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

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

***Abstract*—In this work we present a real-time approach for local trajectory replanning for MAVs. Current trajectory generation methods for multicopters achieve high success rates in cluttered environments, but assume the environment is static and require prior knowledge of the map. In our work we utilize the results of such planners and extend them with local replanning algorithm that can handle unmodeled (possibly dynamic) obstacles while keeping MAV close to the global trajectory. To make our approach real-time capable we maintain information about the environment around MAV in an occupancy grid stored in 3D circular buffer that moves together with a drone, and represent the trajectories using uniform B-splines. This representation ensures that trajectory is sufficiently smooth and at the same time allows efficient optimization.**

I. Introduction

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

The formulation of trajectory generation problem largely depends on the application and assumptions about the environment. For the case where MAV has to navigate in cluttered, possibly indoor, environment we would suggest to subdivide the problem into two layers. At first, we assume that a map of the environment is available and a trajectory from a specified start point to the goal point should be planned in advance.

This task has been a popular research topic for the recent years, with several solutions proposed by Achtelik et al. [1] and Richter et al. [21]. They use occupancy representation of the environment to check for collisions and search for the valid path in visibility graph that is constructed by sampling based planners. After that, they follow the approach 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 kind can compute a trajectory through tens of waypoints in several seconds.

To cope with the fact that there might be unmodeled, possibly dynamic, obstacles a lower planning level is required. It should be able to generate a trajectory that keeps the MAV close to the global path and simultaneously avoids the unpredicted obstacles based on environment representation constructed from the most recent sensor measurements. This replanning level should run in several milliseconds in order to ensure safety of the MAV operating at high velocities.

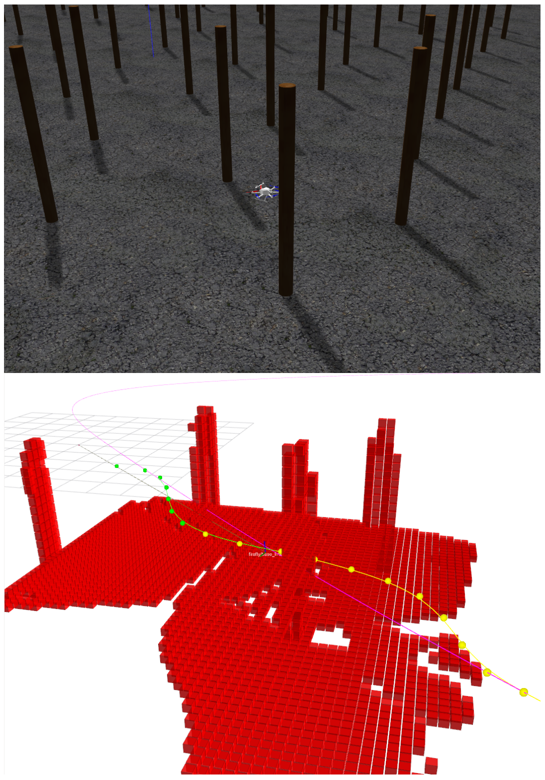
Fig. 1: Example of local trajectory replanning algorithm running in simulator. Global trajectory is visualized in purple and local obstacle map is visualized in red. Local trajectory, represented as a uniform quintic B-spline, and it's control points are visualized in yellow for the fixed parts and green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

The proposed approach solves a similar problem as the approach by Oleynikova et al. [17], but instead of using polynomial splines for representing the trajectory we propose to use B-splines, and discuss their advantages over polynomial splines for this particular task. Further, we propose to use a robocentric, fixed-size 3D circular buffer to maintain the local information about the environment. Even though, it cannot model arbitrarily large occupancy maps, as some octree implementations, faster look-up and measurement insertion operations make it better suited for the real-time replanning tasks.

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

The contributions of the work are as follows:

- Formulation of local trajectory replanning as 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 with performance comparison to existing methods.

Besides 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
```

II. Related Work

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

A. Trajectory generation

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

In search based approaches, first, a non-smooth path is constructed on the graph that represents the environment. This graph can be a fully connected grid as in [6] and [11], or computed by a sampling-based planner (RRT, PRM) as in [21] and [3]. After that, a smooth trajectory represented as 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 the large-scale path planning problems in cluttered environments where a map is a-priory available.

Optimization based approaches rely on minimizing a cost function that consist 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 this work also falls in 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 of ball juggling [15] and motion primitives were successfully applied for flying through the forest [19], but ability of both approaches to find a feasible trajectory largely depends on the chosen discretization.

B. Environment representation

To be able to plan a collision-free trajectory a representation of the environment that stores an information about occupancy is required. The simplest solution that can be used in 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 it's occupancy. The main drawback of this approach is a large memory-footprint, which allows to map only small fixed size volumes. The advantage, however, is a 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 leafs of the trees that contain the same information, but access times for each element become logarithmic in the number of nodes, instead of constant time for the voxel based approaches.

Another popular approach to map the environment is Voxel Hashing proposed by Nießner et al. [16] and used in [18]. It is mainly used for storing Truncated Signed Distance Function (TSDF) representation of the environment. In that 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 have to be inserted, or the dense information has to be stored the advantages of this approach compared to others are not significant.

III. Trajectory Representation using uniform B-splines

We use uniform B-spline representation for the trajectory function $p(t)$. Since, as shown in the works by Mellinger and Kumar [14] and Achtelik et al. [1], the trajectory has to be continuous up to 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 B-spline of degree $k-1$ can be evaluated using the following equation:

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

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

In the particular case of quintic uniform B-splines, for a time $t \in [t_i, t_{i+1})$ the value of $p(t)$ depends only on 6 control points at $[t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}, t_{i+3}]$. To simplify calculations we transform time to uniform representation $s(t) = (t - t_0)/\Delta t$, such that control points transform into $s_i \in [0, .., n]$. We define function $u(t) = s(t) - s_i$ to be a time since the start of the segment. Following the matrix representation of De Boor - Cox formula [20], the value of the function can be evaluated as follows:

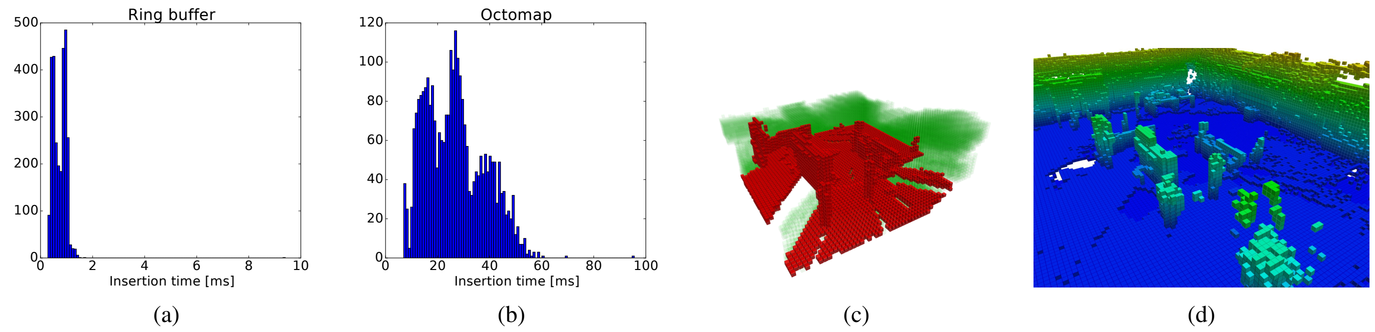

Fig. 2: Comparison between octomap and circular buffer for occupancy mapping on the fr2/pioneer_slam2 sequence of [23]. Being able to map only a local environment around the robot (3m at voxel resolution of 0.1m) circular buffer is more than an order of magnitude faster when inserting pointcloud measurements from depth map subsampled to 160×120 resolution. Subplots (a) and (b) show the histograms of insertion time, and (c) and (d) show qualitative results of circular buffer (red - occupied, green - free) and octomap respectively.

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

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

Given this formula, we can evaluate derivatives with respect to time (velocity, acceleration) in the following way:

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

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

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 closed form. For example, integral over squared acceleration can be computed as follows:

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

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

(8)

where

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

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

Please note that matrix Q for uniform B-spline is constant, so it can be pre-computed in advance for 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 the B-spline trajectory representation compared to the representation based on polynomial splines [21] [17].

To have a trajectory that is continuous up to forth derivative of position we need to use B-splines of degree 5 or greater and polynomial splines of at least degree 9 (we need to set 5 boundary constraints on each endpoint of the segment). Furthermore, for polynomial splines we have to explicitly include

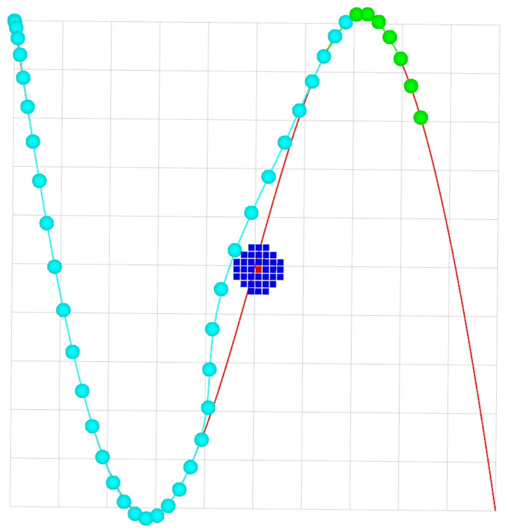

Fig. 3: Example of online trajectory replanning using the proposed optimization objective. Plot shows global trajectory computed by fitting a polynomial spline through the fixed waypoints (red), voxels with distance smaller than 0.5m to the obstacle (blue), computed B-spline trajectory with fixed (cyan)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

boundary constraints into optimization, while B-splines guarantee to generate smooth trajectory for arbitrary set of control points. Another useful property of B-splines is the locality of trajectory changes due to the changes in control points, which means that changes in one control point affect only few segments from the whole trajectory. All these properties result in faster optimization, since we have less variables to optimize and less constraints.

Evaluation of position at particular time, derivatives with respect to time (velocity, acceleration, jerk, snap), and integrals over squared time derivatives are similar for both cases, since closed form solutions for them exist.

The drawback of B-splines, however, is the fact that the trajectory does not pass through the control points. This makes it hard to enforce boundary constraints. In particular, 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 a degree of B-spline. If we need to set an endpoint of trajectory to have non-zero time derivative an iterative optimization algorithm has to be applied.

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

IV. Local Environment Map using 3D Circular Buffer

In order to avoid obstacles during the flight we need to maintain an occupancy model of the environment. On one hand, it should rely on the most recent sensor measurements, and on the other hand it should maintain some information over time, since 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 robocentric 3D circular buffer is beneficial. In the following we discuss implementation details and benefits for the application.

A. Addressing

To enable addressing we discretise 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 inverse operation, that given an index of the voxel outputs it's center point.

Circular buffer consists of continuous array of size N, and offset index o that defines the location of the coordinate system of the volume. With that, we can define the functions to check if a voxel is in the volume and find it's address in the stored array:

$$insideVolume(x) = 0 \leq x - o < N, \tag{11}$$
$$address(x) = (x - o) \mod N. \tag{12}$$

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

$$insideVolume(x) = \; !\,((x - o) \;\&\; (\sim(2^p - 1))), \tag{13}$$
$$address(x) = (x - o) \;\&\; (2^p - 1). \tag{14}$$

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

To keep the volume centered around the camera, we simply have to 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 come from the range sensors, like Lidars, RGB-D cameras or stereo cameras, and can be inserted into occupancy buffer 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 corresponding voxels as occupied. For the points that lie outside the volume we compute the closest point inside the volume and mark corresponding voxels as a free ray. Second, we iterate over all marked voxels, and perform raycasting towards the sensor origin. We use a 3D variant of *Bresenham's line algorithm* [2] to make the raycasting operation efficient.

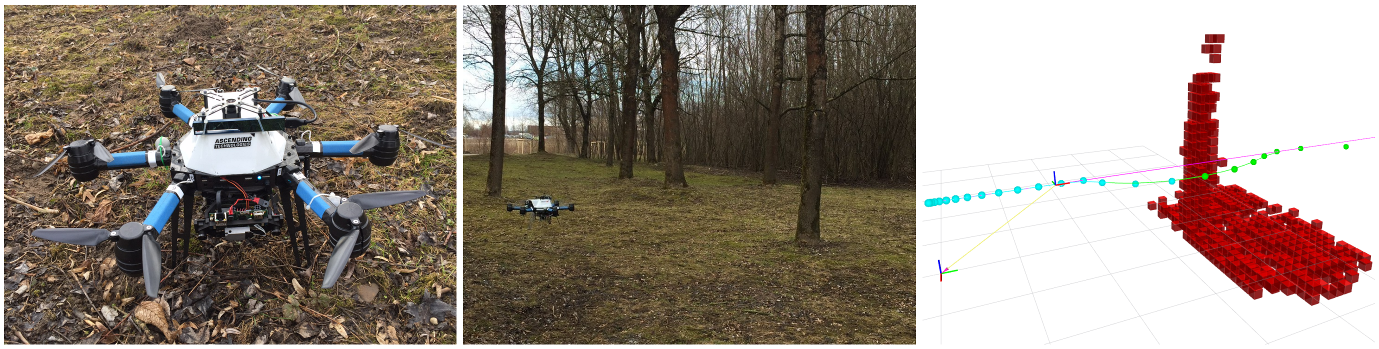
(a) (b) (c)

Fig. 4: Real world experiment performed in the outdoor environment. 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 goal position 30 meters ahead of the drone and trees act as unmapped obstacles that the drone has to avoid. Side view of the scene is shown in (b) and visualization with planned trajectory is shown in (c).

After that, we again iterate over the volume and update the volume elements using the hit and miss probabilities similar to the approach described in [9].

C. Distance Map Computation

In order to allow fast collision checking for the trajectory we compute an Euclidean Distance Transform (EDT) for the occupancy volume. This way, a drone approximated by the bounding sphere can be checked for collision in a single look-up querry. We utilize an efficient $O(n)$ algorithm by Felzenszwalb and Huttenlocher [7] to compute Euclidean Distance Transform for the volume. For querying distance and gradient computation a trilinear interpolation is used.

V. Trajectory Optimization

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

$$E_{total} = E_{ep} + E_c + E_q + E_l, \tag{15}$$

where E_{ep} is an endpoint cost function that penalizes deviation of position and velocity at the end of optimized trajectory segment from the desired values that usually come from the global trajectory; E_c is a collision cost function; E_q is a cost of integral over the squared derivatives (acceleration, jerk, snap); 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 Endpoint Cost Function is to keep the local trajectory close to the global one. This is achieved by penalizing the deviation of position and velocity at the end of the optimized trajectory segment from the desired values that come from global trajectory. Since the property is formulated as a soft constraint, the targeted values might not be achieved, for example, because of the obstacles blocking the path. The function is defined as:

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

where t_{ep} is an end time of the segment, $p(t)$ is the trajectory that we optimize, p_{ep} and p'_{ep} are the desired position and velocity and λ_p and λ_v are the weighting parameters.

B. Collision Cost Function

Collision cost penalizes the trajectory points that come closer than threshold τ to the obstacles. The cost function is computed as the following line integral:

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

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

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

where τ is a distance threshold, $d(x)$ is a distance to the nearest obstacle and λ_c is a weighting parameter.

C. Quadratic Derivative Cost Function

Quadratic derivative cost is penalizing an integral over square derivatives of the trajectory (acceleration, jerk and snap). It is defined as follows:

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

and has a closed form solution for trajectory segments represented as B-splines.

D. Derivative Limit Cost Function

In order to make sure that computed trajectory is feasible we have to ensure that velocity, acceleration and higher derivatives of position stay bounded. It can be included into optimization as a constraint $\forall t : p^{(k)}(t) < p^k_{max}$, but in our approach we formulate it as a soft constraint using the following function:

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

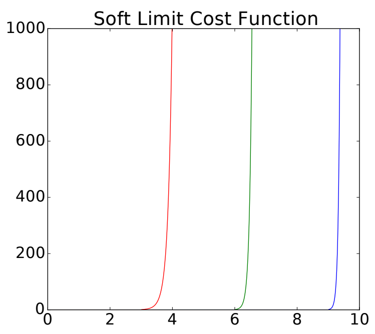

Fig. 5: Soft limit cost function $l(x)$ proposed in Section V-D for p_{max} equals 3 (red), 6 (green) and 9 (blue). This function acts as a soft limit on the time derivatives of the trajectory (velocity, acceleration, jerk, snap) to ensure they are bounded and feasible for the MAV to execute.

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

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

This allows us to use any algorithm designed for unconstrained optimization for minimizing this cost function.

E. Implementation Details

To run the local replanning algorithm on the drone, we first compute a global trajectory using the approach described in [21]. This gives us a polynomial spline trajectory that avoids all mapped obstacles. After that, we initialize our replanning algorithm with 6 control points at the beginning of the global trajectory and C control points that need to be optimized.

At 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. Collision cost (Sec. V-B) for the trajectory is evaluated using a circular buffer that contains the measurements from the RGB-D camera mounted on the drone. Weights for quadratic derivatives cost (Sec. V-C) are set to the same values as used for global trajectory generation and limits (Sec. V-D) are set 20% higher to be able to follow the global trajectory at the right velocity while laterally deviating from it.

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 the 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 MMA [24] and BFGS [13] algorithms for optimization, with both of them showing similar performance.

VI. Results

In this section we present experimental results of the proposed approach. First, we evaluate mapping and trajectory optimization components of the system separately to compare with other approaches and justify their selection. Second, we evaluate the whole system in the realistic simulator with several different environments, and at last, we present an evaluation of the system running on the real hardware.

Algorithm	**Success Fraction**	**Mean Norm. Path Length**	**Mean Compute time** [s]
Inf. RRT* + Poly	**0.9778**	**1.1946**	2.2965
RRT Connect + Poly	0.9444	1.6043	**0.5444**
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	**0.0310**
[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	**0.6333**	1.1398	0.1724
[17] S = 3 snap + Restart	0.6222	1.1230	0.1573
[17] S = 3 snap	0.5000	**1.0733**	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	**1.0668**	**0.0008**
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	**0.5777**	1.3008	0.0072

TABLE I: A table showing a comparison of different path planning approaches. All results except of ours are taken from [17]. Our approach performs similar to polynomial splines without restarts, which indicates that B-splines can represent similar trajectories as polynomial splines. Lower computation times of our approach can be explained by the fact that unconstrained optimization happens directly on the control points, unlike other approaches where the problem first has to be transformed to unconstrained form.

A. 3D 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 0.1m. We insert the depth maps sub-sampled to the resolution 160×120, which come from a real-world dataset [23]. The results (Fig. 2) show that insertion of the data is more than a magnitude faster for the circular buffer, but only a limited space can be mapped with this approach. Since for local replanning we need the map of a bounded neighborhood around the drone, this drawback is not significant for the target application.

B. Optimization performance

To evaluate the trajectory optimization we use a forest dataset from [17]. Each spline configuration is tested on 9 environments with 10 random start and end positions at least 4m away from each other. Each environment is $10 \times 10 \times 10m$ in size and populated with trees with increasing density. The optimization is initialized with straight line and after

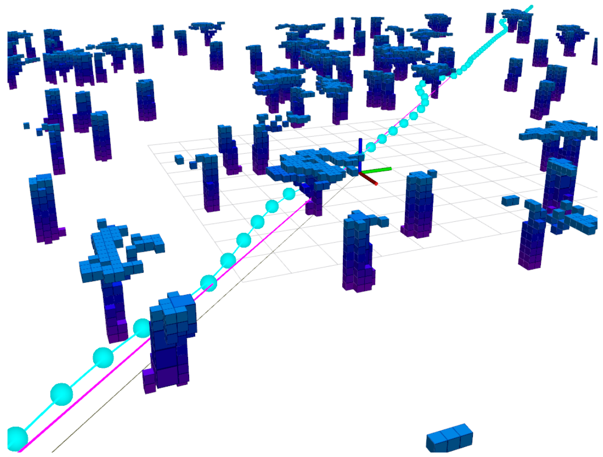

Fig. 6: Result of local trajectory replanning algorithm running in simulator on the forest dataset. Global trajectory is visualized in purple, local trajectory, represented as a uniform quintic B-spline, and it's control points are visualized in cyan. Ground truth octomap forest model is shown for visualization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

optimization is checked for collisions. For all the approaches success fraction, mean normalized length of the path and computation time is reported (Table I).

The results of the proposed approach is similar in success fraction to the polynomial splines from [17] without restarts, but have much faster computation times. This is due to unconstrained optimization involved that directly optimizes the control points, while in [17] a complicated procedure to transform problem to the unconstrained optimization form [21] has to be applied.

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

C. System Simulation

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

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 7 control points optimized.

We present qualitative results of the simulation in Figure 1 and 6. The drone is initialized in the free space and a global path through the world populated with obstacles is computed. In this case, the global path is intentionally computed to intersect the obstacles. 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 was able to compute local trajectory that avoids collisions and keeps it close to the global path. The timing for different operations involved in trajectory replanning is presented in Table II.

D. Real World Experiments

We also evaluate our system on a multicopter in several outdoor experiments (Fig. 4). In the experiment, the drone is initialized without prior knowledge of the map and with a global path set as a straight line with the end point in front of the drone 1m above the ground. The drone has to use onboard sensors to map the environment and follow the global path avoiding trees that act as obstacles.

Our platform is AscTec Neo equipped with stereo camera for estimating the motion of the drone and RGB-D camera (Intel Realsense R200) for mapping the obstacles. All computations are performed on the drone on 2.1 GHz Intel i7 CPU.

In all presented experiments the drone was able to successfully avoid the obstacles and reach the goal position, however we have to point out that robustness of the system is at the moment limited due to the accuracy of available RGB-D cameras that are able to capture outdoor scenes.

VII. Conclusion

In this work we have presented an approach for real-time local trajectory replanning for MAVs. We assume that a global trajectory computed by the off-line algorithm is provided and formulate an optimization problem that replans a local trajectory to follow the global one and simultaneously avoid unmodelled obstacles.

We improve the optimization performance by representing local trajectory using uniform B-splines, which allow us to perform an unconstrained optimization and reduce the number of optimized parameters.

For collision checking we utilize a well known concept of circular buffer to map a fixed area around the MAV with a magnitude faster measurement insertion times than an octree based solution.

We also present an evaluation of the complete system and particular 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*, 31(4):676–698, 2014.

[2] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *In Eurographics 87*, pages 3–10, 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 (IROS 2015)*.

[4] Maurice G Cox. The numerical evaluation of b-splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.

[5] Carl De Boor. On calculating with b-splines. *Journal of Approximation theory*, 6(1):50–62, 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*, 8:415–428, 2012.

[8] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ros). *Studies Comp.Intelligence Volume Number:625*, The Complete Reference (Volume 1)(978-3-319-26052-5):Chapter 23, 2016. ISBN:978-3-319-26052-5.

[9] Armin Hornung, Kai M. 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*, page 7135.

[12] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5420–5425. IEEE, 2010.

[13] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

[14] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, May 2011. doi: 10.1109/ICRA.2011.5980409.

[15] Mark W 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 (IROS)*.

[16] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6):169, 2013.

[17] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 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 A 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*, 34(3):357–377, 2015.

[20] Kaihuai Qin. General matrix representations for b-splines. In *Sixth Pacific Conference on Computer Graphics and Applications, 1998.*, Oct 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 *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2021–2028. IEEE, 2014.

[23] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[24] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM journal on optimization*, 12 (2):555–573, 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.