S : [0, t_F] \to \mathcal{X}_G$ with $t_F \in \mathbb{R}^+$ such that:

$$\begin{cases} [S(0),\ \dot{S}(0),\ \ddot{S}(0)] = \mathbf{x}_0^K \in \mathcal{X}_K \\ [S(t_F),\ \dot{S}(t_F),\ \ddot{S}(t_F)] = \mathbf{x}_F^K \in \mathcal{X}_K \end{cases} \tag{2}$$

Because the trajectory has to be continuous in C, and in order to guarantee continuity of the angular velocity, S has to be of class $\mathcal{C}^3$: $S \in \mathcal{C}^3([0,\ t_F], \mathcal{X}_G)$. This is due to the relationship between the attitude and the linear acceleration imposed by (1). In addition, since these (local) trajectories will be the elements of the solution provided by the (global) motion planner, we want to ensure continuity of the jerk (first derivative of acceleration) between two consecutive trajectories. Thus, another constraint has to be imposed:

$$\dddot{S}(0) = \dddot{S}(t_F) = 0 \tag{3}$$

The trajectory generation problem can be independently solved for each component $i = 1 \ldots 4$ of the geometric state (i.e. each flat output). The parameter $t_{F,i} \in \mathbb{R}^+$ and $S_i \in \mathcal{C}^3([0,\ t_{F,i}], \mathbb{R})$ have to be computed for each component, taking into account bounds on the derivatives.These bounds are directly considered up to the third order. Nevertheless, since jerk has to be continuous and bounded, snap (second derivative of acceleration) has to be bounded too. In summary, the set of constraints is:

$$\begin{cases} S_i(0) = x_0,\ \dot{S}_i(0) = v_0 \\ \ddot{S}_i(0) = a_0,\ \dddot{S}_i(0) = 0 \\ S_i(t_{F,i}) = x_F,\ \dot{S}_i(t_{F,i}) = v_F \\ \dot{S}_i(t_{F,i}) = a_F,\ \dddot{S}_i(t_{F,i}) = 0 \\ \forall t \in [0,\ t_{F,i}], \begin{cases} |\dot{S}_i(t)| \le v_{max} \in \mathbb{R}^{*+} \\ |\ddot{S}_i(t)| \le a_{max} \in \mathbb{R}^{*+} \\ |\dddot{S}_i(t)| \le j_{max} \in \mathbb{R}^{*+} \\ |\ddddot{S}_i(t)| \le s_{max} \in \mathbb{R}^{*+} \end{cases} \end{cases} \tag{4}$$

In order to minimize $t_{F,i}$, full speed has to be reached as soon as possible and maintained as long as possible. Let D refer to this phase of constant velocity and let $v_D \in [-v_{max}, v_{max}]$ be the value of velocity during this phase. This means that time spent to reach it has to be minimized: velocity have to get from v_0 to v_D and then from v_D to v_F as quickly as possible. Hence time spent at maximum acceleration during those two phases has to be maximized. It implies to minimize the durations of acceleration variations. Let B refer to the phase of constant acceleration during the first velocity variation and $a_B \in [-a_{max}, a_{max}]$ be this constant acceleration. Let G be the phase of constant acceleration during the second velocity variation and $a_G \in [-a_{max}, a_{max}]$ this constant acceleration. The phase in which acceleration variates from a_0 to a_B is noted A, and C is used to refer to the phase in which acceleration variates from a_B to zero. The phases where acceleration variates from zero to a_G and from a_G to a_F are noted E and H respectively. All these phases are illustrated with an example in Fig. 3. The principle holds for higher derivatives: time spent at maximum jerk during acceleration variations has to be maximized and durations of jerk variations have to be minimized. This implies to maximize time spent at maximum snap during jerk variations and to minimize the durations of snap variations. This very last part is easy since, following our approach, snap can be discontinuous: snap variation is actually a snap commutation of duration zero ($\forall t \in [0, t_{F,i}]$, $\ddddot{S}_i(t) \in \{-s_{max}, 0, s_{max}\}$). In other words, $\ddddot{S}_i$ is a piecewise constant function, which implies that S_i is a piecewise polynomial function (a spline) of the fourth order.

In the discussion above, we have mentioned seven main phases in the (local) trajectory. For three of them (B, D, G), acceleration is zero and hence snap is also zero. The four others (A, C, E, H) are divided into three sub-phases $(1, 2, 3)$. Phases 1 and 3 correspond to jerk variations and then $|\ddddot{S}_i(t)| = s_{max}$. In phase 2, jerk is constant hence snap is zero. The sign of the snap is opposed during phases 1 and 3, which have the same duration. Tab. I presents the notation used for the durations of the phases and the expressions of the snap as functions of a_B and a_G.

The expression of the snap during phase A_1 is explained as follows: if $a_B > a_0$, $\ddot{S}_i$ has to be increasing, which implies that $\dddot{S}_i$ has to be positive. Since $\dddot{S}_i(0) = 0$, $\dddot{S}_i$ has to be increasing during phase A_1, and thus $\ddddot{S}_i(t) = +s_{max}$. Following the same reasoning, $\ddddot{S}_i(t) = -s_{max}$ if $a_B < a_0$. If $a_B = a_0$, then phase A is not needed, and thus $\ddddot{S}_i(t) = 0$. A similar reasoning can be applied to understand the expressions for phases C, E and H.

Note that this closed-form solution is an intuitive way of minimize flying time. However, there is no guarantee of optimality because of our choices for simplification. The actual optimal solution could be found by applying Pontryagin maximum principle to the corresponding optimal control problem (snap as scalar control input and inequality constraints on the state) but it would imply heavy numerical computation, making it not suited to our needs. The sub-optimality factor induced by our choices is yet to be measured. From now on, both *optimal time/velocity* are to be read as: *optimal time/velocity with respect to our closed-form solution.*

D. Duration of the phases

At this point, the local trajectory for one coordinate (i.e its corresponding spline) is defined by a_B, a_G and the duration

TABLE I
VALUE OF THE SNAP AND DURATION OF EACH PHASE OF THE SPLINE

Phase	Value of the Snap	Duration
A_1	sign$(a_B - a_0).s_{max}$	$t_{A,1}$
A_2	0	$t_{A,2}$
A_3	sign$(a_0 - a_B).s_{max}$	$t_{A,1}$
B	0	t_B
C_1	-sign$(a_B).s_{max}$	$t_{C,1}$
C_2	0	$t_{C,2}$
C_3	sign$(a_B).s_{max}$	$t_{C,1}$
D	0	t_D
E_1	sign$(a_G).s_{max}$	$t_{E,1}$
E_2	0	$t_{E,2}$
E_3	-sign$(a_G).s_{max}$	$t_{E,1}$
G	0	t_G
H_1	sign$(a_F - a_G).s_{max}$	$t_{H,1}$
H_2	0	$t_{H,2}$
H_3	sign$(a_G - a_F).s_{max}$	$t_{H,1}$

of all phases. This subsection explains how it can be defined using a single parameter: v_D. First, $t_{A,1}$, $t_{A,2}$, $t_{C,1}$ and $t_{C,2}$ can be expressed as functions of a_B, a_0, j_{max} and s_{max}. The same goes for $t_{H,1}$, $t_{H,2}$, $t_{E,1}$ and $t_{E,2}$ as functions of a_G, a_F, j_{max} and s_{max}. We provide next explanations for $t_{A,1}$ and $t_{A,2}$. Let us define δ_{B0} =sign$(a_B - a_0)$. From Tab. I and (4), we can write:

$$a_B = \delta_{B0}.s_{max}.t_{A,1}^2 + \delta_{B0}.s_{max}.t_{A,1}.t_{A,2} + a_0 \qquad (5)$$

and $\dddot{S}_i(t_{A,1}) = \delta_{B0}.s_{max}.t_{A,1}$, which implies $|\dddot{S}_i(t_{A,1})| = s_{max}.t_{A,1} \leq j_{max}$, and thus:

$$s_{max}.t_{A,1}^2 \leq \frac{j_{max}^2}{s_{max}} \qquad (6)$$

Let us note $a_{lim} = \frac{j_{max}^2}{s_{max}}$. Phase A_2 is only needed when phases A_1 and A_3 are not enough to reach a_B. If there is no phase A_2, then $t_{A,2} = 0$. Then, using (5) and (6), we can write: $|a_B - a_0| = s_{max}.t_{A,1}^2 \leq a_{lim}$. If $|a_B - a_0| > a_{lim}$, then $t_{A,2} \neq 0$ (phase A_2 is needed), and $t_{A,1} = \frac{j_{max}}{s_{max}}$ (its maximum value). In that case, $t_{A,2} = \frac{|a_B - a_0|}{j_{max}} - \frac{j_{max}}{s_{max}}$. If $t_{A,2} = 0$, then $t_{A,1} = \sqrt{\frac{|a_B - a_0|}{s_{max}}}$. The principle is the same for phases C, E and H and formulas are similar (details are not provided here due to limited space). For phase C, $|a_B - a_0|$ is replaced by $|a_B|$, by $|a_G|$ in phase E and by $|a_G - a_F|$ in phase H. The spline is now defined by $(a_B, a_G, t_B, t_D, t_G)$. Both couples (a_B, t_B) and (a_G, t_G) can actually be expressed as a function of v_D. Using Tab. I and (4), v_D can be expressed as a function of $(a_B, t_{A,1}, t_{A,2}, t_B, t_{C,1}, t_{C,2})$ and hence as a function of (a_B, t_B). Let $V_B : (a_B, t_B) \mapsto v_D$ be this function. Let us also define $v_B^{min} = V_B(-a_{max}, 0)$ and $v_B^{max} = V_B(a_{max}, 0)$. It is possible to show (not detailed here due to limited space) that, for all v_D in $[v_B^{min}, v_B^{max}]$, there is a unique $a_B \in ([-a_{max}, \min(0, a_0)] \cup [\max(0, a_0), a_{max}])$ such that $V_B(a_B, 0) = v_D$. If $v_D < v_B^{min}$, then $a_B = -a_{max}$ and phase B is needed. Its duration is simply $t_B = |v_D - v_B^{min}|/a_{max}$. If $v_D > v_B^{min}$, then $a_B = a_{max}$ and $t_B = |v_D - v_B^{max}|/a_{max}$. Hence, as previously announced (a_B, t_B) can be expressed as a function of v_D. The same goes for (a_G, t_G).

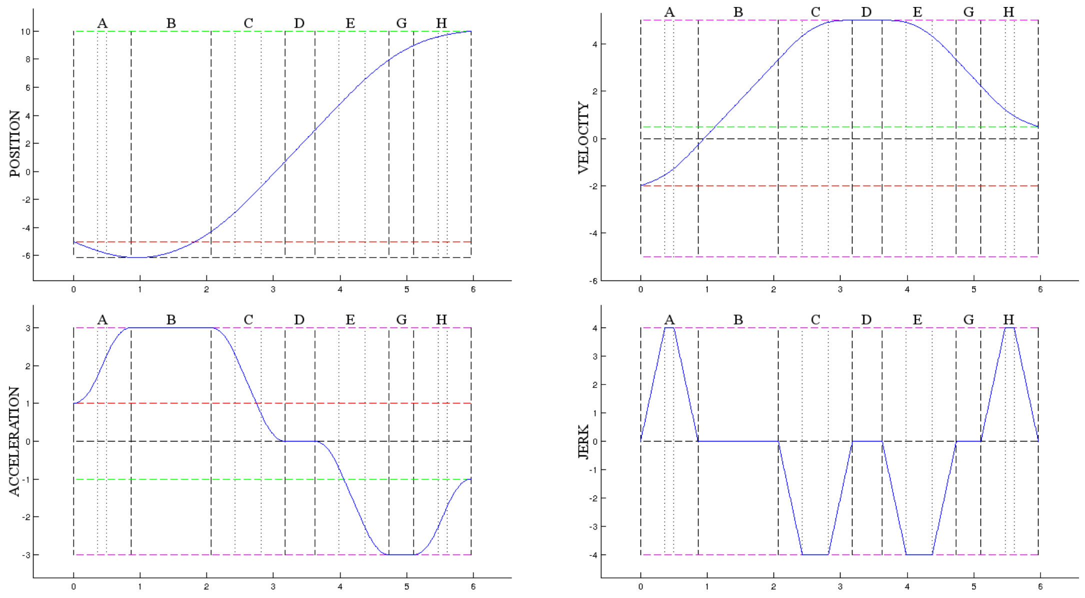

Fig. 3. Example of trajectory provided by the local planner for one coordinate. The bounds of each derivative are represented by pink dashed lines. Red and green dashed lines represent initial and final values, respectively. In phase A, a_B is reached with respect to maximum jerk and snap constraints. In phase B, the system travels towards v_D at constant acceleration a_B. C is a deceleration phase and D the phase of constant velocity. H, G and E are the symmetric phases of A, B and C, respectively. Note that, in order to show all the phases, $|v_D| = v_{max}$ and $|a_B| = |a_D| = a_{max}$ in this example. In a general case, some phases can have zero duration.

At this point the spline is defined by (v_D, t_D). Let t_C be the value of the time parameter at the end of phase C, and t_E its value at the beginning of phase E. As explained previously, these two values can be expressed as a function of v_D. This is also the case for $S_i(t_C)$ and $S_i(t_E)$. Let Δ_S be the function $v_D \mapsto S_i(t_E) - S_i(t_C)$. If $v_D \neq 0$, then $t_D = \Delta_S(v_D)/v_D$. Therefore, the spline is completely defined by the sole value of v_D. Note that, necessarily:

$$v_D.\Delta_S(v_D) \leq 0 \tag{7}$$

E. Optimal velocity

Once the spline is defined by v_D, this section explains how to compute the optimal value v_{opt} that minimizes $t_{F,i}(v_D)$. Let $\mathcal{V}_{valid}$ be the set of values of v_D that meet the constraint (7) and let us note $\mathcal{V}_0 = [\min(0, v_{opt}), \max(0, v_{opt})]$. For synchronization purposes, it is necessary that $\mathcal{V}_0 \subset \mathcal{V}_{valid}$, which implies:

$$\forall v \in \mathcal{V}_0, v.\Delta_S(0) \leq 0 \tag{8}$$

Let us note $\delta_0 = $sign$(\Delta_S(0))$. Since minimizing $t_{F,i}$ requires maximizing $|v_D|$, (8) implies that, if Δ_S has zeros between $v = 0$ and $v = \delta_0.v_{max}$, then v_{opt} is the one of lowest absolute value. If not, $v_{opt} = \delta_0.v_{max}$.

F. Synchronization

The trajectory generation problem is now solved for each component independently. Hence, there are four different values $(t_{F,i})_i, i = 1 \ldots 4$. This subsection explains how to synchronize these one-dimensional trajectories. Solving each problem provides a couple $(t_{F,i}, v_{opt,i})$ and the associated interval $\mathcal{V}_{0,i}$. The purpose of the definition of v_{opt} provided in the previous subsection is to guarantee that $v_D \mapsto t_{F,i}(v_D)$ is continuously strictly monotonic on $\mathcal{V}_{0,i}$. Furthermore, $\lim_{v_D \to 0} t_{F,i}(v_D) = +\infty$. This implies that, for any t in $[t_{F,i}, +\infty[$, there is a unique $v \in \mathcal{V}_{0,i}$ such that $t_{F,i}(v) = t$. This property is used to synchronize the components. The slowest component (of index i_{slow}) is identified and a simple dichotomous search is used to find the unique $v_D \in \mathcal{V}_{0,i}$ such that $t_{F,i}(v_D) = t_F^{i_{slow}}(v_{opt}^{i_{slow}})$ for the other components. As a result, all the components have the same final time $t_F^{i_{slow}}$.

III. TWO GLOBAL PLANNERS

A. Decoupled approach

This subsection explains how the local planner presented in the previous section can be used in a decoupled approach to motion (trajectory) planning. A decoupled approach consists of two stages: 1) planning a geometrically valid path in $\mathcal{X}_G$ for the minimum bounding sphere of the quadrotor; 2) transforming this path into a trajectory in $\mathcal{X}_K$. In our current implementation, a bi-directional RRT algorithm is applied to explore $\mathcal{X}_G$, and linear interpolation is used to connect sampled states. Thus, the resulting path is a concatenation of n collision-free straight line segments in $\mathcal{X}_G$, $\{x_0^G, x_F^G\}_i, i = 1 \ldots n$. The local trajectory planner is then applied to each pair of intermediate states $\{[x_0^G, 0, 0], [x_F^G, 0, 0]\}_i$. The resulting trajectory is collision-free, since the center of mass of the quadrotor follows the initial geometric path. However, it is far from being time-optimal because of the imposed stops at the ends of local paths. A trajectory optimization method can be applied to reduce the time of this initial solution. We have implemented a simple but efficient method based on the random shortcut algorithm [10]. This iterative, anytime algorithm works as follows: at each iteration, two states, x_1^K and x_2^K, are randomly selected from the overall trajectory. Let us call x_A^K the initial state of the local trajectory in which

x_1^K lies, and x_B^K the final state of the local trajectory in which x_2^K lies. The local planner is then applied to generate three new local trajectories between $\{x_A^K, x_1^K\}$, $\{x_1^K, x_2^K\}$ and $\{x_2^K, x_B^K\}$[1]. If they are collision-free, the cost of the trajectory $x_A^K \to x_1^K \to x_2^K \to x_B^K$ is computed, the cost here being the overall flying time. If this cost is lower than the one of $x_A^K \to \ldots \to x_B^K$, this portion of the overall trajectory is replaced by the new one. This step is repeated until a given execution time, a given number of iterations or a given gain of the cost is reached. This simple approach is computationally very fast and efficient. It quickly returns high-velocity, agile trajectories in cluttered environments. Nevertheless, the approach is incomplete, in the sense that it is unable to find solutions to some problems involving aggressive maneuvers, such as the one illustrated in Fig. 1. The approach is also unsuitable to solve problems involving the transportation of a rigidly-attached large object, whose orientation is defined by that of the quadrotor, and is therefore dependent on its velocity and acceleration.

B. Direct use as a steering method

To overcome the limitations of the decoupled approach, the proposed local trajectory planner can be directly used as a *steering method* to connect sampled states within a sampling-based motion planning algorithm. We investigate here its application within the RRT algorithm, but other planners could be used. This approach is able to solve the aforementioned classes of problems involving aggressive maneuvers or large object transportation. Nevertheless, there are drawbacks and difficulties that need to be mentioned. A first issue is the higher dimension of the state space, compared to the decoupled approach in which RRT is applied to explore the geometric state space $\mathcal{X}_G$ (which is of dimensions 4, or 3 if ψ is fixed). Indeed, RRT is now directly applied to explore $\mathcal{X}_K$, which is a 12-dimensional space in the present case. In addition, the local trajectory planner, although computationally efficient, is orders of magnitude more expensive than a simple linear interpolation, which significantly penalizes the overall computational performance of the motion planner. Another important issue is that there is no natural metric in $\mathcal{X}_K$ (as discussed in [1]). Sampling-based motion planners rely on a good metric in order to accurately select the neighbor states. In our case, finding a good metric is as hard as solving the local planning problem itself (i.e. finding the minimal-time trajectory to connect to states with differential constraints). Using a bad metric leads to a bad exploration of the state space and thus impairs convergence of the RRT algorithm. Nevertheless, despite these weaknesses, the proposed local planner used as a steering method allows us to solve difficult problems with a reasonable computation time, as will be shown in the next section.

[1]Note that the portion of the initial trajectory between x_A^K and x_1^K (idem for x_2^K and x_B^K) is different to the local trajectory generated by a new call to the local planner, because zero jerk is imposed on x_1^K (and on x_2^K) when splitting the trajectory.

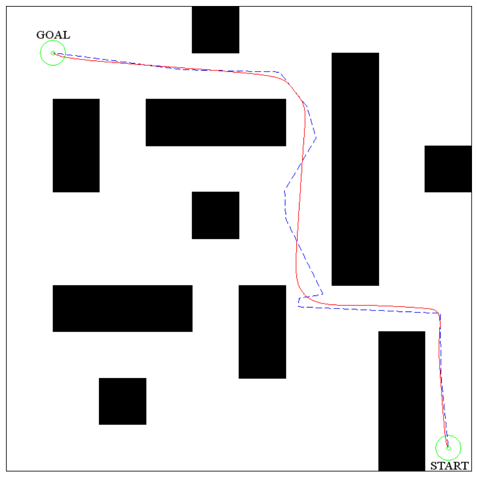

Fig. 4. Motion planning problem in a cluttered workspace. Start and goal positions of the quadrotor are represented by green circles. The dashed blue line is the path of the center of mass before optimization. The plain red line is the resulting trajectory after optimization.

IV. SIMULATIONS

A. Computational performance of the local planner

To evaluate the computational cost of the local trajectory planner, we performed experiments consisting of successive calls from randomly chosen couples of kinodynamic states. The CPU time (averaged over 10^4 calls) on a single core of an Intel Xeon W3520 processor at 2.67GHz is 2.4×10^{-4} seconds. This performance is for an implementation in C, integrated in our motion planning software, Move3D [11]. Although this can appear as extremely fast, it has to be compared to the cost of creating local paths using a basic linear interpolation, which is about 2×10^{-6} seconds using the same software. Thus, computing local trajectories in $\mathcal{X}_K$ is two orders of magnitude more expensive that computing local paths in $\mathcal{X}_G$.

B. Decoupled approach and optimization performances

For the evaluation of the decoupled motion planning approach, we performed several experiments in a relatively simple scenario, represented in Fig. 4. The problem is to find a time-optimal trajectory for a quadrotor to fly between two points at opposed corners of a 10×10 meter workspace filled with box-shaped obstacles. In order to simplify the interpretation of results, we impose constant altitude and fixed yaw angle. The radius of the bounding sphere considered for collision detection is 0.27 meters. The bounds on the derivatives of position are $v_{max} = 5$ m.s^{-1}, $a_{max} = 10$ m.s^{-2}, $j_{max} = 20$ m.s^{-3} and $s_{max} = 50$ m.s^{-4}. Note that these values are for now an arbitrary choice. Bounds corresponding to physical limits of a robot have to be measured experimentally.

The proposed decoupled approach, using RRT as geometric path planner and the random shortcut algorithm for trajectory optimization, is able to provide a time-optimized trajectory in less than 3 seconds of CPU time. One half of this time is spent in the geometric exploration, and the other half in trajectory generation and optimization. As

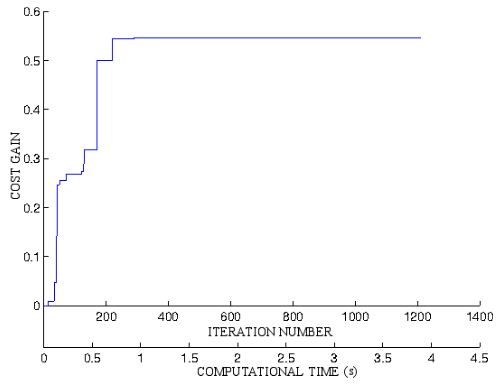

Fig. 5. Performances of the trajectory optimization method in terms of iteration number and computational time.

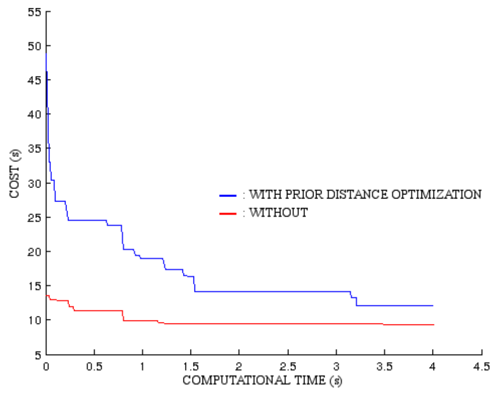

Fig. 6. Evolution of the cost during optimization with and without prior distance optimization.

illustrated in Fig. 5, about 300 iterations of the shortcut algorithm (performed in about 1 second) are sufficient for convergence. The plot represents the cost gain (i.e. the flying time decrease). Flying time of the optimized trajectory is 8.3 seconds.

An alternative strategy to obtain a time-optimized trajectory with a decoupled approach could be to optimize first the length of the geometric path. This could be done using the RRT* algorithms instead of RRT (as proposed in [8]), or applying an optimization post-processing (i.e. path smoothing) to the solution obtained with RRT. However, results of our experiments show that this is a wrong strategy. Fig. 6 shows that the convergence of the trajectory optimization process is much slower when starting from the shortest path than when using the “rough” path provided by RRT.

C. Direct use as a steering method: the slot problem

Finally, we present results obtained using the proposed local planner as steering method inside RRT. The method was applied to solve the difficult *slot problem* illustrated in Fig. 1. The environment is a box of $10 \times 10 \times 3$ meters, divided in two halves by a series of aligned obstacles separated by 0.40 meters. The start position is in one side of this series of obstacles and the goal is on the other side. The diameter of the bounding sphere being of 0.54 meters, this problem can not be solved by the decoupled approach, as there is no collision-free path for the bounding sphere. Solving this problem requires taking the orientation of the quadrotor into account, which implies considering dynamics for motion planning. Bounds for the time-derivatives of the position are the same than in the previous example. The trajectory represented in Fig. 1 has been obtained in 90 seconds with a basic RRT algorithm using our local planner as steering method. This computational time can be explained by the fact that, while going through the slot, velocity and acceleration of the quadrotor must have very specific values taken in a small interval of possibilities. The empty space between two obstacles in the configuration space becomes a very narrow passage when mapped into the kinodynamic state space.

V. CONCLUSION

We have presented a method to compute physically-realistic local trajectories of quadrotors. The method is computationally efficient, which enables its integration into several types of global motion planning approaches. Used within a decoupled approach, based on geometric path planning and subsequent trajectory generation and optimization, motion planning problems in moderately-constrained environments can be solved in very short computing time. Very difficult problems, requiring a direct exploration of the kinodynamic state space, can also be solved in reasonably-short computing time using the proposed local planner as a steering method. As future work, we intend to further investigate the issue of finding an appropriate metric in the kinodynamic state space that could speed up the RRT algorithm. We also plan to compare the trajectories obtained with of our local planner to solutions provided by optimal control methods in order to better evaluate their quality, as it is done in [12]. Finally, experimental validation will soon be conducted in our testbed.

REFERENCES

[1] S. M. LaValle and J. J. Kuffner. Randomized Kinodynamic Planning. The International Journal of Robotics Research, no. 20:378-400, 2001.
[2] S. M. LaValle. Planning algorithms. Cambridge university press, 2006.
[3] Y. Bouktir, M. Haddad and T. Chettibi. Trajectory planning for a quadrotor helicopter. Proc. Mediterranean Conference on Control and Automation, 2008.
[4] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. Proc. ICRA, 2011.
[5] M. Hehn and R. D’Andrea. Quadrocopter Trajectory Generation and Control. IFAC World Congress, vol. 18, no. 1, pp. 1485-1491. 2011.
[6] E. Koyuncu and G. Inalhan. A probabilistic B-spline motion planning algorithm for unmanned helicopters flying in dense 3D environments. Proc. IROS, 2008.
[7] P. M. Bouffard and S. L. Waslander. A hybrid randomized/nonlinear programming technique for small aerial vehicle trajectory planning in 3D. Planning, Perception and Navigation for Intelligent Vehicles (PPNIV) 63, 2009.
[8] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. Proc. ISRR, 2013.
[9] M. W. Mueller, M. Hehn and R. D’Andrea. A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification. Proc. IROS, 2013.
[10] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. The International Journal of Robotics Research, no. 26:845-863, 2007.
[11] T. Siméon, J.-P. Laumond and F. Lamiraux. Move3D: a generic platform for path planning. Proc. ISATP, 2001.
[12] M. Hehn, R. Ritz and R. D’Andrea. Performance Benchmarking of Quadrotor Systems Using Time-Optimal Control. Autonomous Robots, Volume 33,Numbers 1-2, 2012.