

FASTER: Fast and Safe Trajectory Planner for Flights in Unknown Environments

Jesus Tordesillas¹, Brett T. Lopez¹, Michael Everett¹, and Jonathan P. How¹

Abstract

Planning high-speed trajectories for UAVs in unknown environments requires algorithmic techniques that enable fast reaction times to guarantee safety as more information about the environment becomes available. The standard approach to ensure safety is to enforce a “stop” condition in the free-known space. However, this can severely limit the speed of the vehicle, especially in situations where much of the world is unknown. Moreover, the ad-hoc time and interval allocation scheme usually imposed on the trajectory also leads to conservative and slower trajectories. This work proposes FASTER (Fast and Safe Trajectory Planner) to ensure safety without sacrificing speed. FASTER obtains high-speed trajectories by enabling the local planner to optimize in both the free-known and unknown spaces. Safety guarantees are ensured by always having a feasible, safe back-up trajectory in the free-known space at the start of each replanning step. The Mixed Integer Quadratic Program formulation proposed allows the solver to choose the trajectory interval allocation, and the time allocation is found by a line search algorithm initialized with a heuristic computed from the previous replanning iteration. This proposed algorithm is tested extensively both in simulation and in real hardware, showing agile flights in unknown cluttered environments with velocities up to 7.8 m/s. To demonstrate the generality of the proposed framework, FASTER is also applied to a skid-steer robot, and the maximum speed specified for the robot (2 m/s) is achieved in real hardware experiments.

Keywords

UAV, Path Planning, Trajectory Optimization, Convex Decomposition

Supplementary material

- **Code:** To be released soon.
- **Gazebo worlds:** <https://github.com/jtorde>
- **Video:** <https://www.youtube.com/watch?v=fkkkgomkX10>

Introduction

Despite its numerous applications, high-speed UAV navigation through unknown environments is still an open problem. The desired high speeds together with partial observability of the environment and limits on payload weight makes this task especially challenging for aerial robots. Safe operation, in addition to flying fast, is also critical but difficult to guarantee since the vehicle must repeatedly generate collision-free, dynamically feasible trajectories in real-time with limited sensing. Similar to the model predictive control (MPC) literature, safety is guaranteed by ensuring a feasible solution exists indefinitely.

If we consider $\mathbb{R}^3 = \mathcal{F} \cup \mathcal{O} \cup \mathcal{U}$ where \mathcal{F} , \mathcal{O} , \mathcal{U} are disjoint sets denoting free-known, occupied-known, and unknown space respectively, the following hierarchical planning architecture is commonly used: a global planner first finds the shortest piece-wise linear path from the UAV to the goal, avoiding the known obstacles \mathcal{O} . Then, a local planner finds a dynamically feasible trajectory in the direction given by this global plan. This local planner should find a *fast and safe* trajectory that leads the UAV to the goal. These two requirements of **safety** and **speed** represent the following trade-off: on one hand, safety argues for short trajectories completely contained in \mathcal{F} and end points not

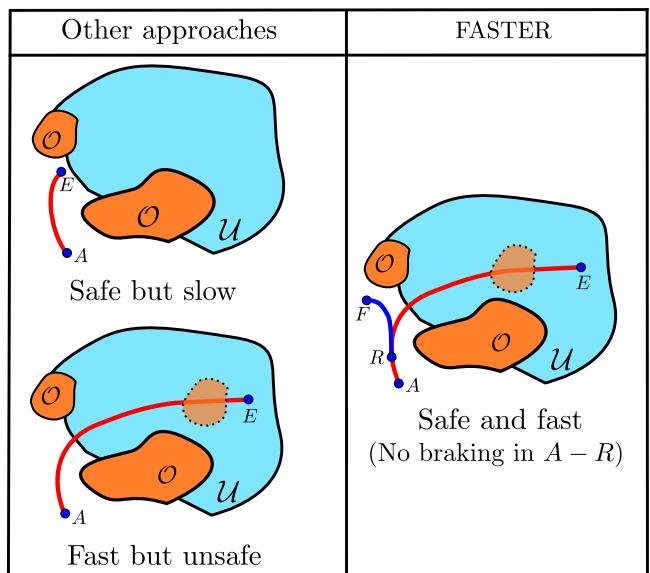


Figure 1. Safety and Speed trade-off. \mathcal{O} is the occupied-known space (orange), and \mathcal{U} is the unknown space (blue). A and E are respectively the start and goal locations of the local plan.

¹Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA

Corresponding author:

Jesus Tordesillas, Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA
Email: jtorde@mit.edu

necessarily near the global plan. As a final stop condition is needed to guarantee safety, short trajectories are generally much slower than long trajectories because the braking maneuver propagates backwards from the end to the initial state of the trajectory. On the other hand, speed argues for longer planned trajectories (usually extending farther than \mathcal{F}) and end points near the global plan.

The typical way to solve the speed versus safety trade-off is to ensure safety by planning only in \mathcal{F} , and then impose a final stop condition near the global plan. This can be achieved by either generating motion primitives that do not intersect $\mathcal{O} \cup \mathcal{U}$ (Mueller et al. 2015; Lopez and How 2017a,b; Tordesillas et al. 2019a), or by constructing a convex representation of \mathcal{F} to be used in an optimization (Deits and Tedrake 2015; Liu et al. 2017b; Preiss et al. 2017). The main limitation of these works is that safety is guaranteed at the expense of higher speeds, especially in scenarios where \mathcal{F} is small compared to $\mathcal{O} \cup \mathcal{U}$. This paper presents an optimization-based approach that solves this limitation by solving for *two* optimal trajectories at every planning step (see Fig. 1): The **first trajectory** is in $\mathcal{U} \cup \mathcal{F}$ and ensures a long planning horizon with an end point on the global plan. The **second trajectory** is in \mathcal{F} , starts from a point along the first trajectory, and it may deviate from the global plan. Only a portion of the first trajectory is actually implemented by the UAV (therefore satisfying the **speed** requirement), while the second trajectory guarantees **safety**, since it is contained in \mathcal{F} and available at the start of every replanning step. This second trajectory is only implemented if the optimization problem becomes infeasible in the next replanning steps.

A second limitation, specially for the optimization-based approaches that use convex decomposition, is the choice of the interval and time allocation method. The interval allocation decides in which polyhedron each interval of the trajectory will be located, whereas the time allocation deals with the time spent on each interval (see Fig. 2). In order to simplify the interval allocation, a common choice is to set the number of intervals to be the same as the number of polyhedra found, forcing each interval to be in one specific polyhedron. This forces the optimizer to select the end points of each trajectory segment within the overlapping area of two consecutive polyhedra, and therefore possibly leading to more conservative or longer trajectories. Moreover, since a different time for each interval has to be found, the time allocation calculation is harder, leading to higher replanning times when using optimization techniques to allocate this time, and to non-smooth or infeasible trajectories when imposing an ad-hoc time allocation. To overcome this limitation, FASTER allows the solver to decide the interval allocation by using a number of intervals higher than the number of polyhedra found (Landry et al. 2016) and by allocating the same time for all the intervals. This time allocation method is efficiently found through a line search algorithm initialized with the solution at the previous replanning iteration.

The planning framework proposed is called **FASTER - FAst and Safe Trajectory PlannER**, and is an extension of our two published conference papers (Tordesillas et al. 2019a,b). In summary, this work has the following contributions:

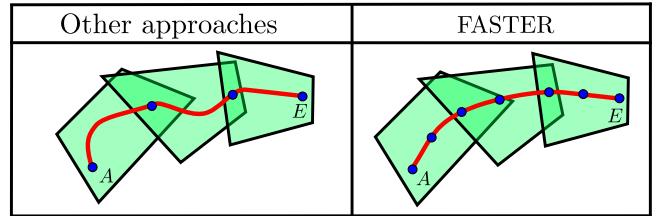


Figure 2. Interval and Time Allocation when using a convex decomposition (■). A and E are respectively the start and goal locations of the local plan.

- A framework that ensures feasibility of the entire collision avoidance algorithm and guarantees safety without reducing the nominal flight speed by allowing the local planner to plan in $\mathcal{F} \cup \mathcal{U}$ while always having a safe trajectory in \mathcal{F} .
- Reduced conservatism of the time and interval allocation compared to prior ad-hoc approaches by efficiently finding the time allocated from the result of the previous replanning iteration and then allowing the optimizer to choose the interval allocation.
- Extension of our previous work (Tordesillas et al. 2019a) by proposing a way to compute very cheaply a heuristic of the cost-to-go needed by the local planner to decide which direction is the best one to optimize towards.
- Simulation and hardware experiments showing agile flights in completely unknown cluttered environments, with velocities up to 7.8 m/s, two times faster than previous state-of-art methods (Tordesillas et al. 2019b,a). FASTER is also tested on a skid-steer robot, showing hardware experiments at the top speed of the robot (2 m/s).

RELATED WORK

Different methods have been proposed in the literature for planning, mapping, and the integration of these two.

Planning for UAVs can be classified according to the specific formulation of the optimization problem and the operating space of the local planner.

As far as the **optimization problem** itself is concerned, most of the current state-of-the-art methods exploit the differential flatness of the quadrotors, and, using an integrator model, minimize the squared norm of a derivative of the position to find a dynamically feasible smooth trajectory (Mellinger and Kumar 2011; Van Nieuwstadt and Murray 1998; Richter et al. 2016). When there are obstacles present, some methods include them in the optimization problem, while others do not.

There are approaches where the obstacle constraints (and sometimes also the input constraints) are checked after solving the optimization problem: some of them use stitched polynomial trajectories that pass through several waypoints obtained running RRT-based methods (Mellinger and Kumar 2011; Richter et al. 2016; Loianno et al. 2017), while others use closed-form solutions or motion-primitive libraries (Mueller et al. 2015; Florence et al. 2016; Lopez and How 2017b,a; Bucki and Mueller 2019; Ryll et al. 2019; Spitzer et al. 2019). These closed-form solutions are also

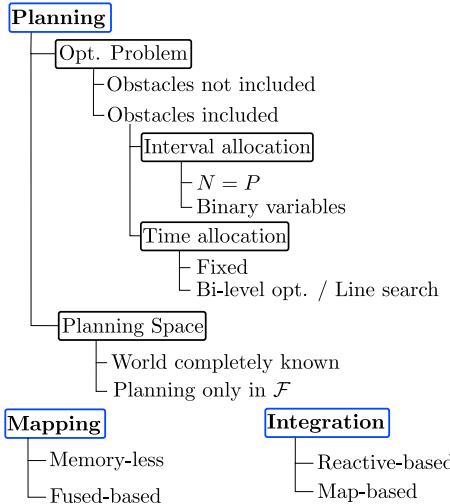


Figure 3. Classification of the state-of-the-art techniques for planning, mapping the the integration between these two.

used to search over the state space (Liu et al. 2017a, 2018b; Zhou et al. 2019). As the optimization problem is unaware of where the obstacles are, these methods either are limited to short trajectories unable to perform complex maneuvers around obstacles, or usually have to do a computationally expensive search to be able to generate a trajectory around obstacles.

The other approach is to include the obstacles directly in the optimization problem. This can be done in the cost function by penalizing the distance to the obstacles (Oleynikova et al. 2016, 2018), but this usually leads to computationally expensive distance fields representations and/or non-convex optimization problems. Another option is to encode the shape of the obstacles in the constraints using successive convexification (Mao et al. 2018; Augugliaro et al. 2012; Schulman et al. 2014) or a convex decomposition of the environment (Liu et al. 2018a, 2017b; Watterson et al. 2018; Gao et al. 2019; Lai et al. 2019; Rousseau et al. 2019). The convergence of successive convexification depends heavily on the initial guess, and is usually not suitable for real-time planning in unknown cluttered environments. The convex decomposition approach is usually done by decomposing the free-known space as a series of P overlapping polyhedra (Liu et al. 2017b; Deits and Tedrake 2015; Preiss et al. 2017). As the trajectory is usually decomposed of N third (or higher)-degree polynomials, to guarantee that the whole trajectory is inside the polyhedra, Bézier Curves (Preiss et al. 2017; Sahingoz 2014), or the sum-of-squares condition (Deits and Tedrake 2015; Landry et al. 2016) are often used. Moreover, for a trajectory there is both an interval (in which polyhedron each interval is) and a time allocation (how much time is assigned to each interval) problem. For the **interval allocation**, a usual decision is to use $N = P$ intervals, and force each interval to be inside its corresponding polyhedron (Preiss et al. 2017). However, this sometimes can be very conservative, since the solver can only choose to place the two extreme points of each interval in the overlapping area of two consecutive polyhedra. Another option, but usually with higher computation times, is to use binary variables (Landry et al. 2016; Deits and Tedrake 2015) to allow the solver to choose the specific interval allocation. For the **time**

allocation, different techniques are used. One is to impose a fixed time allocation using a specific velocity profile (Liu et al. 2017b), which can be very conservative, or cause infeasibility in the optimization problem. Another one is to encode the optimization problem as a bi-level optimization problem, and use line search or gradient descent to iteratively obtain these times (Richter et al. 2016; Preiss et al. 2017).

With regard to the **planning space** of the local planner, several approaches have been developed. One approach is to use only the most recent perception data (Lopez and How 2017b,a), which requires the desired trajectory to remain within the perception sensor field of view. An alternative strategy is to create and plan trajectories in a map of the environment built using a history of perception data. Within this second category, in some works (Schouwenaars et al. 2002; Tordesillas et al. 2019a; Oleynikova et al. 2018), the local planner only optimizes inside \mathcal{F} , which guarantees safety if the local planner has a final stop condition. However, limiting the planner to operating in \mathcal{F} and enforcing a terminal stopping condition can lead to conservative, slow trajectories (especially when much of the world is unknown). Higher speeds can be obtained by allowing the local planner to optimize in both the free-known and unknown space ($\mathcal{F} \cup \mathcal{U}$), but with no guarantees that the trajectory is safe or will remain feasible.

Moreover, two main categories can be highlighted in the **mapping** methods proposed in the literature: memory-less and fused-based methods. The first category includes the approaches that rely only on instantaneous sensing data, using only the last measurement, or weighting the data (Dey et al. 2016; Florence et al. 2018; Gao et al. 2019; Florence et al. 2016). These approaches are in general unable to reason about obstacles observed in the past (Lopez and How 2017a,b), and are specially limited when a sensor with small FOV is used. The second category is the fusion-based approach, in which the sensing data are fused into a map, usually in the form of an occupancy grid or distance fields (Lau et al. 2010; Oleynikova et al. 2017). Two drawbacks of these approaches are the influence of the estimation error, and the fusion time.

Finally, several approaches have been proposed for the **integration** between the planner and the mapper: reactive and map-based planners. Reactive planners often use a memory-less representation of the environment, and closed-form primitives are usually chosen for planning (Lopez and How 2017a,b). These approaches often fail in complex cluttered scenarios. On the other hand, map-based planners usually use occupancy grids or distance fields to represent the environment. These planners either plan all the trajectory at once or implement a Receding Horizon Planning framework, optimizing trajectories locally and based on a global planner. Moreover, when unknown space is also taken into consideration, several approaches are possible: some use optimistic planners that consider unknown space as free (Pivtoraiko et al. 2013; Chen et al. 2016), while in other works an optimistic global planner is used combined with a conservative local planner (Oleynikova et al. 2016, 2018).

FASTER

The notation used throughout the paper is shown in Fig. 4: \mathcal{M} is a sliding map centered on L , the current position

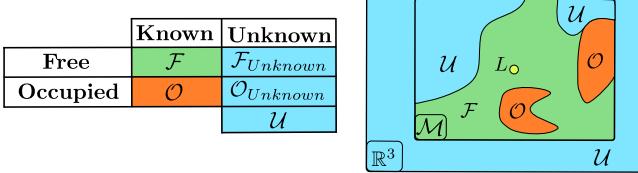


Figure 4. Notation used for the different spaces. L is the current position of the UAV, and \mathcal{M} is the sliding map around the vehicle.

of the UAV. \mathcal{F} and \mathcal{O} will denote the free-known and occupied-known spaces respectively. Similarly, $\mathcal{F}_{Unknown}$ and $\mathcal{O}_{Unknown}$ will denote the free-unknown and occupied-unknown spaces respectively. The total unknown space, denoted as \mathcal{U} , is therefore $\mathcal{U} = \mathcal{F}_{Unknown} \cup \mathcal{O}_{Unknown}$, and \mathcal{F} and \mathcal{O} are completely contained inside the map ($\mathcal{F} \cup \mathcal{O} \subseteq \mathcal{M}$), and all the space outside the map is inside \mathcal{U} ($\mathbb{R}^3 \setminus \mathcal{M} \subseteq \mathcal{U}$). Note also that FASTER is completely in 3D, but some illustrations are in 2D for visualization purposes.

Mapping

A body-centered sliding map \mathcal{M} (in the form of an occupancy grid map) is used in this work. A rolling map is desirable since it reduces the influence of the drift in the estimation error. We fuse a depth map into the occupancy grid using the 3D Bresenham's line algorithm for ray-tracing (Bresenham 1965). Both \mathcal{O} and \mathcal{U} are inflated by the radius of the UAV to ensure safety.

Global Planner

In the proposed framework, Jump Point Search (JPS) is used as a global planner to find the shortest piece-wise linear path from the current position to the goal. JPS was chosen instead of A* because it runs an order of magnitude faster, while still guaranteeing completeness and optimality (Harabor and Grastien 2011; Liu et al. 2017b). The only assumption of JPS is a uniform grid, which holds in our case.

Convex Decomposition

A convex decomposition is done around part of the piece-wise linear path obtained by JPS. To do this convex decomposition, we rely on the approach proposed by (Liu et al. 2017b): A polyhedron is found around each segment of the piece-wise linear path by first inflating an ellipsoid aligned with the segment, and then computing the tangents planes at the points of the ellipsoid that are in contact with the obstacles. The reader is referred to (Liu et al. 2017b) for a detailed explanation. Given a piece-wise linear path with P segments, we will denote the sequence of P overlapping polyhedra as $\{(A_p, c_p)\}$, $p = 0 : P - 1$.

Local Planner

For the local planner, we distinguish these three different jerk-controlled trajectories (see Fig. 5):

- **Whole Trajectory:** This trajectory goes from A to E , and it is contained in $\mathcal{F} \cup \mathcal{U}$. It has a final stop condition.
- **Safe Trajectory:** It goes from R to F , where R is a point in the Whole Trajectory, and F is any

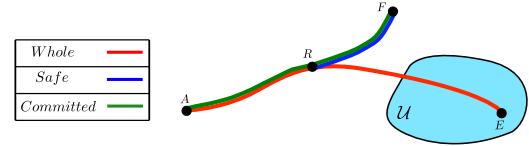


Figure 5. Trajectories used by FASTER: The Committed and Safe Trajectories are inside \mathcal{F} , while the Whole Trajectory is inside $\mathcal{F} \cup \mathcal{U}$

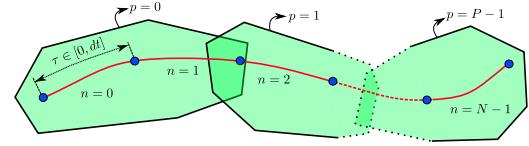


Figure 6. Each interval $n = 0 : N - 1$ of the trajectory is a third degree polynomial, with a total time of dt per interval. $\tau \in [0, dt]$ denotes a local reference of the time inside an interval, and $p = 0 : P - 1$ denotes the polyhedron.

point inside the polyhedra obtained by doing a convex decomposition of \mathcal{F} . It is completely contained in \mathcal{F} , and it has also a final stop condition to guarantee safety.

- **Committed Trajectory:** This trajectory consists of two pieces: The first part is the interval $A \rightarrow R$ of the Whole Trajectory. The second part is the Safe Trajectory. It will be shown later that this trajectory is also guaranteed to be inside \mathcal{F} . This trajectory is the one that the UAV will keep executing in case no feasible solutions are found in the next replanning steps.

The quadrotor is modeled using triple integrator dynamics with state vector $\mathbf{x}^T = [\mathbf{x}^T \dot{\mathbf{x}}^T \ddot{\mathbf{x}}^T] = [\mathbf{x}^T \mathbf{v}^T \mathbf{a}^T]$ and control input $\mathbf{u} = \ddot{\mathbf{x}} = \mathbf{j}$ (where \mathbf{x} , \mathbf{v} , \mathbf{a} , and \mathbf{j} are the vehicle's position, velocity, acceleration, and jerk, respectively).

In the optimization problem solved by the local planner, the trajectory is divided in N intervals (see Fig. 6). Let $n = 0 : N - 1$ denote the specific interval of the trajectory, $p = 0 : P - 1$ the specific polyhedron and dt the time allocated per interval (same for every interval n). If $\mathbf{j}(t)$ is constrained to be constant in each interval $n = 0 : N - 1$, then the whole trajectory will be a spline consisting of third degree polynomials. Matching the cubic form of the position for each interval

$$\mathbf{x}_n(\tau) = \mathbf{a}_n\tau^3 + \mathbf{b}_n\tau^2 + \mathbf{c}_n\tau + \mathbf{d}_n, \quad \tau \in [0, dt]$$

with the expression of a cubic Bézier curve

$$\mathbf{x}_n(\tau) = \sum_{j=0}^3 \binom{3}{j} \left(1 - \frac{\tau}{dt}\right)^{3-j} \left(\frac{\tau}{dt}\right)^j \mathbf{r}_{nj}, \quad \tau \in [0, dt],$$

we can solve for the four control points \mathbf{r}_{nj} ($j = 0 : 3$) associated with each interval n :

$$\begin{aligned} \mathbf{r}_{n0} &= \mathbf{d}_n, & \mathbf{r}_{n1} &= \frac{\mathbf{c}_ndt + 3\mathbf{d}_n}{3} \\ \mathbf{r}_{n2} &= \frac{\mathbf{b}_ndt^2 + 2\mathbf{c}_ndt + 3\mathbf{d}_n}{3} \\ \mathbf{r}_{n3} &= \mathbf{a}_ndt^3 + \mathbf{b}_ndt^2 + \mathbf{c}_ndt + \mathbf{d}_n \end{aligned}$$

Let us introduce the binary variables b_{np} , with $p = 0 : P - 1$ and $n = 0 : N - 1$ (P variables for each interval $n = 0 : N - 1$). As a Bézier curve is contained in the convex hull of its control points, we can ensure that the trajectory will be completely contained in this convex corridor by forcing that all the control points of an interval n are in the same polyhedron (Sahingoz 2014; Preiss et al. 2017) with the constraint $[b_{np} = 1 \implies r_{nj} \in \text{polyhedron } p \ \forall j]$, and at least in one polyhedron with the constraint $\sum_{p=0}^{P-1} b_{np} \geq 1$. With this formulation, the optimizer is free to choose the specific interval allocation (i.e. which interval is inside which polyhedron). The complete MIQP solved in each replanning step for both the Safe and the Whole trajectories is as follows:

$$\begin{aligned} \min_{j_n, b_{np}} \sum_{n=0}^{N-1} \|j_n\|^2 \\ \text{s.t. } \mathbf{x}_0(0) = \mathbf{x}_{init} \\ \mathbf{x}_{N-1}(dt) = \mathbf{x}_{final} \\ \mathbf{x}_n(\tau) = \mathbf{a}_n \tau^3 + \mathbf{b}_n \tau^2 + \mathbf{c}_n \tau + \mathbf{d}_n \quad \forall n, \forall \tau \in [0, dt] \\ \mathbf{v}_n(\tau) = \dot{\mathbf{x}}_n(\tau) \quad \forall n, \forall \tau \in [0, dt] \\ \mathbf{a}_n(\tau) = \ddot{\mathbf{x}}_n(\tau) \quad \forall n, \forall \tau \in [0, dt] \\ \mathbf{j}_n = 6\mathbf{a}_n \quad \forall n \\ b_{np} = 1 \implies \begin{cases} \mathbf{A}_p \mathbf{r}_{n0} \leq \mathbf{c}_p \\ \mathbf{A}_p \mathbf{r}_{n1} \leq \mathbf{c}_p \\ \mathbf{A}_p \mathbf{r}_{n2} \leq \mathbf{c}_p \\ \mathbf{A}_p \mathbf{r}_{n3} \leq \mathbf{c}_p \end{cases} \quad \forall n, \forall p \\ \sum_{p=0}^{P-1} b_{np} \geq 1 \quad \forall n \\ b_{np} \in \{0, 1\} \quad \forall n, \forall p \\ \mathbf{x}_{n+1}(0) = \mathbf{x}_n(dt) \quad n = 0 : N - 2 \\ \|\mathbf{v}_n(0)\|_\infty \leq v_{max} \quad \forall n \\ \|\mathbf{a}_n(0)\|_\infty \leq a_{max} \quad \forall n \\ \|\mathbf{j}_n\|_\infty \leq j_{max} \quad \forall n \end{aligned} \quad (1)$$

This problem is solved using Gurobi (Gurobi Optimization 2018). The decision variables of this optimization problem are the binary variables b_{np} and the jerk along the trajectory \mathbf{j}_n . \mathbf{x}_{init} and \mathbf{x}_{final} denote the initial and final states of the trajectory respectively. The time dt allocated per interval is computed as:

$$dt = f \cdot \max\{T_{v_x}, T_{v_y}, T_{v_z}, T_{a_x}, T_{a_y}, T_{a_z}, T_{j_x}, T_{j_y}, T_{j_z}\}/N \quad (2)$$

where T_{v_i} , T_{a_i} , T_{j_i} are solution of the constant-input motions in each axis $i = \{x, y, z\}$ by applying v_{max} , a_{max} and j_{max} respectively. $f \geq 1$ is a factor that is obtained according to the solution of the previous replanning step (see Fig. 7): Denoting $f_{worked,k-1}$ as the factor that made the optimization feasible in the replanning step $k - 1$, in the replanning step k the optimizer will try values of f (in increasing order) in the interval $[f_{worked,k-1} - \gamma, f_{worked,k-1} + \gamma']$ until the problem converges. Here γ and γ' are constant values chosen by the user. Note that, if $f = 1$, then dt is a lower bound on the minimum time per interval required for the problem to be feasible. Therefore, only factors $f \geq 1$ are tried. This approach tries to keep dt as small as possible (and therefore faster trajectories are

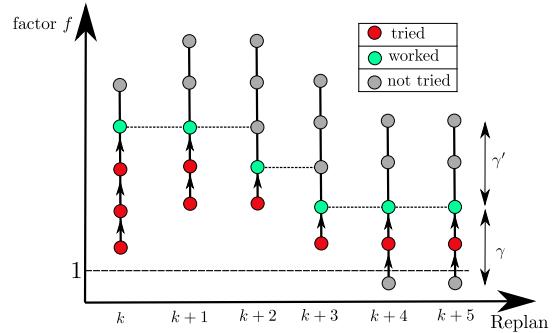


Figure 7. Dynamic adaptation of the factor used to compute the heuristic of the time allocated per interval (dt): For iteration k , the range of factors used is taken around the factor that worked in the iteration $k - 1$. As $f = 1$ is the lower bound that makes the problem feasible, only factors $f \geq 1$ are tried.

Algorithm 1: FASTER

```

1 Function Replan():
2    $k \leftarrow k + 1, \delta t \leftarrow \alpha \Delta t_{k-1}$ 
3   Choose point  $A$  in Committed $_{k-1}$  with offset  $\delta t$  from  $L$ 
4    $G \leftarrow$  Projection of  $G_{term}$  into map  $\mathcal{M}$ 
5    $JPS_a \leftarrow$  Run JPS  $A \rightarrow G$ 
6    $S \leftarrow$  Sphere of radius  $r$  centered on  $A$ 
7    $C \leftarrow JPS_a \cap S, D \leftarrow JPS_{k-1} \cap S$ 
8   if  $\angle CAD > \alpha_0$  then
9      $JPS_b \leftarrow$  Modified  $JPS_{k-1}$  such that  $JPS_{k-1} \cap \mathcal{O} = \emptyset$ 
10     $D \leftarrow JPS_b \cap S$ 
11     $dt_a \leftarrow$  Lower bound on  $dt A \rightarrow C$ 
12     $dt_b \leftarrow$  Lower bound on  $dt A \rightarrow D$ 
13     $J_a = N \cdot dt_a + \frac{\|JPS_a(C \rightarrow G)\|}{v_{max}}$ 
14     $J_b = N \cdot dt_b + \frac{\|JPS_b(D \rightarrow G)\|}{v_{max}}$ 
15     $JPS_k \leftarrow \underset{\{JPS_a, JPS_b\}}{\operatorname{argmin}} \{J_a, J_b\}$ 
16  else
17     $JPS_k \leftarrow JPS_a$ 
18   $JPS_{in} \leftarrow$  Part of  $JPS_k$  inside  $S$ 
19   $Poly_{whole} \leftarrow$  Convex Decomposition in  $\mathcal{U} \cup \mathcal{F}$  using  $JPS_{in}$ 
20   $f_{whole} \leftarrow [f_{whole,k-1} - \gamma, f_{whole,k-1} + \gamma']$ 
21   $Whole \leftarrow$  MIQP in  $Poly_{whole}$  from  $A$  to  $E$  using  $f_{whole}$ 
22   $H \leftarrow Whole \cap \mathcal{U}$ 
23   $R \leftarrow$  Nearest state to  $H$  along  $Whole$  that is not in inevitable collision with  $\mathcal{U}$ 
24   $JPS_{in,known} \leftarrow$  Part of  $JPS_{in}$  in  $\mathcal{F}$ 
25   $Poly_{safe} \leftarrow$  Convex Decomposition in  $\mathcal{F}$  using  $JPS_{in,known}$ 
26   $f_{safe} \leftarrow [f_{safe,k-1} - \gamma, f_{safe,k-1} + \gamma']$ 
27   $Safe \leftarrow$  MIQP in  $Poly_{safe}$  from  $R$  to  $F$  using  $f_{safe}$ 
28   $Committed_k \leftarrow Whole \rightarrow R \cup Safe$ 
29   $f_{whole,k} \leftarrow$  Factor that worked for  $Whole$ 
30   $f_{safe,k} \leftarrow$  Factor that worked for  $Safe$ 
31   $\Delta t_k \leftarrow$  Total replanning time

```

obtained), but at the same time it also tries to minimize the number of trials with different dt needed until convergence.

Complete Algorithm

Algorithm 1 gives the full approach (see also Figs. 8 and 9). Let L be the current position of the UAV. The point A is chosen in the Committed Trajectory of the previous replanning step with an offset δt from L . This offset δt is computed by multiplying the total time of the previous replanning step by $\alpha \geq 1$ (typically $\alpha \approx 1.25$). The idea here is to dynamically change this offset to ensure that most of the time the solver can find the next solution in less than δt .

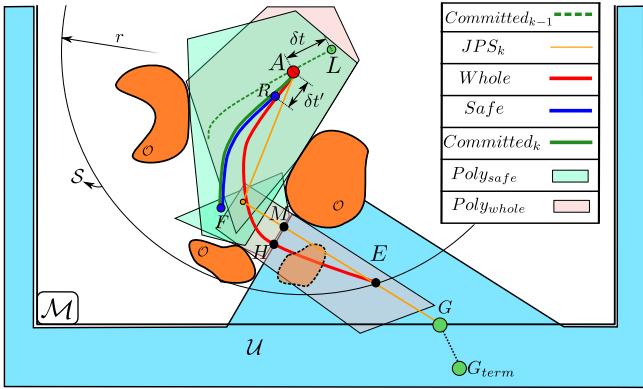


Figure 8. Illustration for Alg.1. One unknown obstacle is shown with dotted line.

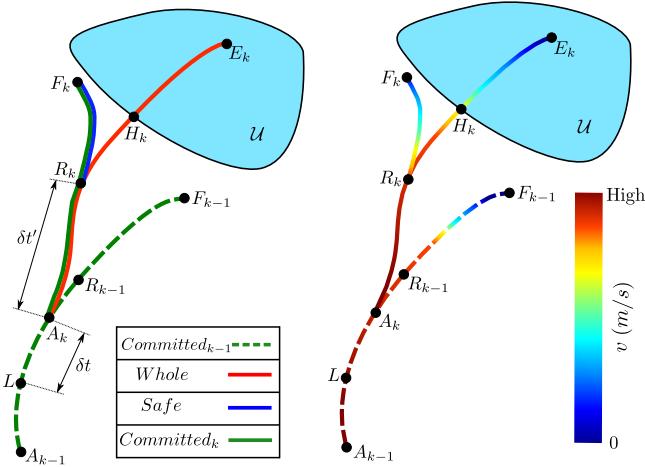


Figure 9. Illustration of all the trajectories involved in Alg. 1 and their associated velocity profiles. \mathcal{U} is the unknown space (■), and k is the replanning step.

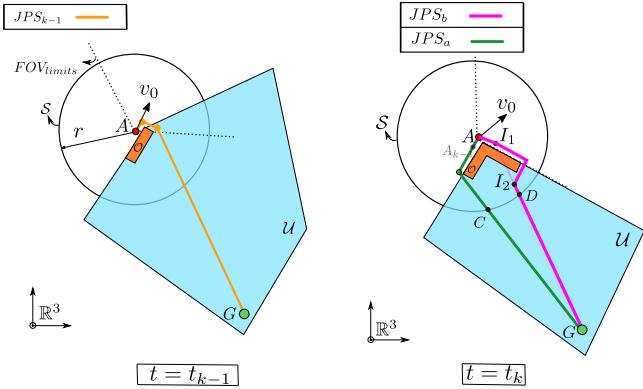


Figure 10. Choice of the direction to optimize. At $t = t_{k-1}$, the JPS solution chosen was JPS_{k-1} . At $t = t_k$, JPS is run again to obtain JPS_a , and JPS_{k-1} is modified so that it does not collide with \mathcal{O} , obtaining JPS_b . A heuristic of the cost-to-go in each direction is computed, and the direction with the lowest cost is chosen as the one towards which the local planner will optimize.

Then, the final goal G_{term} is projected into the sliding map \mathcal{M} (centered on the UAV) in the direction $\overrightarrow{G_{term}A}$ to obtain the point G (line 4). Next, we run JPS from A to G (line 5) to obtain JPS_a .

The local planner then has to decide which direction is the best one to optimize towards (lines 7-17). Instead of blindly trusting the last JPS solution (JPS_a) as the best direction for the local planner to optimize (note that JPS is a zero-order model, without dynamics encoded), we take into account the dynamics of the UAV in the following way: First of all, we modify the JPS_{k-1} so that it does not collide with the new obstacles seen (Fig. 10): we find the points I_1 and I_2 (first and last intersections of JPS_{k-1} with \mathcal{O}) and run JPS three times, so $A \rightarrow I_1$, $I_1 \rightarrow I_2$ and $I_2 \rightarrow G$. Hence, the modified version, denoted by JPS_b , will be the concatenation of these three paths.

Then, we compute a lower bound on dt using Eq. 2 for both $A \rightarrow C$ and $A \rightarrow D$, where C and D are the intersections of the previous JPS paths with a sphere S of radius r centered on A , where r is specified by the user. Next, we find the cost-to-go associated with each direction by adding this dt_a (or dt_b) and the time it would take the UAV to go from C (or D) to G following the JPS solution and flying at v_{max} . Finally, the one with lowest cost is chosen, and therefore $JPS_k \leftarrow \operatorname{argmin}_{\{JPS_a, JPS_b\}} \{J_a, J_b\}$. This will be

the direction towards which the local planner will optimize.

To save computation time, this decision between JPS_a and JPS_b is made only if the angle $\angle CAD$ exceeds a certain threshold α_0 (typically $\approx 15^\circ$). Note that $\angle CAD$ gives a measure of how much the JPS solution has changed with respect to the iteration $k - 1$. A small angle indicates that JPS_a and JPS_{k-1} are very similar (at least within the sphere S), and that therefore the direction of the local plan will not differ much from the iteration $k - 1$.

The **Whole Trajectory** (lines 18-21) is obtained as follows. We do the convex decomposition (Liu et al. 2017b) of $\mathcal{U} \cup \mathcal{F}$ around the part of JPS_k that is inside the sphere S , which we denote as JPS_{in} . This gives a series of overlapping polyhedra that we denote as $Poly_{whole}$. Then, the MIQP in (1) is solved using these polyhedral constraints to obtain the Whole Trajectory.

The **Safe Trajectory** is computed as in lines 22-27. First we compute the point H as the intersection between the Whole Trajectory and \mathcal{U} . Then, we have to choose the point R along the Whole Trajectory as the start of the Safe Trajectory. To do this, note that, on one hand, R should be chosen as far as possible from A , so that δt can be chosen bigger in the next replanning step, which helps to guarantee that A is not chosen on the Safe trajectory (where the braking maneuver happens). On the other hand, however, a point R too close to H may lead to an infeasible problem for the safe trajectory optimizer. We propose two ways to compute R : The first one is to choose it with an offset $\delta t'$ from A , where $\delta t'$ is computed by multiplying the previous replanning time by $\beta \geq 1$. The second (and better) way to solve this trade-off is the following one: we can choose R as the nearest state to H (in the segment $A \rightarrow H$ of Whole) that is not in inevitable collision with \mathcal{U} . To compute an approximation of this state in a very efficient way, we choose R as the last point (going from A to H along the Whole Trajectory) that satisfies

$$\operatorname{sign} [\mathbf{v}_{R,j} (\mathbf{x}_{H,j} - \mathbf{x}_{R,j})] \cdot \frac{\mathbf{v}_{R,j}^2}{2 |a_{max}|} < |\mathbf{x}_{H,j} - \mathbf{x}_{R,j}|$$

where $\mathbf{v}_{R,j}$, $\mathbf{x}_{R,j}$ and $\mathbf{x}_{H,j}$ are respectively the velocity of R , the position of R and the position of H in the

Algorithm 2: FIND INTERSECTION

```

1 Function FindIntersection():
2 while  $JPS_k \neq \emptyset$  do
3    $V \leftarrow$  First element of  $JPS_k$ 
4    $N \leftarrow$  Find nearest neighbour of  $V$  in  $\mathcal{U}$ 
5    $r \leftarrow \|N - V\|$ 
6   if  $r < \epsilon$  then
7     return  $V$ 
8    $S \leftarrow$  Sphere of radius  $r$  centered on  $V$ 
9    $M \leftarrow JPS_k \cap S$ 
10   Remove from  $JPS_k$  the vertexes inside  $S$ 
11   Insert  $M$  at the front of  $JPS_k$ 
12   return No Intersection

```

axes $j = \{x, y\}$. Here we have approximated the system as a double integrator model in each axis, and hence $\frac{v_{R,j}^2}{2|a_{max}|}$ is the minimum stopping distance. Due to these two approximations (double integrator and decoupling in axes x and y), this heuristic may be conservative. We ignore the axis z in this computation to reduce the conservativeness of this heuristic.

Note that even in the case that this heuristic leads to a choice of R for which no feasible collision-free (with $\mathcal{U} \cup \mathcal{O}$) trajectory exists, the optimizer will not find a solution in that replanning step, and therefore will continue executing the solution of the previous replanning step.

After choosing the point R , we do the convex decomposition of \mathcal{F} using the part of JPS_{in} that is in \mathcal{F} , obtaining the polyhedra $Poly_{safe}$. Then, we solve the MIQP from R to any point F inside $Poly_{safe}$ (this point F is chosen by the optimizer).

In both of the convex decompositions presented above, one polyhedron is created for each segment of the piecewise linear paths. To obtain a less conservative solution (i.e. bigger polyhedra), we first check the length of segments of the JPS path, creating more vertexes if this length exceeds certain threshold l_{max} . Moreover, we truncate the number of segments in the path to ensure that the number of polyhedra found does not exceed a threshold P_{max} . This helps reduce the computation times (see Sec. 12).

Finally (line 28), we compute the **Committed Trajectory** by concatenating the piece $A \rightarrow R$ of the Whole Trajectory, and the Safe Trajectory. Note that in this algorithm we have run two *decoupled* optimization problems per replanning step: (1) one for the Whole Trajectory, and (2) one for the Safe Trajectory. This ensures that the piece $A \rightarrow R$ is not influenced by the braking maneuver $R \rightarrow F$, and therefore it guarantees a higher nominal speed on this first piece. The intervals $L \rightarrow A$ and $A \rightarrow R$ have been designed so that at least one replanning step can be solved within that interval.

The UAV will continue executing the trajectory of the previous replanning step ($Committed_{k-1}$) if one of these three scenarios happens:

- **Scenario 1:** Either of the two optimizations is infeasible.
- **Scenario 2:** The piece $A - R$ intersects \mathcal{U} .
- **Scenario 3:** The replanning takes longer than δt .

In Alg. 1, it is required to compute the intersection between a piece-wise linear path (the solution of JPS) and

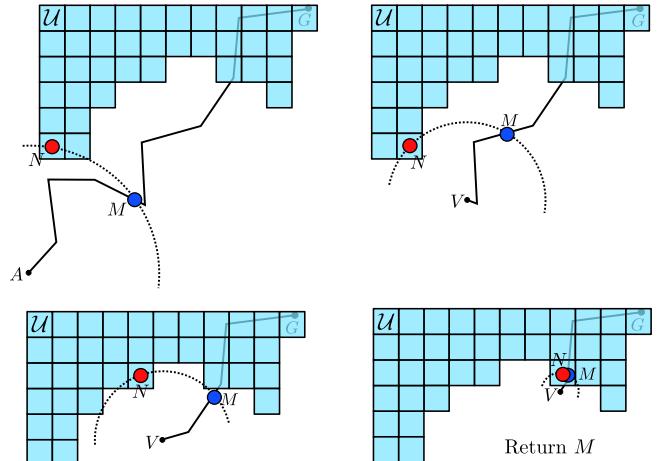


Figure 11. Illustration of Alg. 2 to efficiently find (an approximation of) the intersection between a piece-wise linear path and a voxel grid. \mathcal{U} and JPS_k are used in this figure, but in FASTER this algorithm is also used with \mathcal{O} and JPS_{k-1}

a voxel grid (\mathcal{U} or \mathcal{O}) to obtain the points I_1, I_2 or M . To do this in an efficient way, we use Alg. 2, depicted in Fig. 11. We first find the nearest neighbor N from the beginning of the piece-wise linear path A (line 4), and compute the intersection M between the path and a sphere S centered on A with radius equal to the distance between A and N (line 9). As it is guaranteed that all the points of the path that are inside S do not intersect with the voxel grid, we can repeat the same procedure again, but this time starting from M . This process continues until the distance to the nearest neighbour is below some threshold $\epsilon > 0$ (lines 6–7).

Feasibility Theorem

We can now state the following feasibility theorem for FASTER, which guarantees that all the committed trajectories are completely contained inside free space (known or unknown), and that therefore safety is guaranteed. Here k denotes the replanning step.

Assumption 1. *The map \mathcal{M} is noise-free and the world is static: $\mathcal{F}_k \cup \mathcal{F}_{Unknown,k} = \mathcal{F}_{k+1} \cup \mathcal{F}_{Unknown,k+1}, \forall k$.*

Theorem 1. *Under the assumption 1, Alg. 1 achieves*

$$Committed_k \subseteq \mathcal{F}_k \cup \mathcal{F}_{Unknown,k} \quad \forall k$$

Proof. This theorem can be proven by induction:

1. **Base case:** $Committed_1$ is the union of $A_1 \rightarrow R_1$ and the Safe Trajectory. The interval $A_1 \rightarrow R_1$ is in \mathcal{F}_1 because it has been checked against collision with \mathcal{U}_1 and is contained in a convex corridor that does not intersect \mathcal{O}_1 . The Safe Trajectory is inside \mathcal{F}_1 by construction. Therefore, $Committed_1 \subseteq \mathcal{F}_1 \cup \mathcal{F}_{Unknown,1}$.
2. **Recursion:** If $Committed_k \subseteq \mathcal{F}_k \cup \mathcal{F}_{Unknown,k}$, two different situations can happen in iteration $k + 1$:
 - (a) One of the scenarios 1, 2, or 3 happens. The algorithm will choose $Committed_{k+1} = Committed_k$, and by the assumption 1 we have that $Committed_{k+1} \subseteq \mathcal{F}_{k+1} \cup \mathcal{F}_{Unknown,k+1}$

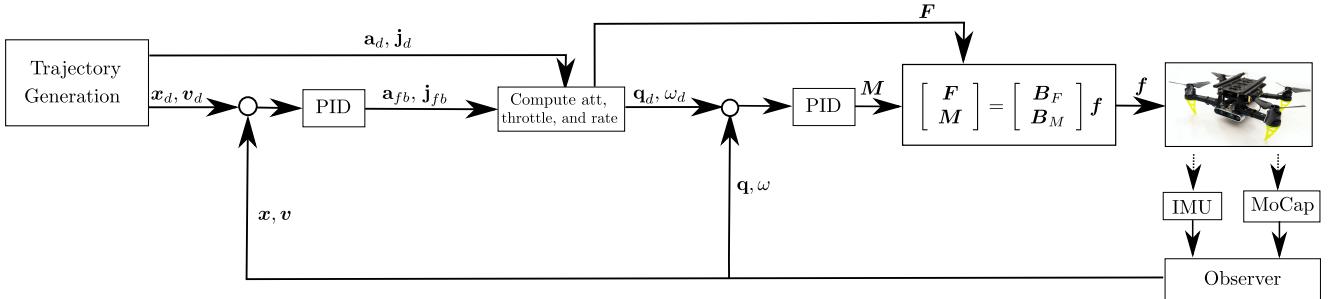


Figure 12. Cascade controller. The inner loop tracks desired attitude and body rates, which are generated by the outer loop. The outer loop tracks position and velocity, using acceleration and jerk as feed-forward commands. The desired position, velocity, acceleration and jerk are generated by FASTER.

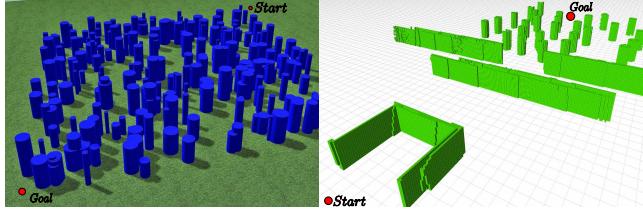


Figure 13. Forest (left) and bugtrap (right) environments used in the simulation. The forest is 50×50 m, and the grid in the bugtrap environment is $1 \text{ m} \times 1 \text{ m}$.

- (b) In any other case, the trajectory obtained (Committed_{k+1}) will be inside \mathcal{F}_{k+1} by construction of the algorithm.

Hence, we conclude that

$$\begin{aligned} \text{Committed}_k &\subseteq \mathcal{F}_k \cup \mathcal{F}_{\text{Unknown},k} \\ \implies \text{Committed}_{k+1} &\subseteq \mathcal{F}_{k+1} \cup \mathcal{F}_{\text{Unknown},k+1} \end{aligned}$$

Remark 1. *The theorem does not assume that $\mathcal{F}_k \subseteq \mathcal{F}_{k+1}$. In other words, it does not assume that the size of the free-known space always increases: $\mathcal{F}_k \subseteq \mathcal{F}_{k+1}$ is not necessarily true due to the sliding map. Note, however, that the proof does not depend on the shape of the map nor on the length of the history kept in this map. Hence, the theorem is also valid for the following two cases:*

- a non-sliding global map $\mathcal{M} \equiv \mathbb{R}^3$.
- a map $\mathcal{M} \equiv \text{FOV}$ (Field of View of the sensor), obtained uniquely by considering the instantaneous sensing data and therefore not keeping history in the map.

Remark 2. *By allowing the algorithm to choose $\text{Committed}_{k+1} = \text{Committed}_k$ (which occurs when one of the scenarios 1, 2 or 3 happen), in iteration $k+1$ the UAV may commit to a trajectory that has some parts outside the map \mathcal{M}_{k+1} . As proven above, it is still guaranteed that $\text{Committed}_{k+1} \subseteq \mathcal{F}_{k+1} \cup \mathcal{F}_{\text{Unknown},k+1}$. This constitutes a form of data compression, where the information of a part of the world being free (which was obtained in iteration k or before) is embedded in the trajectory itself and not directly in the map \mathcal{M}_{k+1} .*

Controller

The cascade controller used to track the trajectory obtained by FASTER is shown in Fig. 12. In the outer PID loop, feedback acceleration is computed to track position and velocity. Combining this feedback acceleration with the desired acceleration as a feed-forward term, the desired attitude q_d is computed. The total commanded acceleration is also used to calculate the desired thrust. Then, given the desired jerk from the trajectory generator and the feedback jerk numerically differentiated from the feedback acceleration, the desired body rates ω_d are computed. The moments M are then calculated by the inner loop using a quaternion-based PD controller expressed as

$$M = \text{sign}(q_{e,w}) K_p \vec{q}_e + K_d (\omega_d - \omega), \quad (3)$$

where $q_e = q^* \otimes q_{\text{des}} = (q_{e,w}, \vec{q}_e)$. Using the motor allocation matrix, the actuator commands f are recovered from the desired thrust and moments. The yaw of the UAV is chosen such that the camera of the UAV points to M (intersection between JPS_k and \mathcal{U} , see Fig. 8). This controller is used in all the UAV simulation and hardware experiments of this paper. In the real hardware experiments, position, velocity, attitude, and IMU biases are estimated by fusing propagated IMU measurements with an external motion capture system.

Results

Simulation

We evaluate the performance of the proposed algorithm in different simulated scenarios. The simulator uses C++ custom code for the dynamics engine, integrating the nonlinear differential equations of the UAV using the Runge-Kutta method. Gazebo (Koenig and Howard 2004) is used to simulate perception data in the form of a depth map. In all these simulations, the depth camera has a horizontal FOV of 90° . The sensing range is 5 m for the first simulation (corner environment), and 10 m for the rest.

We first test FASTER in a simple environment and, for the same replanning step, we compare the velocities of the trajectory found by FASTER (that plans in $\mathcal{U} \cup \mathcal{F}$) with the ones of the trajectory found by a planner that plans only in \mathcal{F} . The environment is shown in Fig. 14, and consists on a corner, with the goal at the other side of the wall, so that the UAV has to turn the corner. The initial velocity at A is 4.8 m/s, and the dynamic constraints imposed are $v_{\max} = 6.5$ m/s, $a_{\max} = 6$ m/s², and $j_{\max} = 20$ m/s³. FASTER achieves a velocity of 6.02 m/s in the segment $A \rightarrow R$

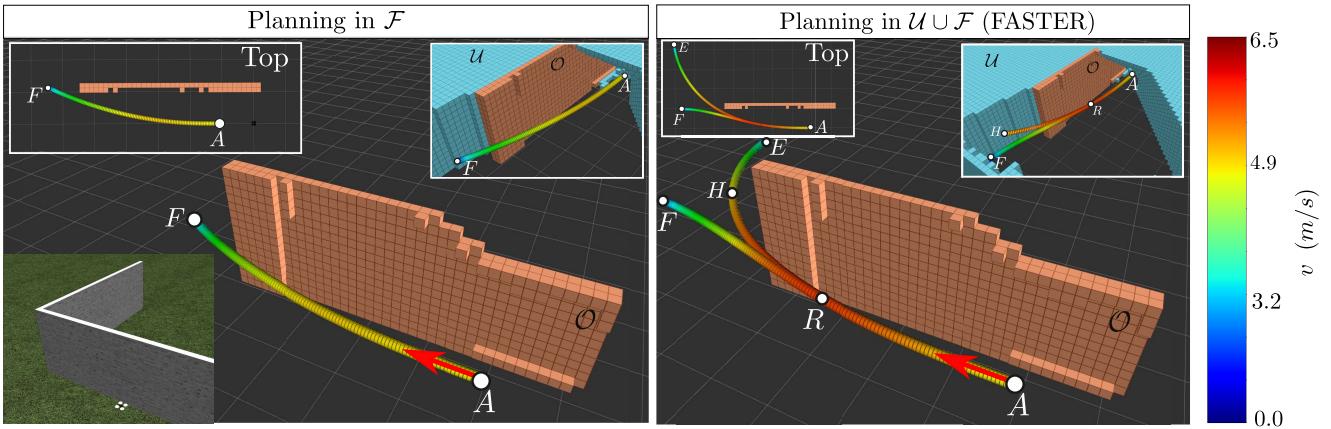


Figure 14. Trajectories obtained when planning only in \mathcal{F} (left) and when planning in $\mathcal{F} \cup \mathcal{U}$ (FASTER, right). The velocity at A is 4.8 m/s. FASTER achieves a velocity of 6.02 m/s in the segment $A \rightarrow R$ (segment that will be actually flown by the UAV), while the other planner achieves a velocity of 5.06 m/s. The ground grid is 1 m \times 1 m.

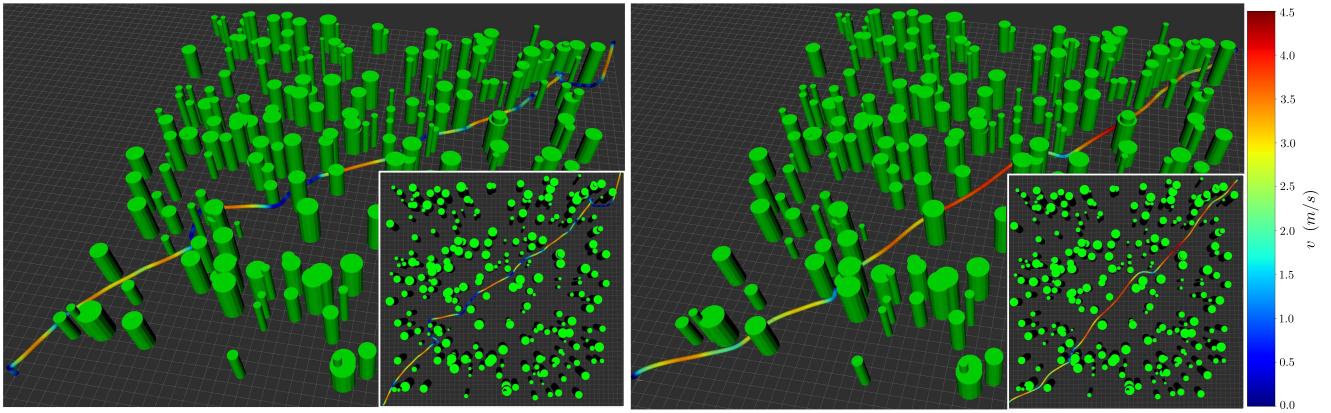


Figure 15. Velocity profile in a random forest simulation. On the left the results of our previous work (Tordesillas et al. 2019a) and on the right FASTER.

Table 2. Distances obtained in 10 random forest simulations. The distance values are computed for the cases that reach the goal. All the results (except the ones of (Tordesillas et al. 2019a) and FASTER) were provided by the authors of (Oleynikova et al. 2018).

Method	Number of Successes	Distance (m)			
		Avg	Std	Max	Min
Incremental	0	-	-	-	-
Rand. Goals	10	138.0	32.0	210.5	105.6
Opt. RRT*	9	105.3	10.3	126.4	95.5
Cons. RRT*	9	155.8	52.6	267.9	106.2
NBVP	6	159.3	45.6	246.9	123.6
SL Expl.	8	103.8	21.6	148.3	86.6
Multi-Fid.	10	84.5	11.7	109.4	73.2
FASTER	10	77.6	5.9	88.0	70.7
Min/Max improv. (%)	8/51	43/89	20/67	3/43	

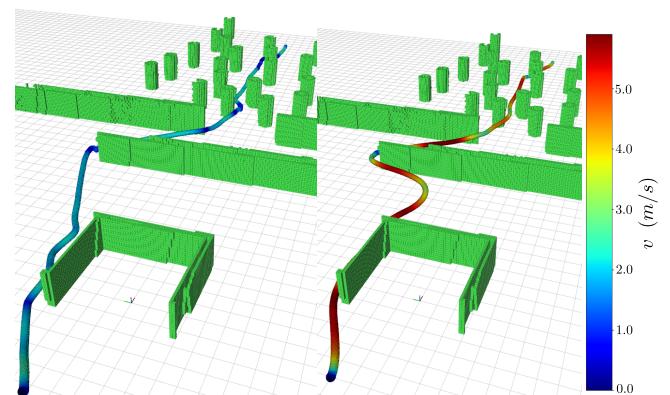


Figure 16. Velocity profile in the bugtrap simulation. On the left the results of our previous work (Tordesillas et al. 2019a) and on the right FASTER.

Table 3. Comparison between (Tordesillas et al. 2019a) and FASTER of flight times in the forest simulation. Results are for 10 random forests.

Method	Time (s)			
	Avg	Std	Max	Min
Multi-Fid.	61.2	16.8	92.5	37.9
FASTER	29.2	4.2	36.8	21.6
Improvement (%)	52.3	75.0	60.2	43.0

Table 1. Comparison between (Tordesillas et al. 2019a) and FASTER of flight distances and times in a bugtrap simulation.

Method	Distance (m)	Time (s)
Multi-Fid.	56.8	37.6
FASTER	55.2	13.8
Improvement (%)	2.8	63.3

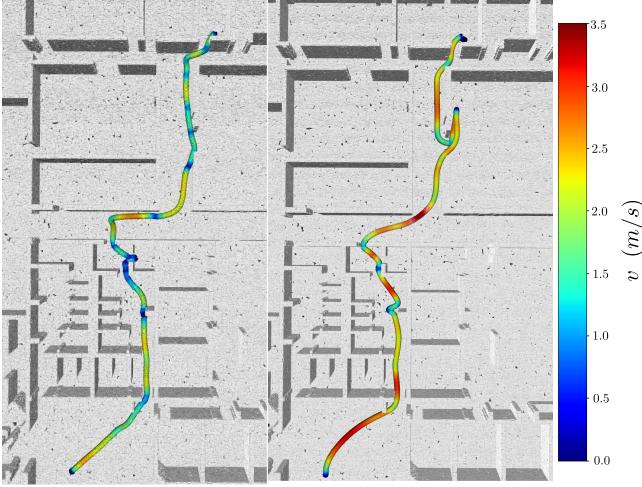


Figure 17. Velocity profile in the office simulation. On the left the results of (Tordesillas et al. 2019a) and on the right FASTER.

Table 4. Comparison between (Tordesillas et al. 2019a) and FASTER of flight distances and times in an office simulation.

Method	Distance (m)	Time (s)
Multi-Fid.	41.5	29.73
FASTER	43.9	20.94
Improvement (%)	-5.8	29.6

(segment that will be actually flown by the UAV), while planning only in \mathcal{F} achieves a velocity of 5.06 m/s. $R \rightarrow F$ is the Safe Trajectory, and $A \rightarrow R \rightarrow F$ is the Committed trajectory. Safety is guaranteed by both planners.

We now test FASTER in 10 random forest environments with an obstacle density of 0.1 obstacles/m² (see Fig. 13 and Extension 1), and compare the flight distances achieved against the following seven approaches:

- Incremental approach (no goal selection).
- Random goal selection.
- Optimistic RRT* (unknown space = free).
- Conservative RRT* (unknown space=occupied).
- “Next-best-view” planner (NBVP) (Bircher et al. 2016).
- Safe Local Exploration (Oleynikova et al. 2018).
- Multi-Fidelity (Tordesillas et al. 2019a).

The first six methods are described deeper in (Oleynikova et al. 2018), while (Tordesillas et al. 2019a) is our previous proposed algorithm. The results are shown in Table 2, which highlights that FASTER achieves a 8 – 51% improvement in the total distance flown. Completion times are compared in Table 3 to (Tordesillas et al. 2019a) (time values are not available for all other algorithms in Table 2). FASTER achieves an improvement of 52% in the completion time. The dynamic constraints imposed for the results of this table are (per axis) $v_{max} = 5$ m/s, $a_{max} = 5$ m/s², and $j_{max} = 8$ m/s³. The velocity profiles obtained for one random forest simulation are shown in Fig. 15.

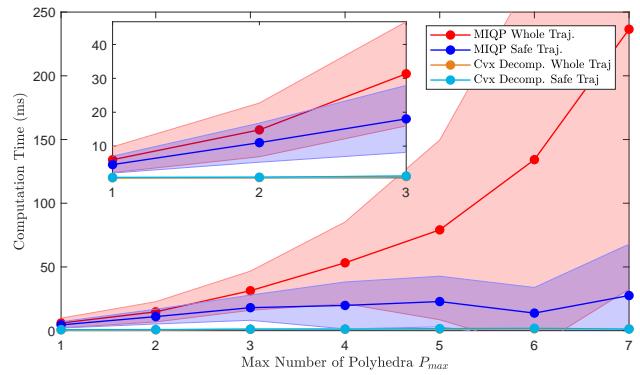


Figure 18. Timing breakdown for the MIQP and Convex Decomposition of the Whole Trajectory and the Safe Trajectory as a function of the maximum number of polyhedra P_{max} . Note that the times for the MIQPs include all the trials until convergence (with different factors f) in each replanning step. The shaded area is the 1- σ interval, where σ is the standard deviation. These results are from the forest simulation.

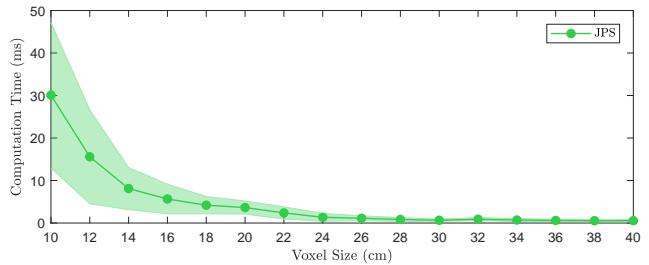


Figure 19. Runtimes of JPS as a function of the Voxel Size. The shaded area is the 1- σ interval, where σ is the standard deviation. These results are from the forest simulation, and for a sliding map of size 20 m × 20 m.

We also test FASTER using the bugtrap environment shown in Fig. 13, and obtain the results that appear on Table 1. Both algorithms have a similar total distance, but FASTER achieves an improvement of 63% on the total flight time. For both cases the dynamic constraints imposed are $v_{max} = 10$ m/s, $a_{max} = 10$ m/s², and $j_{max} = 40$ m/s³. The velocity profile achieved along the trajectory can be seen in Fig. 16.

Finally, we test FASTER in an office environment, obtaining the velocity profile shown in Fig 17 and the distances and flight times shown in Table 4 (see also Extension 1). In this case, the distance flown by FASTER was slightly longer than the one by (Tordesillas et al. 2019a) (note that FASTER entered one of the last rooms, and then turned back), but even with this extra distance, it achieved a 29.6% improvement on the flight time. The dynamic constraints used for the office simulation are $v_{max} = 3$ m/s, $a_{max} = 6$ m/s² and $j_{max} = 35$ m/s³.

The timing breakdown of Alg. 1 as a function of the maximum number of polyhedra P_{max} is shown in Fig. 18. The number of intervals N was 10 for the Whole Trajectory and 7 for the Safe Trajectory. Note that the runtime for the MIQP of the Safe Trajectory is approximately constant as a function of P_{max} . This is due to the fact that the Safe Trajectory is planned only in \mathcal{F} , and therefore most of the times $P < P_{max}$. For the simulations and hardware experiments presented in this paper, $P_{max} = 2 – 4$ was used. The runtimes for JPS as a function of the voxel size

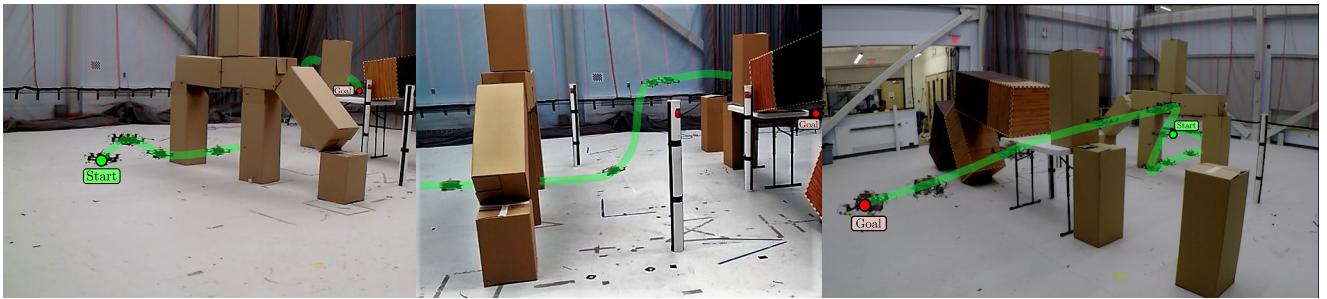


Figure 20. Composite images of Experiment 1. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.

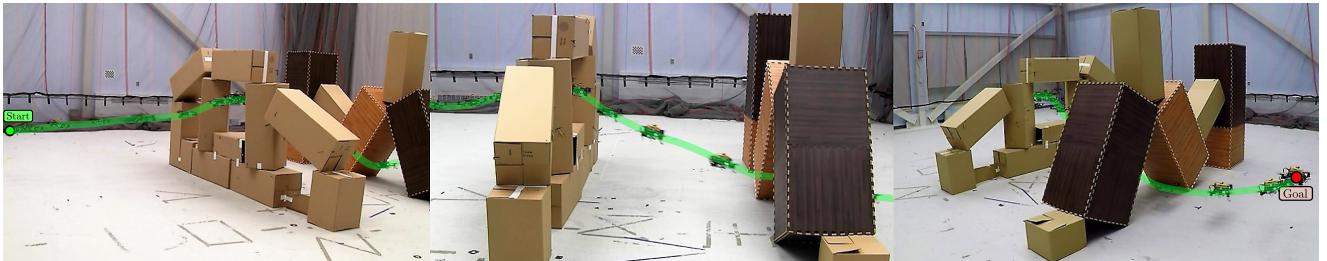


Figure 21. Composite image of Experiment 2. The UAV must fly from start ● to goal ●. Snapshots shown every 330 ms.

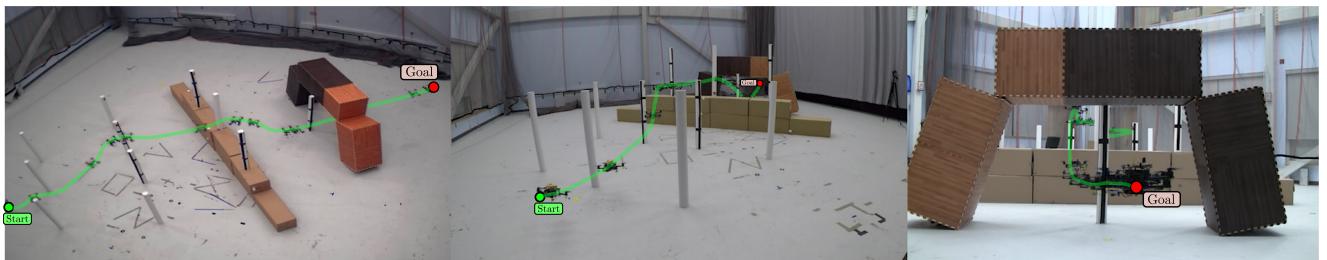


Figure 22. Composite image of Experiment 3. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.

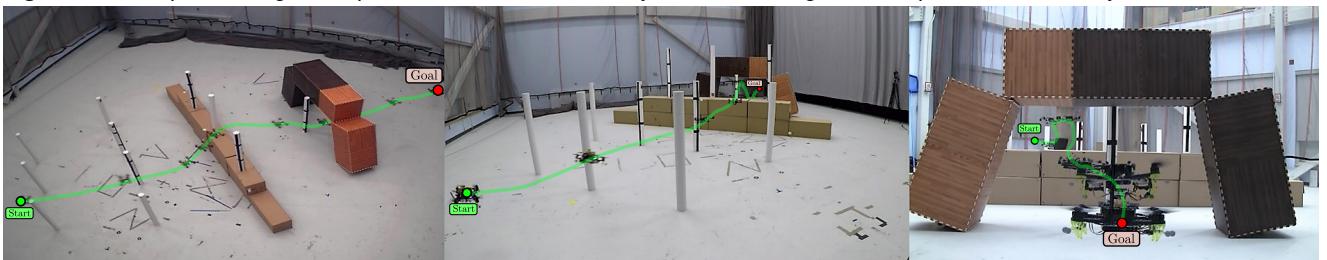


Figure 23. Composite image of Experiment 4. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.

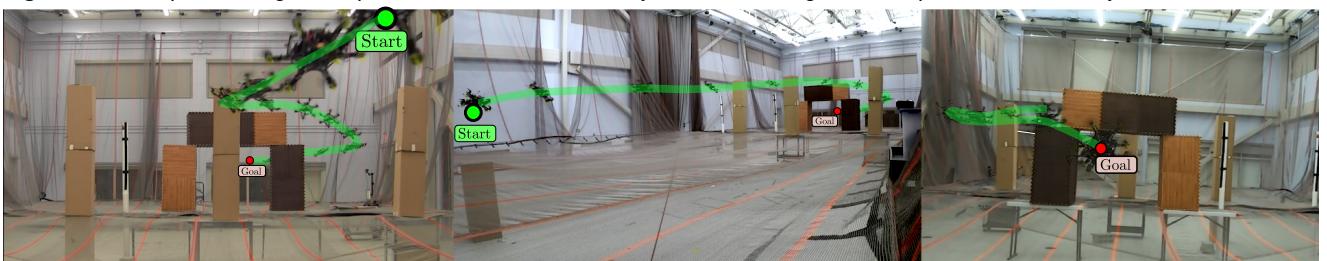


Figure 24. Composite image of Experiment 5. The UAV must fly from start ● to goal ●. Snapshots shown every 330 ms.

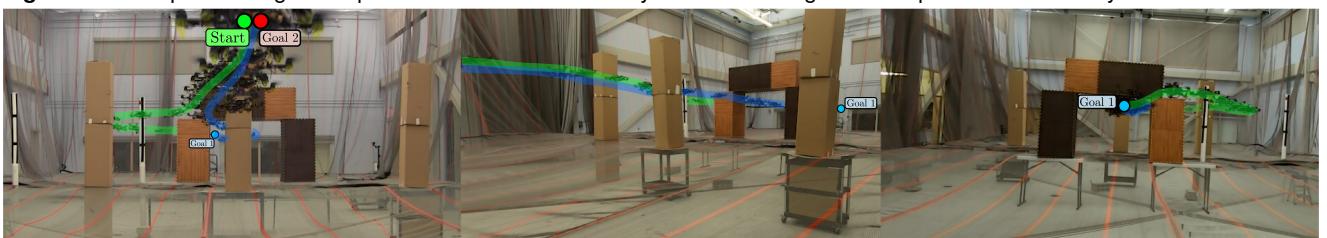


Figure 25. Composite image of Experiment 6. The UAV must fly from start ● to goal 1 ● and then back to goal 2 ●. Snapshots shown every 330 ms.

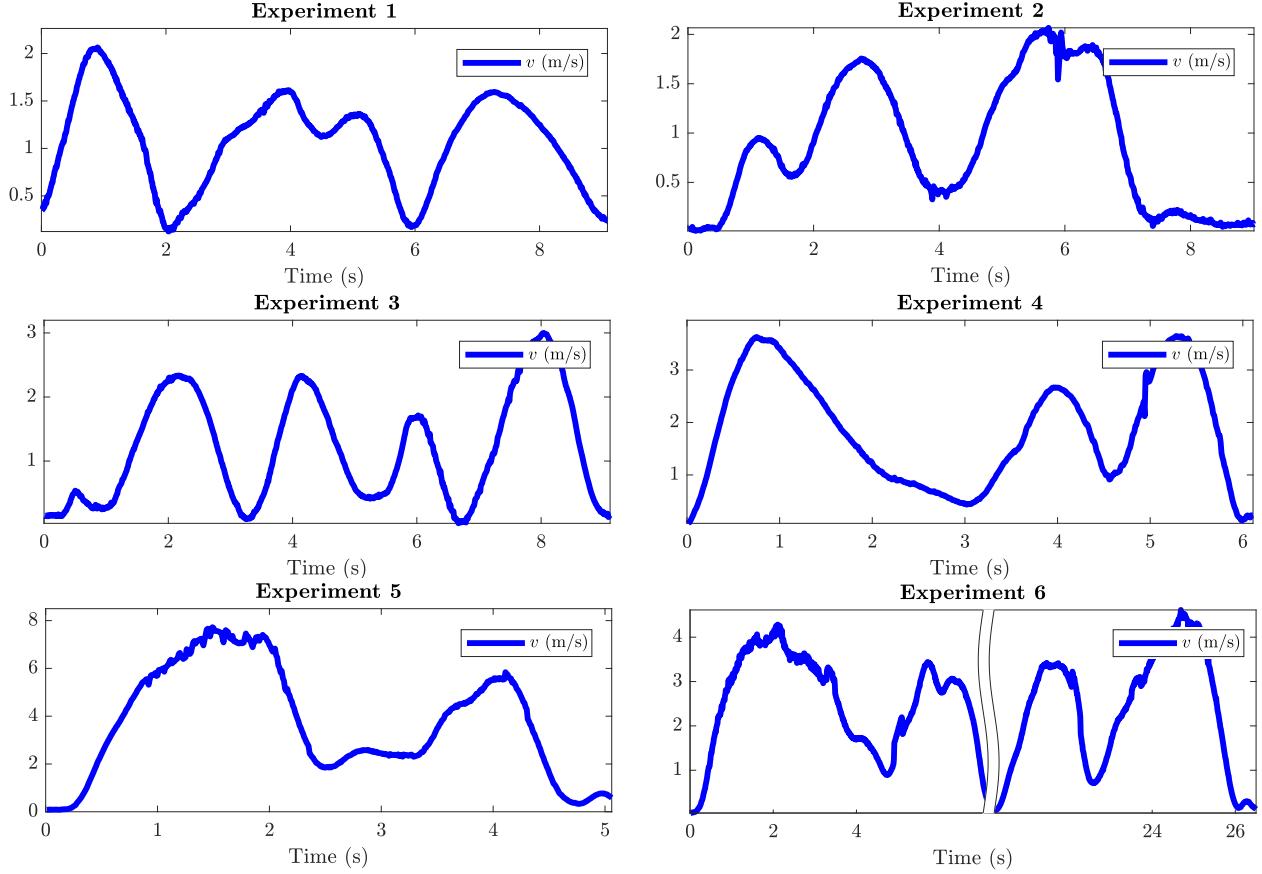


Figure 26. Velocity plots of all the UAV hardware experiments.

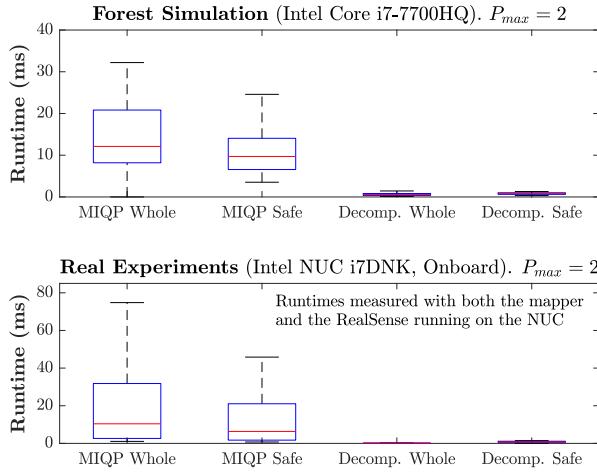


Figure 27. Timing breakdown for the forest simulation and for the real hardware experiments. The parameters used are $P_{max} = 2$, $N = 10$ for the Whole Trajectory, and $N = 7$ for the Safe Trajectory.

of the map for the forest simulation are available in Fig. 19. These times are always < 10 ms for voxel sizes ≥ 14 cm. All these timing breakdowns were measured using an Intel Core i7-7700HQ 2.8GHz Processor.

Hardware

The UAVs used in the hardware experiments are shown in Fig. 28. A quadrotor was used in the experiments 1-4, and



Figure 28. Quadrotor (top) used in the experiments 1-4 and hexarotor (bottom) used in the experiments 5-6. Both are equipped with a Qualcomm® SnapDragon Flight, an Intel® NUC i7DNK and an Intel® RealSense Depth Camera D435.

a hexarotor was used in the experiments 5-6. In both UAVs, the perception runs on the Intel® RealSense, the mapper and planner run on the Intel® NUC, and the control runs on the Qualcomm® SnapDragon Flight.

The six hardware experiments done are shown in Figures 20, 21, 22, 23, 24, and 25 (see also Extension 2). The corresponding velocity profiles are shown in Fig. 26. The maximum speed achieved was 7.8 m/s, in Experiment 5 (Fig. 24).

The first and second experiments (Fig. 20 and 21) were done in similar obstacle environments with the same starting point, but with different goal locations. In the first experiment (Fig. 20), the UAV performs a 3D agile maneuver to avoid the obstacles on the table. In the second experiment (Fig. 21) the UAV flies through the narrow gap of the cardboard boxes structure, and then flies below the triangle-shaped obstacle. In these two experiments, the maximum speed was 2.1 m/s.

In the third and fourth experiments (Fig 22 and 23), the UAV must fly through a space with poles of different heights, and finally below the cardboard boxes structure to reach the goal, achieving a maximum speed of 3.6 m/s.

Finally, in the fifth and sixth experiments (Fig. 24 and 25), the UAV is allowed to fly in a much bigger space, and has to avoid some poles and several cardboard boxes structures. In the fifth experiment (Fig. 24) the UAV achieved a top

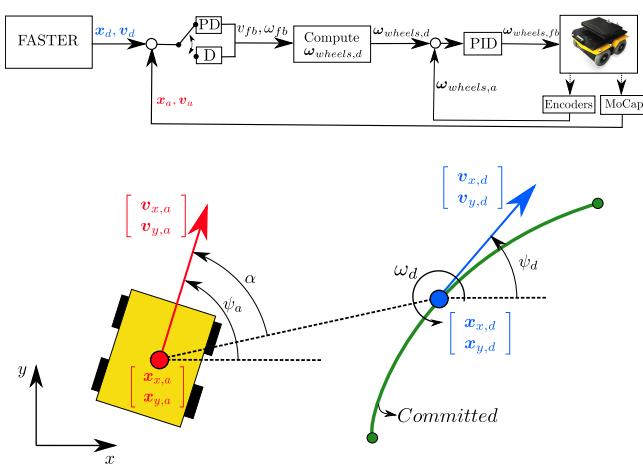


Figure 29. Tracking Controller for the ground robot.



Figure 30. Ground robot used in the experiments. It is equipped with Intel® RealSense Depth Camera D435, and an i7-7700HQ laptop.

speed of 7.8 m/s. In the sixth experiment (Fig. 25) the UAV was first commanded to go to a goal at the other side of the flight space, and then to come back to the starting position, achieving a top velocity of 4.6 m/s.

For $P_{max} = 2$, the boxplots of the runtimes achieved on the forest simulation (measured on an Intel Core i7-7700HQ) and on the hardware experiments (measured on the onboard Intel NUC i7DNK with the mapper and the RealSense also running on it) are shown in Fig. 27. For the runtimes of the MIQP of the Whole and the Safe Trajectories, the 75th percentile is always below 32 ms.

Extension to a ground robot

We now show how, with a different controller, FASTER is also applicable to skid-steer robots.

Controller

Denoting $[.]_d$ the desired value (the value obtained from the trajectory found by FASTER), $[.]_a$ the actual value, ψ the yaw angle, and $\dot{\phi}$ the derivate of the tangential angle of the trajectory (MathWorld 2019) let us define the following variables (see also Fig. 29):

$$\tilde{\psi} := \text{atan2}(\mathbf{v}_{y,d}, \mathbf{v}_{x,d}) - \psi_a$$

$$v_d := \sqrt{\mathbf{v}_{x,d}^2 + \mathbf{v}_{y,d}^2}$$

$$\omega_d \equiv \dot{\phi} = \frac{\mathbf{v}_{x,d} \mathbf{a}_{y,d} - \mathbf{v}_{y,d} \mathbf{a}_{x,d}}{\mathbf{v}_{x,d}^2 + \mathbf{v}_{y,d}^2}$$

$$\tilde{d} := \sqrt{(\mathbf{x}_{x,d} - \mathbf{x}_{x,a})^2 + (\mathbf{x}_{y,d} - \mathbf{y}_{y,a})^2}$$

$$\alpha := \psi_a - \text{atan2}(\mathbf{x}_{y,d} - \mathbf{x}_{y,a}, \mathbf{x}_{x,d} - \mathbf{x}_{x,a})$$

To track the trajectory, we use the cascade controller shown in Fig. 29: First we generate the linear velocity v_{fb} and angular velocity ω_{fb} using the following switching control law: If $\tilde{d} \geq d_0$, v_{fb} and ω_{fb} are obtained using P controllers with \tilde{d} and α respectively. If $\tilde{d} < d_0$, v_{fb} is obtained with a P controller based on v_d , while ω_{fb} is obtained with a PD controller using ω_d and $\tilde{\psi}$. The parameter d_0 was chosen as $d_0 = 15$ cm.

Once obtained the linear and angular velocities v_{fb} and ω_{fb} , they are converted to the desired angular velocities of the wheels $\omega_{wheels,d}$, and compared to the measurements obtained by the encoders. From this error, $\omega_{wheels,fb}$ (sent to the motors) are obtained using a PID controller.

Hardware Experiments

Three different experiments were done with the ground robot (see Figs. 31, 32, and Extension 3). An external motion capture system was used to estimate the position and orientation of the robot. Experiment 7 and 8 were done in obstacles environments similar to the random forest. The maximum speeds achieved for the experiments 7 and 8 were 1.95 m/s and 2.22 m/s respectively. Note that the maximum speed specified for this ground robot is ≈ 2 m/s (Clearpath 2019).

To test the ability of FASTER to reuse the map built, the setup for experiment 9 was a bugtrap environment, and only

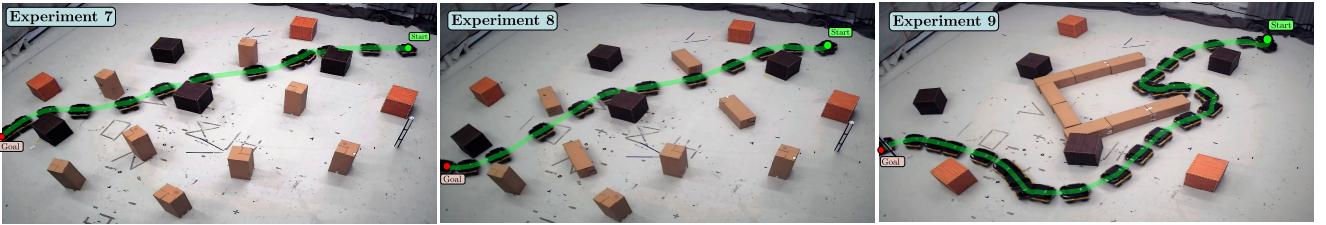


Figure 31. Composite images of Experiments 7, 8 and 9. The ground robot must go from start ● to goal ●. Snapshots shown every 670 ms. To show the ability of FASTER to get out from bugtraps, only points in the depth image closer than 3 meters were used to build the map in experiment 9.

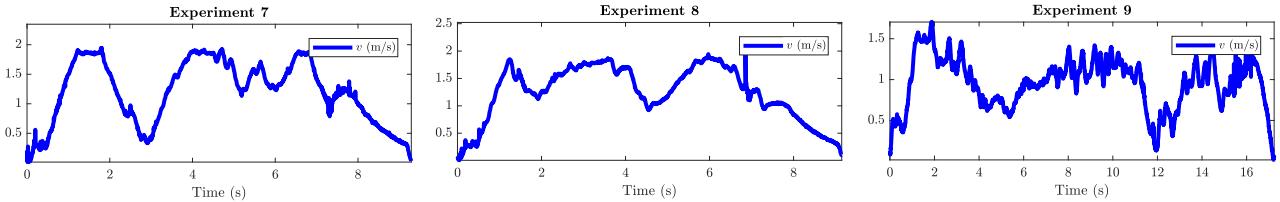


Figure 32. Velocity plots of the experiments 7, 8, and 9.

points in the depth image closer than 3 meters were used to build the map. The robot first enters the bugtrap because it does not see the end of it. Once the robot detects that there is no exit at the end of the bugtrap, it turns back, exits the bugtrap, passes through its left and avoids some new obstacles to finally reach the goal. The maximum speed achieved in this experiment was 1.70 m/s

CONCLUSIONS AND FUTURE WORK

This work presented FASTER, a fast and safe planner for agile flights in unknown environments. The key properties of this planner is that it leads to a higher nominal speed than other works by planning both in \mathcal{U} and \mathcal{F} using a convex decomposition, and ensures safety by having always a Safe Trajectory planned in \mathcal{F} at the beginning of every replanning step. FASTER was tested successfully both in simulated and in hardware flights, achieving velocities up to 7.8 m/s. Finally, we also showed how FASTER is also applicable to skid-steer robots, achieving hardware experiments at 2 m/s.

Future work include the relaxation of the assumption 1: we plan to include the uncertainty associated with the map (due to estimation error and/or sensor noise) in the replanning function, and to extend this planner for dynamic environments. Finally, we also plan to use onboard estimation algorithms like VIO instead of an external motion capture system for the real hardware experiments.

All the GAZEBO worlds used for the simulation are available at <https://github.com/jtorde> for future benchmark.

The videos of the simulation and hardware experiments are available on <https://www.youtube.com/watch?v=fkkkgomkX10>.

Acknowledgements

Thanks to Pablo Tordesillas (ETSAM-UPM) for his help with some figures of this paper, to Parker Lusk and Aleix Paris (ACL-MIT) for their help with the hardware, and to Helen Oleynikova (ASL-ETH) for the data of the forest simulation. The authors would also like to thank John Carter and John Ware (CSAIL-MIT) for their help with the mapper used in this paper.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: Supported in part by Defense Advanced Research Projects Agency (DARPA) as part of the Fast Lightweight Autonomy (FLA) program grant number HR0011-15-C-0110. Views expressed here are those of the authors, and do not reflect the official views or policies of the Dept. of Defense or the U.S. Government. The hardware was supported in part by Boeing Research and Technology. The first author of this paper was also financially supported by La Caixa fellowship.

References

- Augugliaro F, Schoellig AP and D’Andrea R (2012) Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, pp. 1917–1922.
- Bircher A, Kamel M, Alexis K, Oleynikova H and Siegwart R (2016) Receding horizon “next-best-view” planner for 3D exploration. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, pp. 1462–1468.
- Bresenham JE (1965) Algorithm for computer control of a digital plotter. *IBM Systems journal* 4(1): 25–30.
- Bucki N and Mueller MW (2019) Rapid collision detection for multicopter trajectories. *arXiv preprint arXiv:1904.04223* .
- Chen J, Liu T and Shen S (2016) Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, pp. 1476–1483.
- Clearpath (2019) Jackal UGV - Small weatherproof robot. <https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/>. (Accessed on 06/15/2019).
- Deits R and Tedrake R (2015) Efficient mixed-integer planning for uavs in cluttered environments. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 42–49.
- Dey D, Shankar KS, Zeng S, Mehta R, Agcayazi MT, Eriksen C, Daftry S, Hebert M and Bagnell JA (2016) Vision and

- learning for deliberative monocular cluttered flight. In: *Field and Service Robotics*. Springer, pp. 391–409.
- Florence P, Carter J and Tedrake R (2016) Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)*.
- Florence PR, Carter J, Ware J and Tedrake R (2018) Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7631–7638.
- Gao F, Wu W, Gao W and Shen S (2019) Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics* 36(4): 710–733.
- Gurobi Optimization L (2018) Gurobi optimizer reference manual. URL <http://www.gurobi.com>.
- Harabor D and Grastien A (2011) Online graph pruning for pathfinding on grid maps. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 1114–1119.
- Koenig N and Howard A (2004) Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3. IEEE, pp. 2149–2154.
- Lai Sp, Lan MI, Li Yx and Chen BM (2019) Safe navigation of quadrotors with jerk limited trajectory. *Frontiers of Information Technology & Electronic Engineering* 20(1): 107–119.
- Landry B, Deits R, Florence PR and Tedrake R (2016) Aggressive quadrotor flight through cluttered environments using mixed integer programming. In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1469–1475.
- Lau B, Sprunk C and Burgard W (2010) Improved updating of euclidean distance maps and voronoi diagrams. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, pp. 281–286.
- Liu C, Lin CY and Tomizuka M (2018a) The convex feasible set algorithm for real time optimization in motion planning. *SIAM Journal on Control and Optimization* 56(4): 2712–2733.
- Liu S, Atanasov N, Mohta K and Kumar V (2017a) Search-based motion planning for quadrotors using linear quadratic minimum time control. In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, pp. 2872–2879.
- Liu S, Mohta K, Atanasov N and Kumar V (2018b) Search-based motion planning for aggressive flight in $SE(3)$. *IEEE Robotics and Automation Letters* 3(3): 2439–2446.
- Liu S, Watterson M, Mohta K, Sun K, Bhattacharya S, Taylor CJ and Kumar V (2017b) Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Robotics and Automation Letters* 2(3): 1688–1695.
- Loianno G, Brunner C, McGrath G and Kumar V (2017) Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU. *IEEE Robotics and Automation Letters* 2(2): 404–411.
- Lopez BT and How JP (2017a) Aggressive 3-D collision avoidance for high-speed navigation. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, pp. 5759–5765.
- Lopez BT and How JP (2017b) Aggressive collision avoidance with limited field-of-view sensing. In: *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, pp. 1358–1365.
- Mao Y, Szmuk M and Acikmese B (2018) Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems. *arXiv preprint arXiv:1804.06539*.
- MathWorld W (2019) Tangential angle. <http://mathworld.wolfram.com/TangentialAngle.html>. (Accessed on 06/02/2019).
- Mellinger D and Kumar V (2011) Minimum snap trajectory generation and control for quadrotors. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, pp. 2520–2525.
- Mueller MW, Hehn M and D’Andrea R (2015) A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics* 31(6): 1294–1310.
- Oleynikova H, Burri M, Taylor Z, Nieto J, Siegwart R and Galceran E (2016) Continuous-time trajectory optimization for online uav replanning. In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, pp. 5332–5339.
- Oleynikova H, Taylor Z, Fehr M, Siegwart R and Nieto J (2017) Voxblox: Incremental 3D euclidean signed distance fields for on-board mav planning. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.
- Oleynikova H, Taylor Z, Siegwart R and Nieto J (2018) Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters* 3(3): 1474–1481.
- Pivtoraiko M, Mellinger D and Kumar V (2013) Incremental micro-UAV motion replanning for exploring unknown environments. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, pp. 2452–2458.
- Preiss JA, Hausman K, Sukhatme GS and Weiss S (2017) Trajectory optimization for self-calibration and navigation. In: *Robotics: Science and Systems*.
- Richter C, Bry A and Roy N (2016) Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In: *Robotics Research*. Springer, pp. 649–666.
- Rousseau G, Maniu CS, Tebbani S, Babel M and Martin N (2019) Minimum-time B-spline trajectories with corridor constraints. application to cinematographic quadrotor flight plans. *Control Engineering Practice* 89: 190–203.
- Ryll M, Ware J, Carter J and Roy N (2019) Efficient trajectory planning for high speed flight in unknown environments. In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Sahingoz OK (2014) Generation of Bézier curve-based flyable trajectories for multi-UAV systems with parallel genetic algorithm. *Journal of Intelligent & Robotic Systems* 74(1-2): 499–511.
- Schouwenaars T, Féron É and How J (2002) Safe receding horizon path planning for autonomous vehicles. In: *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 40. The University; 1998, pp. 295–304.
- Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K and Abbeel P (2014) Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research* 33(9): 1251–1270.

-
- Spitzer A, Yang X, Yao J, Dhawale A, Goel K, Dabhi M, Collins M, Boirum C and Michael N (2019) Fast and agile vision-based flight with teleoperation and collision avoidance on a multirotor. *arXiv preprint arXiv:1905.13419* .
- Tordesillas J, Lopez BT, Carter J, Ware J and How JP (2019a) Real-time planning with multi-fidelity models for agile flights in unknown environments. In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Tordesillas J, Lopez BT and How JP (2019b) FASTER: Fast and safe trajectory planner for flights in unknown environments. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.
- Van Nieuwstadt MJ and Murray RM (1998) Real-time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal* 8(11): 995–1020.
- Watterson M, Liu S, Sun K, Smith T and Kumar V (2018) Trajectory optimization on manifolds with applications to $SO(3)$ and $\mathbb{R}^3 \times S^2$. *Robotics: Science and Systems (RSS)*
- Zhou B, Gao F, Wang L, Liu C and Shen S (2019) Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters* 4(4): 3529–3536.

Appendix A: Index to Multimedia Extensions

1. Video: Simulation experiments.
2. Video: UAV experiments.
3. Video: Ground robot experiments.