Efficient Trajectory Library Filtering for Quadrotor Flight in Unknown Environments

Vaibhav K. Viswanathan[1], Eric Dexheimer[1], Guanrui Li[2],
Giuseppe Loianno[2], Michael Kaess[1], and Sebastian Scherer[1]

***Abstract*— Quadrotor flight in cluttered, unknown environments is challenging due to the limited range of perception sensors, challenging obstacles, and limited onboard computation. In this work, we directly address these challenges by proposing an efficient, reactive planning approach. We introduce the Bitwise Trajectory Elimination (BiTE) algorithm for efficiently filtering out in-collision trajectories from a trajectory library by using bitwise operations. Then, we outline a full receding-horizon planning approach for quadrotor flight in unknown environments demonstrated at up to 50 Hz on an onboard computer. This approach is evaluated extensively in simulation and shown to collision check up to 4896 trajectories in under 20μs, which is the fastest collision checking time for a MAV planner, to the best of the authors' knowledge. Finally, we validate our planner in over 120 minutes of flights in forest-like and urban subterranean environments.**

I. Introduction

Flying autonomously in cluttered, unknown environments is challenging for micro-aerial vehicles (MAVs). To fly in unknown environments, an MAV must collect perception sensor data, fuse the perception data into a map representation, and then plan a collision-free, dynamically-feasible path all online on its onboard computer. This is challenging due to the limited range of perception sensors, state estimation drift, and limited onboard computation.

Common approaches for online MAV path planning include optimization-based and sampling-based planners. Optimization-based methods have the benefit of solving for an optimal path given an objective function, such as smoothest path or maximizing information gain ([1]–[4]). However, these methods are computationally expensive and can have uncertain solve times. Alternatively, sampling-based planners ([5]–[8]) often have asymptotic guarantees and fixed planning times. The path quality of sampling-based planners is correlated with the planning time, due to the asymptotic guarantees. The most computationally expensive part of sampling-based planning is collision checking. Often, each sampled segment of a path is collision-checked with the entire map representation. This leads to a tradeoff between path quality and solve time. Some approaches mitigate the computational costs by only considering instantaneous sensor messages. However, this approach neglects to use all of the available information and can result in potentially dangerous, suboptimal maneuvers.

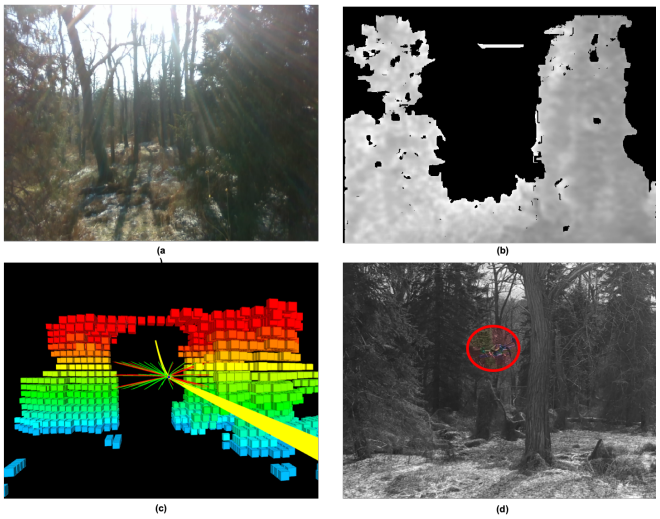


Fig. 1: A field trial of the BiTE planner in a forest-like environment. (a) shows the onboard camera image, (b) shows the corresponding depth image, (c) shows the bitset map and library of trajectories, red trajectories are in-collision, green trajectories are feasible, and the yellow is the selected trajectory, (d) shows the quadrotor performing an aggressive collision-avoidance maneuver.

In this work, we present the Bitwise Trajectory Elimination (BiTE) algorithm, an extremely efficient and reactive receding-horizon planning framework for MAV flight in unknown environments. The key insight is that discretizing the space of trajectories into a trajectory library and using a discrete, robot-centered occupancy grid map representation allows us to pre-compute the expensive collision-checking process. The BiTE algorithm has two stages: 1) a computationally expensive offline stage where collisions between the trajectory library and voxel cells are stored in a bitset library representation and 2) an efficient online stage where in-collision trajectories are filtered out using bitwise operations on the offline computed bitset library.

We use the BiTE algorithm as a key component of a full autonomy pipeline that has been verified in hardware with over 120 minutes of flights in outdoor, forest-like and urban subterranean environments with several robot and sensor configurations. Our approach is capable of collision-checking 4896 trajectories in under 20μs, which is the fastest collision-checking time for an MAV planner that uses a fused map representation, to the best of the author's knowledge.

[1] V. Viswanathan, E. Dexheimer, M. Kaess, S. Scherer are with The Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh PA {vviswan2,edexheim,kaess,basti}@cs.cmu.edu

[2] G. Li, G. Loianno are with The Department of Electrical and Computer Engineering, Tandon School of Engineering, New York University, Brooklyn, NY {gl1871,loiannog}@nyu.edu

The main contributions of this paper are:

- A novel method for efficiently filtering out in-collision trajectories from a trajectory library using bitwise operations, Bitwise Trajectory Elimination (BiTE).
- A full receding horizon perception to planning autonomy system using BiTE demonstrated at up to 50Hz. This is the highest rate for a planner with a map representation, to the best of the authors' knowledge.
- Extensive statistical evaluation of the system in simulation environments.
- Over 120 minutes of field trials demonstrating the ability of the system to control a real quadrotor in forest-like and urban subterranean environments.

This paper is organized as follows: Section II describes related research in the area of MAV motion planning and map representations. Section III describes the BiTE algorithm in detail. Section IV outlines the full online autonomy pipeline for quadrotor flight in unknown environments. Section V presents the simulations conducted and results that demonstrate the algorithm performance. Section VI presents the hardware, field experiments, and results that validate the system can be used for control of real MAVs.

II. Related Work

a) Receding Horizon Planning: Receding horizon approaches are popular for navigation in unknown environments since the replanning compensates for limited sensing horizons and reaction to dynamic obstacles. Some early work in receding horizon planning solves a Mixed Integer Linear Program to find trajectories for a fixed-wing UAV that moves towards a goal position [9]. However, this approach uses a heuristic for collision avoidance and consequently does not do explicit collision-checking. The work in [2] proposes a receding horizon control policy for quadrotors that has guarantees of algorithm completeness and collision avoidance. More recent work in high-speed MAV flight uses motion primitives in a receding horizon manner to follow a global plan [10].

b) Trajectory Libraries: Trajectory libraries and motion primitives are a popular method in robotics for trajectory set generation since they efficiently discretize the space of possible trajectories [11]–[15]. This paradigm has been used heavily for aerial vehicle navigation due to the computation constraints of MAVs. The work in [5] uses a trajectory library for MAV wire avoidance. In [8], Florence et al. present a set of motion primitives sampled in the action space. The approach presented in [16] maintains a trajectory library to ensure there is a set of safe maneuvers for helicopters.

c) Map Representations for Planning: Recent work in planning has emphasized map representations that couple perception and planning to reduce collision-checking time, utilize better sensor error models, and account for state estimation drift.

One vein of planning research is on memoryless algorithms, which reduce collision-checking time by only using the most recent sensor measurements. Both [8] and [17] do collision-checking on a point cloud from a RGB-D sensor measurements, and [18] and [19] use instantaneous stereo images for collision-checking. [20] is another memoryless algorithm that introduces a novel pyramidal partitioning of RGB-D space that enables fast collision checking. In [10], the most recent depth is both used for KD-tree collision checking and fused into a map for longer-horizon planning. A key drawback of using a single frame for planning is that it can result in myopic plans constrained by the sensor FOV or risk collision with a previously seen obstacle.

The most similar method to the work in this paper is Falco [21], which uses an adjacency list to correspond occupied voxels with invalid trajectories. However, this method uses only instantaneous sensor data and is designed for 3D LiDARs, which are heavy and not feasible for many MAVs.

Other works use a history of frames to do collision checking. Nanomap [22] and DROAN [5] do collision-checking with a fixed history of RGB-D and stereo sensor measurements, respectively. Some drawbacks of these methods are that they rely on a fixed number of frames, and are designed for specific sensor modalities (stereo and RGB-D, respectively).

III. Bitwise Trajectory Elimination (BiTE) Algorithm

The principal contribution of this work is the Bitwise Trajectory Elimination (BiTE) algorithm for computationally efficient filtering of in-collision trajectories in a library. This algorithm provides a novel method of integrating perception and planning by introducing the bitset library representation. The bitset library represents a 3D voxel grid in which each voxel has a value for how its occupancy affects the set of feasible trajectories.

The BiTE algorithm has two stages: 1) a computationally expensive offline stage where intersections between the trajectory library and voxel cells are stored in a bitset library, and 2) an efficient online stage in which in-collision trajectories are filtered out using bitwise operations on the offline computed bitset library.

A. Problem Formulation

Suppose the map is represented as a 3D voxel occupancy grid in the robot frame $\mathcal{G}_R = \{g_{ijk} | i = 1, ..., m; j = 1, ..., n; k = 1, ..., o\}$, where $g_{ijk} = \{0, 1\}$. A cell with value $g_{ijk} = 0$ refers to a free cell, and $g_{ijk} = 1$ refers to an occupied cell. The grid is positioned and oriented relative to the robot frame. The total number of cells is $M = m \cdot n \cdot o$ and the map resolution is r.

Let ξ be a trajectory of the UAV, such that $\xi(t) = \{x(t), y(t), z(t), \psi(t), v(t)\}$ where $\{x, y, z\}$ are positions in $\mathbb{R}^3$, ψ is the yaw, and v is the magnitude of the velocity. We assume that the velocity is in the direction of the UAV yaw.

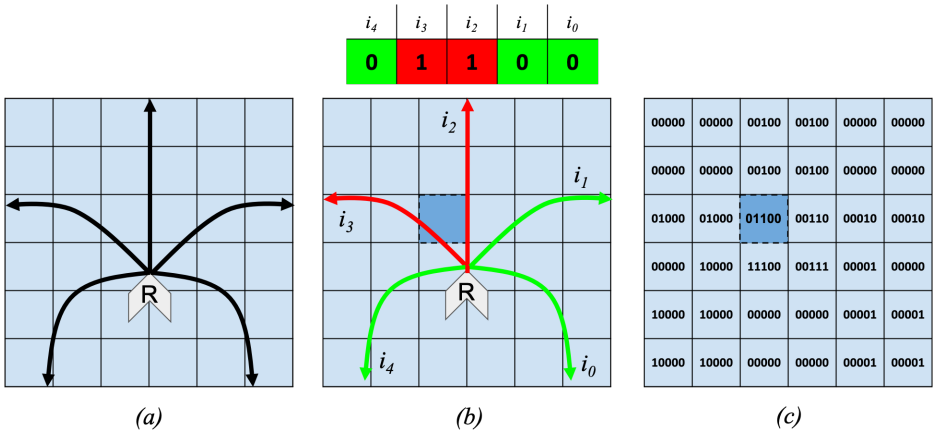


Fig. 2: A toy example of the Bitset Library Representation: (a) shows a trajectory library of size $N = 5$ overlayed on a 2D occupancy grid of size $M = 36$. (b) illustrates how a cell in the bitset library representation stores information about which trajectories in the library intersect it. (c) is the full bitset library structure that results from the offline collision-checking phase.

The space of all trajectories is discretized into a library of N trajectories, $\mathcal{L}$.

The objective of the BiTE algorithm is to find $\mathcal{L}_{free}$, the set of all trajectories that do not intersect the occupied cells in $\mathcal{G}$ such that $\mathcal{L}_{free} \subseteq \mathcal{L}$. The best trajectory, ξ^*, can be selected from $\mathcal{L}_{free}$ and executed by the UAV.

B. Offline Collision Checking

In the bitset library map representation, each voxel stores information about whether each trajectory in the library intersects it. The bitset library is an M-length array of N-bit bitsets, where each element of the array, $\mathcal{B}_i$, is a bitset that corresponds to a voxel in $\mathcal{G}_R$. Each bit in the bitset corresponds to a trajectory in $\mathcal{L}$ and the value is 1 if that trajectory intersects the respective voxel and 0 otherwise. The bitset library data structure is demonstrated in 2D in Figure 2.

The bitset array data structure is built by looping over all voxels. Within each voxel loop, a collision bitset is created by looping over all trajectories and setting the corresponding bit number to 1 if the trajectory intersects the voxel. To account for the robot shape and safety constraints, collision checking is done on expanded configuration space. The collision bitset is added to the bitset library, which is an M-length bitset array. The full algorithm for offline collision checking is shown in Algorithm 1.

The bitset library only needs to be calculated once for a given map resolution r, map dimensions (m,n,0), and trajectory library ($\mathcal{L}$).

C. Online Trajectory Library Filtering

The BiTE online filtering stage leverages the bitset library representation to efficiently collision check all of the trajectories in $\mathcal{L}$. The algorithm for online filtering is shown in Algorithm 2.

In the first step of the online phase, an N-bit bitset $freeTrajectories$ is initialized as zero. Each bit of

Algorithm 1: Offline Collision Checking

```
initialize collisionList = {} ;
for i ← 1 to M do
    c ←bitset< N > ;
    for j ← 1 to N do
        c[j] = checkCollision(ξ_j, g_{i,j,k}) ;
    end
    Add c to collisionList ;
end
Result: collisionList
```

$freeTrajectories$ corresponds to a trajectory in $\mathcal{B}$; the value of the bit is 0 if the trajectory is safe and 1 if it intersects an occupied cell in $\mathcal{G}_R$. The value of $freeTrajectories$ is updated by looping over all cells in $\mathcal{G}_R$, and applying a bitwise OR with $freeTrajectories$ and $\mathcal{B}_i$ if $\mathcal{G}_{R,i}$ is occupied. This bit masking operation sets the bits of all trajectories intersecting occupied cells to 1, effectively collision checking $\mathcal{L}$ to compute $\mathcal{L}_{free}$.

The online filtering stage of the BiTE algorithm is only a step of the full planning system, described in more detail in section IV.

Algorithm 2: Online Trajectory Library Filtering

```
freeTrajectories ←bitset< N > ;
foreach c ∈ collisionList do
    if isOccupied(g_ijk) then
        freeTrajectories = c|freeTrajectories
end
Add c to collisionList ;
Result: freeTrajectories
```

IV. Implementation

A. Autonomy Pipeline

The full online autonomy pipeline consists of a mapping state and a planning stage, as illustrated in Figure 3.

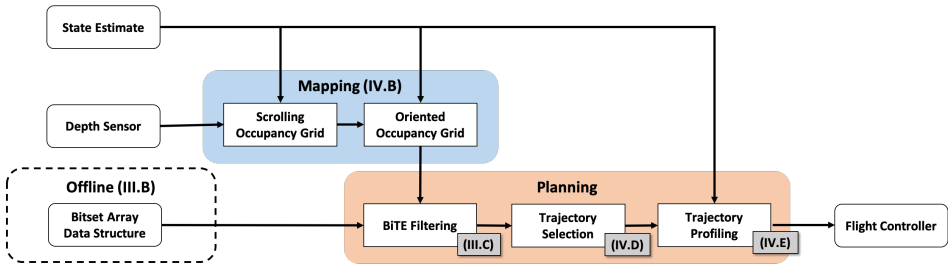


Fig. 3: Autonomy architecture. The output of the state estimate and depth image is used by the mapping subsystem as described in Section IV-C. The oriented occupancy grid and offline-computed bitset map representation is used to filter out in-collision trajectories from the library as described in III. From the filtered trajectories, the best trajectory is selected (Section IV-D) and profiled (Section IV-E).

The mapping step uses the robot state estimate and perception data to create a scrolling occupancy grid that translates with the robot frame, but is oriented in the global frame. The occupancy grid is then rotated into the local frame of the robot. The mapping step has been tested with RGB-D camera, stereo camera, and LiDAR data. The mapping algorithm is described further in Section IV-C.

After fusing the perception data, a set of collision-free trajectories is computed using the online step of the BiTE algorithm described in Section III-C. The best trajectory from this set is then selected as described in section IV-D. The selected trajectory is then profiled and executed as described in section IV-E. The autonomy pipeline is run in a receding-horizon fashion, and has been tested at up to 50Hz.

B. Trajectory Library Generation

The BiTE algorithm is agnostic to the method of trajectory library generation. However, the planner performance in cluttered environments depends significantly on the size of the trajectory library.

We generate a feasible path library in the robot frame by treating the UAV as a point mass with an initial velocity in the x direction and apply accelerations up to a_{max} to the mass at varying angles (θ, ψ). To ensure that the library of trajectories is feasible, we set the initial velocity at generation to be the maximum allowable flight velocity, v_{max}. Due to a quadrotor's agility and ability to hover, the UAV will be able to follow the generated trajectories when $v < v_{max}$. The velocity profile for a selected trajectory is recalculated when $v \neq v_{max}$ as described in Section IV-E.

C. Map Representation

The BiTE algorithm operates on a robot-centric and robot-oriented 3D occupancy grid. The size and resolution of the occupancy grid is fixed, and must be determined before the offline collision checking procedure.

Since local planning only needs information from an immediate surrounding area, a map with efficient updates and constant memory usage is preferred. To allow for various sensor types we use a dense scrolling occupancy grid library [23] centered around the robot. The 3D circular buffer allows for fast indexing operations and constant memory usage, but the voxels are always aligned with the origin of the world frame. This grid is used only for fusing sensor data, while a separate interface is required for the rotation to match the expected bitset for the trajectory library.

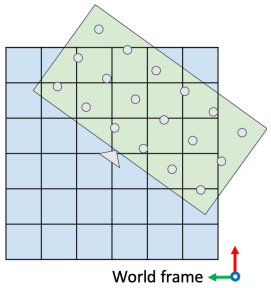


Fig. 4: The blue grid shows a 2D representation of the robot-centric scrolling occupancy grid. The green grid shows the rotated robot-oriented occupancy grid.

To acquire this oriented map, the centers of a precomputed voxel grid are rotated and used to index into the scrolling grid as seen in Figure 4. The bitset can be filled in directly by thresholding the log-odds occupancy value. Note that the dimension of the bitset map and the scrolling grid do not need to be equal. By separating the grids, the sensing horizon and the map size for the precomputed library can be decoupled. Noisy sensor measurements can be used to refine the map further away during high-speed flight, while only the feasible space of trajectories needs to be considered for planning. Efficient sequential processing can be exploited by scaling the rotation of the robot with the voxel resolution such that only additions are needed to iterate over the voxel centers.

D. Trajectory Selection

Given a set of collision-free trajectories, the goal of the planner is to choose the optimal trajectory ξ^*. A simple method we use for all of the simulation trials and the forest-like field trials is selecting the collision-free trajectory that ends the closest to a user-defined goal point. Another method we use for additional safety in the urban subterranean environment is minimizing cost based on an Euclidean Signed Distance Field (ESDF). This approach allows us to to enforce more optimal behavior, like centering the quadrotor in a hallway, etc.

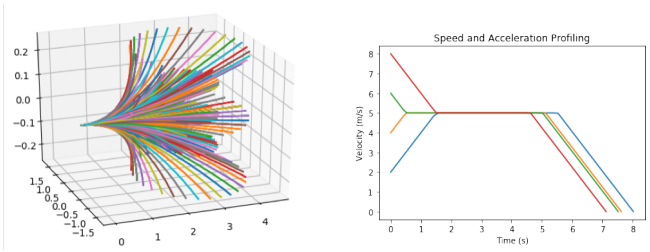
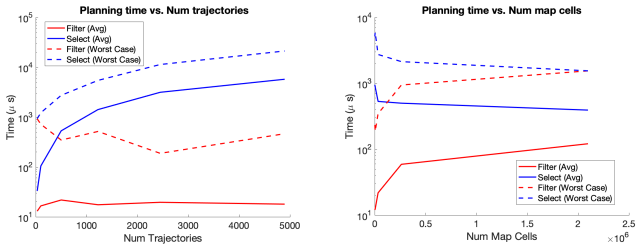


Fig. 5: (a) shows an example of a 3D trajectory library used and (b) shows the velocity profiling of a single path given different starting velocities.



(a) (b)

Fig. 6: The mean and worst case compute time for the BiTE online filtering and select steps of the planning pipeline on an Intel NUC i7. (a) shows how compute time depends on the number of trajectories in the library. (b) shows how compute time varies with map size.

E. Motion Control

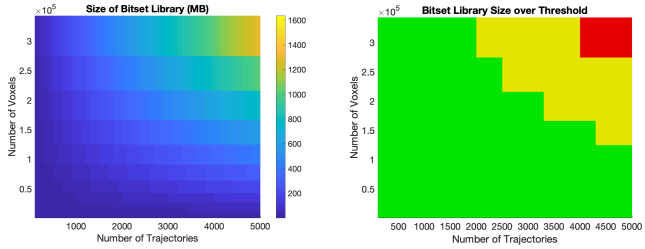
Before the selected trajectory is executed, the time spacing between waypoints is recalculated to create a velocity profile that is dynamically feasible. This velocity profile is generated using a modified version of a time-optimal speed and acceleration profiling algorithm outlined in Section V-B of [24]. The velocity and acceleration profiles are generated for a path given the initial velocity, a goal velocity, and maximum acceleration constraints by assuming linear uniformly accelerated motion between waypoints. We modified the algorithm to include deceleration constraints and to end every trajectory with a zero magnitude velocity. These design choices ensure that if a new trajectory is not found, the current plan can be safely executed to completion.

The quadrotor tracks the profiled trajectory using a path tracker similar to Section IV-B [24]. The feed-forward velocities and pose errors are converted into an desired attitude and thrust that are fed to a low-level attitude controller.

V. Simulation

A. Computational Load

The BiTE algorithm achieves unprecedented planning rates by precomputing the collision-checking and using efficient bitwise operations for trajectory library filtering. We ran two experiments to evaluate the effects of bitset map and trajectory library size on the computational load of the filtering and trajectory selection steps: 1) the trajectory library size (size of bitset) is varied and the map dimension is fixed at $M = 49,152$ cells with a resolution of $0.3m$ and 2) the map dimensions (number of bitsets) are varied and the trajectory library size is fixed at $N = 401$. The executed trajectory, ξ^*, was selected by choosing the collision-free trajectory closest to the goal. The experiments were run on an Intel NUC i7 with 8GB of RAM and 8 CPU cores.



(a) (b)

Fig. 7: (a) The memory requirements of the trajectory library depends on the number of trajectories and the number of voxels. (b) The green indicates bitset libraries that are < 512 MB, the yellow indicates libraries bitset libraries that are 512 MB $\leq \mathcal{B} < 1$ GB, and the red are libraries that are ≥ 1 GB or greater.

Figure 6 shows the mean and worst-case computation time taken for the filtering and selection steps of the BiTE planning pipeline. The results in Figure 6a show that the average time to filter a trajectory library is approximately constant and under 20 μs for up to $N = 4896$. This result is notable because the theoretical, computational complexity of the BiTE online algorithm is linear with N. While the number of operations is linear, the operation is a bitwise OR, which takes less than one CPU cycle. The difference in time for $1 - 5000$ CPU cycles is much less than $1\mu s$; thus, the computation time of the filtering step is likely dominated by supporting operations.

The time of the selection step is linear with N. Intuitively, this result is expected since the absolute size of $\mathcal{L}_{free}$ grows linearly with N, and the executed trajectory, ξ^*, is selected by calculating the cost for all trajectories in $\mathcal{L}_{free}$. The mean and worst-case selection time at $N = 4896$ are $3,960\mu s$ and $21,660\mu s$, respectively.

In general, the selection time is higher than the filtering time. This result is different from most state-of-the-art planning pipelines, in which collision checking is the most expensive step.

B. Bitset Library Memory Requirements

The BiTE algorithm achieves efficient collision checking by making a tradeoff for increased memory requirements. The MAV loads the offline-computed bitset library into memory at the start of each run; thus, the onboard computer's physical memory is the limiting factor. Figure 7a shows the size of the bitset library given the number of voxels and the trajectory library size; the size of the bitset libraries ranges from hundreds of megabytes to greater than one gigabyte.

We can choose the map dimensions and trajectory library size based on the memory constraints of the typical UAV onboard computers, such as the Intel NUC i7 with 8GB RAM used in this work. For these systems, the memory for the bitset library could be limited around 512MB-1GB to allow

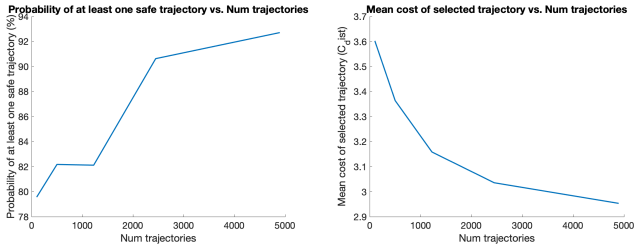
for adequate memory for other necessary operations. Figure 7b shows the acceptable ranges for the map and trajectory library size given the memory constraints of 512MB and 1 GB. The green indicates bitset libraries that are < 512 MB, the yellow indicates libraries bitset libraries that are between $512 \text{ MB} \leq \mathcal{B} < 1$ GB, and the red are libraries that are 1 GB or greater.

The results show that the memory requirements for the bitset library substantially restrict the map size and trajectory library size. This drawback should be addressed in future work.

C. Library Performance

We evaluate the relationship between trajectory library size and resulting performance on two metrics: the probability of finding a safe trajectory and the average cost of the selected trajectory. The experiment was run on 20 randomly generated forest-like environments in which the quadrotor was commanded to fly forward for 500 meters.

As shown in Figure 8a, the probability of finding at least one collision-free trajectory in the library increases with library size. This result highlights the importance of having a large set of candidate trajectories for safety-critical flights. Furthermore, Figure 8b shows that the mean cost of the selected trajectory decreases with trajectory library size. Intuitively, this result matches expectations since using a more granular trajectory library provides more options for selecting a path.



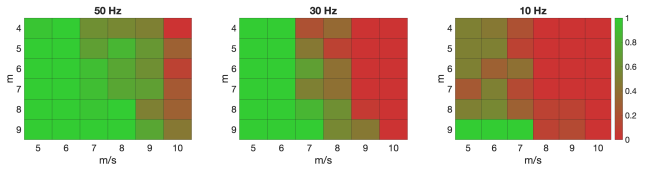
(a) (b)

Fig. 8: (a) The probability of finding at least one feasible trajectory increases with the size of the trajectory library. (b) The mean cost of the selected trajectory decreases with the size of the trajectory library.

D. Reactivity Tests

We evaluate the impact of the planner's re-planning rate on the planner's performance by testing its ability to avoid an instantaneously appearing obstacle at different planning frequencies. The quadrotor is commanded to fly in a straight line trajectory at a fixed velocity ranging from $5-10ms^{-1}$, and a $1m$ diameter cylindrical obstacle is generated instantaneously at a fixed distance $(4-9m)$ in front of the quadrotor. The simulated MAV uses a 50Hz RGB-D camera for perception. We ran the experiment for 100 trials each at 50Hz, 30Hz, and 10Hz.

Figure 9 shows the results of the reactivity experiment. A re-planning rate of 50 Hz results in significantly better reactivity to an instantaneously appearing obstacle. At 50Hz, 75% of the trials successfully avoided the appearing obstacle, including



(a) (b) (c)

Fig. 9: Heatmaps show the probability of successfully avoiding an instantaneously generated cylindrical obstacle with a 1m diameter. The results from the planner run at 50Hz, 30Hz, and 10Hz are shown in (a),(b),(c), respectively.

		Single Trajectory Coll. Check (μs)		Library (N=160) Coll. Check (μs)	
	Computer	Mean	Max.	Mean	Max.
DROAN [5]	i7 NUC	115	818	18,400	131,000
KD-Tree	i7 NUC	80.6	650	12,900	104,000
Florence et al. [8]*	i7 NUC	56	N/A	N/A	N/A
Lopez et al. [17]*	i7-2620M	48	N/A	N/A	N/A
RAPPIDS [20]*	i7-8550U	1.2	N/A	N/A	N/A
Falco[21]*	i7 NUC	0.005^{+}	$\mathbf{0.007}^{+}$	N/A	N/A
BiTE (ours)	i7 NUC	$\mathbf{0.003}^{+}$	0.09^{+}	**16.6**	**722**

TABLE I: BiTE collision-checking outperforms other common planning approaches. The experiments for DROAN, KD-Tree collision checking, and BiTE were conducted on the same hardware with a trajectory library size of $N = 160$. *The results shown for these algorithms are the best reported results in their respective papers. These papers do not report a worst-case result, and the collision checking time for a library of size $N = 160$ is unknown. $^{+}$The single trajectory collision checking time was calculated as the average from the largest trajectory library.

82.5% of the trials at $9ms^{-1}$. Only 40% of the 30Hz trials and 3.3% of the 10Hz trials were successful. The results demonstrate the importance of a high re-planning rate when operating in unknown environments.

E. Collision Check Time Comparisons

We compare the collision checking performance of BiTE to state-of-the-art algorithms through both experimentation and literature review. The results for DROAN [5], KD-Tree checking, and BiTE were evaluated using the 20 randomly generated forest-like environments mentioned previously. KD-tree checking and BiTE simulations use an RGB-D camera for perception, while DROAN uses a stereo pair. We use a trajectory library of size 160 for all three methods. Table I shows the average and worst-case results for both single trajectory and library collision checking.

Other state-of-the-art algorithms introduced in [8], [17], [20], [21] are also compared in Table I. We use the best, self-reported collision check times for comparison. These papers have different representations of the trajectory space, so only the single trajectory collision check is represented for comparability. The results in [8], [17], [20] do not report a worst-case collision check time. We calculate the mean collision check time for [21] from the reported total collision check time for a trajectory library of size 42,875. The same approach was taken for the mean single trajectory collision check time for BiTE.

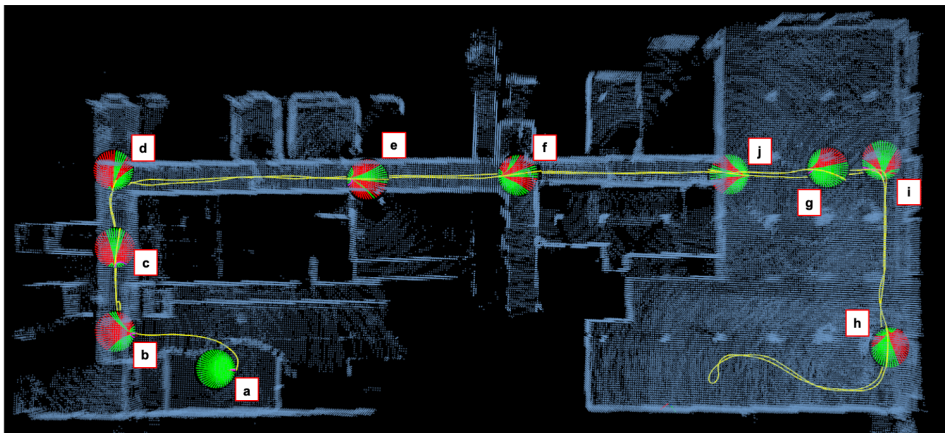
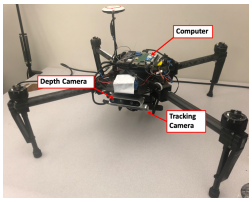


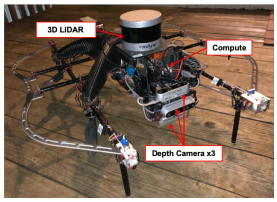
Fig. 10: The flight path of a representative mission in an urban subterranean environment. The trajectory library is visualized at various points throughout the flight path. Green trajectories are feasible and red trajectories are in-collision. The visualization was post-processed from a bag file.

	Forest-like	Urban Sub-T
Num of Trajectories	480	720
Num voxels	49,152	393,216
Voxel size	0.3 m	0.1 m
Collision radius	0.6 m	0.4 m
Max speed	4 m/s	1 m/s

TABLE II: The set of parameters used for field trials.



(a)



(b)

Fig. 11: (a) shows the DJI M100 used in the outdoor forest environment and (b) shows the custom quadrotor used in urban subterranean environments.

VI. Field Trials

We present a series of field trials with over 120 minutes that validate the performance of the BiTE planning approach. For the purpose of this work, we focus on evaluating the performance in two environments: outdoor forest-like environments and urban subterranean environments.

A. Forest Environment

For outdoor forest-like environments, we used a DJI M100 quadrotor shown in Figure 11a. The processing is done on an Intel NUC i7 with 8GB of RAM and 8 CPU cores. The platform is equipped with an Intel Realsense T265 tracking camera for state estimation and an Intel Realsense D400 depth camera for mapping. The raw output of the depth camera was processed with a speckle filter to remove spurious points.

1) Overview of trials: We conducted 24 flights in a dense, forest-like area with goal velocities ranging from $2-4ms^{-1}$. The goal point was set to be 100 meters from the takeoff location. Figure 1 shows an example of the forest environment and the corresponding mapping & planning visualization.

There were two notable failure cases. In one case, a thin horizontal branch was filtered from the depth image by the speckle filter. This resulted in a collision since the branch did not appear in the bitset map. After this trial, the speckle filter parameters were updated to more conservative values. This parameter update occasionally caused the map to have false positive obstacles due to sensor noise. In another case, the T265 tracking camera driver stopped publishing an odometry, causing an immediate crash.

B. Subterranean Urban Environment

We tested in subterranean urban environments using a custom quadrotor design shown in Figure 11b. This quadrotor was designed specifically for indoor flight for the DARPA Subterranean Challenge [25]. The platform is equipped with a Velodyne Puck VLP-16 Lite, with 30° vertical field of view, for mapping. All of the computation is done on an Intel NUC i7 with 8GB of RAM and 8 CPU cores. We use LOAM [26] for SLAM, and the output is converted into the Bitset Map representation.

1) Overview of trials: We conducted 8 flights with an average time of 11 minutes. Goal waypoints were set by a global planner that balances exploration and remaining in communication range. The environments ranged from open spaces to branching hallways. During these trials, the quadrotor propeller guards occasionally made contact with the wall due to tunnel wind effects and control tracking error. None of these collisions resulted in the missions ending prematurely.

2) Analysis of a single flight: In this section, we qualitatively analyze the BiTE algorithm in a long flight. The results used in this analysis are post-processed for a bag file. This specific mission was chosen since it demonstrates the benefit of having a large trajectory library in environments ranging from open spaces to narrow hallways. The flight path is overlayed on an occupancy grid shown in Figure 10. For visualization purposes, the trajectory library was downsampled to 72 2-D trajectories.

(a) The mission starts with a takeoff in an open area where

none of the trajectories are in collision. (b) The global plan guides the quadrotor towards a doorway to a narrow hallway, where a few trajectories turning into the hallway and turning into another room are feasible. (c) There are several feasible trajectories that allow the quadrotor to center itself in the hallway as well as turn around. (d) The hallway has an entrance to a small room and a 90 degree turn. The resulting set of feasible trajectories allows the quadrotor to fly straight into the room or take a right turn. (e) In addition to the hallway trajectories, there is a single feasible trajectory to enter a tight doorway. This result would not be possible with a sparser trajectory library. (f) There are a set of trajectories for each possible direction at the intersection. (g) (h) (i) The quadrotor explores an open space with the main obstacles being the walls and structural pillars. (j) The quadrotor returns from open space into a narrow hallway. The set of trajectories allows the quad to safely navigate into the hallway or turn around to stay in the open room.

VII. Conclusion & Future Work

In this work, we present BiTE: an efficient algorithm demonstrated to collision-check up to 4896 trajectories in under $20\mu s$ on an onboard computer. This algorithm is part of a full autonomy pipeline that was validated in over 120 minutes of flight in forest-like and urban subterranean environments. We conducted extensive statistical simulation experiments of this planning framework, and demonstrate that it out performs state-of-the-art approaches with respect to computation time.

Future work could include reducing the memory requirements and optimizing the algorithm for GPU and/or FPGA parallelization. Furthermore, BiTE can be extended to other robotic systems. Systems with high degrees of freedom, such as manipulators, could benefit from the use of large trajectory libraries that BiTE facilitates.

Acknowledgment

This work was partially supported by DARPA agreement HR00111820044. We thank Rogerio Bonatti and Ian Higgins for their help with field trials.

References

[1] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 344–351.

[2] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive MAV flight with limited range sensing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3235–3240.

[3] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast incremental euclidean distance fields for online motion planning of aerial robots," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019.

[4] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017.

[5] G. Dubey, R. Madaan, and S. Scherer, "DROAN - disparity-space representation for obstacle avoidance: Enabling wire mapping avoidance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018, pp. 6311–6318.

[6] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1474–1481, 2018.

[7] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[8] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps," in *Proceedings of Workshop Algorithmic Foundations of Robotics*, 2016.

[9] J. Bellingham, A. Richards, and J. P. How, "Receding horizon control of autonomous aerial vehicles," in *American Control Conference (IEEE Cat. No. CH37301)*, vol. 5. IEEE, 2002, pp. 3741–3746.

[10] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *International Conference on Robotics and Automation (ICRA)*, May 2019, pp. 732–738.

[11] M. Stolle and C. G. Atkeson, "Policies based on trajectory libraries," in *IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 3344–3349.

[12] D. Berenson, R. Diankov, Koichi Nishiwaki, Satoshi Kagami, and J. Kuffner, "Grasp planning in complex scenes," in *7th IEEE-RAS International Conference on Humanoid Robots*, Nov 2007, pp. 42–48.

[13] D. Dey, T. Y. Liu, B. Sofman, and J. A. Bagnell, "Efficient optimization of control libraries," in *AAAI*, 2012.

[14] C. Goldfeder, M. Ciocarlie, J. Peretzman, H. Dang, and P. K. Allen, "Data-driven grasping with partial sensor data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2009, pp. 1278–1283.

[15] E. Frazzoli, M. A. Dahleh, and E. Feron, "Robust hybrid control for autonomous vehicle motion planning," in *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No.00CH37187)*, vol. 1, Dec 2000, pp. 821–826 vol.1.

[16] S. Arora, S. Choudhury, D. Althoff, and S. Scherer, "Emergency maneuver library-ensuring safe navigation in partially known environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 6431–6438.

[17] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5759–5765.

[18] R. Brockers, A. Fragoso, and L. Matthies, "Stereo vision-based obstacle avoidance for micro air vehicles using an egocylindrical image space representation," in *Micro-and Nanotechnology Sensors, Systems, and Applications VIII*, vol. 9836. International Society for Optics and Photonics, 2016.

[19] A. J. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3046–3052.

[20] N. Bucki, J. Lee, and M. W. Mueller, "Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation," 2020.

[21] J. Zhang, C. Hu, R. G. Chadha, and S. Singh, "Maximum likelihood path planning for fast aerial maneuvers and collision avoidance," in *Proceedings of (IROS) IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2019.

[22] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "NanoMap: Fast, uncertainty-aware proximity queries with lazy search over local 3D data," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7631–7638.

[23] D. Maturana, "dimatura/scrollgrid: First public version," Jul. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.832978

[24] G. Hoffmann, S. Waslander, and C. Tomlin, *Quadrotor Helicopter Trajectory Tracking Control*.

[25] "DARPA subterranean challenge." [Online]. Available: http://www.subtchallenge.com/

[26] J. Zhang and S. Singh, "LOAM: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, 2014.