2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Continuous-Time Trajectory Optimization for Online UAV Replanning

Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart and Enric Galceran
Autonomous Systems Lab, ETH Zürich

***Abstract*— Multirotor unmanned aerial vehicles (UAVs) are rapidly gaining popularity for many applications. However, safe operation in partially unknown, unstructured environments remains an open question. In this paper, we present a continuous-time trajectory optimization method for real-time collision avoidance on multirotor UAVs. We then propose a system where this motion planning method is used as a local replanner, that runs at a high rate to continuously recompute safe trajectories as the robot gains information about its environment. We validate our approach by comparing against existing methods and demonstrate the complete system avoiding obstacles on a multirotor UAV platform.**

I. Introduction

Multirotor UAVs are gaining wide acceptance not only as research platforms, but also for use in various real-world applications. Despite recent progress in on-board state estimation, planning, and control, many current UAV systems still require either an empty environment or perfect knowledge of one *a priori*. This limits their safety and utility in unstructured, unknown environments.

In this paper, we focus on the problem of planning safe avoidance trajectories for a multirotor helicopter (multicopter) in partially known or unknown environments. For example, use cases such as high-speed forest flight require low-latency motion planning, as these environments are often densely populated and obstacles frequently occlude one another [1].

Motion primitive methods have been a common choice for online replanning on fixed-wing and multirotor platforms, since they can be executed quickly and each "primitive" can be constructed to be dynamically feasible [2] [3]. However, such methods require discretizing the state-space, which requires a huge motion library or having the controller track from nearest start state, which [3] cites as being responsible for up to 20% of the failures of their fixed-wing collision avoidance system. For our approach, we overcome the need for state-space or time discretization by choosing a continuous-time basis function to express our trajectories, and plan from arbitrary points in the state space to allow greater flexibility for online replanning.

We draw inspiration from trajectory optimization methods, such as CHOMP [4], which locally minimize collision and smoothness costs on a discrete-time trajectory. These planners are most commonly used for solving manipulation problems, where most of the constraints are kinematic rather than dynamic. Kinematic constraints can be simply expressed in discrete time, while dynamic constraints are more naturally suited to continuous-time representations (and avoid unnecessary numerical differentiation errors).

The research leading to these results has received funding from the European Communitys Seventh Framework Programme (FP7) under grant-agreement n.608849 (EuRoC) and n.644128, (AEROWORKS).

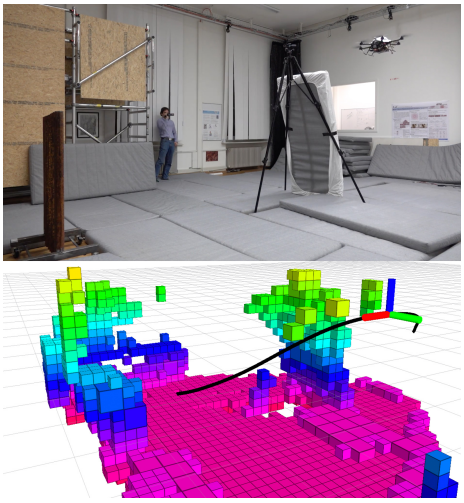


Fig. 1: Results of an experiment showing our online local replanning system. The UAV starts from behind the large mattress, which blocks its view of the iron obstacle (left) behind. The planning goal is set inside the iron obstacle, but as soon as the UAV observes that it is occupied, it replans to stop in front (planned path seen in black). This shows the ability of our system to avoid newly-detected obstacles and find feasible goal positions online.

For UAV flight, it is advantageous to use continuous-time trajectory representations like polynomials [5]. These basis functions provide continuity up to a high derivative, fast evaluation at any given time, and a very small number of parameters are needed to describe even long and complex trajectories. By varying only one parameter (segment time), it is possible to ensure that these trajectories are dynamically feasible given a simplified model of multicopter dynamics.

We propose a method of continuous-time trajectory optimization that allows the real-time generation of safe avoidance trajectories. Our method uses a two-part objective function, minimizing both a derivative of position and cost of collision with the environment. We also show this method as integrated into a local replanning system, where we use this local trajectory optimization to modify an initial plan in the presence of previously unknown obstacles.

Our approach runs in real-time (under 50 ms for a complete planning cycle), produces continuous-time trajectories, and is able to plan from and to arbitrary states without a

need for discretization in the workspace or state-space.

The contributions of this work are as follows:

- A continuous-time polynomial formulation of a local trajectory optimization problem for collision avoidance, capable of running in real-time on a real multicopter.
- A complete system with this as local replanning component, which continuously computes collision-free trajectories around any newly detected obstacles.
- Evaluation against existing trajectory optimization and planning algorithms, and experiments on a real world platform (Fig. 1).

II. Related Work

3D path planning approaches for UAVs can be broadly classified into several categories, such as sampling-based methods (often followed by smoothing), trajectory optimization methods, and method based on motion primitives. Here we discuss motion planning methods related to our work, with emphasis in suitability for real-time replanning with a dynamically updating map.

Sampling-based planning followed by a smoothing step which ensures dynamic constraints are met is commonly used for 3D global planning on UAVs [6]. Approaches such as running RRT-based methods to generate a visibility graph, followed by fitting high-order polynomials through the waypoints (graph vertices) are shown to outperform traditional RRT methods in control space in terms of execution time [6]. Recent speed-ups to the polynomial optimization have also allowed such combined planners to run in almost real-time, taking only a few seconds to generate long global plans [7]. While these methods generally produce high-quality plans and are probabilistically complete, they are still too slow for some online applications like real-time avoidance.

Closest to our proposed approach are discrete-time trajectory optimization methods. CHOMP, a trajectory optimization-based motion planner that revived interest in this class of planner in recent years, uses a two-part objective function with a smoothness and collision cost, and performs gradient descent with positions of discrete waypoints as parameters [4]. In order to speed up convergence time to a feasible plan, and ensure smoothness of the final solution, each gradient descent step is multiplied by a Riemennian metric in order to ensure smooth, incremental updates. Also based on trajectory optimization, STOMP is a gradient-free method that samples candidate trajectories and minimizes a cost function by creating linear combinations of the best-scoring candidates [8]. A more recent advent in trajectory optimization for collision-free planning breaks up the workspace into free convex regions and performs Sequential Quadratic Programming (SQP) to converge to a solution faster than the previous two methods [9]. Unfortunately, this requires a pre-built map with pre-computed convex regions, which is difficult to achieve in real-time.

Another approach to finding a low-cost path is to cheaply generate many path candidates, and choose the best of the candidates based on an objective function. This has been done for finding good polynomial trajectories to enable quadrotor ball juggling [10], selecting locally lower-cost trajectories to track a global plan in rough terrain [11], and choosing the safest trajectories for autonomous vehicles in traffic [12]. However, these approaches rely on randomly-sampled trajectories finding collision-free paths, which is an assumption that may not hold in very cluttered environments.

An alternative is to solve the optimal control problem using mixed-integer programming, where the workspace is again broken up into convex regions and a global optimum including some linearized or simplified version of the system dynamics is found [13], [14]. These approaches generally give dynamically-feasible and collision-free trajectories, and it is even possible to make guarantees on their safety [15], [16], however require a map representation which is very costly to compute and generally have long runtimes (on the order of magnitude of minutes).

A class of methods commonly used for replanning are those based on motion primitives. The state-space of the UAV is discretized into a state lattice with motion primitives forming edges in the graph, and standard graph search algorithms such as A* and AD* are used to find a feasible solution through this graph. This has been shown in multicopters for navigating through partially known environments [2], [17] and on fixed-wing airplanes for navigating through a forest while always safely being able to perform an emergency turn-around maneuver [18]. Another work shows flying high-speed through a forest using only on-board vision and planning, picking collision-free next maneuvers from a motion library [3], where they cite insufficient richness of the motion primitive library and discretization in the start state as responsible for 50% of the experimental failures.

A drawback of these approaches is the need to discretize both the workspace and state space (for example, motion primitives can only be generated for a finite number of start velocities and end velocities), and the performance of the algorithm is tightly linked to how many motion primitives are generated. Although we do use a discretized workspace representation, our approach does not require such discretization, nor does it require discretization in time or state-space, giving the possibility of a wider range of solutions to be found.

III. Continuous-Time Trajectory Optimization Algorithm

Our approach focuses on optimizing high-degree polynomial trajectories made out of several segments, as inspired by [6]. The trajectory is essentially a high-order polynomial spline, with C^D continuity, where D is the derivative we attempt to minimize. These high-order splines are generally used for global trajectory generation, and have many advantages including the ability to specify velocities, accelerations, and lower derivatives at waypoints, very fast evaluation times, and compact representation of long and complex trajectories. While a closed-form solution exists to minimizing the sum of squared derivatives of such a spline,

we expand the problem to also contain information about the environment to generate a locally optimal safe trajectory.

A. Problem Formulation

Instead of considering the full dynamics of a multicopter, we follow the work of Mellinger and Kumar [5] to plan in a reduced space of *differentially flat outputs*. This allows us to plan only in $\mathbf{R}^3$ and handle yaw separately.

Therefore, we will consider a polynomial trajectory in K dimensions, with S segments, and each segment of order N. Each segment has K dimensions, each of which is described by an Nth order polynomial:

$$f_k(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \dots a_N t^N \tag{1}$$

with the polynomial coefficients:

$$\mathbf{p}_k = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_N \end{bmatrix}^\top. \tag{2}$$

Given this trajectory representation, we seek to find the set of coefficients $\mathbf{p}^*$ that minimize an objective function J. In our case, similar to CHOMP [4], our objective function has two components: a part that attempts to minimize a derivative D, J_d, and a part that attempts to minimize collisions with the environment, J_c.

$$\mathbf{p}^* = \underset{\mathbf{p}}{\operatorname{argmin}}\, w_d J_d + w_c J_c \tag{3}$$

The following sections will present our choices of objective costs, J_d and J_c, optimization method, and map representation to solve this problem in real-time.

B. Method

As described in [6], we express the polynomial not in terms of its $N+1$ coefficients, but in terms of its end-derivatives to allow us to pose the derivative minimization problem as an unconstrained quadratic program (QP), which is significantly faster to solve than the constrained dual of this problem.

We can map between polynomial coefficients and end-derivatives using the $\mathbf{A}$ matrix, and rearrange the end-derivatives into a free ($\mathbf{d}_P$) and fixed ($\mathbf{d}_F$) blocks using a mapping matrix $\mathbf{M}$:

$$\mathbf{p} = \mathbf{A}^{-1}\mathbf{M}\begin{bmatrix}\mathbf{d}_F \\ \mathbf{d}_P\end{bmatrix}. \tag{4}$$

The construction of matrices $\mathbf{A}$, $\mathbf{M}$ (and $\mathbf{R}$ below) is addressed in [6]. The fixed derivatives $\mathbf{d}_F$ are given from the fixed end-constraints, like start and end velocities and accelerations, while the free derivatives $\mathbf{d}_P$ are the parameters we optimize.

In order to incorporate costs from collisions with the environment, we use a minimization problem with two costs:

$$\mathbf{d}_P^* = \underset{\mathbf{d}_P}{\operatorname{argmin}}\; w_d J_d + w_c J_c \tag{5}$$

where J_d is the cost due to integrated squared derivative terms (if minimizing snap, integral of squared snap along the trajectory), J_c is the cost due to collisions, and w_d and w_c are the weighing terms for each part of the cost.

The objective J_d can be calculated via the following:

$$\begin{aligned} J_d &= \mathbf{d}_F^\top \mathbf{R}_{FF}\mathbf{d}_F + \mathbf{d}_F^\top \mathbf{R}_{FP}\mathbf{d}_P + \\ &\quad \mathbf{d}_P^\top \mathbf{R}_{PF}\mathbf{d}_F + \mathbf{d}_P^\top \mathbf{R}_{PP}\mathbf{d}_P \end{aligned} \tag{6}$$

where $\mathbf{R}$ is the augmented cost matrix, and $\mathbf{R}_{XX}$ denotes the appropriate blocks within this matrix.

The Jacobian of J_d with respect to the parameter vector can be computed as follows:

$$\frac{\partial J_d}{\partial \mathbf{d}_P} = 2\mathbf{d}_F^\top \mathbf{R}_{FP} + 2\mathbf{d}_P^\top \mathbf{R}_{PP}. \tag{7}$$

The derivative costs are independent for each axis, and are accumulated over all K dimensions of the problem.

To represent collision costs, we use a line integral of the potential function $c(\mathbf{f}(t))$ over the arc length of the trajectory.

Since our environment is represented as a discrete voxel grid (see Section III-C), we need to sample the trajectory and test at least one point within each voxel along the trajectory.

To do so, we transform the trajectory from end-derivatives into workspace coordinates. For each axis k at a time t:

$$\mathbf{T} = [t^0, t^1, t^2, ..., t^N] \tag{8}$$

$$f_k(t) = \mathbf{T}\mathbf{p}_k \tag{9}$$

$$\mathbf{f}(t) = \begin{bmatrix} f_x(t) & f_y(t) & \dots \end{bmatrix} \tag{10}$$

We also compute the velocity at each time t, using a matrix $\mathbf{V}$, which maps a vector of polynomial coefficients of a function to the polynomial coefficients of its derivative.

$$v_k(t) = \dot{f}_k(t) = \mathbf{T}\mathbf{V}\mathbf{p}_k \tag{11}$$

$$\mathbf{v}(t) = \begin{bmatrix} v_x(t) & v_y(t) & \dots \end{bmatrix} \tag{12}$$

The collision cost is then the line integral below, integrated over each segment m (where t_m is the end time of the segment):

$$\begin{aligned} J_c &= \int_S c(\mathbf{f}(t))ds \\ &= \int_{t=0}^{t_m} c(\mathbf{f}(t)) \left\| \dot{\mathbf{f}}(t) \right\| dt \\ &= \sum_{t=0}^{t_m} c(\mathbf{f}(t)) \left\| \mathbf{v}(t) \right\| \Delta t \end{aligned} \tag{13}$$

where $c(\mathbf{f}(t))$ is the potential cost described in [4].

Finally, using the product and chain rules, we obtain the Jacobian for each axis k:

$$\begin{aligned} \frac{\partial J_c}{\partial \mathbf{d}_{Pk}} &= \sum_{t=0}^{t_m} \|\mathbf{v}(t)\|\ \nabla_k c\, \mathbf{T}\mathbf{L}_{PP}\Delta t + \\ &\quad c(\mathbf{f}(t)) \frac{v_k(t)}{\|\mathbf{v}(t)\|} \mathbf{T}\mathbf{V}\mathbf{L}_{PP}\Delta t. \end{aligned} \tag{14}$$

Here $\mathbf{L} = \mathbf{A}^{-1}\mathbf{M}$, or the complete mapping matrix between end-derivatives and polynomial coefficients. $\mathbf{L}_{PP}$ refers to the block of the right-side columns of the matrix, corresponding to the columns which operate on the free parameters $\mathbf{d}_P$.

We use a heuristic to estimate the segment times, t_m, to meet dynamic constraints and we hold these times fixed during the optimization.

C. Map Representation

Map representation and choice of potential cost function is central to the algorithm described above. Naturally, the potential cost function must be smooth, but its gradient must also be able to push trajectories out of collision. We use the potential described in [4], which is a function of an Euclidean Signed Distance Field (ESDF) value, $d(\mathbf{x})$, at a point in 3D space, $\mathbf{x}$, and ϵ is a constant value specifying the obstacle clearance past which space is considered free.

$$c(\mathbf{x}) = \begin{cases} -d(\mathbf{x}) + \frac{1}{2}\epsilon & \text{if } d(\mathbf{x}) < 0 \\ \frac{1}{2\epsilon}(d(\mathbf{x}) - \epsilon)^2 & \text{if } 0 \leq d(\mathbf{x}) \leq \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

We use a voxel-based map representation, as they can be built and maintained quickly. To ensure that a trajectory does not collide with the environment, we must check each voxel along the trajectory. Note that our continuous-time approach still presents advantages over discrete-time methods, as we are flexible in how often we sample the trajectory for collisions, and can change this interval between iterations. We choose to evaluate the function along every arc length point Δs equal to the map voxel resolution. This significantly speeds up computation without compromising safety.

D. Optimization

In any nontrivial environment, the optimization problem in (5) is likely to be non-convex and highly nonlinear. For minimizing the function, we choose to use a quasi-Newton method like BFGS [19] (though other simpler methods like gradient descent can also be used).

However, all solutions found with such methods are inherently *local* solutions – and they are prone of falling into local minima depending on the initialization. Therefore, in order to increase the chances of finding a feasible (non-colliding) solution, we do several random restarts where we perturb the initial state by a random quantity, and then select the lowest-cost trajectory as the final solution. A more thorough discussion on the necessity of random restarts for local trajectory optimization is offered in [9].

IV. Replanning System

In this section, we introduce the complete system that makes it possible to run our local replanning method online, in real-time on dynamically updating map data. First, we introduce how to build the map and its companion distance field, then we discuss using a global plan as input into the local replanner, and finally, how to select start and end points for replanning.

A. Incremental Mapping

As mentioned in Section III-C, we require an Euclidean Signed Distance Field (ESDF) to compute the collision potentials. Our map representation is an Octomap [20], which contains voxels in one of three states: free, unknown, or occupied.

When first constructing the map, we fill in the occupancy of all cells and compute the distances for the complete map, which is computationally expensive. In order to allow our algorithm to run in real-time, we track changed nodes in our Octomap representation and invalidate all voxels in the ESDF that have those nodes as parents (i.e., nearest neighbor of a different state). This allows us to recompute the distance values of only a few tens to hundred voxels per map update, instead of having to recompute the full dense grid of millions of voxels.

One important point about the map representation is that although Octomap allows three states (free, unknown, and occupied) with full probabilities, in order to construct a distance field, we must discretize to only two states — free and occupied. How to deal with unknown voxels is a question of safety: we cannot safely plan through them unless the robot is able to stop in known free space. Therefore, we choose to treat unknown as occupied, creating a very conservative planner. A more thorough discussion of this choice is offered in Section VI.

B. Global Planning

Next, we built a global plan to an end point in the original Octomap, while treating unknown space as free. This creates a high-level optimistic planner, while the local replanner is conservative and therefore safer. This plan will then be used as a prior for the replanner, and also allows us to use a replanner that is not complete – in case no solution is found by the local trajectory optimization, we simply stop and wait for the global planner to find a new path.

We use a 2-stage global planner: first, we find a topologically feasible straight-line path using Informed RRT* [21], and then we plan a dynamically feasible polynomial trajectory through it [7].

C. Local Replanning

To perform the local replanning, we start with the global plan (if available) or a straight-line plan to the next waypoint as prior, and incrementally update the ESDF.

We then select appropriate start and end points for the replanning algorithm. As start point, we choose the point on the current trajectory t_R seconds in the future, where t_R is the update rate of the replanner. Since our planner allows continuity and smoothness even in low derivatives, we are able to use the full state of the UAV at the start point, including velocity and acceleration, guaranteeing a smooth path even with changing plans.

The goal point is chosen as a point on the global trajectory that is h meters ahead of the start point, where h is a planning horizon. If unoccupied, we accept that point as the goal, otherwise we attempt to find the nearest unoccupied neighbor in the ESDF, and as a final fallback we shorten the planning horizon until a free goal point is found.

We can then run the local optimization procedure between these two points. Either the optimization succeeds in finding a collision-free path, or we attempt random restarts until either a collision-free trajectory is found or the vehicle stops and waits for the global planner to select a new path.

V. Experimental Results

In this section we first evaluate the proposed continuous-time local planner and compare it to existing planning algorithms. Then, we validate our complete system in both a long, realistic simulation scenario where the robot only has local information about the environment, and then in a real-world test on an UAV avoiding newly detected objects in a room.

A. Evaluation

We validate our approach as a local start-to-goal point planner in simulation on 100 random 2D forest environments.

To analyze how our algorithm behaves in different densities of clutter, we generate $5 \times 5\ m$ Poisson forests [1] of densities between $0.2\,trees/m^2$ to $0.8\,trees/m^2$. We then analyze the success fraction of our algorithm versus CHOMP [4], a discrete-time local optimization method.

We initialize both algorithms with a straight-line path between opposite corners of the map. For our continuous-time algorithm, we use between 1 and 5 segments of 11th order polynomials ($N = 11$) minimizing snap, and optionally use 10 random restarts. For CHOMP, we use a fixed N of 100 points and minimize velocity.

Fig. 2a shows typical paths generated by the algorithms. As can be seen, 1 segment does not have sufficient degrees of freedom to solve this planning problem, but 5 segments are able to find a short, smooth, feasible solution. CHOMP is also able to find a solution for this problem, but falls into a different local minima from our approach.

We can further analyze the behavior of the algorithms at different forest densities (number of obstacles in the environment), as seen in Fig. 2b. As the density of the environment increases, the chance of all methods finding a valid solution decreases. However, there is a large increase on success rate if random restarts are used, and a larger number of segments is able to handle denser environments (as in the case in Fig. 2a). Fig. 2c shows the effect of increasing the number of segments on success across all test cases.

For a more representative evaluation, we simulate arealistic 3D forest environment using real tree models, as shown in Fig. 3. The environments are $10 \times 10 \times 10\,m$, populated with a density of $0.2\ trees/m^2$. The trees are of random scale and height, adding the additional complexity of navigating in 3D and avoiding the tree canopies. We generate 9 such environments, and select 10 random start and goal points at least 4 meters apart, for a total of 90 test cases.

We evaluate several parameter settings of our algorithm and compare to CHOMP (which minimizes velocity), and sampling-based visibility graph search (RRT-based methods) with polynomial smoothing using 9th order minimum snap polynomials. For CHOMP and our method, since both feature a derivative cost term and a collision cost term, we use the same weights ($w_d = 0.1$, $w_c = 10$) to make as fair of a comparison as possible. Both algorithms are allowed to run for up to 50 iterations.

The results are shown in Table I. Though RRT-based algorithms with smoothing are clearly able to solve a larger number of problems, Informed RRT* takes too long to run in real-time at a high rate, and RRTConnect, while significantly faster, is still exceeding the time budget and producing much longer paths.

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
Ours $S = 2$ jerk	0.4889	1.1079	**0.0310**
Ours $S = 3$ vel	0.4778	1.1067	0.0793
Ours $S = 3$ jerk	0.5000	1.0996	0.0367
Ours $S = 3$ jerk + Restart	**0.6333**	1.1398	0.1724
Ours $S = 3$ snap + Restart	0.6222	1.1230	0.1573
Ours $S = 3$ snap	0.5000	**1.0733**	0.0379
Ours $S = 4$ jerk	0.5000	1.0917	0.0400
Ours $S = 5$ jerk	0.5000	1.0774	0.0745

TABLE I: A table showing comparison of RRT variants with polynomial smoothing, CHOMP, and our approach on a set of 90 forest planning problems, as shown in Fig. 3. We compare the success fraction, normalized path length (solution path length divided by straight-line path length), and computation time. As can be seen, adding random restarts significantly improves success fraction but at the cost of higher computation time. RRT* and RRT Connect are able to solve a higher percentage of problems, but at the cost of slower performance.

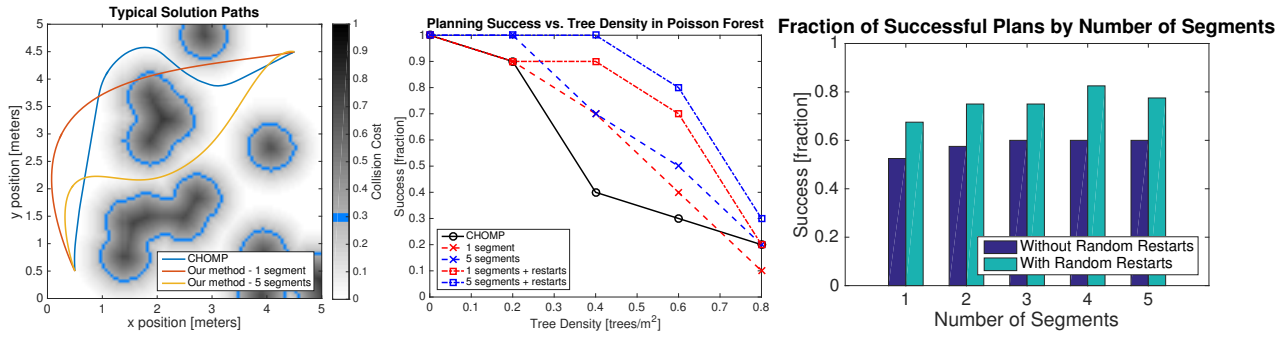
For $N = 100$ (where N is the number of discretized waypoints) in the CHOMP algorithm, the results are comparable both in run time, success rate, and path length. However, in order to fully safely verify the trajectory, there should be a waypoint for every voxel in the 3D occupancy grid. Therefore, $N = 500$ is a more appropriate comparison from a safety perspective (as the mean path length is approximately 5 meters), and since the execution time grows approximately with $O(N^2)$, this method performs much slower in such cases. It also has a lower success rate, as it does not converge to a collision-free solution within the limited iteration steps.

On the other hand, our method has a fixed number of parameters for a given number of segments, regardless of trajectory length or map resolution. Therefore, we are able to keep a low computation time, as long as the number of segments chosen is appropriate for the density of the environment, and our results show that 3 segments is enough for the realistic forest scenario tested. As a result, our approach has only 10 free parameters where CHOMP has 500 per axis.

We chose to use 3 segments and minimize jerk in our final real-world experiments, as this has the smallest number of free parameters for the highest success rate in our comparison.

B. System Simulation

Next, we validate our local replanning in the context of a complete system and only a partially known map.



(a) (b) (c)

Fig. 2: Results of 2D evaluations of our algorithm against CHOMP. (a): Typical paths generated by our algorithm with different number of segments, compared to CHOMP. With only one segment (red), there are not enough degrees of freedom to avoid all the obstacles, but it is able to find a solution (and one different from CHOMP) with 5 segments. The potential cost map is in gray, with the original obstacle edges in blue. (b): Success rate of the different local planners vs. density of the environment. As density increases, success rate decreases, but more so for a smaller number of segments, and some of this decrease can be counteracted by doing 10 random restarts. (c): Fraction of successful plans (taken over environments of all densities) by number of segments. This also shows a significant increase in success rate by doing random restarts, which allows the algorithm to avoid local minima that are in collision.

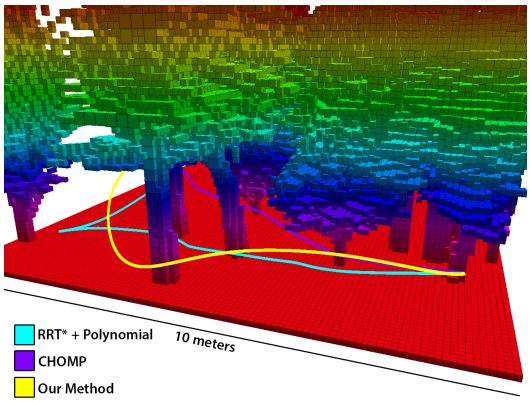


Fig. 3: Figure showing the simulation setup for evaluations. The forest is $10 \times 10 \times 10$ meters, with a density of 0.2 $tree/m^2$. The paths are planned between two random points in the space, at least $4\,m$ apart. Yellow is our method, cyan is Informed RRT* + polynomial optimization (discussed in global planner section), and purple is CHOMP.

We set up a realistic simulation experiment in RotorS [22] simulator, using a model of our multicopter platform. We approximate filling a blank map from sensor data by only giving the UAV access to a small radius of the map around itself while flying through a large forest environment. The map is $50 \times 50\,m$ with a density of 0.1 $trees/m^2$.

We use 4 meters as a planning horizon for our local replanning and give the algorithm access to 5 meters around its current position (to emulate a stereo system with a 5 meter maximum range). A new plan, minimizing jerk in a 3-segment trajectory, is generated at 4 Hz as the UAV is flying.

Fig. 4 shows the results of our experiment, compared to a global plan made from a fully-known map using Informed RRT* and polynomial smoothing.

As can be seen, both algorithms produce similar paths, with the local replanning finding a solution that is only $0.5m$ longer than the global path. The RRT* plus smoothing algorithm ran with complete knowledge of the map and took 30 seconds to compute, 20 of which were spent on finding the visibility graph and 10 were spent on finding a collision-free polynomial path. On the other hand, our algorithm was able to find a comparable path while considering only a 4 meter region around itself and continuous replanning at 4 Hz.

C. Real World Experiments

Finally, we show our complete system running in real-time on a multicopter, starting from a completely blank map and filling it from sensor data. The experiment is done in an indoor environment with two obstacles: a large one directly in front of the robot, obscuring the robot's vision, and a smaller second obstacle behind the first. A goal point is placed inside the second obstacle, and the UAV must avoid the large obstacle, fly behind it, detect the second obstacle, and stop short of collision. The physical setup is shown in Fig. 1.

Our platform is an Asctec Firefly[1] using a visual-inertial stereo sensor [23], running at 20 Hz, for both state estimation and perception, both of which are done entirely on-board on an 2.1 GHz Intel i7 CPU.

The UAV starts with a blank map and builds it online from dense stereo reconstruction data. The map is updated at 5 Hz, and replanning is done at 4 Hz, and we use the same parameters as in the previous section. The key difference is that due to the narrow field of view of the camera, we treated unknown space as free. A further discussion of this decision is offered in Section VI.

[1] http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-firefly/

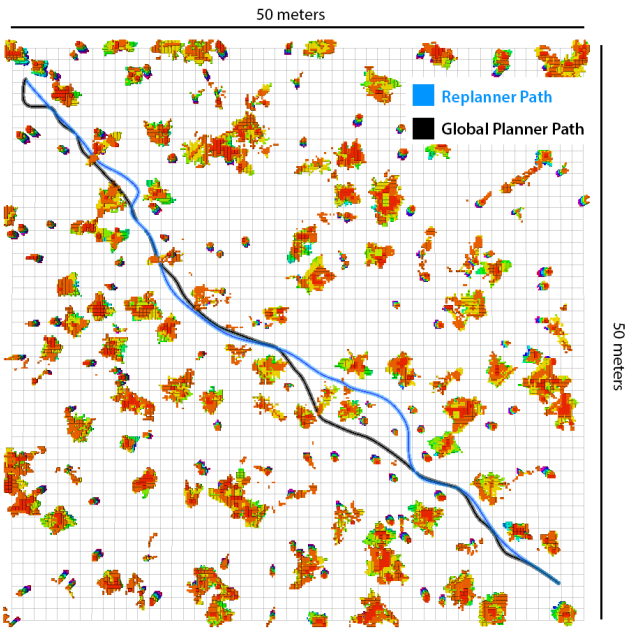


Fig. 4: Here we show a comparison of our local replanning method (cyan - 67.9 m), operating with a planning horizon of 4 meters, compared to a single global plan (black - 67.5 m) generated with prior knowledge of the entire map. The environment is 50×50 m with a density of $0.1 trees/m^2$, and we show only the tree trunks of the obstacles for clarity. The local replanning algorithm is running at a rate of 4 Hz, while the global planner, using Informed RRT* with polynomial smoothing, runs in 30 seconds.

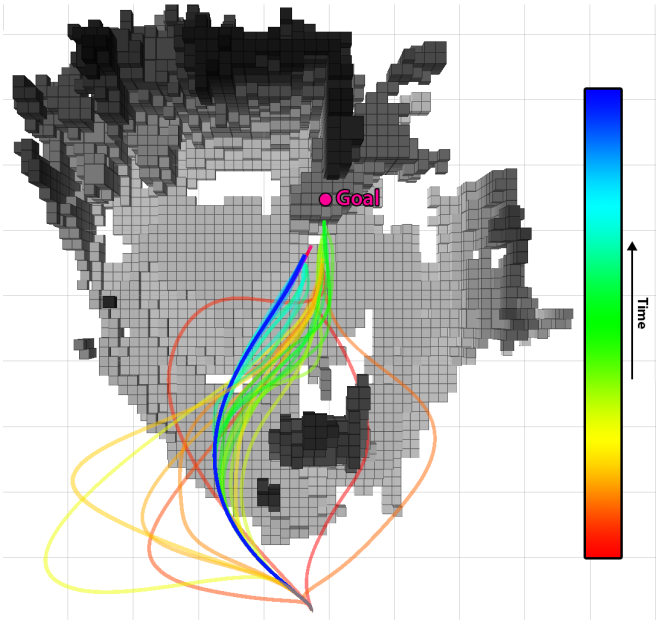


Fig. 5: Local plans over time for the real-world experiments. The original goal point is embedded in the second obstacle (pink), which is not visible from the start position. Over time, the paths stop short of collision with the obstacle and the final path (blue) is a shorter, lower curvature path than many of the plans earlier in the experiment.

Fig. 5 shows the path evolution over time of the trajectory, with the color of the trajectory going from red to blue with time. As can be seen, though initial trajectory candidates are both in collision and often very far from obstacles, but as the UAV approaches the goal, its path gets smoother and out of collision. Also note that until the trajectory color reaches cyan, the trajectory goal is still placed in collision since the UAV has not seen the obstacle yet. This experiment can be seen in the video attachment.

We also show average timings for the complete system in Table II. As can be seen, the complete system is fast enough to run in real-time at 4 Hz, and has a mean latency of only 40 ms between acquiring depth data from the sensors to generating a feasible collision-free trajectory.

VI. Discussion

Our experiments have shown that our approach is able to find solutions to local path-planning problems successfully, at a comparable rate to existing trajectory optimization methods. While sampling-based methods are still able to solve a much larger percentage of the problems posed, they are prohibitively slow for our target application.

The main advantage of our method compared to discrete-time trajectory optimization methods lies in the inherently smooth, compact representation. For example, as can be seen in Table I, the number of waypoints CHOMP requires to check every voxel along a 5 m long path with a map resolution of 10 cm is $N = 500$, and with this many variables, the convergence rate is significantly slower and the execution time is significantly longer than for our algorithm, even with 5 polynomial segments.

Our approach allows better control of end derivatives, which makes it much easier to integrate this into a continuous replanning framework, as shown in Section V-B and Section V-C. We are able to continue planning from the exact current (or future) state of the UAV, leading to smooth, continuous paths. This is an advantage over both motion primitive methods, which must discretize the state, and discrete-time methods, which can only encode lower-derivative continuity as a cost rather than a hard constraint.

However, the main drawback of this approach is the required map representation (ESDF) in which space is treated as either occupied or unoccupied, and unknown space must be treated as one of the two. The obvious choice is to treat unknown space as impassable. While this can work well in simulation, real sensors often have measurements which are not completely dense, leading to blocks of unknown space even in areas that have been observed. Treating these as occupied leads to the UAV rarely being able to find areas where the entire bounding box of the UAV contains no unoccupied voxels. Treating these as free, on the other hand, encourages the UAV to travel into unknown space to avoid obstacles. This can have disastrous consequences depending on the sensor configuration; for example, flying straight into

Step	Time [*ms*]
Mapping	
Octomap Insert	10
ESDF Initial Map Creation*	110
ESDF Incremental Update	<1
Local Replanning	
Select Start and End	1
Optimization (Total)	28
Compute Der. Gradient (per 100 evals)	0.6
Computer Col. Gradient (per 100 evals)	16
Total Time per Planner Iteration	40

TABLE II: Timings for our complete replanning system, taken from the real-world experiment. We present mean timings over the entire experiment, which is why the total optimization time is shorter than the maximum time with 10 restarts (most planner iterations find a feasible solution without any restarts). Gradients timings are given over 100 evaluations of the cost function. *Initial map creation runs only once and is not included in the total.

a ceiling that the sensors can not observe. There is also the additional cost of computing a dense distance field over each voxel of the original map, which does not scale to very large environments.

However, our algorithm can be adapted to use other map representations, as long as a smooth, continuous penalty for collisions can be defined. Future work will focus on finding more compact potential cost representations without these drawbacks.

VII. Conclusions

We presented a motion planning method that uses trajectory optimization in continuous time to find collision-free paths between obstacles. We then constructed a complete replanning system, from mapping to trajectory generation, which allows us to replan at a high rate and respond to previously unknown or unseen obstacles with low delay. We verified that our method runs comparably to discrete-time trajectory optimization, while having the advantages of continuous-time representation to minimize the number of parameters and allow arbitrary start and goal states. Our experiments showed the system running both in simulation and on a real multicopter platform at 4 Hz, though timing analysis shows that it could run at upwards of 25 Hz.

References

[1] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2899–2906, IEEE, 2012.

[2] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2452–2458, IEEE, 2013.

[3] A. J. Barry, *High-Speed Autonomous Obstacle Avoidance with Pushbroom Stereo*. PhD thesis, Massachusetts Institute of Technology, 2016.

[4] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "Chomp: Covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[5] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525, IEEE, 2011.

[6] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2013.

[7] M. Burri, H. Oleynikova, , M. W. Achtelik, and R. Siegwart, "Real-time visual-inertial mapping, re-localization and planning onboard mavs in unknown environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015.

[8] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4569–4574, IEEE, 2011.

[9] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[10] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[11] P. Krüsi, B. Bücheler, F. Pomerleau, U. Schwesinger, R. Siegwart, and P. Furgale, "Lighting-invariant adaptive route following using iterative closest point matching," *Journal of Field Robotics*, vol. 32, no. 4, pp. 534–564, 2015.

[12] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *Intelligent Vehicles Symposium (IV)*, pp. 391–396, IEEE, 2013.

[13] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the American Control Conference*, vol. 3, pp. 1936–1941, IEEE, 2002.

[14] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," 2016.

[15] A. J. Barry, A. Majumdar, and R. Tedrake, "Safety verification of reactive controllers for uav flight in cluttered environments using barrier certificates," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 484–490, IEEE, 2012.

[16] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, 2010.

[17] B. MacAllister, J. Butzke, A. Kushleyev, H. Pandey, and M. Likhachev, "Path planning for non-circular micro aerial vehicles in constrained environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3933–3940, IEEE, 2013.

[18] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson, "Motion primitives and 3d path planning for fast flight through a forest," *The International Journal of Robotics Research*, p. 0278364914558017, 2015.

[19] D. F. Shanno, "On broyden-fletcher-goldfarb-shanno method," *Journal of Optimization Theory and Applications*, vol. 46, no. 1, pp. 87–94, 1985.

[20] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[21] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2997–3004, IEEE, 2014.

[22] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The Complete Reference (Volume 1)*, ch. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. Springer International Publishing, 2016.

[23] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 431–437, IEEE, 2014.