Real-Time Planning with Multi-Fidelity Models for Agile Flights in Unknown Environments

Jesus Tordesillas[1], Brett T. Lopez[1], John Carter[2], John Ware[2] and Jonathan P. How[1]

***Abstract*— Autonomous navigation through unknown environments is a challenging task that entails real-time localization, perception, planning, and control. UAVs with this capability have begun to emerge in the literature with advances in lightweight sensing and computing. Although the planning methodologies vary from platform to platform, many algorithms adopt a hierarchical planning architecture where a slow, low-fidelity global planner guides a fast, high-fidelity local planner. However, in unknown environments, this approach can lead to erratic or unstable behavior due to the interaction between the global planner, whose solution is changing constantly, and the local planner; a consequence of not capturing higher-order dynamics in the global plan. This work proposes a planning framework in which multi-fidelity models are used to reduce the discrepancy between the local and global planner. Our approach uses high-, medium-, and low-fidelity models to compose a path that captures higher-order dynamics while remaining computationally tractable. In addition, we address the interaction between a fast planner and a slower mapper by considering the sensor data not yet fused into the map during the collision check. This novel mapping and planning framework for agile flights is validated in simulation, showing replanning times of 5-40 ms in cluttered environments, a value that is 3-30 times faster than similar state-of-the-art planning algorithms.**

I. INTRODUCTION

UAV autonomous navigation in unknown environments has received special interest in the last few years because of its unlimited applications, ranging from aerial surveying and inspection to search and rescue. However, these applications are often reduced to low-speed flights due to the current limitations and low rates of the state-of-the-art mappers and planners. The inherent non-convexity of the path planning optimization problem, together with the high mapping and planning rate needed for agile flights make this problem especially hard. This work presents a novel framework to perform high-rate mapping and planning in unknown environments suitable for agile maneuvers, addressing the fundamental problem between the interaction of a global planner and a local planner.

Computational tractability of the planning problem leads to the use of a low-fidelity global planner that computes a cost-to-go (CTG) needed by the high-fidelity local planner. However, the fact that the global planner does not account for the dynamics results in erratic behaviors when the world model is changing rapidly. There is therefore a need of an accurate CTG calculation that captures both the global environment and the dynamic feasibility, maintaining relatively low computation times at the same time.

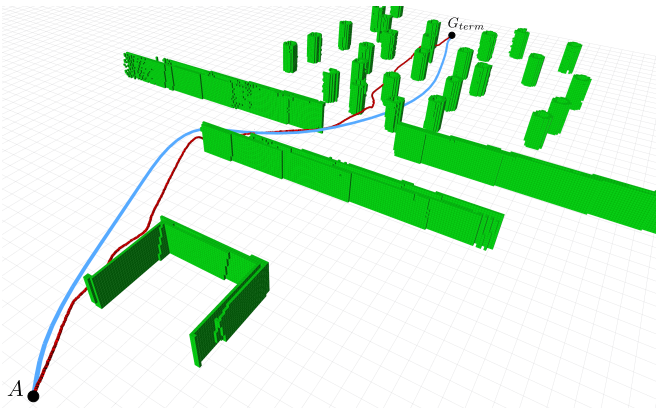

Fig. 1: Global optimum and our method. When the map is completely known, the optimal trajectory computed using the approach of [1] is shown in blue (■). The red trajectory (■) is the solution found by our method, where the world is not known and it is being discovered as the UAV flies forward. The grid is 1m $\times$1m, and the sensing range is 10 m.

Moreover, the choice of the representation of the environment and the size of the "global" map (larger scale than the sensor FOV and the local representation, but typically does not contain all information observed to reduce effort) have a significant impact on the computational cost, but for most systems updates of these models cannot be done at the sensor frame rates ($\sim$30 Hz) and updates are typically slower than the re-plan rate. Thus a second design challenge is how to combine the global knowledge (available at a slower rate) with the high-rate local information in the planner representation of the environment. Finally, the state-of-the-art mappers and planners run onboard at $\sim$5 Hz, so the the third key challenge is how to optimize the planning and mapping algorithms to achieve higher rates, suitable for aggressive flights.

This work addresses these challenges with the following contributions:

- A novel formulation of the planning problem that takes into account the dynamics of the vehicle in the cost-to-go calculation to solve the negative interaction that usually occurs between the global and local planners when operating in unknown environments.
- A lightweight fused-based mapping framework using a sliding map to reduce the estimation error influence that runs onboard fusing a depth image in 50 ms.
- An integration of a high-rate planner with a slower-rate mapper, with a collision check algorithm that accounts

[1]J. Tordesillas, B. Lopez, J. How are with the Aerospace Controls Laboratory, MIT, 77 Massachusetts Ave., Cambridge, MA, USA {jtorde, btlopez, jhow}@mit.edu
[2] J. Carter and J. Ware are with the MIT Robust Robotics Group. {jakeware, jcarter}@csail.mit.edu

for both the most recent fused information and the available sensed data not included in that map.
- Simulation experiments showing agile flights in completely unknown cluttered environments, achieving replanning times of 5-40 ms.

II. RELATED WORK

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

For **planning**, most of the current state-of-the-art methods exploit the differential flatness of the quadrotors, and solve the planning problem minimizing the squared norm of a derivative of the position [2], [3] to find a dynamically feasible smooth trajectory [4]. On the one hand, there are approaches where the obstacle constraints, and sometimes also 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 [2], [4], [5]. Others use Pontryagin's Minimum Principle to find closed-form solutions [6]. These closed-form solutions are also used to search over the state space [7], [1]. Alternatively, there are works that use the cost function to penalize the distance to the obstacles [8], [9], requiring usually computationally expensive distance fields representations. Finally, there are also approaches that add the obstacle constraints in the optimization problem, like Mixed-Integer Programming (MIP) [10], convex decompositions [11], [12], [13] and Successive Convexification [14], [15], [16].

Most of the approaches described above either assume that the global map is known, and the optimization is done using the global map. Others assume that there is a global planner (RRT or exploration based planners) that gives the waypoints to the local planner. In agile flights, this leads to oscillatory behaviors, since the world is being discovered and the solution of the global planner changes constantly.

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 [17], [18]. These approaches are in general unable to reason about obstacles observed in the past [19], [20]. 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 [21], [22]. Two drawbacks of these approaches are the influence of the estimation error, and the fusion time (which means that the planner usually uses an out-of-date fused map).

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 [19] and [20]. 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: [23] and [24] used optimistic planners (considering unknown space as free), while in [8] and [9], an optimistic global planner is used combined with a conservative local planner.

III. PROBLEM FORMULATION

A. Planning

Computing a dynamically feasible trajectory from the start to the goal is typically intractable, which is the standard argument for the distinction between the global and local planners. The global planner gives the local planner a notion of cost-to-go (CTG) or traversability in certain directions; while the local planner deviates around the nearby obstacles and chooses the terminal point accordingly. Key to this CTG calculation is the trade-off between accuracy and computation time. Issues include the sophistication of the dynamics used in the calculation and the number of points at which the CTG is computed.

In our proposed framework, Jump Point Search (JPS) is used as a global planner to find the shortest 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 [25], [12]. The only assumption of JPS is a uniform grid, which holds in our case. JPS is only done for position (not velocity or acceleration) to reduce the computational burden.

Our local planner monitors the JPS solution for drastic changes between each replan. If a large change is detected, a new path, composed of a high, medium, and low fidelity model (where the model order is reduced farther away from the vehicle) is created for the current and last JPS solution. This procedure is able to capture a subset of the dynamics while maintaining computational tractability. The resulting multi-fidelity paths are compared and the path with lowest cost is selected for execution. This hierarchical trajectory consists of a jerk-controlled part, a velocity-controlled part, and a geometric part.

The jerk-controlled primitive is the part of the trajectory nearest to the current position, and it is the one that will be actually executed. The quadrotor is modeled using triple integrator dynamics with state $\mathbf{x}^T = \begin{bmatrix}\boldsymbol{x}^T & \dot{\boldsymbol{x}}^T & \ddot{\boldsymbol{x}}^T\end{bmatrix} = \begin{bmatrix}\boldsymbol{x}^T & \boldsymbol{v}^T & \boldsymbol{a}^T\end{bmatrix}$ and control input $\mathbf{u} = \dddot{\boldsymbol{x}} = \boldsymbol{j}$ (where $\boldsymbol{x}$, $\boldsymbol{v}$, $\boldsymbol{a}$, and $\boldsymbol{j}$ are the vehicle's position, velocity, acceleration, and jerk, respectively), and the following convex optimization problem is solved using CVXGEN [26] to find this trajectory:

$$\min_{u_{0:N-1}} \sum_{i=0}^{N-1} \|\mathbf{u}_i\|^2 + (\mathbf{x}_N - \mathbf{x}_f)^T Q (\mathbf{x}_N - \mathbf{x}_f) \qquad (1)$$

$$\begin{aligned}
\text{subject to } & \mathbf{x}_0 = \mathbf{x}_{init} \\
& \mathbf{x}_{k+1} = M_1 \mathbf{x}_k + M_2 \mathbf{u}_k \quad \forall k = 0 : N-1 \\
& \|\boldsymbol{v}_k\|_\infty \leq v_{max} \qquad \forall k = 1 : N \\
& \|\boldsymbol{a}_k\|_\infty \leq a_{max} \qquad \forall k = 1 : N \\
& \|\boldsymbol{u}_k\|_\infty \leq j_{max} \qquad \forall k = 0 : N-1.
\end{aligned}$$

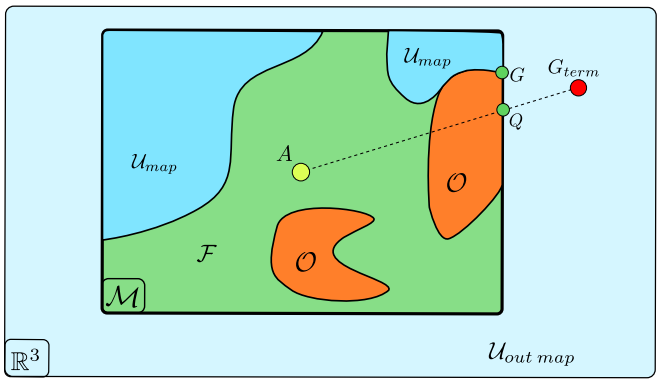

Fig. 2: Sliding map and goal projection. The sliding map $\mathcal{M}$ has occupied space $\mathcal{O}$, unknown space $\mathcal{U}_{map}$ and free space $\mathcal{F}$. Total unknown space is $\mathcal{U} = \mathcal{U}_{map} \cup \mathcal{U}_{out\ map}$. $\boldsymbol{Q}$ is the projection into the map of the terminal goal $\boldsymbol{G}_{term}$ in the direction of $\overrightarrow{\boldsymbol{AG}_{term}}$. The closest free or unknown frontier point to $\boldsymbol{Q}$ ($\boldsymbol{G}$ in the figure) is selected as the goal.

In this problem, the number of discretization steps N is fixed. The time step dt (embedded in M_1 and M_2) is computed as

$$dt = \max\{T_v, T_{a_x}, T_{a_y}, T_{a_z}, T_{j_x}, T_{j_y}, T_{j_z}\}$$

where $T_v = \|\boldsymbol{x}_{init} - \boldsymbol{x}_f\|_2 / v_{max}$, and T_{a_i}, T_{j_i} are the solution for each axis $i = \{x, y, z\}$ of the constant-acceleration and constant-jerk motion equations applying a_{max} and j_{max} respectively. This dt is a tight lower bound that is increased in each iteration until the problem converges.

The second key part of the trajectory is a velocity-controlled primitive. It is a trade-off between the sophistication of the dynamic model (lower order than jerk input model), but sufficiently accurate to capture the UAV CTG (more accurate than the distance-based cost). Moreover, it ensures that the computation times are maintained in the order of 300 μs, four times faster than the ones required when the input is higher. The final part of the trajectory is the part of the JPS solution that goes from the end of the velocity-controlled primitive to the goal. This provides an indication of how to avoid traps and avoid obstacles, but there is little attempt to capture to vehicle dynamics at that distance away – that is done when the receding horizon controller gets closer.

B. Mapping

A sliding map, which moves with the UAV, is used to represent the world. This is a compromise between storing the whole world and relying only on local maps. It also tries to minimize the accumulated estimation error. This map $\mathcal{M}$ contains free space $\mathcal{F}$, (known) obstacles $\mathcal{O}$ and unknown space $\mathcal{U}_{map}$ (see Fig. 2). In this way,

$$\mathbb{R}^3 = \mathcal{O} \cup \mathcal{F} \cup \mathcal{U}_{map} \cup \mathcal{U}_{Out\ map} = \mathcal{M} \cup \mathcal{U}_{Out\ map}$$

Using this map, the collision check for each of the three primitives presented above is done as follows: the jerk-controlled trajectory is considered collision-free if it does not intersect $\mathcal{O} \cup \mathcal{U}$. The velocity-controlled primitive is forced to pass through the waypoints of the JPS solution. Finally, the JPS path is guaranteed not to hit $\mathcal{O}$. In this way, and

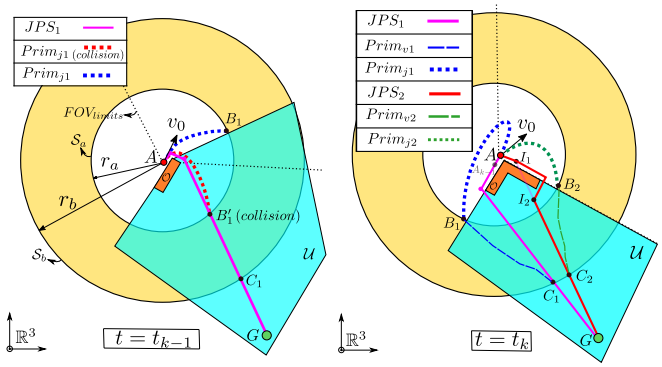

Fig. 3: Illustration of the Alg. 1. The radii r_a and r_b define the spherical surfaces $\mathcal{S}_a$ and $\mathcal{S}_b$. JPS is used to compute the paths JPS_1 and JPS_2. The UAV chooses the jerk-controlled primitive $Prim_{ji}$ that has lowest cost, considering the cost of that primitive, the velocity-controlled primitive $Prim_{vi}$ and the distance from $\boldsymbol{C}_i$ to $\boldsymbol{G}$ following JPS_i. Unknown space $\mathcal{U}$ is shown in blue. Note that the figure is in 2D for visualization purposes, but the planning is in 3D.

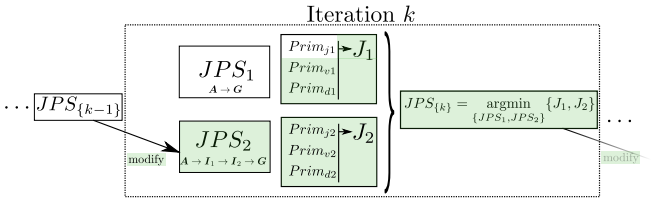

Fig. 4: Iteration k in the Alg. 1. JPS_2 is the modified version of $JPS_{\{k-1\}}$, that avoids the new obstacles detected. The algorithm chooses the jerk-controlled primitive $Prim_{ji}$ that has the lowest associated cost-to-go J_i. The terms in green are only computed if $\angle \boldsymbol{B'}_1 \boldsymbol{A} \boldsymbol{B'}_{\{k-1\}}$ is greater than α_0.

similar to [9], JPS is an optimistic global planner, while the local planner is conservative.

IV. ALGORITHM

Let us introduce some notation first (see also Fig. 3): Let $\mathbf{A}$ be the current position of the UAV. Let $\mathcal{S}_a$ and $\mathcal{S}_b$ be two concentric spheres with center in $\mathbf{A}$, and with radius r_a and r_b respectively. JPS will denote the shortest piece-wise linear path found by running JPS between $\mathbf{A}$ and the goal. For this path, define the tuples $JPS_{wp} := (\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_n)$ and $JPS_{path} := (\overline{\boldsymbol{q}_1\boldsymbol{q}_2}, \dots, \overline{\boldsymbol{q}_{n-1}\boldsymbol{q}_n})$, where JPS_{wp} has the n waypoints $\boldsymbol{q}_i \in \mathbb{R}^3$ of the solution of JPS (being $\boldsymbol{q}_1$ the start and $\boldsymbol{q}_n$ the goal) and JPS_{path} contains the segments between these waypoints.

Define the point $\boldsymbol{B'} := \cap_{\boldsymbol{f}}(\mathcal{S}_a, JPS_{path})$, where $\cap_{\boldsymbol{f}}(\mathcal{S}, JPS_{path})$ is a function that computes the **f**irst intersection between the spherical surface $\mathcal{S}$ and the path JPS_{path}. Similarly, $\boldsymbol{C} := \cap_{\boldsymbol{l}}(\mathcal{S}_b, JPS_{path})$ will be the ***l***ast intersection. In an analogous way, $\cap_{\boldsymbol{int}}(\mathcal{S}_a, \mathcal{S}_b, JPS_{wp})$ will denote all the elements of JPS_{wp} that are ***int***ermediate between $\boldsymbol{B'}$ and $\boldsymbol{C}$. Hence, JPS_{wp} can be written as:

$$JPS_{wp} = (\boldsymbol{q}_1, \dots \boldsymbol{q}_{r-1}, \underbrace{\boldsymbol{q}_r, \dots, \boldsymbol{q}_{s-1}}_{\cap_{\boldsymbol{int}}(\mathcal{S}_a, \mathcal{S}_b, JPS_{wp})}, \boldsymbol{q}_s, \dots, \boldsymbol{q}_n)$$

where $\boldsymbol{q}_r$ is the first point outside the sphere $\mathcal{S}_a$ and $\boldsymbol{q}_s$ is the first point outside the sphere $\mathcal{S}_b$. The subindex $\{\cdot\}$ will indicate the iteration number. Finally, the subindex 1 in JPS

will denote the JPS solution of the current iteration, while 2 will be the modified (i.e not intersecting any obstacles) version of $JPS_{\{k-1\}}$ (chosen in the previous iteration). Same applies to $\boldsymbol{B'}$ and $\boldsymbol{C}$. The concept of modified is explained later on.

In our algorithm, the whole trajectory planned is divided into three primitives: $Prim_j \cup Prim_v \cup Prim_d$, which are defined as follows:

1) $Prim_j$ is a jerk-controlled primitive that has as initial state the current state of the vehicle, and as a final state a stop condition in the point $\boldsymbol{B}$, which is defined as the point in the sphere $\mathcal{S}_a$ near to $\boldsymbol{B'}$ that guarantees that $Prim_j \cap (\mathcal{O} \cup \mathcal{U}) = \varnothing$.
2) $Prim_v$ is defined as the composition of several velocity-controlled primitives between each pair of consecutive points of $(\boldsymbol{B}, \boldsymbol{q}_r, \ldots, \boldsymbol{q}_{s-1}, \boldsymbol{C})$.
3) Finally, $Prim_d$ is the part of JPS that goes from $\boldsymbol{C}$ to the goal, and therefore $Prim_d \cap \mathcal{O} = \varnothing$.

The time-normalized cost associated with each one of these primitives are:

$$J_{Prim_j} = \frac{N \cdot dt}{j_{max}^2} \cdot \sum_{i=0}^{N-1} \|\boldsymbol{j}_i\|_2^2$$

$$J_{Prim_v} = \sum_{k=r}^{s-r+1} \left(\frac{N \cdot dt_k}{v_{max}^2} \cdot \sum_{i=0}^{N-1} \|\boldsymbol{v}_{k_i}\|_2^2 \right)$$

$$J_{Prim_d} = \frac{\|\boldsymbol{q}_s - \boldsymbol{C}\|_2}{v_{max}} + \sum_{k=s}^{n-1} \left(\frac{\|\boldsymbol{q}_{k+1} - \boldsymbol{q}_k\|_2}{v_{max}} \right)$$

The total cost of the planned trajectory is then $J := J_{Prim_j} + J_{Prim_v} + J_{Prim_d}$. In this total cost we combine a high-fidelity model near the UAV, a medium-fidelity model in the medium part, and model without dynamics for the farthest part of the trajectory. This cost is an approximation of the total cost of the jerk-controlled trajectory that goes from $\boldsymbol{A}$ to $\boldsymbol{G}$, but it is much faster to compute. Also, it is more accurate than relying only on jerk for the first part, and distance in the JPS path for the rest, since an intermediate velocity-controlled primitive is included.

The proposed approach is shown in Alg. 1 and Fig. 3. For iteration k, it proceeds as follows — first (line 3), the terminal goal $\boldsymbol{G_{term}}$ is projected into $\mathcal{M}$ in the direction of $\overrightarrow{\boldsymbol{AG_{term}}}$ to obtain $\boldsymbol{Q}$ (see Fig. 2). The nearest unknown or free point to $\boldsymbol{Q}$ is selected as the (intermediate) goal $\boldsymbol{G}$, and JPS is run from the actual position $\boldsymbol{A}$ to $\boldsymbol{G}$ to obtain JPS_1. The intersection of JPS_1 with the inner sphere $\mathcal{S}_a$ defines the point $\boldsymbol{B'}_1$. This point indicates the direction towards which the jerk-controlled planner should optimize. As a jerk-controlled primitive from $\boldsymbol{A}$ to $\boldsymbol{B'}_1$ is not guaranteed to be collision free, we sample $\approx$ 30 points in the sphere $\mathcal{S}_a$ around $\boldsymbol{B}'_1$ to obtain a final position that makes this primitive collision-free, storing them in the priority queue $\mathcal{K}$ (line 19). We prioritize points near to $\boldsymbol{B'}$ that have at the same time a high probability of the primitive being collision-free. As the geometry of JPS encodes where the obstacles are, we sample these points in the following way (Fig. 5): First, the points $\boldsymbol{q}_2, \ldots, \boldsymbol{q}_{r-1}$ are projected onto $\mathcal{S}_a$ to obtain $\boldsymbol{P}_{\boldsymbol{q}_2}, \ldots, \boldsymbol{P}_{\boldsymbol{q}_{r-1}}$. Then we sample from $\boldsymbol{B'}$ to $\boldsymbol{P}_{\boldsymbol{q}_{r-1}}$, from $\boldsymbol{P}_{\boldsymbol{q}_{r-1}}$ to $\boldsymbol{P}_{\boldsymbol{q}_{r-2}}$, and so on. Finally, we append to this priority queue some samples taken in concentric circles to $\boldsymbol{B'}$, to increase the probability of finding a feasible final condition.

Algorithm 1: Replan

Data: A, G_{term}, $\mathcal{O}$, $\mathcal{F}$, $\mathcal{U}$, $\delta_a < \delta_b < 1$, $\alpha_0 > 0$, $R_b > R_{a,max} > R_{a,min} > 0$,

1 **Function** `Replan()`:
2 $k \leftarrow k+1$, $J_1 \leftarrow 0$ and $J_2 \leftarrow \infty$
3 $G \leftarrow GetIntermediateGoal(G_{term})$
4 $JPS_1 \leftarrow RunJPS(A, G)$
5 $r_a = \min(sat_{R_{a,min}}^{R_{a,max}}(\|\overrightarrow{Aq_2}\|_2), \delta_a \cdot \|\overrightarrow{AG}\|_2)$
6 $r_b = \min(Rb, \delta_b \cdot \|\overrightarrow{AG}\|_2)$
7 $Prim_{j1} \leftarrow GetPrimj(1)$
8 **if** $\angle B'_1 A B'_{\{k-1\}} > \alpha_0$ **then**
9 $JPS_2 \leftarrow JPS_{\{k-1\}}$
10 **if** $Intersects(JPS_{\{k-1\}})$ **then**
11 $JPS_2 \leftarrow RunJPS(A \rightarrow I_1 \rightarrow I_2 \rightarrow G)$
12 $Prim_{j2} \leftarrow GetPrimj(2)$
13 $J_1 \leftarrow GetCost(1)$ and $J_2 \leftarrow GetCost(2)$
14 Choose i with lowest cost J_i
15 $JPS_{\{k\}} \leftarrow JPS_i$ and $B'_{\{k\}} \leftarrow B'_i$
16 **return** $Prim_{ji}$
17 **Function** `GetPrimj(i)`:
18 $B'_i \leftarrow \cap_f(\mathcal{S}_a, JPS_{i,path})$
19 $\mathcal{K} \leftarrow SamplePoints(B'_i)$
20 **for** $Size(\mathcal{K})$ *times* **do**
21 $B_i \leftarrow Pop(\mathcal{K})$
22 $Prim_{ji} \leftarrow Cvx_{Jerk}(InitialCond(), B_i)$
23 **return** $Prim_{ji}$ **if** $IsFree(Prim_{ji})$
24 **Function** `GetCost(i)`:
25 $C_i \leftarrow \cap_f(\mathcal{S}_b, JPS_{i,path})$
26 $\mathcal{WP} \leftarrow \cap_{int}(\mathcal{S}_a, \mathcal{S}_b, JPS_{i,wp})$
27 $Prim_{vi} \leftarrow Cvx_{Vel}(B_i, \mathcal{WP}, C_i)$
28 **return** $J_{Prim_{j_i}} + J_{Prim_{v_i}} + J_{Prim_{d_i}}$

Then we iterate over each point in $\mathcal{K}$ (setting it as the final position in the optimization problem 1) and stop when a collision free primitive (i.e that does not intersect $\mathcal{O} \cup \mathcal{U}$) is found (lines 20-23). At this point, the algorithm computes the angle $\angle \boldsymbol{B'}_1 \boldsymbol{A} \boldsymbol{B'}_{\{k-1\}}$, where $\boldsymbol{B'}_{\{k-1\}}$ is the intersection of $\mathcal{S}_a$ with $JPS_{\{k-1\}}$. This angle gives a measure of how much the JPS solution has changed from the iteration $k-1$. A small angle indicates that JPS_1 and $JPS_{\{k-1\}}$ are very similar (at least within the sphere $\mathcal{S}_a$), and that therefore the local plan will not differ much from the iteration $k-1$. Hence, if this angle is smaller than a threshold α_0 (typically $\approx 15°$), the algorithm finishes, and $Prim_{j1}$ is returned. If it is bigger, the algorithm needs to decide whether obtaining the local plan using JPS_1, or relying on the previous path $JPS_{\{k-1\}}$. To do this, in lines 10-11 first we modify $JPS_{\{k-1\}}$ by obtaining $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ (first and last intersections of $JPS_{\{k-1\}}$ with $\mathcal{O}$) and run JPS three times to obtain the paths $\boldsymbol{A} \rightarrow \boldsymbol{I}_1$, $\boldsymbol{I}_1 \rightarrow \boldsymbol{I}_2$, and $\boldsymbol{I}_2 \rightarrow \boldsymbol{G}$. The union of these paths is JPS_2. Once JPS_2 is obtained, a very similar process as explained before is done, but with JPS_2 in this case, to obtain $Prim_{j2}$ (line 12).

The decision between $Prim_{j1}$ or $Prim_{j2}$ is made (lines 14-16) choosing the one that has the lowest cost $J_i := J_{Prim_{ji}} + J_{Prim_{vi}} + J_{Prim_{di}}$, $i = \{1, 2\}$.

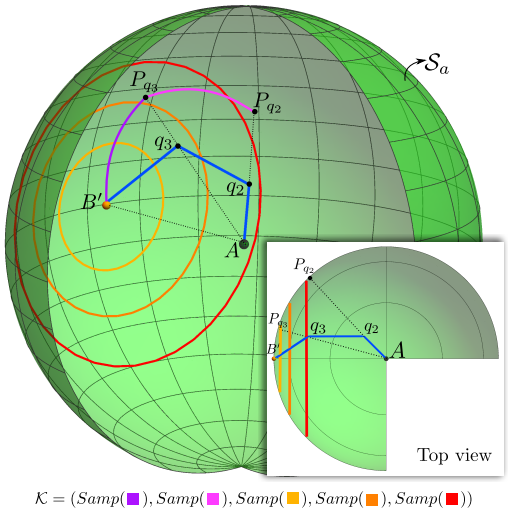

Fig. 5: Priority queue $\mathcal{K}$. Given JPS (■), the priority queue $\mathcal{K}$ returned by $SamplePoints(\boldsymbol{B'})$ contains points in this order: First samples along the spherical arc $\boldsymbol{B'} \rightarrow \boldsymbol{P_{q_3}}$. Then along the arc from $\boldsymbol{P_{q_3}}$ to the next projection, and so on. After that, several samples are taken from concentric circumferences to $\boldsymbol{B'}$.

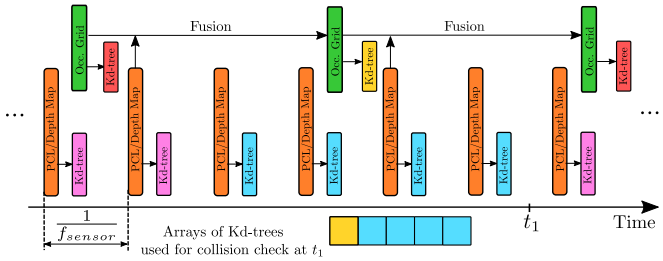

Fig. 6: Instantaneous sensing data and occupancy grid pipeline. Sensing data from the depth sensor (■) is received at f_{sensor}. New point clouds are fused into the Occupancy Grid (■). Collision check at t_1 is done using an array of k-d trees that contains the k-d tree of the last map fused (■), and the k-d trees of the last point clouds received that are not included in the map (■).

A. Mapping and Planning Integration

We fuse a depth map into the occupancy grid using the 3D Bresenham's line algorithm for ray-tracing [27]. However, as discussed earlier, the mapper update rate is slower than the sensor frame rate, and problems can arise when the local planner only relies on this (out-of-date) map to generate a primitive. This issue is addressed here by storing the k-d trees of the point clouds that have arrived since the most recent map was published (see Fig. 6). Collision checks are then done using the occupancy grid and some of the saved k-d trees of the instantaneous point clouds. This combination ensures that the local planner relies both on the most recent fused map with global knowledge of the world, and on up-to-date point clouds that contain the instantaneous sensing data.

V. EXPERIMENTAL RESULTS

A. Simulation

We evaluate the performance of the proposed algorithm in different simulated scenarios. The simulator uses C++ custom code for the dynamics engine, and Gazebo [28] to simulate perception data (in the form of a depth map and a point cloud). In all these simulations, the depth camera has a horizontal FOV of 90°, and a sensing range of 10 m.

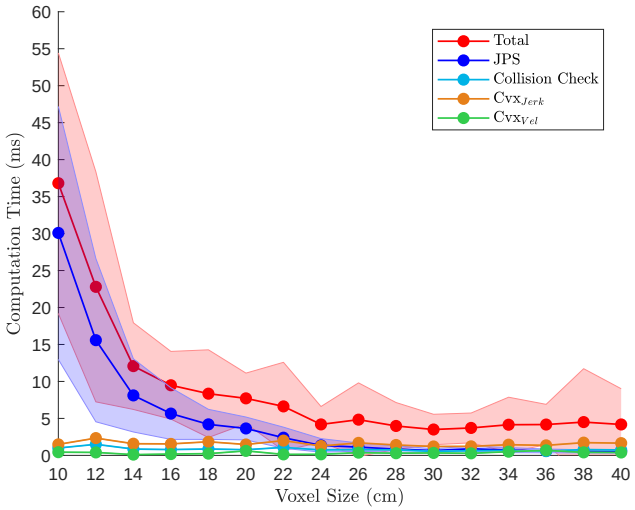

Fig. 7: Timing breakdown for different voxel sizes in the forest simulation. The replan function takes less than 10 ms when the voxel size is bigger than 15 cm. The sliding map is 20m×20m.

We first compare our method against six different methods: Incremental approach (no goal selection), random goal selection, optimistic RRT* (unknown space = free), conservative RRT* (unknown space=occupied), "next-best-view" planner (NBVP) [29], and Safe Local Exploration [9]. These six methods are described deeper in [9]. The scenario setting is a random cluttered forest with an obstacle density of 0.1 obstacles/m^2 (see Fig. 8) and a sliding map size of 20m×20m, with a voxel size of 10 cm. The results for ten different random forests are shown in Table I. Our method succeeds in all 10 simulations and obtains a path that is in average 19–47 % shorter than the other methods.

The map and voxel size chosen have a strong impact on the computation times and performance. For a given map, a small voxel size provides a more accurate solution, at the expense of more computation time running JPS and collision check. For a map size of 20m×20m, the timing breakdown of the replan function in the forest simulation for different map voxel sizes is shown in Fig. 7. The replan function takes 37 ms in average when the voxel size is 10 cm, and is reduced to less than 10 ms when the voxel size is bigger than 15 cm. JPS takes $\approx 80\%$ of the total replanning time when the voxel size is low, and $\approx 15\%$ when the voxel size is higher. This is due to the fact that the computation time of JPS depends on the number of cells, which is reduced by the cube of the voxel size. Note that the values of Cvx$_{jerk}$ and Cvx$_{vel}$ indicate the total time for all the jerk-controlled and velocity-controlled primitives computed in each replanning step. The mean time per primitive is 1.3 ms for jerk and 0.3 ms for velocity. The distances of these primitives range from 0.5 to 4 m and from 2 to 5.5 m (depending on the geometry of the obstacles) for the jerk-controlled and velocity-controlled primitives respectively.

TABLE I: Distances obtained by seven different methods in the cluttered forest simulation. The distance values are computed for the cases that reach the goal. The improvement percentages are computed for the minimum and the maximum of each column. The results for the other planners were provided by the authors of [9].

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 [29]	6	159.3	45.6	246.9	123.6
SL Expl. [9]	8	103.8	21.6	148.3	86.6
Ours	**10**	**84.5**	11.7	**109.4**	**73.2**
Min/Max improvement (%)		**19/47**	**-14/78**	**13/59**	**16/41**

To compare the trajectory found by our approach (in which the map is **discovered** as the UAV proceeds) with the optimal trajectory when the map is completely **known**, we use two simulation environments: a *bugtrap* scenario (Fig. 1) and a cluttered office scenario (Fig. 9). In the *bugtrap* scenario, our method produces a trajectory of 56.8 m, approximately the same length as the optimal trajectory (56.3 m). In the office simulation, the total length with our approach is 41.5 m (optimal trajectory is 35.9 m). In this simulation, the UAV enters two rooms, but when it detects that there is no exit, turns back and finds another path to the goal. In both simulations, the optimal trajectory has been obtained using the approach proposed in [1], which is optimal in the discretized space. As in our approach the world is being discovered gradually, our solution is not globally optimal, and it requires more control effort than the optimal one. However, the similarity between these two paths reflects the performance of our algorithm, able to obtain a near-optimal path even when the world is discovered gradually.

B. Hardware Experiments

The UAV used for the future hardware experiments is shown in Fig. 10. All the perception, planning and control runs onboard, and the position, velocity, attitude, and IMU biases are estimated by fusing propagated IMU measurements with an external motion capture system via a Kalman filter. The mapping fusion times achieved onboard are 50ms and 80ms for depth image resolutions of 480×270 and 640×480 respectively. All these experiments are available in this link and also in the video accompanying this work.

VI. CONCLUSIONS

This work presented a novel planning a mapping framework suitable for agile flights in unknown environments. The key properties of this framework is its ability to solve the interaction between the global planner and the local planner considering the dynamics of the vehicle, and its ability to address efficiently the integration between a fast planner and a slower mapper. The replanning and mapping rates are several times faster than the state of the art.

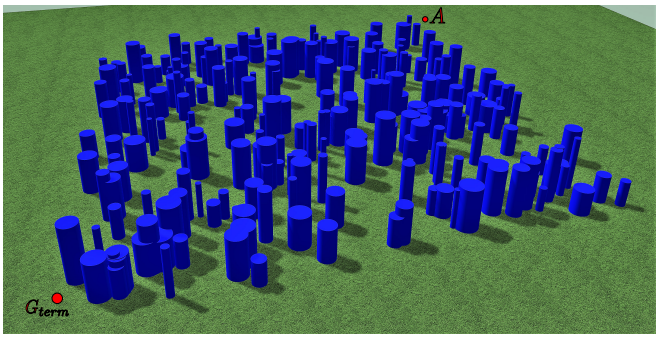

Fig. 8: Forest simulation. The UAV must fly from $\boldsymbol{A}$ to $\boldsymbol{G_{term}}$ in a 50m×50m forest with an obstacle density of 0.1 obstacles/m^2.

Fig. 9: Office simulation. The UAV must fly from $\boldsymbol{A}$ to $\boldsymbol{G_{term}}$ in an office environment. The optimal trajectory is shown in blue (■). The red trajectory (■) is the solution found by our method.

Fig. 10: UAV used in the experiments. It is equipped with a Qualcomm® SnapDragon Flight, an Nvidia® Jetson TX2 and an Intel® RealSense Depth Camera D435.

ACKNOWLEDGMENT

Thanks to Boeing Research & Technology for support of the hardware, to Helen Oleynikova (ASL-ETH) for the data of the forest simulation, and to Pablo Tordesillas (ETSAM-UPM) for his help with some figures of 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] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018.

[2] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.

[3] Michiel J Van Nieuwstadt and Richard M Murray. Real-time trajectory generation for differentially flat systems. *International Journal of Robust and Nonlinear Control: IFAC-Affiliated Journal*, 8(11):995–1020, 1998.

[4] 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.

[5] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. 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, 2017.

[6] 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.

[7] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 2872–2879. IEEE, 2017.

[8] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5332–5339. IEEE, 2016.

[9] 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.

[10] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *American Control Conference, 2002. Proceedings of the 2002*, volume 3, pages 1936–1941. IEEE, 2002.

[11] Changliu Liu, Chung-Yen Lin, and Masayoshi Tomizuka. The convex feasible set algorithm for real time optimization in motion planning. *SIAM Journal on Control and Optimization*, 56(4):2712–2733, 2018.

[12] 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.

[13] Michael Watterson, Sikang Liu, Ke Sun, Trey Smith, and Vijay Kumar. Trajectory optimization on manifolds with applications to so (3) and r 3× s2. *Robotics: Science and Systems (RSS)*, 2018.

[14] Yuanqi Mao, Michael Szmuk, and Behcet Acikmese. Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems. *arXiv preprint arXiv:1804.06539*, 2018.

[15] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. 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*, pages 1917–1922. IEEE, 2012.

[16] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[17] Debadeepta Dey, Kumar Shaurya Shankar, Sam Zeng, Rupesh Mehta, M Talha Agcayazi, Christopher Eriksen, Shreyansh Daftry, Martial Hebert, and J Andrew Bagnell. Vision and learning for deliberative monocular cluttered flight. In *Field and Service Robotics*, pages 391–409. Springer, 2016.

[18] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. *arXiv preprint arXiv:1802.09076*, 2018.

[19] 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.

[20] 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.

[21] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Improved updating of euclidean distance maps and voronoi diagrams. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 281–286. IEEE, 2010.

[22] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[23] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar. Incremental micro-uav motion replanning for exploring unknown environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2452–2458. IEEE, 2013.

[24] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1476–1483. IEEE, 2016.

[25] Daniel Damir Harabor, Alban Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, 2011.

[26] Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[27] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[28] Nathan P Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer, 2004.

[29] 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.