

# Real-Time Trajectory Replanning for MAVs using Uniform B-splines and a 3D Circular Buffer

Vladyslav Usenko, Lukas von Stumberg, Andrej Pangercic and Daniel Cremers  
Technical University of Munich

**Abstract**—In this paper, we present a real-time approach to local trajectory replanning for microaerial vehicles (MAVs). Current trajectory generation methods for multicopters achieve high success rates in cluttered environments, but assume that the environment is static and require prior knowledge of the map. In the presented study, we use the results of such planners and extend them with a local replanning algorithm that can handle unmodeled (possibly dynamic) obstacles while keeping the MAV close to the global trajectory. To ensure that the proposed approach is real-time capable, we maintain information about the environment around the MAV in an occupancy grid stored in a three-dimensional circular buffer, which moves together with a drone, and represent the trajectories by using uniform B-splines. This representation ensures that the trajectory is sufficiently smooth and simultaneously allows for efficient optimization.

## I. INTRODUCTION

In recent years, microaerial vehicles (MAVs) have gained popularity in many practical applications such as aerial photography, inspection, surveillance and even delivery of goods. Most commercially available drones assume that the path planned by the user is collision-free or provide only limited obstacle-avoidance capabilities. To ensure safe navigation in the presence of unpredicted obstacles a replanning method that generates a collision-free trajectory is required.

Formulation of the trajectory generation problem largely depends on the application and assumptions about the environment. In the case where an MAV is required to navigate a cluttered environment, possibly an indoor one, we would suggest subdividing the problem into two layers. First, we assume that a map of the environment is available and a trajectory from a specified start point to the goal point is planned in advance.

This task has been a popular research topic in recent years, and several solutions have been proposed by Achtelik et al. [1] and Richter et al. [21]. They used occupancy representation of the environment to check for collisions and searched for the valid path in a visibility graph constructed using sampling based planners. Thereafter, they followed the approach proposed by Mellinger and Kumar [14] to fit polynomial splines through the points of the planned path to generate a smooth feasible trajectory. The best algorithms of this type can compute a trajectory through tens of waypoints in several seconds.

To cope with any unmodeled, possibly dynamic, obstacle a lower planning level is required, which can generate a trajectory that keeps the MAV close to the global path

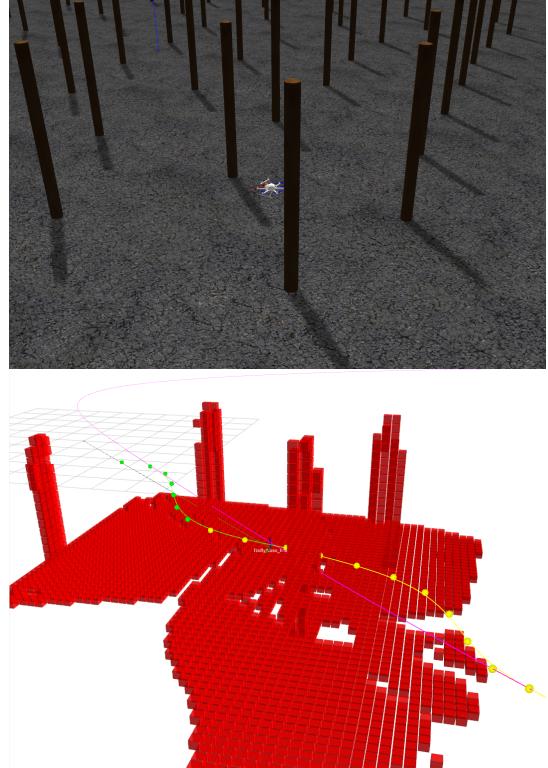


Fig. 1: Example of local trajectory replanning algorithm running in the simulator. Global trajectory is visualized in purple and the local obstacle map is visualized in red. The local trajectory is represented by a uniform quintic B-spline, and its control points are visualized in yellow for the fixed parts and in green for the parts that can still change due to optimization.

and simultaneously avoids unpredicted obstacles based on an environment representation constructed from the most recent sensor measurements. This replanning level should run in several milliseconds to ensure the safety of MAVs operating at high velocities.

The proposed approach solves a similar problem as that solved by Oleynikova et al. [17], but instead of using polynomial splines for representing the trajectory we propose the use of B-splines and discuss their advantages over polynomial splines for this task. Furthermore, we propose the use of a robocentric, fixed-size three-dimensional (3D) circular buffer to maintain local information about the environment. Even

though the proposed method cannot model arbitrarily large occupancy maps, as some octree implementations, faster look-up and measurement insertion operations make it better suited for real-time replanning tasks.

We demonstrate the performance of the system in several simulated and real-world experiments, and provide open-source implementation of the software to community.

The contributions of the present study are as follows:

- Formulation of local trajectory replanning as a B-spline optimization problem and thorough comparison with alternative representations (polynomial, discrete).
- High-performance 3D circular buffer implementation for local obstacle mapping and collision checking and comparison with alternative methods.
- System design and evaluation on realistic simulator and real hardware, in addition to performance comparison with existing methods.

In addition to analyzing the results presented in the paper, we encourage the reader to watch the demonstration video and inspect the available code, which can be found at

[https://vision.in.tum.de/research/  
robotvision/replanning](https://vision.in.tum.de/research/robotvision/replanning)

## II. RELATED WORK

In this section, we describe the studies relevant to different aspects of collision-free trajectory generation. First, we discuss existing trajectory generation strategies and their applications to MAV motion planning. Thereafter, we discuss the state-of-the-art approaches for 3D environment mapping.

### A. Trajectory generation

Trajectory generation strategies can be subdivided into three main approaches: search-based path planning followed by smoothing, optimization-based approaches and motion-primitive-based approaches.

In search-based approaches, first, a non-smooth path is constructed on a graph that represents the environment. This graph can be a fully connected grid as in [6] and [11], or be computed using a sampling-based planner (RRT, PRM) as in [21] and [3]. Thereafter, a smooth trajectory represented by a polynomial, B-spline or discrete set of points is computed to closely follow this path. This class of approaches is currently the most popular choice for large-scale path planning problems in cluttered environments where a map is available a priori.

Optimization-based approaches rely on minimizing a cost function that consists of smoothness and collision terms. The trajectory itself can be represented as a set of discrete points [25] or polynomial segments [17]. The approach presented in the present work falls into this category, but represents a trajectory using uniform B-splines.

Another group of approaches is based on path sampling and motion primitives. Sampling-based approaches were successfully used for challenging tasks such as ball juggling [15], and motion primitives were successfully applied to flight through the forest [19], but the ability of both approaches

to find a feasible trajectory depends largely on the selected discretization scheme.

### B. Environment representation

To be able to plan a collision-free trajectory a representation of the environment that stores information about occupancy is required. The simplest solution that can be used in the 3D case is a voxel grid. In this representation, a volume is subdivided into regular grid of smaller sub-volumes (voxels), where each voxel stores information about its occupancy. The main drawback of this approach is its large memory-footprint, which allows for mapping only small fixed-size volumes. The advantage, however, is very fast constant time access to any element.

To deal with the memory limitation, octree-based representations of the environment are used in [9] [22]. They store information in an efficient way by pruning the leaves of the trees that contain the same information, but the access times for each element become logarithmic in the number of nodes, instead of the constant time as in the voxel-based approaches.

Another popular approach to environment mapping is voxel hashing, which was proposed by Nießner et al. [16] and used in [18]. It is mainly used for storing a truncated signed distance function representation of the environment. In this case, only a narrow band of measurements around the surface is inserted and only the memory required for that sub-volume is allocated. However, when full measurements must be inserted or the dense information must be stored the advantages of this approach compared to those of the other approaches are not significant.

## III. TRAJECTORY REPRESENTATION USING UNIFORM B-SPLINES

We use uniform B-spline representation for the trajectory function  $p(t)$ . Because, as shown in [14] and [1], the trajectory must be continuous up to the forth derivative of position (snap), we use quintic B-splines to ensure the required smoothness of the trajectory.

### A. Uniform B-splines

The value of a B-spline of degree  $k - 1$  can be evaluated using the following equation:

$$p(t) = \sum_{i=0}^n p_i B_{i,k}(t), \quad (1)$$

where  $p_i \in \mathbb{R}^n$  are control points at times  $t_i, i \in [0, \dots, n]$  and  $B_{i,k}(t)$  are basis functions that can be computed using the De Boor – Cox recursive formula [5] [4]. Uniform B-splines have a fixed time interval  $\Delta t$  between their control points, which simplifies computation of the basis functions.

In the case of quintic uniform B-splines, at time  $t \in [t_i, t_{i+1})$  the value of  $p(t)$  depends only on six control points, namely  $[t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}, t_{i+3}]$ . To simplify calculations we transform time to a uniform representation  $s(t) = (t - t_0)/\Delta t$ , such that the control points transform into  $s_i \in [0, \dots, n]$ . We define function  $u(t) = s(t) - s_i$  as time

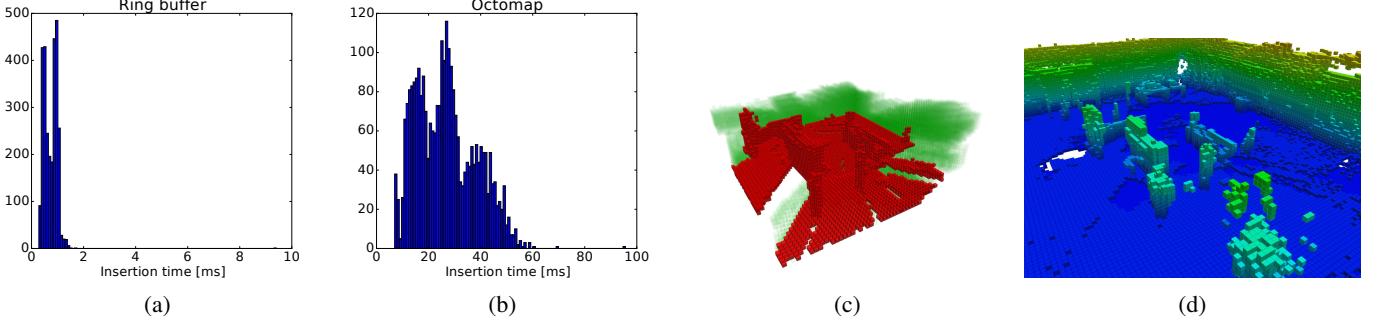


Fig. 2: Comparison between octomap and circular buffer for occupancy mapping on fr2/pioneer\_slam2 sequence of [23]. Being able to map only a local environment around the robot (3 m at voxel resolution of 0.1 m) circular buffer is more than an order of magnitude faster when inserting point cloud measurements from a depth map subsampled to a resolution of  $160 \times 120$ . Subplots (a) and (b) show the histograms of insertion time, and (c) and (d) show the qualitative results of the circular buffer (red: occupied, green:free) and the octomap, respectively.

elapsed since the start of the segment. Following the matrix representation of the De Boor – Cox formula [20], the value of the function can be evaluated as follows:

$$p(u(t)) = \begin{pmatrix} 1 \\ u \\ u^2 \\ u^3 \\ u^4 \\ u^5 \end{pmatrix}^T M_6 \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}, \quad (2)$$

$$M_6 = \frac{1}{5!} \begin{pmatrix} 1 & 26 & 66 & 26 & 1 & 0 \\ -5 & -50 & 0 & 50 & 5 & 0 \\ 10 & 20 & -60 & 20 & 10 & 0 \\ -10 & 20 & 0 & -20 & 10 & 0 \\ 5 & -20 & 30 & -20 & 5 & 0 \\ -1 & 5 & -10 & 10 & -5 & 1 \end{pmatrix}. \quad (3)$$

Given this formula, we can compute derivatives with respect to time (velocity, acceleration) as follows:

$$p'(u(t)) = \frac{1}{\Delta t} \begin{pmatrix} 0 \\ 1 \\ 2u \\ 3u^2 \\ 4u^3 \\ 5u^4 \end{pmatrix}^T M_6 \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}, \quad (4)$$

$$p''(u(t)) = \frac{1}{\Delta t^2} \begin{pmatrix} 0 \\ 0 \\ 2 \\ 6u \\ 12u^2 \\ 20u^3 \end{pmatrix}^T M_6 \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}. \quad (5)$$

The computation of other time derivatives and derivatives with respect to control points is also straightforward.

The integral over squared time derivatives can be computed in the closed form. For example, the integral over squared acceleration can be computed as follows:

$$E_q = \int_{t_i}^{t_{i+1}} p''(u(t))^2 dt \quad (6)$$

$$= \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}^T M_6^T Q M_6 \begin{pmatrix} p_{i-2} \\ p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \\ p_{i+3} \end{pmatrix}, \quad (7)$$

where

$$Q = \frac{1}{\Delta t^3} \int_0^1 \begin{pmatrix} 0 \\ 0 \\ 2 \\ 6u \\ 12u^2 \\ 20u^3 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 2 \\ 6u \\ 12u^2 \\ 20u^3 \end{pmatrix}^T du \quad (9)$$

$$= \frac{1}{\Delta t^3} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 8 & 12 & 16 & 20 \\ 0 & 0 & 12 & 24 & 36 & 48 \\ 0 & 0 & 16 & 36 & 57.6 & 80 \\ 0 & 0 & 20 & 48 & 80 & 114.286 \end{pmatrix}. \quad (10)$$

Please note that matrix  $Q$  is constant in the case of uniform B-splines. Therefore, it can be computed in advance for determining the integral over any squared derivative (see [21] for details).

### B. Comparison with polynomial trajectory representation

In this subsection, we discuss the advantages and disadvantages of B-spline trajectory representation compared to polynomial-splines-based representation [21] [17].

To obtain a trajectory that is continuous up to the forth derivative of position, we need to use B-splines of degree five

or greater and polynomial splines of at least degree nine (we need to set five boundary constraints on each endpoint of the segment). Furthermore, for polynomial splines we must explicitly include boundary constraints into optimization, while the use of B-splines guarantees the generation of a smooth trajectory for an arbitrary set of control points. Another useful property of B-splines is the locality of trajectory changes due to changes in the control points, which means that a change in one control point affects only a few segments in the entire trajectory. All these properties result in faster optimization because there are fewer variables to optimize and fewer constraints.

Evaluation of position at a given time, derivatives with respect to time (velocity, acceleration, jerk, snap), and integrals over squared time derivatives are similar for both cases because closed-form solutions are available for both cases.

The drawback of B-splines, however, is the fact that the trajectory does not pass through the control points. This makes it difficult to enforce boundary constraints. The only constraint we can enforce is a static one (all time derivatives are zero), which can be achieved by inserting the same control point  $k + 1$  times, where  $k$  is the degree of the B-spline. If we need to set an endpoint of the trajectory to have a non-zero time derivative, an iterative optimization algorithm must be used.

These properties make polynomial splines more suitable for the cases where the control points come from planning algorithms (RRT, PRM), which means that the trajectory must pass through them, else the path cannot be guaranteed to be collision-free. For local replanning, which must account for unmodeled obstacles, this property is not very important; thus, the use of B-spline trajectory representation is a better option.

#### IV. LOCAL ENVIRONMENT MAP USING 3D CIRCULAR BUFFER

To avoid obstacles during flight, we need to maintain an occupancy model of the environment. On one hand, the model should rely on the most recent sensor measurements, and on the other hand it should store some information over time because the field of view of the sensors mounted on the MAV is usually limited.

We argue that for local trajectory replanning a simple solution with a robocentric 3D circular buffer is beneficial. In what follows, we discuss details pertaining to implementation and advantages from the application viewpoint.

##### A. Addressing

To enable addressing we discretize the volume into voxels of size  $r$ . This gives us a mapping from point  $p$  in 3D space to an integer valued index  $x$  that identifies a particular voxel, and the inverse operation that given an index outputs its center point.

A circular buffer consists of a continuous array of size  $N$  and an offset index  $o$  that defines the location of the coordinate system of the volume. With this information, we can define the functions to check whether a voxel is in the volume and

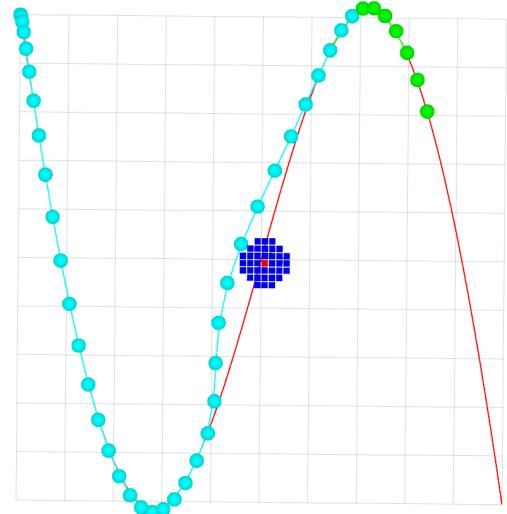


Fig. 3: Example of online trajectory replanning using proposed optimization objective. The plot shows a global trajectory computed by fitting a polynomial spline through fixed waypoints (red), voxels within 0.5 m of the obstacle (blue), computed B-spline trajectory with fixed (cyan) and still optimized (green) segments and control points. In the areas with no obstacles, the computed trajectory closely follows the global one, while close to an obstacle, the proposed method generates a smooth trajectory that avoids the obstacle, and rejoins the global trajectory.

find its address in the stored array:

$$\text{insideVolume}(x) = 0 \leq x - o < N, \quad (11)$$

$$\text{address}(x) = (x - o) \mod N. \quad (12)$$

If we restrict the size of the array to  $N = 2^p$ , we can rewrite these functions to use cheap bitwise operations instead of divisions:

$$\text{insideVolume}(x) = !((x - o) \& (\sim(2^p - 1))), \quad (13)$$

$$\text{address}(x) = (x - o) \& (2^p - 1). \quad (14)$$

where  $\&$  is a "bitwise and,"  $\sim$  is a "bitwise negation," and  $!$  is a "boolean not."

To ensure that the volume is centered around the camera, we must simply change the offset  $o$  and clear the updated part of the volume. This eliminates the need to copy large amounts of data when the robot moves.

##### B. Measurement insertion

We assume that the measurements are performed using range sensors, such as Lidar, RGB-D cameras, and stereo cameras, and can be inserted into the occupancy buffer by using raycast operations.

We use an additional flag buffer to store a set of voxels affected by insertion. First, we iterate over all points in our measurements, and for the points that lie inside the volume, we mark the corresponding voxels as occupied. For the points



Fig. 4: Real-world experiment performed outdoors. The drone (AscTec Neo) equipped with RGB-D camera (Intel Realsense R200) is shown in (a). In the experiment, the global path is set to a straight line with the goal position 30 m ahead of the drone, and trees act as unmapped obstacles that the drone must avoid. Side view of the scene is shown in (b), and visualization with the planned trajectory is shown in (c).

that lie outside the volume, we compute the closest point inside the volume and mark the corresponding voxels as free rays. Second, we iterate over all marked voxels and perform raycasting toward the sensor origin. We use a 3D variant of *Bresenham's line algorithm* [2] to increase the efficiency of the raycasting operation.

Thereafter, we again iterate over the volume and update the volume elements by using the hit and miss probabilities, similarly to the approach described in [19].

### C. Distance map computation

To facilitate fast collision checking for the trajectory, we compute the Euclidean distance transform (EDT) of the occupancy volume. This way, a drone approximated by the bounding sphere can be checked for collision by one look-up query. We use an efficient  $O(n)$  algorithm written by Felzenszwalb and Huttenlocher [7] to compute EDT of the volume, where  $n = N^3$  is the number of voxels in the grid (the complexity is cubic in the size of the volume along a single axis). For querying distance and computing gradient, trilinear interpolation is used.

## V. TRAJECTORY OPTIMIZATION

The local replanning problem is represented as an optimization of the following cost function:

$$E_{total} = E_{ep} + E_c + E_q + E_l, \quad (15)$$

where  $E_{ep}$  is an endpoint cost function that penalizes position and velocity deviations at the end of the optimized trajectory segment from the desired values that usually come from the global trajectory;  $E_c$  is a collision cost function;  $E_q$  is the cost of the integral over the squared derivatives (acceleration, jerk, snap); and  $E_l$  is a soft limit on the norm of time derivatives (velocity, acceleration, jerk and snap) over the trajectory.

### A. Endpoint cost function

The purpose of the endpoint cost function is to keep the local trajectory close to the global one. This is achieved by

penalizing position and velocity deviation at the end of the optimized trajectory segment from the desired values that come from the global trajectory. Because the property is formulated as a soft constraint, the targeted values might not be achieved, for example, because of obstacles blocking the path. The function is defined as follows:

$$E_{ep} = \lambda_p(p(t_{ep}) - p_{ep})^2 + \lambda_v(p'(t_{ep}) - p'_{ep})^2, \quad (16)$$

where  $t_{ep}$  is the end time of the segment,  $p(t)$  is the trajectory to be optimized,  $p_{ep}$  and  $p'_{ep}$  are the desired position and velocity, respectively,  $\lambda_p$  and  $\lambda_v$  are the weighting parameters.

### B. Collision cost function

Collision cost penalizes the trajectory points that are within the threshold distance  $\tau$  to the obstacles. The cost function is computed as the following line integral:

$$E_c = \lambda_c \int_{t_{min}}^{t_{max}} c(p(t)) \|p'(t)\| dt, \quad (17)$$

where the cost function for every point  $c(x)$  is defined as follows:

$$c(x) = \begin{cases} \frac{1}{2\tau}(d(x) - \tau)^2 & \text{if } d(x) \leq \tau \\ 0 & \text{if } d(x) > \tau, \end{cases} \quad (18)$$

where  $\tau$  is the distance threshold,  $d(x)$  is the distance to the nearest obstacle, and  $\lambda_c$  is a weighting parameter. The line integral is computed using the rectangle method, and distances to the obstacles are obtained from the precomputed EDT by using trilinear interpolation.

### C. Quadratic derivative cost function

Quadratic derivative cost is used to penalize the integral over square derivatives of the trajectory (acceleration, jerk, and snap), and is defined as follows:

$$E_{ep} = \sum_{i=2}^4 \int_{t_{min}}^{t_{max}} \lambda_{qi}(p^{(i)}(t))^2 dt. \quad (19)$$

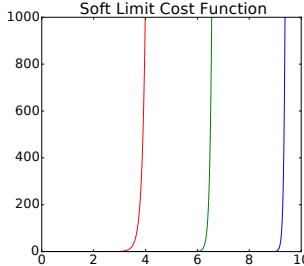


Fig. 5: Soft limit cost function  $l(x)$  proposed in Section V-D for  $p_{max}$  equals three (red), six (green), and nine (blue). This function acts as a soft limit on the time derivatives of the trajectory (velocity, acceleration, jerk, and snap) to ensure they are bounded and are feasible to execute by the MAV.

The above function has a closed-form solution for trajectory segments represented using B-splines.

#### D. Derivative limit cost function

To make sure that the computed trajectory is feasible, we must ensure that velocity, acceleration and higher derivatives of position remain bounded. This requirement can be included into the optimization as a constraint  $\forall t : p^{(k)}(t) < p_{max}^k$ , but in the proposed approach, we formulate it as a soft constraint by using the following function:

$$E_{ep} = \sum_{i=2}^4 \int_{t_{min}}^{t_{max}} l(p^{(i)}(t)) dt, \quad (20)$$

where  $l(x)$  is defined as follows:

$$l(x) = \begin{cases} \exp((p^{(k)}(x))^2 - (p_{max}^k)^2) - 1 & \text{if } p^{(k)}(x) > p_{max}^k \\ 0 & \text{if } p^{(k)}(x) \leq p_{max}^k \end{cases} \quad (21)$$

This allows us to minimize this cost function by using any algorithm designed for unconstrained optimization.

#### E. Implementation details

To run the local replanning algorithm on the drone, we first compute a global trajectory by using the approach described in [21]. This gives us a polynomial spline trajectory that avoids all mapped obstacles. Thereafter, we initialize our replanning algorithm with six fixed control points at the beginning of the global trajectory and  $C$  control points that need to be optimized.

In every iteration of the algorithm we set the endpoint constraints (Sec. V-A) to be the position and velocity at  $t_{ep}$  of the global trajectory. The collision cost (Sec. V-B) of the trajectory is evaluated using a circular buffer that contains measurements obtained using the RGB-D camera mounted on the drone. The weights of quadratic derivatives cost (Sec. V-C) are set to the same values as those used for global trajectory generation, and the limits (Sec. V-D) are set 20% higher to ensure that the global trajectory is followed with the appropriate velocity while laterally deviating from it.

Algorithm	Success Fraction	Mean Norm. Path Length	Mean Compute time [s]
Inf. RRT* + Poly	<b>0.9778</b>	<b>1.1946</b>	2.2965
RRT Connect + Poly	0.9444	1.6043	<b>0.5444</b>
CHOMP N = 10	0.3222	1.0162	0.0032
CHOMP N = 100	0.5000	1.0312	0.0312
CHOMP N = 500	0.3333	1.0721	0.5153
[17] S = 2 jerk	0.4889	1.1079	<b>0.0310</b>
[17] S = 3 vel	0.4778	1.1067	0.0793
[17] S = 3 jerk	0.5000	1.0996	0.0367
[17] S = 3 jerk + Restart	<b>0.6333</b>	1.1398	0.1724
[17] S = 3 snap + Restart	0.6222	1.1230	0.1573
[17] S = 3 snap	0.5000	<b>1.0733</b>	0.0379
[17] S = 4 jerk	0.5000	1.0917	0.0400
[17] S = 5 jerk	0.5000	1.0774	0.0745
Ours C = 2	0.4777	<b>1.0668</b>	<b>0.0008</b>
Ours C = 3	0.4777	1.0860	0.0011
Ours C = 4	0.4888	1.1104	0.0015
Ours C = 5	0.5111	1.1502	0.0021
Ours C = 6	0.5555	1.1866	0.0028
Ours C = 7	0.5222	1.2368	0.0038
Ours C = 8	0.4777	1.2589	0.0054
Ours C = 9	<b>0.5777</b>	1.3008	0.0072

TABLE I: Comparison of different path planning approaches. All results except those of the presented study are taken from [17]. Our approach performs similarly to approaches using polynomial splines without restarts, which indicates that B-splines can represent trajectories similar to those represented by polynomial splines. Lower computation times of our approach can be explained by the fact that unconstrained optimization occurs directly on the control points, unlike other approaches where the problem must first be transformed into an unconstrained form.

After optimization, the first control point from the points that were optimized is fixed and sent to the MAV position controller. Another control point is added to the end of the spline, which increases  $t_{ep}$  and moves the endpoint further along the global trajectory.

For optimization we use [10], which provides an interface to several optimization algorithms. We have tested the MMA [24] and BFGS [13] algorithms for optimization, with both algorithms yielding similar performance.

## VI. RESULTS

In this section, we present experimental results obtained using the proposed approach. First, we evaluate the mapping and the trajectory optimization components of the system separately for comparison with other approaches and justify their selection. Second, we evaluate the entire system in a realistic simulator in several different environments, and finally, present an evaluation of the system running on real hardware.

#### A. Three-dimensional circular buffer performance

We compare our implementation of the 3D circular buffer to the popular octree-based solution of [9]. Both approaches use the same resolution of 0.1 m. We insert the depth maps

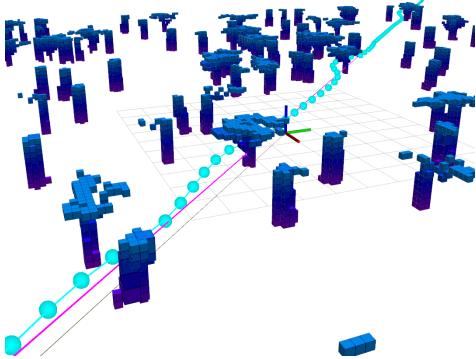


Fig. 6: Result of local trajectory replanning algorithm running in a simulator on the forest dataset. The global trajectory is visualized in purple, local trajectory is represented as a uniform quintic B-spline, and its control points are visualized in cyan. Ground-truth octomap forest model is shown for visualization purposes.

sub-sampled to the resolution of  $160 \times 120$ , which come from a real-world dataset [23]. The results (Fig. 2) show that insertion of the data is more than an order of magnitude faster with the circular buffer, but only a limited space can be mapped with this approach. Because we need the map of a bounded neighborhood around the drone for local replanning, this drawback is not significant for target application.

### B. Optimization performance

To evaluate the trajectory optimization we use the forest dataset from [17]. Each spline configuration is tested in 9 environments with 10 random start and end positions that are at least 4 m away from each other. Each environment is  $10 \times 10 \times 10 \text{ m}^3$  in size and is populated with trees with increasing density. The optimization is initialized with a straight line and after optimization, we check for collisions. For all the approaches, the success fraction, mean normalized path length, and computation time are reported (Table I).

The results of the proposed approach are similar in terms of success fraction to those achieved with polynomial splines from [17] without restarts, but the computation times with the proposed approach are significantly shorter. This is because the unconstrained optimization employed herein directly optimizes the control points, while in [17], a complicated procedure to transform a problem to the unconstrained optimization form [21] must be applied.

Another example of the proposed approach for trajectory optimization is shown in Figure 3, where a global trajectory is generated through a pre-defined set of points with an obstacle placed in the middle. The optimization is performed as described in Section V-E, with the collision threshold  $\tau$  set to 0.5 m. As can be seen in the plot, the local trajectory in the collision free regions aligns with the global one, but when an obstacle is encountered, a smooth trajectory is generated to avoid it and ensure that the MAV returns to the global trajectory.

Operation	Computing 3D points	Moving volume	Inserting measurements	SDF computation	Trajectory optimization
Time [ms]	0.265	0.025	0.518	9.913	3.424

TABLE II: Mean computation time for operations involved in trajectory replanning in the simulation experiment with depth map measurements sub-sampled to  $160 \times 120$  and seven control points optimized.

### C. System simulation

To further evaluate our approach, we perform a realistic simulation experiment by using the Rotors simulator [8]. The main source of observations of the obstacles is a simulated RGB-D camera that produces VGA depth maps at 20 FPS. To control the MAV, we use the controller developed by Lee et al. [12], which is provided with the simulator and modified to receive trajectory messages as control points for uniform B-splines. When there are no new commands with control points, the last available control point is duplicated and inserted into the B-spline. This is useful from the viewpoint of failure case because when an MAV does not receive new control points, it will slowly stop at the last received control point.

We present the qualitative results of the simulations shown in Figures 1 and 6. The drone is initialized in free space and a global path through the world populated with obstacles is computed. In this case, the global path is computed to intersect the obstacles intentionally. The environment around the drone is mapped by inserting RGB-D measurements into the circular buffer, which is then used in the optimization procedure described above.

In all presented simulation experiments, the drone can compute a local trajectory that avoids collisions and keeps it close to the global path. The timings of the various operations involved in trajectory replanning are presented in Table II.

### D. Real-world experiments

We evaluate our system on a multicopter in several outdoor experiments (Fig. 4). In these experiments, the drone is initialized without prior knowledge of the map and the global path is set as a straight line with its endpoint in front of the drone 1 m above the ground. The drone is required to use onboard sensors to map the environment and follow the global path avoiding trees, which serve as obstacles.

We use the AscTec Neo platform equipped with a stereo camera for estimating drone motion and an RGB-D camera (Intel Realsense R200) for obstacle mapping. All computations are performed on the drone on a 2.1 GHz Intel i7 CPU.

In all presented experiments, the drone can successfully avoid the obstacles and reach the goal position. However, the robustness of the system is limited at the moment owing to the accuracy of available RGB-D cameras that can capture outdoor scenes.

## VII. CONCLUSION

In this paper, we presented an approach to real-time local trajectory replanning for MAVs. We assumed that the global trajectory computed by an offline algorithm is provided and formulated an optimization problem that replans the local trajectory to follow the global one while avoiding unmodeled obstacles.

We improved the optimization performance by representing the local trajectory with uniform B-splines, which allowed us to perform unconstrained optimization and reduce the number of optimized parameters.

For collision checking we used the well-known concept of circular buffer to map a fixed area around the MAV, which improved the insertion times by an order of magnitude compared to those achieved with an octree-based solution.

In addition, we presented an evaluation of the complete system and specific sub-systems in realistic simulations and on real hardware.

## ACKNOWLEDGMENTS

This work has been partially supported by grant CR 250/9-2 (Mapping on Demand) of German Research Foundation (DFG) and grant 608849 (EuRoC) of European Commission FP7 Program.

We also thank the authors of [17] for providing their dataset for evaluation of the presented method.

## REFERENCES

- [1] Markus W Achtelik, Simon Lynen, Stephan Weiss, Margarita Chli, and Roland Siegwart. Motion-and uncertainty-aware path planning for micro aerial vehicles. *Journal of Field Robotics*, 2014.
- [2] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics 87*, 1987.
- [3] Michael Burri, Helen Oleynikova, , Markus W. Achtelik, and Roland Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments. In *Intelligent Robots and Systems*, 2015.
- [4] Maurice G Cox. The numerical evaluation of B-splines. *IMA Journal of Applied Mathematics*, 1972.
- [5] Carl De Boor. On calculating with B-splines. *Journal of Approximation theory*, 1972.
- [6] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 2008.
- [7] Pedro F Felzenszwalb and Daniel P Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 2012.
- [8] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ROS). *Studies in Computational Intelligence*, 2016.
- [9] Armin Hornung, Kai Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.
- [10] Steven G. Johnson. The nlopt nonlinear-optimization package. URL <http://ab-initio.mit.edu/nlopt>.
- [11] Dongwon Jung and Panagiotis Tsiotras. On-line path generation for small unmanned aerial vehicles using B-spline path templates. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008.
- [12] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on SE(3). In *Conference on Decision and Control*, 2010.
- [13] Dong Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 1989.
- [14] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *International Conference on Robotics and Automation*, 2011.
- [15] Mark Mueller, Markus Hehn, and Raffaello D’Andrea. A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification. In *Intelligent Robots and Systems*, 2013.
- [16] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *Transactions on Graphics*, 2013.
- [17] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online UAV replanning. In *International Conference on Intelligent Robots and Systems*, 2016.
- [18] Helen Oleynikova, Zachary Taylor, Marius Fehr, Juan Nieto, and Roland Siegwart. Voxblox: Building 3d signed distance fields for planning. *arXiv preprint arXiv:1611.03631*, 2016.
- [19] Aditya Paranjape, Kevin C Meier, Xichen Shi, Soon-Jo Chung, and Seth Hutchinson. Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research*, 2015.
- [20] Kaihuai Qin. General matrix representations for B-splines. In *Sixth Pacific Conference on Computer Graphics and Applications*, 1998.
- [21] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*. 2016.
- [22] Frank Steinbrücker, Jürgen Sturm, and Daniel Cremers. Volumetric 3D mapping in real-time on a cpu. In *IEEE Conference on Robotics and Automation*, 2014.
- [23] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intelligent Robots and Systems*, 2012.
- [24] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *Journal on Optimization*, 2002.
- [25] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. CHOMP: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 2013.