

# An Iterative Planner with Provable Safety for Quadcopter Navigation

Gaofeng Su<sup>1</sup>, Teaya Yang<sup>2</sup>, Raja Sengupta<sup>1</sup> and Mark W. Mueller<sup>2</sup>

**Abstract**—Safe Flight Corridor planning is a state-of-the-art paradigm for real-time drone navigation in unknown environments, valued for its computational efficiency and reactivity. However, when using sensors with limited fields of view, the Safe Flight Corridor planning methods suffer from perceptual blind spots and must rely on assumptions about unobserved space, assumptions that cannot be formally guaranteed and may fail in critical corner cases. This paper addresses these limitations by introducing an iterative planner that augments a reactive core with a compact spatial memory, represented by lightweight convex primitives. We establish a formal proof of safety for the proposed algorithm in static environments and validate the implementation through simulation.

## I. INTRODUCTION

Real-time motion planning in unknown and cluttered environments presents a fundamental challenge for the autonomy of drones. This capability is critical for a growing range of applications, from infrastructure inspection [1] to search operation. The core difficulty lies in generating dynamically feasible trajectories that guarantee safety under severe perceptual limitations and strict onboard computational constraints.

To address this challenge, the Safe Flight Corridor (SFC) paradigm has emerged as a dominant and effective strategy. Its core idea is to represent large, complex collision-free spaces using simple convex geometry primitives, such as polyhedra and ellipsoids, to represent the large, complex collision-free space [2]. This simplified representation facilitates efficient and reliable collision-free trajectory optimization. While the SFC framework is now widely adopted, methods for generating the corridor itself fall into several distinct categories, each associated with a fundamental trade-off.

On one end of the spectrum are map-based SFC planners. These methods generate corridors from an explicit, locally-consistent environmental map, usually represented as occupancy grid maps and optimize collision-free trajectory as needed [3]–[6]. By maintaining a map, these planners can explicitly reason about known free, occupied, and unknown space, allowing them to provide strong safety guarantees. Specifically, [4] provide a mathematical proof of the safety guarantee. However, this robustness comes at the cost of significant computational and memory overhead required to build and maintain the map, which can limit the replanning frequency needed for highly agile flight and increase computation resource requirement on hardware.

As a solution to the high computational cost of map-based planning, memory-less SFC planners generate corridors solely from the immediate sensor measurement, discarding information between planning cycles. For example, [7] use the depth camera’s field of view (FOV) to filter out trajectories outside the FOV and employ an efficient sampling strategy to construct a chain of collision-free spheres along the trajectory to ensure safety. Building on this idea, [8] partition the collision-free space within the FOV into pyramids, enabling verification of a large number of randomly sampled trajectories. [9] further improve the efficiency of [8] by proposing a faster pyramid construction method. However, the memory-less nature of these planners introduces a critical vulnerability due to the limited FOV of onboard perception sensors. For instance, when using a single depth camera, the drone may need to traverse an unobservable region, we called it “blind spot”, before entering the FOV region (illustrated in Figure 2). Consequently, planners must rely on non-guaranteed assumptions about these blind spots, potentially compromising safety.

Seeking to combine the speed of reactive methods with the situational awareness of map-based planners, a third category of “hybrid” SFC planners has emerged. These approaches are map-less but attempt to retain some memory by maintaining a collection of convex geometry representations of free space, such as overlapping spheres or polyhedra [10]–[12]. While computationally efficient and able to preserve more temporal information than memory-less planners, these methods still face challenges due to the limited FOV of onboard perception sensors. Since they typically maintain only a set of convex regions representing free space, extra care is required to ensure that newly constructed convex regions are indeed collision-free. In practice, however, such guarantees are rarely formalized and often rely on explicit or implicit assumptions.

Attempts to provide formal safety guarantees for lightweight SFC systems often rely on idealized assumptions. For instance, some recent work [12] derives a guarantee based on an omni-directional sensor model. In practice, however, the implementation uses LiDAR sensors with limited vertical FOV, so the formal assumptions do not fully capture real-world sensing conditions. This highlights the need for a formal proof that jointly accounts for the drone, its realistic perception sensor, and the convex free-space approximation used to represent the safe flight corridor.

This paper proposes an iterative planning method in the map-less safe flight corridor category that preserves computational efficiency while explicitly accounting for perceptual blind spots. Our primary contribution is a formal proof

The authors are with the <sup>1</sup>CalUmaned Lab, Department of Civil and Environmental Engineering, and the <sup>2</sup>High Performance Robotics Lab, Department of Mechanical Engineering, University of California, Berkeley. Contact at {koufongso, teaya.yang, rajasengupta, mwm}@berkeley.edu

of safety for this framework. Our analysis is conducted under the assumptions of a static environment, a sensor that accurately detects all obstacles within its perception range and the drone has perfect tracking so it will strictly follow the planned safe trajectory. Given these assumptions, our method only requires an initial startup condition and makes no further assumptions about unobserved space during operation. The main contributions of this paper are:

- 1) A novel geometric primitive for representing safe flight corridors.
- 2) An iterative planning algorithm that extends the RAPIDS framework [8] to eliminate blind-spot vulnerabilities, validated in simulation.
- 3) A formal mathematical proof guaranteeing the safety of the proposed algorithm.

The rest of this paper is organized as follows. Section II presents the problem setup and key definitions. Section III describes the proposed algorithm and its procedure. Section IV provides a formal safety proof of the algorithm, while Section V discusses its current limitations. Section VI presents two simulation flights demonstrating the proposed approach. Finally, Section VII concludes the paper with discussions and directions for future work.

## II. TERMINOLOGIES

In this work, we address the problem of real-time motion planning for a quadcopter equipped with a forward-facing depth camera. We consider a static environment, where all obstacles are stationary and the depth camera can capture all obstacles within its perception range and the drone can perfectly track the trajectory. These assumption ensures that information added to our spatial memory map remains valid over the planning horizon, allowing us to formally prove the safety of the algorithm.

Let  $k \in \mathbb{N}$  denote the discrete iteration step, our setup is as follows:

- 1) **Quadcopter model:** The quadcopter's configuration (position and orientation) in a world-fixed coordinate system at step  $k$  is  $(x_k, R_k) \in SE(3)$ . We model the geometry of the drone as a sphere  $S_k(x_k, r)$ , centered at  $x_k$  with fixed radius  $r > 0$ . A spherical model is a common conservative approximation of the drone in motion planning. It simplifies collision detection to a simple distance query, enabling real-time computation, and provides a safety guarantee for the drone's true, more complex geometry.
- 2) **Environment and obstacles:** The environment contains static obstacle regions, denoted as the set  $\mathcal{C}_{obs} \subset \mathbb{R}^3$ .
- 3) **Collision-free/safe:** A quadcopter with position  $x_k$  is considered collision-free/safe if its spherical model  $S_k$  does not intersect the obstacle set, i.e.  $S_k \cap \mathcal{C}_{obs} = \emptyset$ .
- 4) **Collision-free space decomposition:** We decompose the known collision-free space as a set of convex polyhedra. Each polyhedron is denoted as  $f_{safe}$ , and the set

of all known polyhedra at iteration  $k$  is  $\mathcal{F}_k = \{f_{safe,1}, f_{safe,2}, \dots, f_{safe,n}\}$ . Each polyhedron is fully contained in the free space, e.g.,  $f_{safe,i} \subset \mathcal{C}_{free}$  and  $\mathcal{C}_{free} = \mathbb{R}^3 \setminus \mathcal{C}_{obs}$ .

In our implementation, the convex polyhedra are custom-defined frustums (see Figure 1). A frustum is defined by two components: (i) a sphere  $S$ , specified by its center  $x$  and radius  $r$ . (ii) a plane  $P$ , we called it far plane, specified by its normal  $n$  and offset  $d$ , given by the equation  $n \cdot x + d = 0$ , which contains a convex quadrilateral  $Q$ . The plane  $P$  does not intersect the sphere, i.e., the distance from  $x$  to  $P$  is strictly greater than  $r$ . The frustum is then the convex volume bounded by six planes: the far plane  $P$  containing quadrilateral  $Q$ ; four side planes, each formed by an edge of  $Q$  and tangent to the sphere  $S$ ; four side edges, each formed by intersecting adjacent side planes; and a capping plane, we called it near plane, parallel to  $P$ , tangent to the sphere on the side opposite  $Q$ . If  $Q$  is a rectangle, then we call this frustum a rectangular frustum.

- 5) **Perception model:** The drone is equipped with a forward-looking depth camera. Its field of view (FOV), denoted by  $V_k$ , is modeled as a rectangular pyramid with the apex located at the drone's position  $x_k$ . The distance from the apex to the base corresponds to the sensing range of the depth camera. The depth image acquired at time step  $k$  is denoted by  $I_k$ . By definition,  $V_k$  is convex.

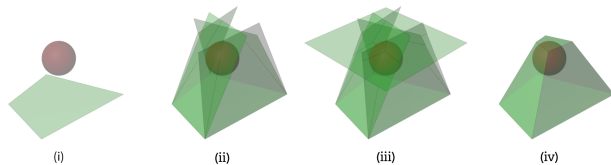


Fig. 1. Definition of the customized frustum primitive and its construction process. From left to right: (i) given a sphere and a convex quadrilateral on a plane that does not intersect the sphere. (ii) construct four side planes passing through the edges of the quadrilateral and tangent to the sphere. (iii) add a capping plane parallel to the quadrilateral's plane and tangent to the sphere; the frustum is the convex volume enclosed by these six planes. If the quadrilateral is rectangular, the frustum is called a rectangular frustum.

## III. ALGORITHMS

Our method is an iterative planning loop designed to guarantee safety by explicitly managing unobserved regions. At the beginning of each planning iteration, the drone must account for a blind spot, defined as the region outside the sensor's current FOV that a planned trajectory passes through (see Figure 2). Since this region is unobserved, its safety cannot be immediately verified by sensor data and must be explicitly managed.

Our algorithm addresses this challenge in two primary stages, as illustrated in Figure 3 and detailed in Algorithm 1:

- 1) **Boundary frustum generation:** First, a boundary frustum is identified from the set of previously validated safe frustums. This new frustum provides a safe

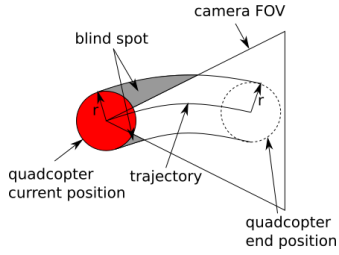


Fig. 2. Illustration of the blind spot. It is the swept volume of the drone along the portion of a planned trajectory that lies outside the camera’s current FOV.

geometric bound for the blind spot area within the current camera FOV (Algorithm 2).

- 2) **Recursive trajectory verification:** Second, for each sampled trajectory, the algorithm recursively constructs new safe frustums within this boundary to certify that the trajectory is collision-free (Algorithm 3).

All newly generated frustums are aggregated for subsequent planning cycles. We assume a static environment, as reasoning about dynamic obstacles in unobserved regions is intractable without additional assumptions.

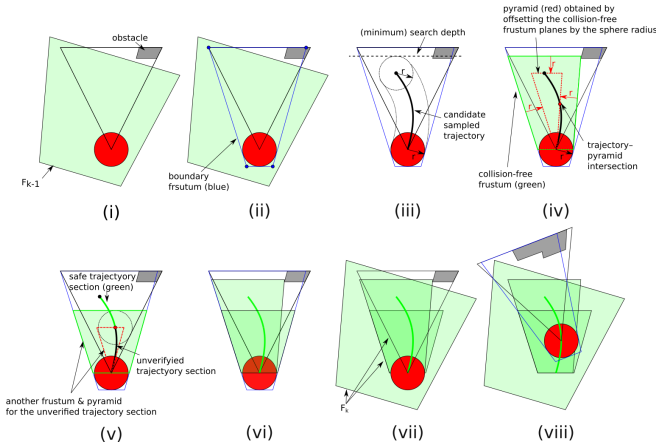


Fig. 3. (i) At planning iteration  $k$ , the algorithm starts with a set of known safe frustums  $\mathcal{F}_{k-1}$ . (ii) A boundary frustum is constructed from the intersection between the known safe frustums and the FOV to limit the search space for new safe frustums and to ensure safety in the blind spot. (iii) A candidate trajectory is generated by selecting a sampled position as its endpoint, where the black sphere represents the quadcopter’s bounding sphere at that location. This process also determines the minimum required searching depth of the potential collision-free frustum, since a valid frustum must fully enclose the sphere at the trajectory endpoint. (iv - v) To verify the candidate trajectory, safe frustums are recursively chained backward from the goal toward the start, with each new frustum certifying the safety of a preceding trajectory segment. (vi) The process repeats until the trajectory is fully verified or the time budget expires. All newly generated frustums are added to the set, yielding  $\mathcal{F}_k$  for the next iteration. (vii - viii) At planning iteration  $k + 1$ , given  $\mathcal{F}_k$ , repeat the same process.

### A. Boundary Frustum for Blind Spot Safety

The first stage of our algorithm, `FINDBOUNDARYFRUSTUM` (Algorithm 2), aims to find the largest possible volume that safely covers the blind spot. It iterates through the set of known safe frustums,  $\mathcal{F}_{k-1}$ , and for each one containing the drone, it calls the `CONSTRUCTBOUNDARY` function

(Line 6). This function, illustrated in Figure 4, generates a candidate boundary frustum. The largest valid candidate is selected as the final boundary frustum,  $B_k$ , for the current planning cycle. This procedure involves the following steps:

- 1) Compute intersection points  $p_i$  between the edges of pyramid  $V_k$  and frustum  $f_{safe}$ . If no intersection exists on an edge, use the corresponding base vertex of  $V_k$  for that edge.
- 2) Construct a convex frustum  $B_{candidate}$  inscribing  $S_k$ , with the far plane determined by the intersection points (or far endpoints), and extend the side planes so that the far plane reaches the base of  $V_k$ . The near plane is chosen tangent to the sphere and parallel to  $V_k$ ’s base.
- 3) Verify geometric validity: (i) all four near-plane vertices lie inside  $f_{safe}$ . (ii) all four far-plane vertices lie inside  $V_k$ . and (iii) the frustum expands along the FOV depth direction, i.e., the near-plane quadrilateral has a smaller area than the far-plane quadrilateral. If any condition fails, set  $B_{candidate} \leftarrow \emptyset$ .

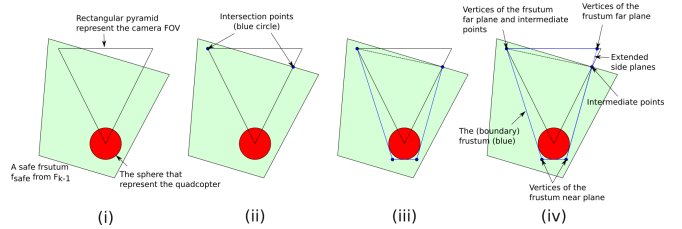


Fig. 4. 2D illustration of the function `CONSTRUCTBOUNDARY` in Algorithm 2 (Line 6). From left to right: (i) given a convex, safe frustum  $f_{safe}$  from the set  $\mathcal{F}_{k-1}$ . (ii) compute the intersection points between the FOV pyramid and  $f_{safe}$  along the pyramid edges, if no intersection exists on an edge, use the corresponding base vertex for that edge. (iii) construct the frustum side planes from these intersection points and define a near plane parallel to the base of the FOV pyramid. (iv) extend the side planes so that the far plane coincides with the base of the FOV pyramid, forming a candidate frustum, and verify the validity of its vertices.

### B. Recursive Trajectory Verification

With a safe boundary  $B_k$  established, the algorithm can verify new candidate trajectories, which are generated using a fifth-degree polynomial sampler, as detailed in [13]. The core of the verification process `CHECKANDADDFRUSTUMFUNCTION` (Algorithm 3) adapts the recursive, frustum-based approach from RAPPIDS [8].

The RAPPIDS method efficiently certifies a trajectory by decomposing it into monotonic sections and finding a chain of safe pyramid to cover them. However, as a memoryless framework, it cannot guarantee safety in unobserved regions (i.e., the blind spot). The original method relies on a heuristic that assumes the space is clear for a fixed distance ahead of the drone. While effective in many scenarios, this approach lacks a formal safety guarantee.

Our work replaces this heuristic with a provably correct method. By first establishing a provably safe boundary frustum  $B_k$  that covers the blind spot, we provide a guaranteed safe volume to operate within. This allows us to use the

efficient frustum-chaining logic of RAPPIDS on a solid foundation of safety.

The key novelty of our work is the CONSTRUCTSAFE-FRUSTUM function, which replaces the risky heuristics of prior methods with a rigorous construction process. This function builds a new, provably safe rectangular frustum within the boundary frustum  $B_k$  by performing the following three steps:

- 1) Searching for a maximal collision-free rectangular base in the depth image, constrained by both perceived obstacles and the boundary frustum  $B_k$ .
- 2) Extruding this base into a rectangular frustum and verifying that all eight of its vertices are contained within  $B_k$ , which is crucial for the safety proof.
- 3) Efficiently checking the entire frustum volume for collisions by using a GPU-accelerated point cloud partitioned into a grid, limiting checks to relevant cells.

Once a containing safe frustum is found, its interior pyramid is used to check for collisions with the trajectory segment. This check reduces to solving a fourth-degree polynomial, which has an efficient closed-form solution. If a collision is found, the safe portion of the trajectory is certified, and the function runs on the remaining segment recursively. If the entire trajectory is certified, it is collision-free.

Finally, to ensure long-term performance and scalability, the set of safe frustums  $\mathcal{F}_k$  is actively managed. To prevent this set from growing indefinitely, we cap its size (30 in our implementation) and employ two pruning strategies. First, frustums that are far from the quadcopter’s current position are discarded as they are irrelevant for near-term planning. Second, geometrically redundant frustums are filtered to maintain a compact and diverse set. These management techniques are essential for maintaining the real-time feasibility of the planner over extended operational periods.

#### IV. SAFETY PROOF

**Lemma 1.** *The boundary frustum,  $B_k$ , is convex and  $B_k \subseteq f_{safe} \cup V_k$ ,  $f_{safe} \in \mathcal{F}_{k-1}$ .*

The constructed boundary frustum  $B_k$  has four near plane vertices  $v_{near} = \{v_1, v_2, v_3, v_4\}$ , four far plane vertices  $v_{far} = \{v'_1, v'_2, v'_3, v'_4\}$  and four intermediate point  $v_m = \{m_1, m_2, m_3, m_4\}$ . By construction  $v_{near} \subset f_{safe}$ , and  $v_m \subset f_{safe}$ . Since  $f_{safe}$  is collision-free and convex, the frustum  $ConvexHull(v_{near}, v_m) \subseteq f_{safe}$ . On the other side,  $v_m \subset V_k$  and  $v_{far} \subset V_k$ , where  $V_k$  is the pyramid that represents the camera FOV, which is convex, the frustum  $ConvexHull(v_m, v_{far}) \subseteq V_k$ . Since  $B_k = ConvexHull(v_{near}, v_m) \cup ConvexHull(v_m, v_{far})$ , therefore,  $B_k \subseteq f_{safe} \cup V_k$ .

Moreover,  $B_k$  is also convex. As detailed in the previous section,  $B_k$  is formed by extending the side planes of an initial frustum to the base of  $V_k$ . This construction ensures the resulting side faces are planar, which guarantees the convexity of  $B_k$ .

---

#### Algorithm 1 FindCollisionFreeTrajectory

---

```

1: Input: Drone pose  $(x_k, R_k)$ , drone sphere model  $S_k(x_k, r)$ , perception data  $(V_k, I_k)$ , previous set of collision-free frustums  $\mathcal{F}_{k-1}$ 
2: Output: Collision-free trajectory  $\xi_{best}$  with lowest cost, updated set of collision-free frustums  $\mathcal{F}_k$ 
3:  $\xi_{best} \leftarrow \emptyset$ ,  $\mathcal{F}_{new} \leftarrow \emptyset$ 
4:  $B_k \leftarrow \text{FINDBOUNDARYFRUSTUM}(S_k(x_k, r), V_k, \mathcal{F}_{k-1})$ 
5: if  $B_k \neq \emptyset$  then
6:   while planning time remains do
7:      $\xi_{sample} \leftarrow \text{SAMPLEFEASIBLETRAJECTORY}(V_k)$ 
8:      $(is\_safe, \mathcal{F}_{new}) \leftarrow \text{CHECKANDADDFRUSTUM}(\xi_{sample}, I_k, B_k, \mathcal{F}_{new})$ 
9:     if  $is\_safe$  and  $Cost(\xi_{sample}) < Cost(\xi_{best})$  then
10:       $\xi_{best} \leftarrow \xi_{sample}$ 
11:    end if
12:  end while
13: end if
14:  $\mathcal{F}_{add} \leftarrow \text{COMMIT}(\mathcal{F}_{new})$ 
15:  $\mathcal{F}_k \leftarrow \mathcal{F}_{k-1} \cup \mathcal{F}_{add}$ 
16: return  $\xi_{best}$ ,  $\mathcal{F}_k = 0$ 

```

---



---

#### Algorithm 2 FindBoundaryFrustum

---

```

1: Input: Drone sphere model  $S_k$ , FOV model  $V_k$ , previous set of collision-free frustums  $\mathcal{F}_{k-1}$ 
2: Output: Frustum  $B_{best}$  with  $S_k \subseteq B$  and  $(B \setminus V_k) \subseteq C_{free}$ 
3:  $B_{best} \leftarrow \emptyset$ 
4: for each rectangular frustum  $f_{safe}$  in  $\mathcal{F}_{k-1}$  do
5:   if  $S_k \subseteq f_{safe}$  then
6:      $B_{candidate} \leftarrow \text{CONSTRUCTBOUNDARY}(S_k, V_k, f_{safe})$ 
7:     if  $B_{candidate} \neq \emptyset$  and  $B_{candidate} \succ B_{best}$  then
8:        $B_{best} \leftarrow B_{candidate}$ 
9:     end if
10:  end if
11: end for=0

```

---

**Corollary 1.** *The constructed rectangular frustum  $f$  is convex and collision-free.*

The frustum  $f$  is constructed as a rectangular frustum, which is a convex shape by definition.

By construction, all eight vertices of  $f$  are contained in the convex set  $B_k$ , which ensures that  $f \subseteq B_k$ . From Lemma 1, we know that  $B_k \subseteq f_{safe} \cup V_k$ , it follows by transitivity that  $f \subseteq f_{safe} \cup V_k$ .

The frustum  $f$  can thus be expressed as the union of its intersections with these two regions:  $f = (f \cap f_{safe}) \cup (f \cap V_k)$ . We verify each region is collision-free:

- 1) The region  $(f \cap f_{safe})$  is a subset of  $f_{safe}$ , which is given as collision-free.
- 2) The region  $(f \cap V_k)$  is verified to be collision-free by checking against obstacle from the depth image,  $I_k$ .

Since both constituent parts of  $f$  are collision-free, the entire frustum  $f$  is proven to be collision-free.

---

**Algorithm 3** CheckAndAddFrustum

---

```
1: Input: Testing trajectory  $\xi$ , drone sphere model  $S_k$ ,  
depth image  $I_k$ , boundary frustum  $B_k$ , the set of convex,  
collision-free rectangular frustum  $\mathcal{F}_{\text{new}}$   
2: Output: boolean value  $is\_safe$ , and the updated  $\mathcal{F}_{\text{new}}$   
3:  $is\_safe \leftarrow false$   
4:  $\mathcal{M} \leftarrow \text{GETMONOTONICSECTION}(\xi)$   
5: while  $\mathcal{M} \neq \emptyset$  do  
6:    $\xi_m \leftarrow \text{POP}(\mathcal{M})$   
7:    $x_{\text{end}} \leftarrow \text{GETDEEPESTPOINT}(\xi_m)$   
8:    $f \leftarrow \text{FINDCONTAININGFRUSTUM}(\mathcal{F}_{\text{new}}, S(x_{\text{end}}, r))$   
9:   if  $f = \emptyset$  then  
10:     $f \leftarrow \text{CONSTRUCTSAFEFRUSTUM}(S(x_{\text{end}}, r), I_k, B_k)$   
11:    if  $f = \emptyset$  then  
12:      return  $false, \mathcal{F}_{\text{new}}$   
13:    else  
14:       $\mathcal{F}_{\text{new}} \leftarrow \text{PUSH}(f)$   
15:    end if  
16:  end if  
17:   $p \leftarrow \text{CONTRACTTOPYRAMID}(f, S(x_s, r))$   
18:   $t \leftarrow \text{FINDDEEPESTCOLLISIONTIME}(p, \xi_m)$   
19:  if  $t \neq \emptyset$  then  
20:     $\mathcal{M} \leftarrow \text{PUSH}(\text{GETSUBSECTION}(\xi_m, t))$   
21:  end if  
22: end while  
    =0
```

---

Corollary 1 is the foundation for the overall safety guarantee of our algorithm. It allows us to establish a key safety invariant by induction:

- 1) **Base case ( $k=0$ ):** We begin with an initial set of convex, collision-free rectangular frustums  $\mathcal{F}_0$ .
- 2) **Inductive step:** At every subsequent iteration  $k$  ( $k>0$ ), assume the given rectangular frustums  $\mathcal{F}_{k-1}$  is safe and each element is convex. From Corollary 1, any new frustum  $f_i$  generated by our method is also guaranteed to be convex and collision-free. And the updated set  $\mathcal{F}_k = \mathcal{F}_{k-1} \cup f_1, f_2, \dots$  is also safe and all elements are convex. This invariant holds for the entire process, providing a formal guarantee that any trajectory planned within these regions will be collision-free.

## V. LIMITATIONS

The proposed algorithm guarantees safety but does not ensure feasibility. The construction of a new safe frustum depends on the presence of a valid boundary frustum. As the trajectory approaches the side planes or the far plane of the current frustum, this feasibility may diminish, particularly in cluttered environments. This limitation reflects the lack of information needed to guarantee safety within blind spots. Figure 5 illustrates a representative scenario where this issue can arise.

One possible mitigation is to introduce a cost function that penalizes trajectories passing too close to frustum boundaries, even when they remain collision-free. Alternatively,

more advanced planning strategies could explicitly address these boundary conditions. Designing such approaches and evaluating their impact on feasibility remain important directions for future research.

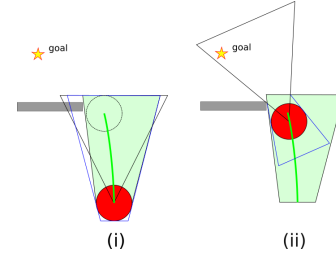


Fig. 5. Illustration of one possible scenario where the boundary frustum cannot be constructed, causing the quadcopter to become stuck. (i) A collision-free trajectory is successfully planned with an endpoint closest to the goal. (ii) Because the quadcopter flies too close to the side plane of the safe frustum, it becomes infeasible to construct a valid boundary frustum.

## VI. SIMULATION VERIFICATION

The proposed algorithm was implemented and verified in a simulator developed by [14]. The test environment follows a setup similar to [10], where cylinders of 0.6 m diameter are randomly placed in a 30 m by 60 m area to mimic a forest with density 1/25 [tree/m<sup>2</sup>]. Cylinder locations are sampled randomly for each trial. The quadcopter starts at (0, 0, 0) m, hovers at 1.5 m altitude, and aims to reach a goal at (65, 0, 1.5) m.

The drone is equipped with a forward-looking depth camera with a 4 m sensing range, and the maximum number of safe frustums maintained is 30. We adopt an unbiased random sampling strategy when generating trajectory candidates, without any global reference. A goal-driven cost function is used, where trajectories whose endpoints are closer to the goal have lower cost. The yaw is controlled to remain aligned with the velocity direction.

Simulations were executed on a laptop with an Intel(R) Core(TM) i7-6820HQ CPU. Our planner's core verification stage operates within a fixed 60 ms budget, while the pre-processing time varies with scene complexity. Over our test runs, this resulted in a planning frequency ranging from 10 Hz to a maximum of 15 Hz, a rate limited by the simulated depth camera's frame rate.

While we have not performed onboard deployment, the planner's computation structure is similar to RAPPIDS [8]. The primary additional cost arises from frustum collision checking, which can be efficiently handled by projecting the depth image into a point cloud, which could be accelerated using a GPU, and by employing spatial data structures such as k-d trees or voxel grids for fast queries. Frustum management and boundary frustum computation are negligible (less than 1 ms on the reference laptop). Given that the core planning pipeline already runs within a fixed 60 ms budget, these considerations suggest that real-time onboard implementation is feasible on modern embedded platforms such as Nvidia Jetson or Qualcomm RB5. Evaluating the planner on actual embedded hardware is left for future work.

To provide an initial quantitative evaluation, we conducted a total of 20 simulation runs. These tests were performed across 10 unique environments, each generated with a different random seed. To assess the consistency of the planner’s stochastic sampling strategy, each environment was tested twice. The primary finding from these trials was the practical validation of our theoretical safety guarantee. Critically, zero collisions were recorded across all 20 runs. The planner successfully navigated to the goal in 30% of trials. This success rate is consistent with the known limitations of a local, random-sampling planner. Furthermore, it is a direct consequence of the feasibility limitation discussed in Section V, where a new boundary frustum cannot be constructed if the drone approaches the existing corridor walls.

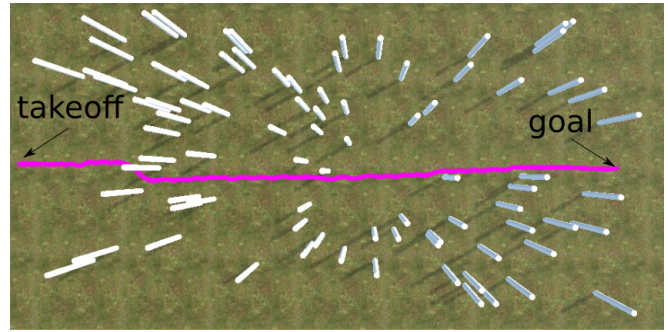
Figure 6 shows two representative runs in different random scenes. In the first run, the quadcopter successfully reaches the goal without collision. In the second, the quadcopter becomes trapped because a valid boundary frustum could not be constructed. However, it remains entirely within safe space throughout the flight. These examples illustrate that, even in challenging scenarios with perceptual blind spots, the planner preserves the safety guarantee derived in our theoretical analysis. Figure 7 visualizes the generated safe frustum set, the boundary frustum, the FOV, and the trajectory during operation.

These preliminary results demonstrate the robustness of the safety framework. A clear direction for future work is to incorporate a more advanced sampling strategy to improve the goal-reaching rate and to enhance the robustness of the boundary frustum construction, all while retaining the formal safety proofs of the underlying framework.

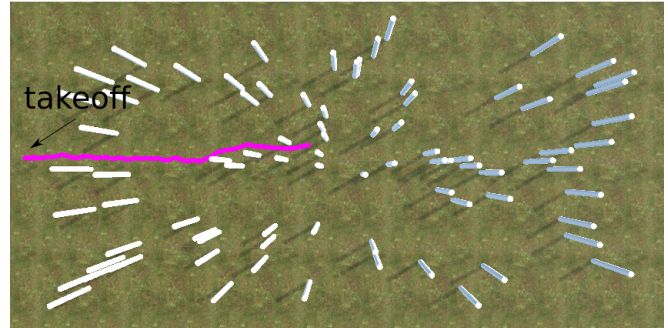
## VII. CONCLUSION AND FUTURE WORK

In this paper, we introduced a novel iterative planner that leverages frustums as a lightweight geometric representation of free space. The framework provides a formal safety guarantee that explicitly accounts for the limitations of a limited FOV sensor, such as a depth camera, which is a critical challenge in autonomous navigation. We formally established the algorithm’s safety properties and validated them through simulation studies.

Our design prioritizes safety, which can cause the planner to behave conservatively in complex scenarios where guaranteed forward progress is not possible. This limitation suggests several directions for future work. First, the computational performance of our proof-of-concept implementation could be improved, enabling more trajectories to be evaluated within a given computational budget, as frustum construction remains the primary bottleneck. Second, the framework could be extended with more advanced planning strategies to enhance feasibility and robustness in challenging environments, while maintaining formal safety guarantees. Finally, we plan to implement and test the framework on a physical quadcopter to validate the safety guarantees under real-world conditions, including sensor noise and unmodeled dynamics.



(i)



(ii)

Fig. 6. Two runs in the forest scene. Flight trajectories are shown in purple. (i) The quadcopter successfully reaches the goal. (ii) The quadcopter becomes trapped due to inability to construct valid boundary frustums, but remains collision-free.

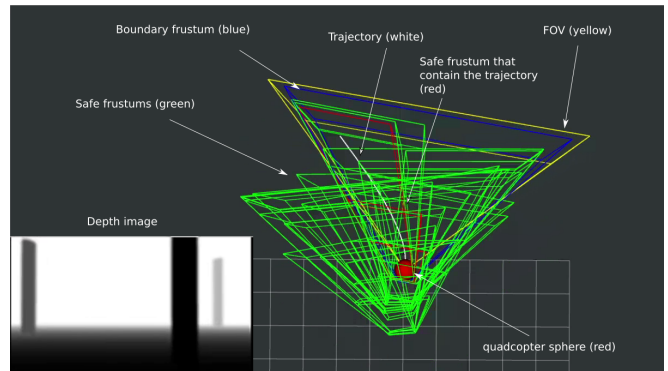


Fig. 7. Visualization of the iterative corridor construction process.

## ACKNOWLEDGMENT

This work was supported by the California Department of Transportation (Caltrans).

## REFERENCES

- [1] S. Jordan, J. Moore, S. Hovet, J. Box, J. Perry, K. Kirsche, D. Lewis, and Z. T. H. Tse, “State-of-the-art technologies for uav inspections,” *IET Radar, Sonar & Navigation*, vol. 12, no. 2, pp. 151–164, 2018.
- [2] R. Deits and R. Tedrake, *Computing Large Convex Regions of Obstacle-Free Space Through Semidefinite Programming*. Cham: Springer International Publishing, 2015, pp. 109–124. [Online]. Available: [https://doi.org/10.1007/978-3-319-16595-0\\_7](https://doi.org/10.1007/978-3-319-16595-0_7)
- [3] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, “Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments,” *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

- [4] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 922–938, 2021.
- [5] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.
- [6] C. Toumih and D. Floreano, "High-speed motion planning for aerial swarms in unknown and cluttered environments," *IEEE Transactions on Robotics*, vol. 40, pp. 3642–3656, 2024.
- [7] B. T. Lopez and J. P. How, "Aggressive 3-d collision avoidance for high-speed navigation," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5759–5765.
- [8] N. Bucki, J. Lee, and M. W. Mueller, "Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4626–4633, 2020.
- [9] T. B. Nguyen, M. Murshed, T. Choudhury, K. Keogh, G. Kahan-dawa Appuhamillage, and L. Nguyen, "A depth-based hybrid approach for safe flight corridor generation in memoryless planning," *Sensors*, vol. 23, no. 16, p. 7206, 2023.
- [10] Y. Ren, F. Zhu, W. Liu, Z. Wang, Y. Lin, F. Gao, and F. Zhang, "Bubble planner: Planning high-speed smooth quadrotor trajectories using receding corridors," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 6332–6339.
- [11] J. Ji, Z. Wang, Y. Wang, C. Xu, and F. Gao, "Mapless-planner: A robust and fast planning framework for aggressive autonomous flight without map fusion," in *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2021, pp. 6315–6321.
- [12] Y. Ren, F. Zhu, G. Lu, Y. Cai, L. Yin, F. Kong, J. Lin, N. Chen, and F. Zhang, "Safety-assured high-speed navigation for mavs," *Science Robotics*, vol. 10, no. 98, p. eado6187, 2025. [Online]. Available: <https://www.science.org/doi/abs/10.1126/scirobotics.ado6187>
- [13] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadcopter trajectory generation," *IEEE transactions on robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.
- [14] T. Yang and M. W. Mueller, "Agrinav: Uav simulator for vision-based navigation in agricultural environments," in *Proceedings of the 9th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture (AGRICONTROL)*, 2025, to appear.