

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

Jesus Tordesillas, Brett T. Lopez and Jonathan P. How

Abstract— High-speed trajectory planning through unknown environments requires algorithmic techniques that enable fast reaction times while maintaining safety as new information about the operating environment is obtained. The requirement of computational tractability typically leads to optimization problems that do not include the obstacle constraints (collision checks are done on the solutions) or use a convex decomposition of the free space and then impose an ad-hoc time allocation scheme for each interval of the trajectory. Moreover, safety guarantees are usually obtained by having a local planner that plans a trajectory with a final “stop” condition in the free-known space. However, these two decisions typically lead to slow and conservative trajectories. We propose FASTER (Fast and Safe Trajectory Planner) to overcome these issues. 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. Furthermore, we present a Mixed Integer Quadratic Program formulation in which the solver can choose the trajectory interval allocation, and where a time allocation heuristic is computed efficiently using the result of 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 3.6 m/s.

SUPPLEMENTARY MATERIAL

Code: FASTER (<https://github.com/mit-acl/faster>) and simulation worlds (<https://github.com/jtorde>)

Video: <https://youtu.be/gwV0YRs5IWs>.

I. INTRODUCTION

Navigating through unknown environments entails repeatedly generating collision-free, dynamically feasible trajectories that are executed over a finite horizon. Similar to that in the Model Predictive Control (MPC) literature, safety is guaranteed by ensuring a feasible solution exists indefinitely. If we consider $\mathbb{R}^3 = \mathcal{O} \cup \mathcal{F} \cup \mathcal{U}$ where \mathcal{F} , \mathcal{O} , \mathcal{U} are disjoint sets denoting free-known, occupied-known, and unknown space respectively, safety is guaranteed by constructing trajectories that are entirely contained in \mathcal{F} with a final stop condition. This can be achieved by generating motion primitives that do not intersect $\mathcal{O} \cup \mathcal{U}$ [1]–[4], or by constructing a convex representation of \mathcal{F} to be used in an optimization [5]–[7]. However, both approaches lead to slow trajectories in scenarios where \mathcal{F} is small compared to $\mathcal{U} \cup \mathcal{O}$. This paper presents an optimization-based approach that reduces the aforementioned limitations by solving for

The authors are with the Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA {jtorde, btlopez, jhow}@mit.edu

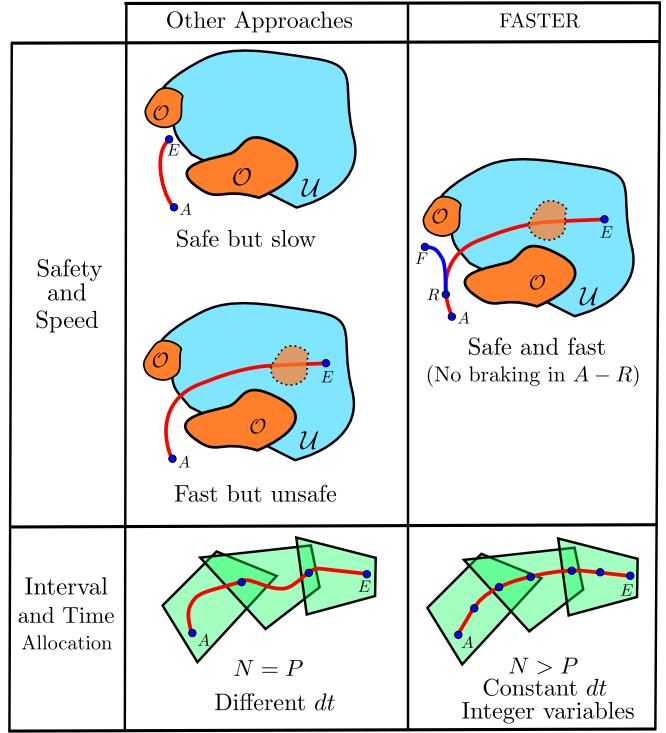


Fig. 1: Contributions of this work.

two optimal trajectories at every planning step (see Fig. 1): one in $\mathcal{U} \cup \mathcal{F}$, and another one in \mathcal{F} .

Decomposing the free space into P overlapping polyhedra along a path connecting a start A to goal E location (see Fig. 1), the usual approach is to divide the total trajectory into $N = P$ intervals [5]. On one hand, this simplifies the problem because no integer variables are needed, as each interval is forced to be in one specific polyhedron. On the other hand, the time allocation problem becomes much harder, as there are N different dt_n (time allocated for each interval n). The trajectory is also more conservative since the optimizer is only allowed to move the end points of each interval of the trajectory in the overlapping areas. To overcome these two problems, we propose the use of the same dt for all the intervals, and use $N > P$ intervals, encoding the optimization problem as a Mixed Integer Quadratic Program (MIQP). Moreover, and as the minimum feasible dt depends on the state of the UAV and on the specific shape of \mathcal{F} and \mathcal{U} at a specific replanning step, we also propose an efficient way to compute a heuristic of this dt using the result obtained in the previous replanning iteration.

In summary, this work has the following contributions:

- 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 MIQP formulation for the interval and time allocation problem of the flight trajectories compared to prior work.
- Extension of our previous work [8], (where we considered the interaction between the global and local planners) 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 3.6 m/s.

II. RELATED WORK

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

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 [2]–[4], 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 using a history of perception data. Within this second category, some works [8]–[10] limit the local planner to generate trajectories only in *free-known space* (\mathcal{F} in Fig. 1), which guarantees safety if the local planner has a final stop condition. However, limiting the planner to operating in free-known space and enforcing a terminal stopping condition can lead to conservative, slow trajectories (especially when much of the world is unknown). While allowing the local planner to optimize in both the free-known and unknown space ($\mathcal{F} \cup \mathcal{U}$), higher speeds can be obtained but with no guarantees that the trajectory is safe or will remain feasible.

As far as the *optimization formulation* is concerned, two approaches can be highlighted. The first does not include the obstacles in the optimization problem [8], leading to a closed-form solution for the trajectory [1], [3], [4] or in general to very small computation times [8]. The computation time for these approaches are very low since obstacles are not explicitly considered in the trajectory generation. This enables multiple candidate trajectories to be generated (via sampling) and evaluated for collisions (using nearest-neighbor search) at each planning stage. While these approaches are computationally efficient, they are unable to construct sophisticated maneuvers due to the discretization of the candidate trajectories, leading to slower trajectories in cluttered environments. The second approach is to include obstacle constraints directly in the optimization. This is usually done describing the free space by a set of overlapping polyhedra (also known as convex decomposition) [5]–[7]. The trajectory can then be parameterized by a sequence of third (or higher)-degree polynomials. Bézier Curves [7],

[11] or the sum-of-squares condition [6], [12] can be used to guarantee that the trajectory remains in the overlapping polyhedra. Subsequently, there will be an interval (which polytope each polynomial is in) and a time allocation (how much time is assigned to each interval) problem. For the **interval allocation**, a typical solution is to use the same number of trajectory segments as number of polyhedra, and then each polynomial segment is forced to be inside its corresponding polyhedron [7]. However, this can be very restrictive since the solver only has the freedom to select where the two endpoint points of each interval are placed in the overlapping regions. Another option, but with higher computation times, is to use binary decision variables [6], [12] to allow the solver to choose the specific interval allocation. For the **time allocation**, one can either use a fixed time allocation [5] or formulate a bi-level optimization to find the times [13], [7]. However, the first approach can be very conservative and can cause infeasibility in the optimization problem while the seconds leads to longer replanning times.

III. FAST AND SAFE TRAJECTORY PLANNER

A. Planning

The Fast and Safe Trajectory Planner (FASTER) uses hierarchical architecture where a long-horizon global planner guides a short-horizon local planner to a desired goal location. The global planner used in this work is Jump Point Search (JPS). JPS finds the shortest piecewise linear path between two points in a 3D uniformly-weighted voxel grid, guaranteeing optimality and completeness but running an order of magnitude faster than A* [14], [5].

For the local planner, we distinguish these three different jerk-controlled trajectories (some of the points will be precisely defined later, see Fig. 3):

- **Whole Trajectory:** This trajectory goes from a start location A to goal location 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 point inside the polyhedra obtained by doing a convex decomposition of \mathcal{F} . It is completely contained in \mathcal{F} (free-known space), 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 is also guaranteed to be inside \mathcal{F} (see explanation below). This trajectory is the one that the UAV will execute 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).

Let $n = 0 : N - 1$ denote the specific interval of the trajectory and $p = 0 : P - 1$ the specific polyhedron. If $j(t)$ 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 :

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

Let us denote the sequence of P overlapping polyhedra as $\{\mathbf{A}_p, \mathbf{c}_p\}$, $p = 0 : P - 1$, and introduce binary variables b_{np} (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 whole trajectory will be inside this convex corridor by forcing that all the control points are in the same polyhedron [7], [11] with the constraint $[b_{np} = 1 \implies \mathbf{r}_{nj} \in \text{polyhedron } p \ \forall j]$, and at least in one polyhedron with the constraint $\sum_{p=0}^{P-1} b_{np} \geq 1$. The optimizer is free to choose in which polyhedron exactly. The complete MIQP solved in each replanning step (using Gurobi, [15]) for both the Safe and the Whole trajectories is this one:

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

In the optimization problem above, dt (same for every interval n) is computed as

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

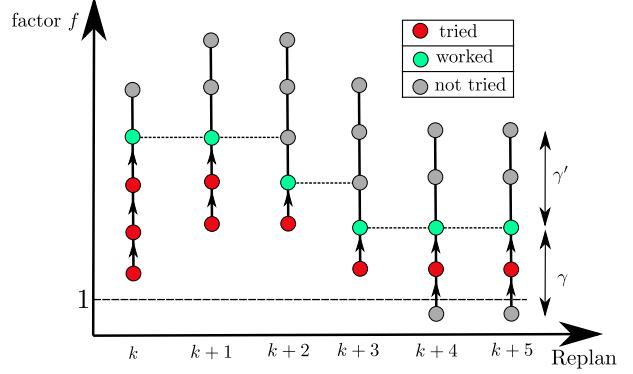


Fig. 2: Dynamic adaptation of the factor used to compute the heuristic of the time allocation per interval (dt): For iteration k , the range of factors used is taken around the factor that worked in the iteration $k - 1$.

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. 2): The optimizer will try values of f (in increasing order) in the interval $[f_{\text{worked}, k-1} - \gamma, f_{\text{worked}, k-1} + \gamma']$ until the problem converges. Here, $f_{\text{worked}, k-1}$ is the factor that made the problem feasible in the previous replanning step. Note that, if $f = 1$, then dt is a lower bound on the minimum time per interval required for the problem to be feasible.

Algorithm 1: FASTER

Data: Current Position of the UAV L , Committed_{k-1} , JPS_{k-1} , G_{term} , \mathcal{O} , \mathcal{F} , \mathcal{U} , r

Function Replan():

- 1 $k \leftarrow k + 1$, $\delta t \leftarrow \alpha \Delta t_{k-1}$, $\delta t' \leftarrow \beta \Delta t_{k-1}$
- 2 Choose point A in Committed_{k-1} with offset δt from L
- 3 $G \leftarrow$ Projection of G_{term} into map \mathcal{M}
- 4 $JPS_a \leftarrow$ Run JPS $A \rightarrow G$
- 5 $C \leftarrow JPS_a \cap \mathcal{S}$
- 6 $JPS_b \leftarrow$ Modified JPS_{k-1} such that $JPS_{k-1} \cap \mathcal{O} = \emptyset$
- 7 $D \leftarrow JPS_b \cap \mathcal{S}$
- 8 $dt_a \leftarrow$ Lower bound on dt $A \rightarrow C$
- 9 $dt_b \leftarrow$ Lower bound on dt $A \rightarrow D$
- 10 $J_a = N \cdot dt_a + \frac{v_{max}}{\|JPS_a(C \rightarrow G)\|}$
- 11 $J_b = N \cdot dt_b + \frac{v_{max}}{\|JPS_b(D \rightarrow G)\|}$
- 12 $JPS_k \leftarrow \underset{\{JPS_a, JPS_b\}}{\operatorname{argmin}} \frac{v_{max}}{\{J_a, J_b\}}$
- 13 $JPS_k \leftarrow JPS_a$
- 14 $JPS_{in} \leftarrow$ Part of JPS_k inside \mathcal{S}
- 15 $Poly_{whole} \leftarrow$ Convex Decomposition in $\mathcal{U} \cup \mathcal{F}$ using JPS_{in}
- 16 $f_{whole} \leftarrow [f_{whole, k-1} - \gamma, f_{whole, k-1} + \gamma']$
- 17 $\text{Whole} \leftarrow$ MIQP in $Poly_{whole}$ from A to G using f_{whole}
- 18 $JPS_{in, known} \leftarrow$ Part of JPS_{in} in \mathcal{F}
- 19 $Poly_{safe} \leftarrow$ Convex Decomposition in \mathcal{F} using $JPS_{in, known}$
- 20 $f_{safe} \leftarrow [f_{safe, k-1} - \gamma, f_{safe, k-1} + \gamma']$
- 21 $\text{Safe} \leftarrow$ MIQP in $Poly_{safe}$ from R to F using f_{safe}
- 22 $\text{Committed}_k \leftarrow \text{Whole}_{A \rightarrow R} \cup \text{Safe}$
- 23 $f_{whole, k} \leftarrow$ Factor that worked for Whole
- 24 $f_{safe, k} \leftarrow$ Factor that worked for Safe
- 25 $\Delta t_k \leftarrow$ Total replanning time

Algorithm 1 summarizes the full approach (see also Fig. 3). 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

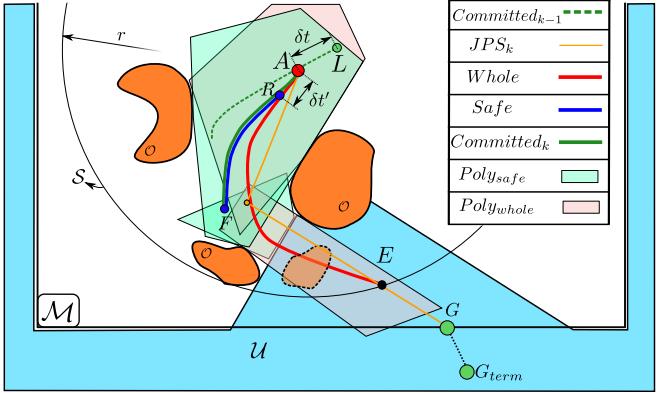


Fig. 3: Illustration for Alg.1. \mathcal{U} is the unknown space (□), and \mathcal{O} are the known obstacles (■). One unknown obstacle is shown with dotted line.

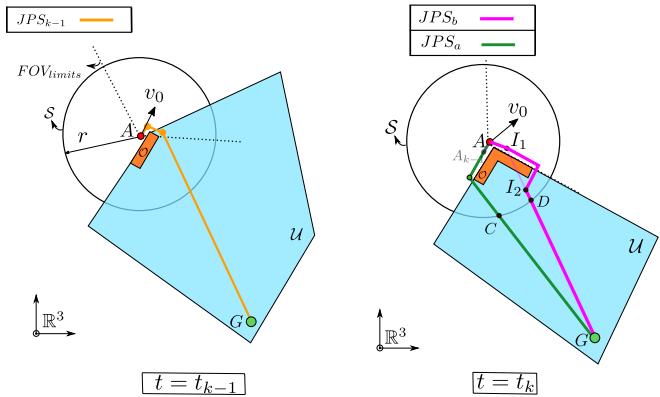


Fig. 4: Choice of the direction to optimize. At $t = t_{k-1}$, the JPS solution chosen was JPS_{k-1} . At $t = t_k$, JPS is run again to obtain JPS_a , and JPS_{k-1} is modified so that it does not collide with \mathcal{O} , obtaining JPS_b . A heuristic of the cost-to-go in each direction is computed, and the direction with the lowest cost is chosen as the one towards which the local planner will optimize.

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 times the solver is able to find the next solution in less than δt . 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 must decide if the current JPS solution should be used to guide the optimization (lines 6–14). 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. 4): 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 $I_2 \rightarrow I_G$. 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 . 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.

The **Whole Trajectory** (lines 15–18) is obtained as follows. We do the convex decomposition [5] of $\mathcal{U} \cup \mathcal{F}$ around JPS_{in} , which is the part of JPS_k that is inside the sphere S . 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 19–22. First we choose the point R along the Whole Trajectory with an offset $\delta t'$ from A (this $\delta t'$ is computed by multiplying the previous replanning time by $\beta \geq 1$), and run convex decomposition in \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. IV).

Finally (line 23), 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 guarantees a higher nominal speed on this first piece. The intervals $L \rightarrow A$ and $A \rightarrow R$ have been designed so that, with high probability, at least one replanning step can be solved within that interval. Moreover, to prevent the (very rare) cases where both A and R are in \mathcal{F} , but the piece $A \rightarrow R$ is not, we check that piece $A \rightarrow R$ against collision with \mathcal{U} . If any of the two optimizations in this algorithm fails, or the piece $A \rightarrow R$ intersects \mathcal{U} , or the replanning step takes longer than δt , the UAV does not commit to a new trajectory in that replanning step, and continues executing the Committed Trajectory of the previous replanning step. Thus safety is guaranteed by construction: the UAV will only fly Committed Trajectories, which are always guaranteed to be in \mathcal{F} with a terminal stopping condition.

B. Mapping

For the mapping, we use a sliding map centered on the UAV that moves as the UAV flies. We fuse a depth map into the occupancy grid using the 3D Bresenham's line algorithm for ray-tracing [16], and \mathcal{O} and \mathcal{U} are inflated by the radius of the UAV to ensure safety.

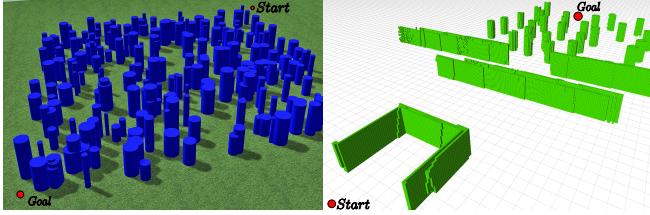


Fig. 5: Forest (left) and bugtrap (right) environments used in the simulation. The forest is 50×50 m, and the grid in the bugtrap environment is $1 \text{ m} \times 1 \text{ m}$.

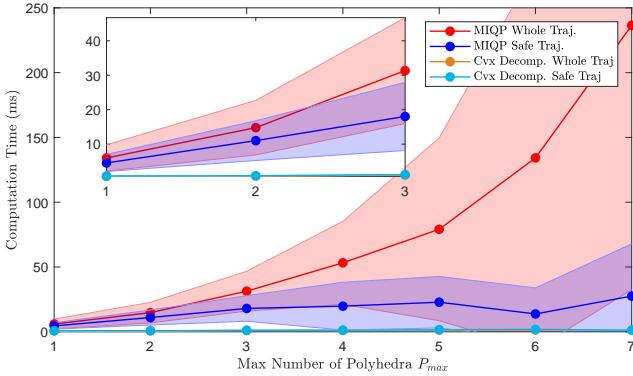


Fig. 6: Timing breakdown for the MIQP and Convex Decomposition of the Whole Trajectory and the Safe Trajectory as a function of the maximum number of polyhedra P_{max} . Note that the times for the MIQPs include all the trials until convergence (with different factors f) in each replanning step. The shaded area is the $1-\sigma$ interval, where σ is the standard deviation. These results are from the forest simulation.

IV. RESULTS

A. Simulation

We first test FASTER in 10 random forest environments with an obstacle density of 0.1 obstacles/m² (see Fig. 5) 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) [17], Safe Local Exploration [10], (see [10] for details of all these approaches), and Multi-Fidelity [8].

The results are shown in Table I, which highlights that FASTER achieves a 8 – 51% improvement in the distance. Completion times are compared in Table II to our previous proposed algorithm [8] (time values are not available for all other algorithms in Table I). FASTER achieves an improvement of 52% in the completion time. The dynamic constraints imposed for the results of this table are (per axis) $v_{max} = 5 \text{ m/s}$, $a_{max} = 5 \text{ m/s}^2$, and $j_{max} = 8 \text{ m/s}^3$.

We also test FASTER using the bugtrap environment shown in Fig. 5, and obtain the results that appear on Table III. 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 $v_{max} = 10 \text{ m/s}$, $a_{max} = 10 \text{ m/s}^2$, and $j_{max} = 40 \text{ m/s}^3$.

The timing breakdown of Alg. 1 as a function of the

TABLE I: Distances obtained in 10 random forest simulations. Improvement percentages are computed for the minimum and the maximum of each column. Some results were provided by the authors of [10].

Method	Number of Successes	Distance (m)			
		Avg	Std	Max	
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 [17]	6	159.3	45.6	246.9	123.6
SL Expl. [10]	8	103.8	21.6	148.3	86.6
Mult-Fid [8]	10	84.5	11.7	109.4	73.2
FASTER	10	77.6	5.9	88.0	70.7
Min/Max improvement (%)		8/51	43/89	20/67	3/43

TABLE II: Comparison between [8] and FASTER of flight times in the forest simulation. Results are for 10 random forests.

Method	Time (s)			
	Avg	Std	Max	Min
Mult-Fid [8]	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 III: Comparison between [8] and FASTER of flight distances and times in a bugtrap simulation.

Method	Distance (m)		Time (s)	
	Mult-Fid [8]	FASTER	Avg	Time
Mult-Fid [8]	56.8	55.2	37.6	37.6
FASTER	55.2	29.2	13.8	13.8
Improvement (%)	2.8	75.0	60.2	63.3



Fig. 7: UAV used in the experiments. It is equipped with a Qualcomm® SnapDragon Flight, an Intel® NUC and an Intel® RealSense Depth Camera D435.

maximum number of polyhedra P_{max} is shown in Fig. 6. 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, $P_{max} = 2 – 3$ was used. The runtimes for JPS as a function of the voxel size of the map for the forest simulation are available in Fig. 7 of [8]. All these timing breakdowns were measured using an Intel Core i7-7700HQ 2.8GHz Processor.

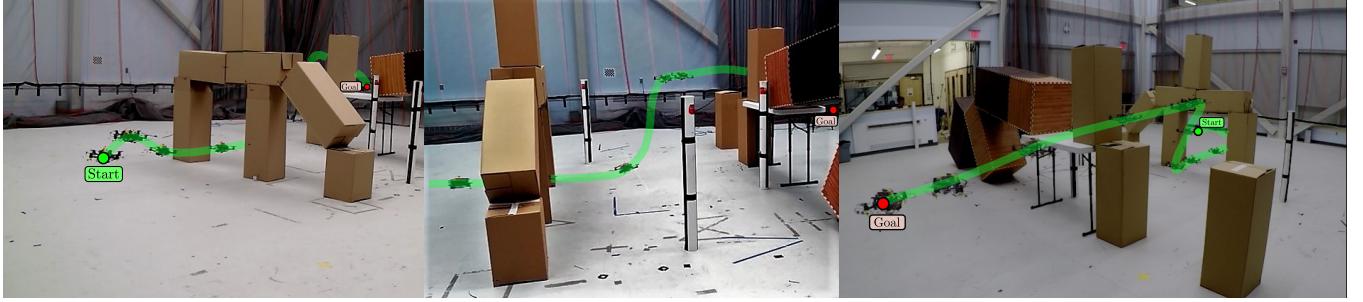


Fig. 8: Composite images of Experiment 1. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.

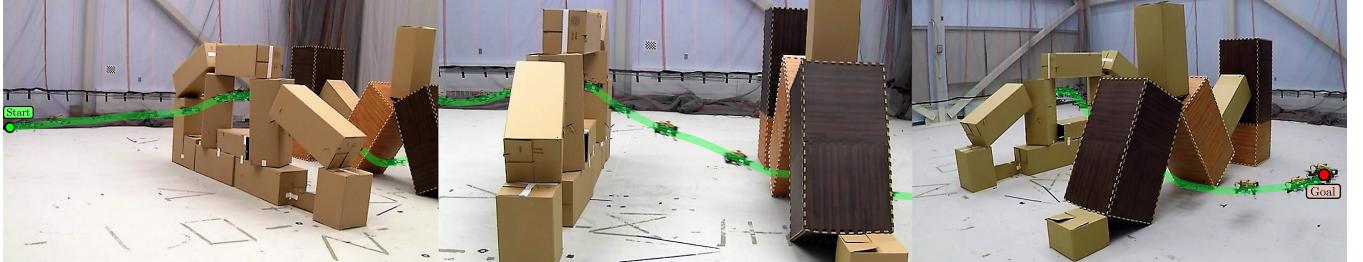


Fig. 9: Composite image of Experiment 2. The UAV must fly from start ● to goal ●. Snapshots shown every 330 ms.

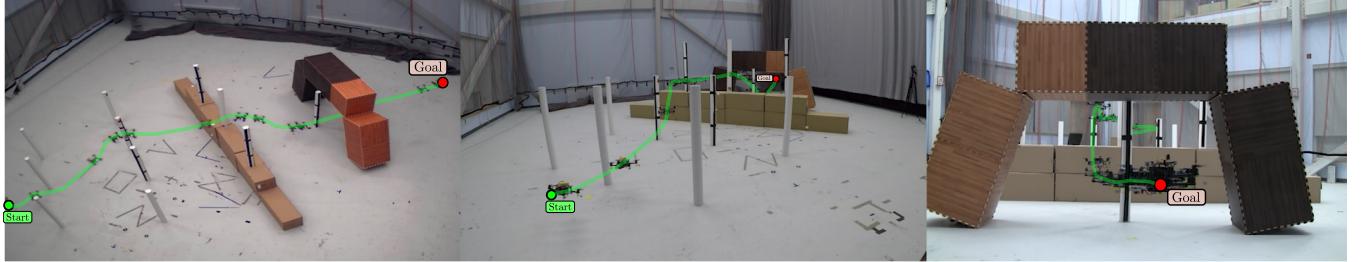


Fig. 10: Composite image of Experiment 3. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.



Fig. 11: Composite image of Experiment 4. The UAV must fly from start ● to goal ●. Snapshots shown every 670 ms.

B. Hardware

The UAV used in the hardware experiments is shown in Fig. 7. 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 attitude, IMU biases, position and velocity are estimated by fusing (via a Kalman filter) propagated IMU measurements with an external motion capture system.

The first and second experiments (Fig. 8 and 9) were done in similar obstacle environments with the same starting point, but with different goal locations. In the first experiment (Fig. 8), the UAV performs a 3D agile maneuver to avoid the obstacles on the table. In the second experiment (Fig. 9) 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 10 and 11), 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.

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 with the mapper and the RealSense also running on it) are shown in Fig. 12. For the runtimes of the MIQP for the Whole and the Safe Trajectories, the 75th percentile is always below 32 ms.

V. CONCLUSIONS

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

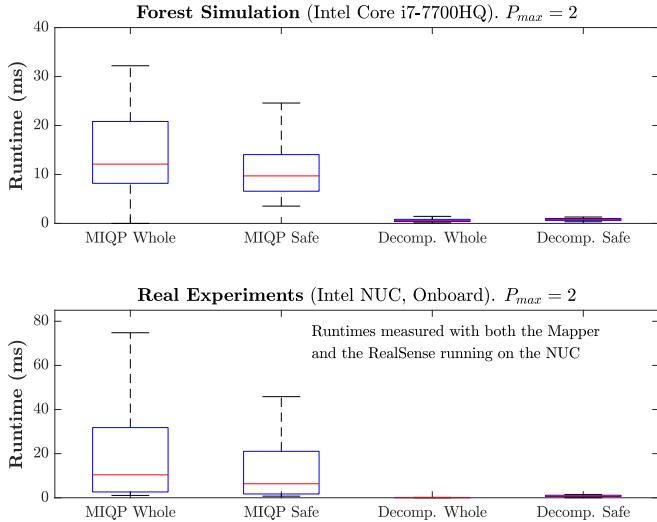


Fig. 12: Timing breakdown for the forest simulation and for the real hardware experiments. The parameters used are $P_{max} = 2$, $N = 10$ for the Whole Trajectory, and $N = 7$ for the Safe Trajectory.

than other works by planning both in \mathcal{U} and \mathcal{F} , 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 3.6 m/s.

ACKNOWLEDGMENT

Thanks to Boeing Research & Technology for support of the hardware, to my brother Pablo Tordesillas (ETSAM-UPM) for his great help with some figures of this paper and to Parker Lusk (ACL-MIT) for his help with the hardware. The authors would also like to thank John Carter and John Ware (CSAIL-MIT) for their help with the mapper used in this paper. Supported in part by Defense Advanced Research Projects Agency (DARPA) as part of the Fast Lightweight Autonomy (FLA) program, 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.

REFERENCES

- [1] Mark W Mueller, Markus Hehn, and Raffaello D’Andrea. A computationally efficient motion primitive for quadrocopter trajectory

- generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.
- [2] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.
- [3] Brett T Lopez and Jonathan P How. Aggressive 3-D collision avoidance for high-speed navigation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5759–5765. IEEE, 2017.
- [4] Brett T Lopez and Jonathan P How. Aggressive collision avoidance with limited field-of-view sensing. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 1358–1365. IEEE, 2017.
- [5] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.
- [6] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for uavs in cluttered environments. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 42–49. IEEE, 2015.
- [7] James A Preiss, Karol Hausman, Gaurav S Sukhatme, and Stephan Weiss. Trajectory optimization for self-calibration and navigation. In *Robotics: Science and Systems*, 2017.
- [8] Jesus Tordesillas, Brett T Lopez, John Carter, John Ware, and Jonathan P How. Real-time planning with multi-fidelity models for agile flights in unknown environments. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.
- [9] Tom Schouwenaars, Éric Féron, and Jonathan How. Safe receding horizon path planning for autonomous vehicles. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 40, pages 295–304. The University; 1998, 2002.
- [10] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters*, 3(3):1474–1481, 2018.
- [11] Ozgur Koray Sahingoz. Generation of bezier curve-based flyable trajectories for multi-uav systems with parallel genetic algorithm. *Journal of Intelligent & Robotic Systems*, 74(1-2):499–511, 2014.
- [12] Benoit Landry, Robin Deits, Peter R Florence, and Russ Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1469–1475. IEEE, 2016.
- [13] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.
- [14] Daniel Damir Harabor, Alain Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, 2011.
- [15] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.
- [16] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.
- [17] Andreas Bircher, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart. Receding horizon “next-best-view” planner for 3D exploration. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1462–1468. IEEE, 2016.