Faster Algorithms for Growing Collision-Free Convex Polytopes in Robot Configuration Space

Peter Werner, Thomas Cohn⋆, Rebecca H. Jiang⋆, Tim Seyde, Max Simchowitz, Russ Tedrake, and Daniela Rus

Massachusetts Institute of Technology, Cambridge, MA 02139
{wernerpe,tcohn,rhjiang,tseyde,msimchow,russt,rus}@mit.edu

Abstract. We propose two novel algorithms for constructing convex collision-free polytopes in robot configuration space. Finding these polytopes enables the application of stronger motion-planning frameworks such as trajectory optimization with Graphs of Convex Sets [1] and is currently a major roadblock in the adoption of these approaches. In this paper, we build upon IRIS-NP (Iterative Regional Inflation by Semidefinite & Nonlinear Programming) [2] to significantly improve tunability, runtimes, and scaling to complex environments. IRIS-NP uses nonlinear programming paired with uniform random initialization to find configurations on the boundary of the free configuration space. Our key insight is that finding near-by configuration-space obstacles using sampling is inexpensive and greatly accelerates region generation. We propose two algorithms using such samples to either employ nonlinear programming more efficiently (*IRIS-NP2*) or circumvent it altogether using a massively-parallel zero-order optimization strategy (*IRIS-ZO*). We also propose a termination condition that controls the probability of exceeding a user-specified permissible fraction-in-collision, eliminating a significant source of tuning difficulty in IRIS-NP. We compare performance across eight robot environments, showing that IRIS-ZO achieves an order-of-magnitude speed advantage over IRIS-NP. IRIS-NP2, also significantly faster than IRIS-NP, builds larger polytopes using fewer hyperplanes, enabling faster downstream computation.

Keywords: collision-free motion planning, robot configuration space

1 Introduction

A major challenge in robot motion planning is the need to simultaneously consider *task space*, the world in which the robot physically resides, and *configuration space*, $\mathcal{C}$, the set of all possible robot configurations. The planner must produce a trajectory in the configuration space, but many constraints are formulated in the task space. Collision avoidance is particularly challenging, because even geometrically simple obstacles in task space can have intractably complicated descriptions when transformed into configuration space through the robot's inverse

---

⋆ Denotes equal contribution.

kinematics. While there is work on constructing explicit configuration-space representations of workspace obstacles [3,4,5], these methods are intractable for the high degree-of-freedom (dof) robotic systems being used today.

The unavailability of obstacles' configuration-space descriptions has not prevented the development of a rich literature of motion planning algorithms. Many approaches approximate the set of collision-free configurations ($\mathcal{C}^{\text{free}}$) without explicitly constructing the individual obstacles. Interval analysis and cell decompositions can approximate $\mathcal{C}^{\text{free}}$ as the union of boxes [6][7, §5-6], but these methods are computationally intractable for high-dimensional configuration spaces. Perhaps the most widely used technique has been sampling-based planning, in which samples are drawn from $\mathcal{C}^{\text{free}}$ and connected into a graph structure.

These representations of $\mathcal{C}^{\text{free}}$ are popular due to their simplicity and versatility. However, they all struggle with the "curse of dimensionality" – the memory use of the representation may grow exponentially with the dimension of the configuration space. This has led the motion planning community to explore volumetric approximations of $\mathcal{C}^{\text{free}}$ such as the union of spheres used by Yang and LaValle [8] or the polytopes constructed by Deits and Tedrake [9]. In contrast to the grid- or sampling-based approaches, the individual sets used in these approaches describe free-space regions rather than points. Although these sets are harder to construct, they often enable a more concise approximation of $\mathcal{C}^{\text{free}}$ and yield convex (probabilistic) collision-avoidance constraints.

Motion planning algorithms such as trajectory optimization with *Graph of Convex Sets* (GCSTrajOpt) [1] leverage these representations to quickly produce high-quality, collision-free trajectories for high-dimensional robotic systems. However, the performance of these planners is highly dependent on the properties of the convex sets. It is desirable for these sets to have large volumes in order to reduce the number of sets required to approximate $\mathcal{C}^{\text{free}}$, while retaining simple descriptions to make downstream planning more efficient. While creating perfectly collision-free sets is very costly, the new algorithms should provide a straightforward way to trade off between precision and runtime.

Recent results for planning with GCSTrajOpt leverage the IRIS-NP algorithm [1,10,11,12]. IRIS-NP takes in a collision-free *seed* configuration and attempts to build a convex, collision-free polytope surrounding it. IRIS-NP, however, falls short of meeting the aforementioned criteria in practice. Its runtime is substantial and trading off between runtime and correctness (how much of the region is collision-free) proves challenging. IRIS-NP only terminates after failing to solve a user-specified number of nonlinear programs in succession, a time-consuming process dependent on this user-specified parameter acting as a proxy for correctness, when the actual relationship is unclear. Furthermore, these nonlinear programs must be run separately for every object in the scene, causing IRIS-NP to scale poorly with environment complexity.

In this paper, we improve upon IRIS-NP, focusing on a key subroutine that constructs hyperplanes to separate the seed point from obstacles. Our improvements leverage the fact that we can evaluate thousands of configurations for collisions in the time it takes to solve a single nonlinear optimization problem.

We employ random sampling and collision checking to estimate the proportion of the polytope that is collision-free, providing a rigorous probabilistic certificate, and enabling intuitive tradeoffs between region correctness and computation times. We further present two algorithms that utilize sampling and collision checking to improve polytope generation: IRIS-ZO rapidly generates polytopes using a simple parallelized zero-order optimization strategy that requires no gradient computations and is easy to implement. IRIS-NP2 uses sampled collisions to seed nonlinear optimizations, increasing search success, and dramatically reducing the required number of programs. We demonstrate that both algorithms outperform IRIS-NP in terms of computation time and region quality.

2   Problem Formulation

In this section, we introduce the problem formulation and discuss the required inputs and provided outputs of our proposed algorithms.

We aim to generate large convex polytopes in configuration space whose fraction in collision is less than a user-provided constant. More precisely, let λ denote the Lebesgue measure in the free configuration space $\mathcal{C}^{\text{free}}$. Given a user-specified admissible fraction in collision $\varepsilon \in (0,1)$ and confidence $\delta \in (0,1)$, we will compute positive-volume convex polytopes $\mathcal{R} \subseteq \mathcal{C}$ that satisfy

$$\mathbf{Pr}\left[\frac{\lambda(\mathcal{R} \setminus \mathcal{C}^{\text{free}})}{\lambda(\mathcal{R})} > \varepsilon\right] \leq \delta. \tag{1}$$

Since obtaining a closed-form description of $\mathcal{C}^{\text{free}}$ is intractable for general robotic systems [13, §4.3.3],[7, §3], our algorithms utilize a task-space description. Such descriptions are readily provided via common robot description formats such as URDFs [14] or SDFs [15]. We expect the robot to be described as M sets $\mathcal{G}_i \subseteq \mathbb{R}^{N_{\text{ts}}}$ representing collision geometries in task space. Here, $N_{\text{ts}} \in \{2,3\}$ is the dimension of the task space, and $q \in \mathcal{C}$ is the configuration. The sets $\mathcal{G}_i$ are represented in their body frame B_i. Using the monogram notation of [16, §3.1], each $\mathcal{G}_i$ is paired with a configuration-dependent rigid-body transformation $^W X^{B_i}(q) \in SE(N_{\text{ts}})$ that defines the forward kinematics of $\mathcal{G}_i$ in the world frame W. We use the shorthand $\mathcal{G}_i(q)$ for $\mathcal{G}_i$ expressed in world frame:

$$\mathcal{G}_i(q) = \{p_W \in \mathbb{R}^{N_{\text{ts}}} |\ p_{B_i} =\ ^{B_i} X^W(q) \cdot p_W \in \mathcal{G}_i\}.$$

We let $\mathcal{G}(q) = \{\mathcal{G}_1(q), \mathcal{G}_2(q), \ldots, \mathcal{G}_M(q)\}$ be the set of all collision geometries.

For a configuration q, the system is *in collision* if there exists a *valid* [1]pair of collision geometries $\mathcal{A}(q), \mathcal{B}(q) \in \mathcal{G}(q)$ such that $\mathcal{A}(q) \cap \mathcal{B}(q) \neq \emptyset$. For a valid collision pair $\mathcal{A}(q), \mathcal{B}(q) \in \mathcal{G}(q)$, the corresponding *configuration-space obstacle* $\mathcal{O}^{\mathcal{AB}}(q)$ is implicitly defined as

$$\mathcal{O}^{\mathcal{AB}}(q) := \{q \mid \mathcal{A}(q) \cap \mathcal{B}(q) \neq \emptyset\} \tag{2}$$

We avoid explicitly describing $\mathcal{O}^{\mathcal{AB}}$ by using nonconvex task-space constraints, as in [2].

---

[1] Typically, only a subset of all collision pairs is considered for practical reasons.

3   Related Works

Algorithms for computing positive-volume subsets of $\mathcal{C}^{\text{free}}$ can be divided into two categories: those which require explicit descriptions of the obstacles in configuration space, and those that can use implicit descriptions. Explicit descriptions of obstacles are generally only available for robots with simple kinematics (e.g. only prismatic joints).

If descriptions of all obstacles are given as convex sets, the original IRIS algorithm [9] can construct large collision-free polytopes about a seed points using a series of convex optimizations. Such descriptions can be obtained from arbitrary meshes via approximate convex decomposition techniques [17]. Wu et. al. [18] use a similar approach to grow convex polytopes around an existing trajectory, towards producing a shorter, collision-free path.

Due to the complex kinematics of robotic manipulators, obstacles in configuration space are frequently given by implicit descriptions. Yang and LaValle leveraged the kinematic Jacobian to relate motion in configuration-space and task space, allowing the construction of collision-free ellipsoids [8]. Unfortunately, large numbers of ellipsoids are required to approximate even simple, low-dimensional configuration spaces.

The original IRIS algorithm has been extended to handle such implicit descriptions in two ways. IRIS-NP uses nonlinear programming to find multiple locally separating hyperplanes to each obstacle, until the program becomes infeasible [2] – an imprecise termination condition that often leaves some obstacle volume in regions. Jaitly and Farzan instead used a nonuniform sampling strategy to seed the collision search program [10]. The other option is to use a rational reparametrization of the kinematics to construct regions that are rigorously certified to be collision-free with sums-of-squares programming [19]. However, such optimizations are computationally expensive, and the regions are grown in a stereographic projection of configuration space, which distorts distances.

Alternatively, [20] directly decomposes three-dimensional spaces into polytopes only using sample-based collision-checking. Unfortunately, this approach requires dense sampling of the configuration space to produce large sets which is intractable in all but the simplest cases.

4   Background on IRIS Algorithms

In this section, we review the key algorithms our approach builds upon. The IRIS algorithm [9] takes in a list of convex obstacles and produces a collision-free polytope around a seed point in the same space as these obstacles. IRIS-NP [2] generalizes IRIS to operate in configuration space using nonlinear programming.

4.1   IRIS with Convex Obstacles

IRIS [9], takes as input N_o convex obstacles $\mathcal{O}_i$ and a collision-free seed point $s \in \mathcal{C}^{\text{free}}$. IRIS then computes a large collision-free polytope by alternating between two convex optimizations: the SEPARATINGPLANES step, that produces a

polytope conditioned on an initial ellipsoid, and the INSCRIBEDELLIPSOID step, which updates the ellipsoid to the maximum-volume inscribed ellipsoid (MVIE) [21, §8.4.2] in the current polytope. These alternations guarantee the containment of the ellipsoid, and hence, guarantee that the volume of the MVIE is monotonically increasing across alternations. To understand the upcoming modifications to IRIS-NP, we review the SEPARATINGPLANES step in detail.

The SEPARATINGPLANES step starts with a collision-free ellipsoid $\mathcal{E}$,

$$\mathcal{E} = \{x \mid (x - c)^T E(x - c) \leq 1, \ E \succ 0\}, \tag{3}$$

that is centered at c with a symmetric, positive-definite matrix E. For each of the N_o obstacles, IRIS computes a hyperplane that separates c from $\mathcal{O}_i$ and passes through the point in $\mathcal{O}_i$ that lies closest to the ellipsoid center c in the metric of the ellipsoid. This is done by solving

$$\textbf{minimize } ||x_i - c||_E^2 \textbf{ subject to } x_i \in \mathcal{O}_i, \tag{4}$$

where $||x||_E^2 := x^T E x$. Given the optimum $x_i^\star$, the hyperplane

$$\mathcal{H}_i = \{x | a_i^T x - b_i = 0\}, \qquad a_i = E(x_i^\star - c), \ b_i = -a_i^T x_i^\star \tag{5}$$

separates c from $\mathcal{O}_i$ and does not intersect $\mathcal{E}$ [9, §3.5]. This step is performed for each obstacle. Finally, the intersection of the halfspaces $a_i^T x \leq b_i$ yields an updated collision-free polytope.

4.2 Computing Separating Planes in Robot Configuration Space

$$\textbf{minimize } ||q - c||_E^2 \textbf{ subject to } t \in \mathcal{A}(q) \cap \mathcal{B}(q), \ q \in \mathcal{P} \tag{6}$$
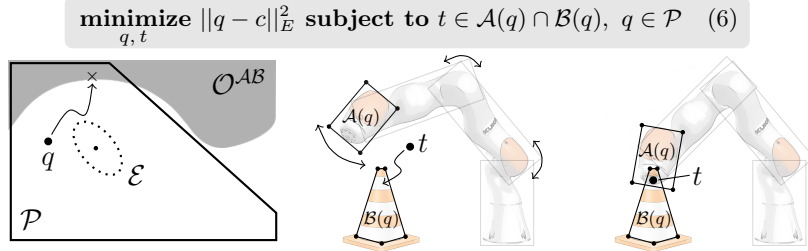
Fig. 1: The nonlinear closest-point-in-obstacle program seeks to find the closest configuration q, under the ellipsoidal metric $\mathcal{E}$, that lies inside of the current polytope $\mathcal{P}$ and causes the collision pair $\mathcal{A}$, $\mathcal{B}$ to collide. Instead of constraining $q \in \mathcal{O}^{\mathcal{AB}}$ directly, we encode the collision constraint in task space with an auxiliary point t that is constrained to lie in the in the intersection of both collision geometries. The picture on the right shows a feasible solution. A locally optimal solution to the program is indicated by the cross in the cartoon on the left.

IRIS requires convex obstacles to ensure the convexity of (4). However, configuration-space obstacles are generally non-convex. IRIS-NP [2] generalizes the closest-point-in-obstacle program (4) for the configuration-space obstacle

$\mathcal{O}^{\mathcal{AB}}$. To avoid intractable descriptions of $\mathcal{O}^{\mathcal{AB}}$, the collision constraint is encoded in task space. Optimization variables represent a point in each of the collision geometries, $t_{\mathcal{A}} \in \mathcal{A}$, $t_{\mathcal{B}} \in \mathcal{B}$, in their body frames, constrained by $^W X^{B_{\mathcal{A}}} t_{\mathcal{A}} = {}^W X^{B_{\mathcal{B}}} t_{\mathcal{B}}$, to coincide in task space when transformed through the forward kinematics. For clarity, in this paper, we write this as requiring a single point t to lie in the intersection of the associated pair of collision geometries in task space. This is shown in Fig. 1. In order to ensure that new hyperplanes address obstacles still relevant to the current polytope $\mathcal{P}$, IRIS-NP requires the found points in collision to lie inside $\mathcal{P}$. The full nonlinear program then reads:

$$\underset{q,t}{\textbf{minimize}} \quad ||q - c||_E^2 \tag{6a}$$

$$\textbf{subject to} \quad t \in \mathcal{A}(q) \cap \mathcal{B}(q), \quad q \in \mathcal{P}. \tag{6b}$$

Because configuration-space obstacles are generally non-convex, multiple hyperplanes may be necessary to separate c from $\mathcal{O}^{\mathcal{AB}}$.
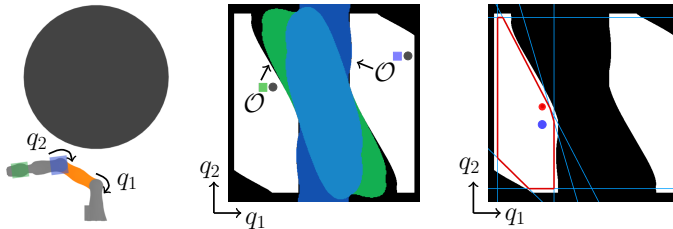

Fig. 2: A visualization of an IRIS-NP region for a simple two dof system. *Left*: Two dof robot arm with a disk shaped obstacle. Two selected collision geometries of the system are highlighted in green and blue. *Center*: Configuration space of the system. The black regions correspond to collisions. Two of the configuration-space obstacles are highlighted in blue and green, corresponding to the configurations where the blue and green rectangle intersect the disk obstacle. *Right*: Resulting region (red outline), when seeding at the red dot. The configuration in left frame is shown by the blue dot.

To construct the separating hyperplanes for a single collision pair, $\mathcal{A}$ and $\mathcal{B}$, IRIS-NP repeatedly finds locally optimal solutions $q^\star$ to (6), adding the face

$$\mathcal{H}_i = \{x | a_i^T q^\star - b_i + \Delta = 0\}, \qquad a_i = E(q^\star - c), \ b_i = -a_i^T q^\star, \tag{7}$$

to $\mathcal{P}$ for each of the found points $q^\star$ in collision. More precisely, the polytope $\mathcal{P}$ is updated by intersecting $\mathcal{P}$ with the halfspace $\{q | a_i^T q \leq b_i - \Delta\}$, where $\Delta > 0$ is a user-specified stepback. This stepback ensures that the collision configuration $q^\star$ is excluded from $\mathcal{P}$, and prevents the need for an infinite number of faces to exclude a non-convex obstacles. This is repeated until a fixed number of attempted solves of (6) fail; IRIS-NP interprets this failure as a suggestion that $\mathcal{P}$ is sufficiently collision-free. Note that due to the local nature of nonlinear programming, a single failed solve often provides inconclusive information about the feasibility of the overall program.

To summarize, the IRIS-NP strategy to approximating SEPARATINGPLANES in robot configuration space is to create a list of all valid collision pairs ordered by their task-space distance, and, for each collision pair, repeatedly find locally optimal solutions to (6) and update $\mathcal{P}$ until a user-specified number of consecutive solve attempts fail. Through this procedure, IRIS-NP strives to make $\mathcal{P}$ collision-free with respect to all valid collision pairs. See Fig. 2 for an illustration.

5   Improving the Separating Hyperplanes Routine

This section discusses how we leverage sampling to produce a rigorous termination condition for SEPARATINGPLANES in robot configuration space (see sec. 4.2), and two improved approaches for solving the step, yielding the new algorithms IRIS-ZO and IRIS-NP2. Both of these new formulations follow the same alternation scheme as IRIS-NP, given in Alg. 1. In particular, we still initialize the

---

Algorithm 1: Template for the new IRIS algorithms

> **Input:** Domain $\mathcal{D} \subseteq \mathcal{C}$, collision-free seed $s \in \mathcal{D}$, options O.
> **Output:** Polytope $\mathcal{P} \subseteq \mathcal{D}$
> **Algorithm:** $\mathcal{E} \leftarrow \text{BALL}(s, r_{\text{start}})$, $i \leftarrow 1$
> **while** *not done* **do**
> > $\mathcal{P} \leftarrow \text{SEPARATINGPLANES}(\mathcal{D}, \mathcal{E}, i, O)$ ▷ **employs new term. cond.**
> > $\mathcal{E} \leftarrow \text{INSCRIBEDELLIPSOID}(\mathcal{P})$
> > $i \leftarrow i + 1$
>
> **end**
> **return** $\mathcal{P}$

---

ellipsoid with a small ball of radius r_{start} and employ the same overall termination conditions for the alternations as [2, §II.D], such as the seed point s no longer being contained in $\mathcal{P}$, reaching a maximum number of alternations, or achieving convergence of the ellipsoid volume. Our proposed termination condition for SEPARATINGPLANES decides when $\mathcal{P}$ meets a correctness criterion based on polytope fraction in collision, not when to terminate the alternations. We discuss the proposed termination condition in Sec. 5.1, before discussing IRIS-ZO in Sec. 5.2, and IRIS-NP2 in Sec. 5.3.

Parameter	Description	IRIS-ZO	IRIS-NP2
ε	Admissible fraction of the region in collision	✓	✓
δ	Max admissible uncertainty	✓	✓
Δ	Configuration margin, i.e. "step back"	✓	✓
T	Termination condition (as in [2])	✓	✓
N_p	Number of optimized particles per inner iteration	✓	✓
N_b	Number of bisection steps	✓	✗
N_f	Max number of faces added per inner iteration	✓	✓

Table 1: Glossary of algorithm parameters.

5.1   Termination Condition for the Separating Planes Step

In this section, we discuss a termination condition that allows a user to specify a desired bound on the fraction of the volume of the polytope $\mathcal{P}$ in collision. In

the following, let $\varepsilon_{tr} := \lambda(\mathcal{P} \setminus \mathcal{C}^{\text{free}})/\lambda(\mathcal{P})$ denote the true fraction in collision of $\mathcal{P}$, where λ is the Lebesgue measure over $\mathcal{C}^{\text{free}}$, and ε is the specified admissible fraction in collision. A naive approach may be to sample uniformly in the polytope $\mathcal{P}$, estimate the fraction of $\mathcal{P}$ in collision $\hat{\varepsilon}$ to be equal to the fraction of these samples that are in collision, and terminate if $\hat{\varepsilon} \leq \varepsilon$. However, each time we perform this check, there is some chance of underestimating ε_{tr} such that we incorrectly terminate with a polytope with $\varepsilon_{tr} > \varepsilon$. The probability of false termination accumulates over the multiple evaluations of this condition.

Instead, we propose a statistical test that controls the probability of falsely claiming a polytope is sufficiently collision-free and terminating prematurely. For some user-specified uncertainty δ, a correct termination condition guarantees $\mathbf{Pr}\left[\varepsilon_{tr} > \varepsilon\right] \leq \delta$ for the region we return. To accomplish this, we pair union bounds with a simple statistical test based on a Chernoff bound.

We sample a batch of M points $q_i \in \mathcal{P}$ uniformly. We then assign $X_i = 0$ if $q_i \in \mathcal{C}^{\text{free}}$ and $X_i = 1$ otherwise. Note that $X_i \sim \text{Bernoulli}(\varepsilon_{tr})$. The employed bound reads as follows.

Theorem 1 (Sample Bound). *Let $\varepsilon_{tr} \geq \varepsilon \geq 0$, $(X_i)_{M \geq i \geq 1} \sim \text{Bernoulli}(\varepsilon_{tr})$, with $M \geq 2\log(1/\delta)/(\varepsilon\tau^2)$, for any fixed parameter $\tau > 0$, and $\delta \in (0,1]$. Define $\bar{X}_M := \frac{1}{M}\sum_{i=1}^{M} X_i$. It holds that*

$$\mathbf{Pr}\left[\bar{X}_M \leq (1-\tau)\varepsilon\right] \leq \delta. \tag{8}$$

Proof Observe that $\mathbf{Pr}\left[\bar{X}_M \leq (1-\tau)\varepsilon\right] \leq \mathbf{Pr}\left[\bar{X}_M \leq (1-\tau)\varepsilon_{tr}\right]$. If $\tau > 1$ both sides evaluate to 0, and otherwise $(1-\tau)\varepsilon \leq (1-\tau)\varepsilon_{tr}$. Next, we use the standard multiplicative Chernoff bound, $\mathbf{Pr}\left[\bar{X}_M \leq (1-\tau)\varepsilon_{tr}\right] \leq e^{-M\varepsilon_{tr}\tau^2/2}$, e.g. given in [22, §A, Thm A.1.15], and use the fact that $e^{-M\varepsilon_{tr}\tau^2/2} \leq e^{-M\varepsilon\tau^2/2}$. We have now shown that $\mathbf{Pr}\left[\bar{X}_M \leq (1-\tau)\varepsilon_{tr}\right] \leq e^{-M\varepsilon\tau^2/2}$, and the final result follows from evaluating $M = 2\log(1/\delta)/(\varepsilon\tau^2)$. $\qquad\square$

Definition 1 (Unadaptive Test). *Let $\mathsf{UnadaptiveTest}(\delta, \varepsilon, \tau)$ collect $M = \lceil 2\log(1/\delta)/\varepsilon\tau^2 \rceil$ samples $(X_i)_{1 \leq i \leq M}$. $\mathsf{UnadaptiveTest}$ returns `reject` if $\bar{X}_M > (1-\tau)\varepsilon$, and `accept` otherwise.*

Corollary 1 (Controlling False Accept). *Suppose $\varepsilon_{tr} \geq \varepsilon$, i.e. the true fraction in collision is higher than allowed, and we have $(X_i)_{1 \leq i \leq M} \sim \text{Bernoulli}(\varepsilon_{tr})$. Then, $\mathsf{UnadaptiveTest}(\delta, \varepsilon, \tau)$ rejects with probability at least $1 - \delta$.*

Thus, our test behaves as follows: Assume we are given a polytope $\mathcal{P}$, an admissible fraction in collision ε, an admissible target uncertainty δ, and a batch of M uniform samples in $\mathcal{P}$ with $M = \lceil 2\log(1/\delta)/(\varepsilon\tau^2) \rceil$. If it holds that $\mathcal{P}$ does *not* meet the specifications, i.e. $\varepsilon_{tr} \geq \varepsilon$, then $\mathsf{UnadaptiveTest}(\delta, \varepsilon, \tau)$ rejects $\mathcal{P}$ with probability greater than $1 - \delta$; the test only falsely accepts $\mathcal{P}$ with probability less than δ. Observe that if τ decreases, we require many samples and the test becomes expensive. Conversely, if we increase τ, the number of samples decreases, but the test starts rejecting every $\mathcal{P}$. In practice, we choose $\tau = 0.5$ to balance power and computational cost of the test.

Within SeparatingPlanes, for both algorithms, faces are iteratively added to $\mathcal{P}$ until sufficient separation from the configuration-space obstacles is achieved. We refer to these iterations as *inner iterations*. In order to decide whether the polytope is sufficiently collision-free at the k-th inner iteration, we run UnadaptiveTest$(\delta_k, \varepsilon, \tau)$. We can bound the probability of SeparatingPlanes terminating falsely after K iterations by the union bound

$$\mathbf{Pr}(\text{``false accept''} \text{ if } \lambda(\mathcal{P} \setminus \mathcal{C}^{\text{free}})/\lambda(\mathcal{P}) > \varepsilon) \leq \sum_{k=1}^{K} \delta_k. \tag{9}$$

In order to control the probability of falsely accepting, i.e. terminating, after an unknown number of inner iterations, in the case where a single outer iteration is to be run, we simply select δ_k according to the sequence $\delta_k = 6\delta/(\pi^2 k^2)$ for the k-th test, as this sequence sums to δ as $k \to \infty$. If an unknown number of *outer iterations* (i.e. alternations between updating the polytope and the ellipsoid) are run, we must account for the additional UnadaptiveTest queries. In order for the probability of falsely accepting to sum to δ over infinite inner and outer iterations, we run UnadaptiveTest$(\delta_{i,k}, \varepsilon, \tau)$, in the i-th outer iteration, after the k-th inner iteration, with $\delta_{i,k} := 36\delta/(\pi^4 i^2 k^2)$.

Note that this termination condition assumes uniform samples in $\mathcal{P}$. In practice, we generate approximately uniform samples using hit-and-run sampling [23] with a sufficiently high number of mixing steps.
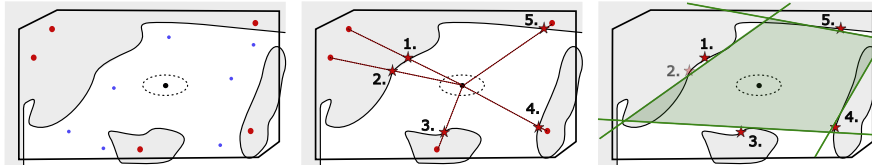
5.2 The IRIS-ZO Algorithm

Fig. 3: One iteration of ZeroOrderSeparatingPlanes. *Left*: Configuration space with grey configuration-space obstacles. First, a batch of samples is drawn uniformly in the current polytope and checked for collisions. *Center*: The found collisions $\mathcal{S}_{\text{col}}$, shown in red, are brought closer to the center of the ellipsoid using bisection. The updated candidate points $\mathcal{S}_{\text{col}}^{\star}$ (red stars), are sorted by their weighted squared distance to the center of the ellipsoid under the ellipsoidal metric. *Right* - Hyperplanes are iteratively added for the non-redundant candidates in order. Note that the second candidate is made redundant by the first hyperplane, and therefore only 4 faces are added to $\mathcal{P}$.

The IRIS-ZO algorithm uses a simple parallelized zero-order optimization strategy to directly solve SeparatingPlanes in Alg. 1 for all collision pairs simultaneously. We call this subroutine ZeroOrderSeparatingPlanes and summarize it in Alg. 2. Fig. 3 illustrates how ZeroOrderSeparatingPlanes optimizes hyperplanes. The ZeroOrderSeparatingPlanes step constructs a

---

Algorithm 2: ZEROORDERSEPARATINGPLANES
___

Input: Domain $\mathcal{D} \subseteq \mathcal{C}$, ellipsoid $\mathcal{E} = (E, c)$, current outer iteration $i \in \mathbb{N}$.
Output: Polytope $\mathcal{P} \subseteq \mathcal{D}$ satisfying (1) for (ε, δ_i).
Algorithm: $k \leftarrow 0$, $\mathcal{P} \leftarrow \mathcal{D}$
while TRUE **do**
    $\mathcal{S} \sim$ UNIFORMSAMPLE$(\mathcal{P}, M)$
    $\mathcal{S}_{\text{col}} \leftarrow$ POINTSINCOLLISION$(\mathcal{S})$
    If UNADAPTIVETEST$(\delta_{i,k}, \varepsilon, \tau)$ returns `accept` **then** *break.*
    $\mathcal{S}_{\text{col}}^{\star} \leftarrow$ UPDATEPOINTSVIABISECTION$(\mathcal{S}_{\text{col}}, c)$
    $\mathcal{P} \leftarrow$ ORDERANDPLACENONREDUDNANTFACES$(\mathcal{P}, \mathcal{E}, \mathcal{S}_{\text{col}}^{\star}, \Delta)$
    $k \leftarrow k + 1$
end
return $\mathcal{P}$

---

probabilistically collision-free polytope $\mathcal{P}$ by repeating 5 steps until the statistical test passes. We take as input the domain $\mathcal{D}$, the current ellipsoid $\mathcal{E}$, the current outer iteration i and options that are summarized in Tab. 1. Here, we assume that the center of the current ellipsoid is collision-free. We then initialize $\mathcal{P}$ with the domain $\mathcal{D} = \{q | A_{\mathcal{D}} q \leq b_{\mathcal{D}}\}$ (typically a polytope describing the joint limits).

The first step is to uniformly sample a batch of configurations $\mathcal{S}$ in $\mathcal{P}$ via hit-and-run sampling [23] with a batch size of $\max\{M, N_p\}$. Then, we find all configurations $\mathcal{S}_{\text{col}} \subseteq \mathcal{S}$ that are in collision by running a collision checker. These samples are used to check the termination condition by running the unadaptive test, returning $\mathcal{P}$ if it accepts. More precisely, counting the collisions $|\mathcal{S}_{\text{col}}^M|$ in the first M samples in $\mathcal{S}$ and accepting if $|\mathcal{S}_{\text{col}}^M| \leq M(1 - \tau)\varepsilon$.

If the probabilistic test fails, we then add hyperplanes to the polytope. We produce candidate solutions for the closest point in collision program (6) based on the first N_{p} configurations q in $\mathcal{S}_{\text{col}}$. For each q, we use bisection search to step toward the center c of the ellipsoid, while ensuring that q is still in collision. In practice, we bisect a fixed number N_b of times and pick the configuration $q^{\star}$ that is closest to c and still in collision. This yields our updated batch $\mathcal{S}^{\star}$. This strategy is guaranteed to converge to the boundary of an obstacle for each configuration q, but not necessarily the obstacle containing the original q or the one that is closest to c. Because this optimization procedure is gradient-free and only relies on a collision checker, it is highly parallel and its performance scales with available computational resources. Finally, we order all candidates $q^{\star} \in \mathcal{S}_{\text{col}}^{\star}$ by ellipsoidal metric in ascending order. We repeatedly pick the closest candidate $q^{\star}$ in the polytope and add a hyperplane at that point tangent to the ellipsoid, until we hit N_f non-redundant hyperplanes or run out of candidates.

5.3 The IRIS-NP2 Algorithm

The IRIS-NP2 algorithm (Alg. 3) places hyperplanes more carefully than IRIS-ZO, increasing runtime in favor of larger regions with fewer hyperplanes, while still improving upon IRIS-NP runtimes. Like IRIS-NP, IRIS-NP2 uses nonlinear programming, solving (6) to find separating hyperplanes. In contrast to IRIS-NP, IRIS-NP2 always initializes NLP searches with feasible initial guesses q_0,
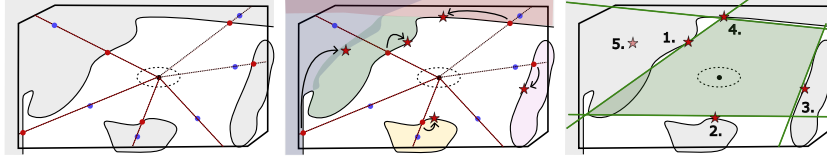
Fig. 4: Execution of NP2-SEPARATINGPLANES using the ray collision finder for GETCONFIGINCOLLISION. Blue dots represent uniform samples in the polytope. Red dots represent the results of the corresponding discrete line searches. Stars represent the results of local nonlinear searches (6). On the right, the corresponding hyperplanes (green) form the updated polytope.

prioritizing initial guesses with lower ellipsoid metric $||q_0 - c||_E^2$. These initial guesses, along with the corresponding pair of bodies in collision, $(\mathcal{A}, \mathcal{B})$ such that $\mathcal{A}(q) \cap \mathcal{B}(q) \neq \emptyset$, are obtained via a subroutine GETCONFIGINCOLLISION, two options for which are outlined at the end of this section.

Algorithm 3: NP2-SEPARATINGPLANES

Input: Domain $\mathcal{D} \subseteq \mathcal{C}$, ellipsoid $\mathcal{E} = (E, c)$, current outer iteration $i \in \mathbb{N}$
Output: Polytope $\mathcal{P} \subseteq \mathcal{D}$ satisfying (1) for (ε, δ_i).
Algorithm: $k \leftarrow 0$, $\mathcal{P} \leftarrow \mathcal{D}$
repeat
    $i \leftarrow 0$, $n_f \leftarrow 0$
    while $i < N_p$ *and* $n_f < N_f$ **do**
        $q_0, (\mathcal{A}, \mathcal{B}) \leftarrow$ GETCONFIGINCOLLISION
        if q_0 *is not None* **then**
            $q^* \leftarrow$ solve (6) with initial guess q_0, with pair $(\mathcal{A}, \mathcal{B})$
            $a = \frac{E(q^*-c)}{||E(q^*-c)||}$, $b = a^\top q^* - \Delta$
            Add hyperplane defined by (a, b) to $\mathcal{P}$
            $n_f \leftarrow n_f + 1$
        end
        $i \leftarrow i + 1$
    end
    $k \leftarrow k + 1$
until UnadaptiveTest$(\delta_{i,k}, \varepsilon, \tau)$ *returns* `accept`;
return $\mathcal{P}$

This strategy of always using a feasible initial guess presents a major advantage over IRIS-NP, as IRIS-NP spends substantial time in NLP solves that return infeasible. In part, this property of IRIS-NP is due to initializing NLP searches with uniform samples q_0, in the hopes that the nonlinear program solver can pull those q_0 not in collision into collision to return feasible optimized solutions $q^\star$. In practice, this frequently fails even when the polytope does contain collisions. Furthermore, as discussed in Sec. 4.2, IRIS-NP *requires* that many consecutive NLP solves return infeasible in order to terminate. Assessing termination readiness via the Bernoulli trials is faster, allowing the NLP, which is slow to solve, to be used only to construct hyperplanes. Another advantage of this strategy is it scales better with environment complexity. IRIS-NP must solve a minimum number of NLPs for every valid collision pair. This quantity grows quickly with scene complexity, while many of these pairs never actually collide, resulting in

infeasible NLPs and wasted computation time. IRIS-NP2 only considers pairs known to collide, and hence solves only feasible NLPs. We present two options for GETCONFIGINCOLLISION, with differing costs and benefits.

For the *Greedy Collision Finder*, all samples from the Bernoulli trial that are in collision are aggregated and sorted in ascending order of distances to the center point of the ellipsoid, w.r.t. the ellipsoidal metric. (In effect, N_p is dynamically set to equal the number of collision particles.) When queried, this subroutine returns the sample with the next-lowest ellipsoid metric, until all in-collision samples from the Bernoulli trial have been exhausted. If a sample is not in the polytope (due to faces that have been added from earlier samples), it would no longer be a feasible initial guess for Equation (6), so `None` is returned.
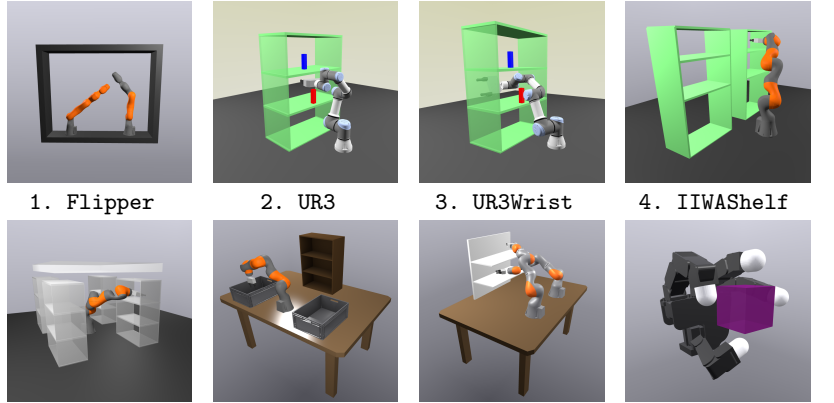
The *Ray Collision Finder* (Fig. 4) prioritizes the quality of the constructed hyperplanes via an inexpensive line search to find configurations in collision near the ellipsoid center. The ray collision finder takes a subset of the samples drawn for the Bernoulli trial, and, for each sample, steps outward in discrete steps along the ray from the ellipsoid center through the sample. The first step in collision is returned. If a step has exited the polytope, `None` is returned. This discrete search serves a purpose similar to the objective of (6): to limit the faces needed, we desire to add hyperplanes at the closest (in the ellipsoid metric) points in collision. Because the discrete search is not restricted to a specific collision pair, it may find configuration-space obstacles closer than (6). In contrast to the bisection search performed by IRIS-ZO, this line search sweeps monotonically outward from the ellipsoid center in an attempt to find the closest collision; the IRIS-ZO bisection search steps inward from the sample, finding a point on a boundary of a configuration-space obstacle. (The ray may pass through several such boundaries.)

6   Experiments

To evaluate our proposed changes to the IRIS-NP algorithm, we designed a comprehensive benchmark. This benchmark includes eight different robotic systems, described in Fig. 5. For each benchmark, we select 10 seed configurations manually, placing the robot in a variety of configurations in its environment. Each experiment is run for a single outer iteration, to isolate performance differences in the SEPARATINGPLANES steps.

We compare the approaches for two different settings: a "fast" setting, where we request the region to be 90% collision-free, with 90% confidence, and a "precise" setting, where we request the region to be 99% collision-free with 95% confidence. IRIS-NP cannot take these arguments directly, so we hand-tune the required number of consecutive infeasible solves to roughly match the average collision fractions achieved by our algorithms. For IRIS-NP2, we tuned the algorithm hyperparameters of Tab. 1 on a per-experiment basis.

To mitigate the effects of randomness of the algorithms in our comparisons, we constructed regions around each seed point 10 times. This yields 100 trials per robot environment, or 800 total, for each algorithm. For the parallelized components of the algorithms, an Intel Core i9-10850K (10 cores, 20 threads)

Fig. 5: The systems we use to benchmark our algorithms, along with the number of degrees of freedom, number of collision pairs, and the ratio of the volume of the free configuration space relative to the entire configuration space $\mathcal{C}$. The ratio $\mathrm{vol}(\mathcal{C}^{\mathrm{free}})/\mathrm{vol}(\mathcal{C})$ is estimated with 10^7 samples uniformly drawn from $\mathcal{C}$.

Domain	Flipper	UR3	UR3Wrist	IIWAShelf	4Shelves	IIWABins	2IIWAs	Allegro
# dof	3	5	6	7	7	7	14	15
# col. pairs	137	142	142	172	302	212	264	187
$\mathrm{vol}(\mathcal{C}^{\mathrm{free}})/\mathrm{vol}(\mathcal{C})$	0.149	0.268	0.254	0.480	0.117	0.577	0.536	0.261

is used. All convex programs are solved using MOSEK [24] and the nonlinear programs are solved using SNOPT [25]. The volume of the maximum-volume inscribed ellipsoid is used as a proxy for polytope volume. We report these volumes normalized by the averaged IRIS-NP results.

For each algorithm and robotic system, we include the runtime, number of hyperplanes, volume, and proportion of the region in collision, averaged over all trials on all seed points in Tab. 2. Region volume and number of hyperplanes are both dependent on the seed configuration – if the robot is in a narrow passageway, the region may be small, and if it is near obstacles, many hyperplanes may be required. In Fig. 6, we visualize the runtime and hyperplane data across the 10 trials for the 8 corresponding seed configurations shown in Fig. 5. Our algorithms show significant improvements relative to IRIS-NP in runtime and number of hyperplanes. In particular, IRIS-ZO gains approximately an order-of-magnitude time advantage with a comparable number of hyperplanes. While IRIS-NP2 is slower than IRIS-ZO, it generally remains faster than IRIS-NP and uses significantly fewer hyperplanes.

7   Discussion

In recent years, region generation has posed a major roadblock for the adoption of new motion planning approaches such as GCSTrajOpt. Toward alleviating this roadblock, we have presented significant improvements to the IRIS-NP al-

Environment	Time [s]				Num Hyperplanes				Rel Volume				Frac in collision			
	NP	ZO	Greedy	Ray	NP	ZO	Greedy	Ray	NP	ZO	Greedy	Ray	NP	ZO	Greedy	Ray
Fast																
Flipper	0.168	**0.025**	0.045	0.847	22.7	15.8	13.6	**13.4**	1.0	0.627	**2.167**	2.017	0.040	0.013	0.022	0.016
UR3	0.994	**0.041**	0.113	0.221	41.6	27.8	25.4	**22.7**	1.0	0.971	**3.598**	3.296	0.040	0.029	0.020	0.028
UR3Wrist	1.227	**0.050**	0.168	0.948	51.4	34.0	30.8	**26.5**	1.0	3.824	**5.376**	1.663	0.029	0.031	0.023	0.023
IIWAShelf	1.345	**0.091**	0.442	0.633	77.3	47.2	32.7	**31.0**	1.0	6.196	**8.73e3**	16.249	0.068	0.034	0.027	0.052
4Shelves	1.317	**0.092**	0.313	0.845	62.9	46.9	35.1	**30.3**	1.0	0.139	1.388	**1.724**	0.102	0.038	0.025	0.033
IIWABins	1.536	**0.074**	0.627	0.861	84.0	44.3	33.4	**31.1**	1.0	542.228	**4.99e4**	604.161	0.092	0.035	0.027	0.021
2IIWAs	20.482	**2.024**	62.060	46.527	242.1	246.2	77.7	**65.7**	1.0	0.0008	**316.777**	95.052	0.061	0.057	0.037	0.055
Allegro	2.277	0.264	**0.101**	0.106	75.3	83.7	41.9	**39.6**	1.0	0.036	**152.040**	60.554	0.0007	0.052	0.015	0.033
Precise																
Flipper	1.002	0.172	**0.152**	0.269	26.1	17.8	**14.3**	14.4	1.0	0.621	1.169	**1.236**	0.003	0.001	0.001	0.0004
UR3	5.222	**0.304**	0.325	0.642	52.9	44.5	28.1	**27.0**	1.0	3.698	**13.656**	2.301	0.003	0.002	0.002	0.003
UR3Wrist	4.664	**0.359**	0.359	0.410	61.8	56.1	31.5	**28.6**	1.0	3.811	**183.345**	4.959	0.002	0.003	0.001	0.002
IIWAShelf	11.261	**0.787**	1.087	3.008	126.3	92.0	41.0	**37.8**	1.0	**763.826**	7.817	9.260	0.004	0.003	0.002	0.004
4Shelves	25.315	**0.699**	0.838	1.002	85.2	81.3	40.5	**36.1**	1.0	1.592	1.592	1.592	0.012	0.003	0.002	0.003
IIWABins	15.487	**0.763**	0.997	2.188	128.6	98.6	44.1	**39.6**	1.0	504.123	**603.431**	92.268	0.003	0.003	0.002	0.003
2IIWAs	202.913	**43.194**	84.973	77.181	395.1	738.7	90.8	**82.7**	1.0	0.0003	37.507	**48.984**	0.006	0.005	0.004	0.004
Allegro	2.164	7.455	**0.499**	1.271	75.3	290.6	45.5	**42.2**	1.0	0.005	8.237	**20.434**	0.0007	0.005	0.001	0.003

Table 2: Statistics averaged over all 10 seed points. "Rel Volume" corresponds to the volumes of the MVIE, normalized by that of IRIS-NP. For the "Fast" settings, on average, IRIS-ZO was 15.5 times faster than IRIS-NP with 1.4 times fewer hyperplanes. IRIS-NP2 with the greedy strategy was 6.55 times faster with 2.1 times fewer hyperplanes, and with the ray strategy was 4.2 times faster with 2.3 times fewer hyperplanes. For the "Precise" settings, IRIS-ZO was 14 times faster than IRIS-NP with around the same number of hyperplanes. IRIS-NP2 with the greedy strategy was 12.3 times faster with 2.4 times fewer hyperplanes, and with the ray strategy was 8 times faster with 2.7 times fewer hyperplanes.

Fast settings: Time [s]

Fast settings: Number of hyperplanes

Precise settings: Time [s]

Precise settings: Number of hyperplanes

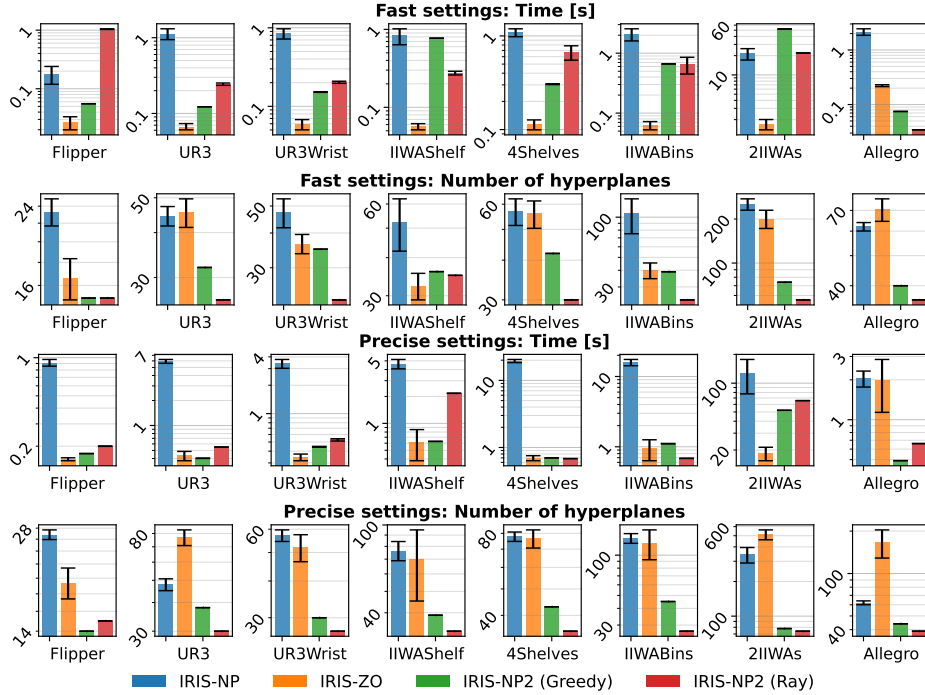Legend: IRIS-NP, IRIS-ZO, IRIS-NP2 (Greedy), IRIS-NP2 (Ray)

Fig. 6: The mean and standard deviation of the runtime and hyperplane counts across ten trials per environment for the 8 seed configurations shown in Fig. 5.

gorithm, reducing its runtime, making it more user-friendly, and increasing the quality of the resulting regions. We derive a probabilistic test for the proportion of a given region that is collision-free, allowing the user to more directly trade off algorithmic precision and speed. We have also presented new approaches to the SEPARATINGPLANES subroutine, which show significantly better performance than IRIS-NP, while consistently achieving the user-specified collision-free threshold. IRIS-NP2 is generally faster and requires fewer hyperplanes to define the polytopes. IRIS-ZO is easy to implement is around 15 times faster and often uses fewer hyperplanes than IRIS-NP, although the regions are smaller. Furthermore, these new approaches scale more favorably with the complexity of the environment. As they are deployed in environments with increasingly many collision geometries, we anticipate the performance gap will grow even further. Besides the quantitative improvements, the hyperparameters are straightforward to tune and have clear effects on the behavior of the algorithm.

There are myriad directions for future research. Better parallelization is a promising direction to speed up both new approaches – beyond CPU-level multi-threading, SIMD instructions [26] and GPU programming [27] have shown great promise for collision-checking. In particular, we regard better hardware usage for IRIS-ZO particularly promising, as both the sampling and the particle updates are trivially parallelizeable and currently bottle-necked by the number of threads in the CPU. Furthermore, using samples that are almost in-collision to initialize IRIS-NP2 might aid in finding small obstacles when requesting regions with a very small proportion in collision. Finally, we have focused on runtime and number of hyperplanes as our primary objectives, but more closely examining the relationship between the generated regions and the downstream motion planning algorithms will be essential for further improving the results. In particular, we aim to investigate how to improve efficient cover generation as in [28].

References

1. T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science robotics*, vol. 8, no. 84, p. 7843, 2023.
2. M. Petersen and R. Tedrake, "Growing convex collision-free regions in configuration space using nonlinear programming," *arXiv preprint arXiv:2303.14737*, 2023.
3. T. Lozano-Perez, *Spatial planning: A configuration space approach.* Springer, 1990.
4. C. Bajaj and M.-S. Kim, "Generation of configuration space obstacles: Moving algebraic surfaces," *The International Journal of Robotics Research*, vol. 9, no. 1, pp. 92–112, 1990.
5. J. Pan and D. Manocha, "Efficient configuration space construction and optimization for motion planning," *Engineering*, vol. 1, no. 1, pp. 046–057, 2015.
6. L. Jaulin, "Path planning using intervals and graphs," *Reliable computing*, vol. 7, no. 1, pp. 1–15, 2001.

7. J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.
8. L. Yang and S. M. LaValle, "The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 419–432, 2004.
9. R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 109–124.
10. A. Jaitly and S. Farzan, "Paamp: Polytopic action-set and motion planning for long horizon dynamic motion planning via mixed integer linear programming," *arXiv preprint arXiv:2403.10924*, 2024.
11. T. Cohn, M. Petersen, M. Simchowitz, and R. Tedrake, "Non-Euclidean Motion Planning with Graphs of Geodesically-Convex Sets," in *Proceedings of Robotics: Science and Systems*, Daegu, Republic of Korea, July 2023.
12. T. Cohn, S. Shaw, M. Simchowitz, and R. Tedrake, "Constrained bimanual planning with analytic inverse kinematics," *arXiv preprint arXiv:2309.08770*, 2023.
13. S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
14. ROS.org, "Urdf - unified robot description format," 2024, accessed: 2024-06-09. [Online]. Available: http://wiki.ros.org/urdf
15. O. S. R. Foundation, "Sdf specification," 2024, accessed: 2024-06-09. [Online]. Available: http://sdformat.org
16. R. Tedrake, *Robotic Manipulation*, 2023. [Online]. Available: http://manipulation.mit.edu
17. K. Mamou and F. Ghorbel, "A simple and efficient approach for 3d mesh approximate convex decomposition," in *2009 16th IEEE international conference on image processing (ICIP)*. IEEE, 2009, pp. 3501–3504.
18. Y. Wu, I. Spasojevic, P. Chaudhari, and V. Kumar, "Optimal convex cover as collision-free space approximation for trajectory generation," *arXiv preprint arXiv:2406.09631*, 2024.
19. H. Dai, A. Amice, P. Werner, A. Zhang, and R. Tedrake, "Certified polyhedral decompositions of collision-free configuration space," *The International Journal of Robotics Research*, 2023.
20. A. Sarmientoy, R. Murrieta-Cidz, and S. Hutchinsony, "A sample-based convex cover for rapidly finding an object in a 3-d environment," in *Proceedings of the 2005 IEEE ICRA*. IEEE, 2005, pp. 3486–3491.
21. S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
22. N. Alon and J. H. Spencer, *The probabilistic method*. John Wiley & Sons, 2016.
23. L. Lovász, "Hit-and-run mixes fast," *Mathematical programming*, vol. 86, 1999.
24. M. ApS, "MOSEK Optimization Suite," 2019.
25. P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.
26. T. Tan, R. Weller, and G. Zachmann, "Simdop: Simd optimized bounding volume hierarchies for collision detection," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 7256–7263.
27. C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast bvh construction on gpus," in *Computer Graphics Forum*, vol. 28, no. 2. Wiley Online Library, 2009, pp. 375–384.
28. P. Werner, A. Amice, T. Marcucci, D. Rus, and R. Tedrake, "Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs," *arXiv preprint arXiv:2310.02875*, 2023.