Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

Superfast Configuration-Space Convex Set Computation on GPUs for Online Motion Planning

Peter Werner[1], Richard Cheng[2], Thomas Stewart[3], Russ Tedrake[1,2], and Daniela Rus[1]
[1]MIT CSAIL [2]Toyota Research Institute [3]Woven by Toyota
{wernerpe, russt, rus}@mit.edu
richard.cheng@tri.global, tom.stewart@woven.toyota

***Abstract*—In this work, we leverage GPUs to construct probabilistically collision-free convex sets in robot configuration space on the fly. This extends the use of modern motion planning algorithms that leverage such representations to changing environments. These planners rapidly and reliably optimize high-quality trajectories, without the burden of challenging nonconvex collision-avoidance constraints. We present an algorithm that inflates collision-free piecewise linear paths into sequences of convex sets (SCS) that are probabilistically collision-free using massive parallelism. We then integrate this algorithm into a motion planning pipeline, which leverages dynamic roadmaps to rapidly find one or multiple collision-free paths, and inflates them. We then optimize the trajectory through the probabilistically collision-free sets, simultaneously using the candidate trajectory to detect and remove collisions from the sets. We demonstrate the efficacy of our approach on a simulation benchmark and a KUKA iiwa 7 robot manipulator with perception in the loop. On our benchmark, our approach runs 17.1 times faster and yields a 27.9% increase in reliability over the nonlinear trajectory optimization baseline, while still producing high-quality motion plans. Website: `https://sites.google.com/view/GPUPolytopes`**

I. Introduction

Finding high-quality collision-free motion plans remains one of the most fundamental and challenging problems in robotics [1, 2]. Systems with high demands on reliability and performance, e.g. on production lines or in warehouses, often require carefully tailored ad hoc solutions, which afford reliability at the cost of engineering effort and performance.

Over the years, sampling-based motion planning, trajectory optimization using nonlinear programming, and combinations thereof, have become the most established approaches for motion planning. Sampling-based motion planners are simple to implement [2, 3], work well in lower dimensions, can have completeness guarantees, and can be massively accelerated using large-scale parallelization [4, 5]. See [6] for an overview on sampling-based motion planning. Unfortunately, these methods become impractical as the dimension increases, yielding suboptimal motion plans and long planning times.

Directly transcribing motion planning as a general nonlinear optimization problem and finding solutions with local methods, e.g. [7–9], has the benefit of scaling to higher-dimensional problems and allowing the user to encode a wider variety of costs and constraints. Unfortunately, such local optimization approaches are sensitive to initialization and often struggle to find a feasible solution when one exists.

To capture the strengths of both trajectory optimization and sampling-based motion planning, a series of recent works [10–17] initially proposed in [18] investigate planning through collections of collision-free convex sets that approximately decompose the free space. For the remainder of this paper, we will refer to this class of planners as decomposition-based motion planners (DBMPs). A challenging nonconvexity in trajectory optimization stems from the collision-avoidance constraints. Given a collection of collision-free convex sets, DBMPs replace these nonconvex constraints with simple convex constraints, which allows them to profit from the fast solve times and reliability of convex optimization. This shifts the challenges that stem from these collision-avoidance constraints to constructing a high-quality representation of the collision-free space through a union of convex sets.

For a candidate approximate convex decomposition to be effective, it should consist of a small number of sets, contain high-quality trajectories in the union of the sets, and its individual sets should be described by few constraints in order to keep the trajectory optimization problems small.

Many robot motion planning problems are most naturally formulated in the robot's configuration space $\mathcal{C}$. Therefore, we seek to construct the convex sets in $\mathcal{C}^{\text{free}}$, the collision-free subset of $\mathcal{C}$. Unfortunately, this is particularly challenging because configuration spaces tend to be high-dimensional with complicated geometries, and we often only have implicit descriptions of $\mathcal{C}^{\text{free}}$ [1, §4.3.3]. Therefore, the go-to algorithms used by practitioners for constructing convex decompositions of $\mathcal{C}^{\text{free}}$ are slow, scale poorly with the environment complexity, and are difficult to use [19, 20], mainly due to their complexity or because they are difficult to tune. Additionally, these algorithms only consider static environments. Generally, the sets need to be recomputed in case the environment changes. So far, these challenges have made set construction a bottleneck, and have prevented DBMPs from being applied successfully to scenarios involving changing environments like mobile manipulation.

In this paper, our aim is to tackle two key challenges: (1) computing convex sets in robot configuration space for motion planning in real-time, and (2) reliable and effective positioning of the sets such that they contain high-quality trajectories. By addressing these challenges, our aim is to unlock the benefits of DBMPs and enable their use in general, changing environments.

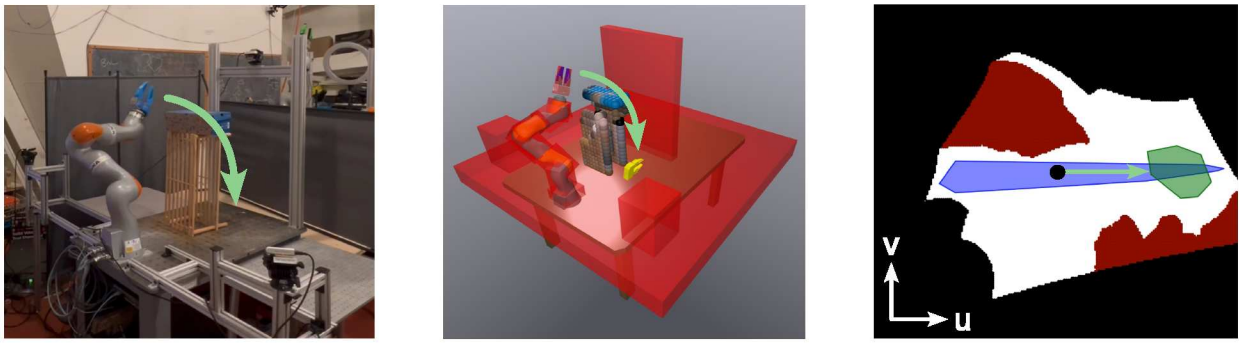

Fig. 1: On the left, our hardware setup with a KUKA LBR iiwa 7 R800 robotic manipulator and three Intel RealSense D415 depth cameras for perceiving obstacles is shown. In the center, the simulated system with the perceived obstacles is shown. The target end effector pose is indicated by the yellow gripper and the collision geometries of the system are shown in red. The perceived obstacles are approximated by a union of spheres shown in the center of the table. The figure on the right shows a two-dimensional slice of the seven-dimensional configuration space, along with a slice of the safe sets (blue and green), and a slice of the configuration obstacles (self-collisions in black, collisions with perceived obstacles in red). The slice lies tangent to the trajectory such that the u-direction is co-linear with the velocity and the v-direction is a random orthogonal vector. The current configuration is indicated by the black dot.

To tackle challenge (1), we build on the recently developed IRIS-ZO algorithm [21], which lends itself to large-scale parallelism. In particular, we propose Zero-Order Edge Inflation (EI-ZO), and demonstrate that GPU acceleration of our algorithm enables real-time set construction in changing configuration spaces.

EI-ZO also addresses challenge (2) by inflating collision-free line segments rather than points, yielding probabilistically collision-free polytopes which are guaranteed to contain the seed line segment. This enables us to inflate collision-free, piecewise-linear (PWL) paths between the start and the goal into sequences of convex sets (SCS) in which successive sets in the sequence intersect. By construction, this SCS connects the start to the goal and guarantees the containment of a collision-free path. In comparison to approximating the entire free space with a union of convex sets [22], this approach dramatically reduces the computational burden by reducing the number of required sets. Perhaps more importantly, it makes the step of finding a collection of sets that connect the start to the goal reliable, even in complicated, changing configuration spaces.

Finally, we propose a motion planning pipeline that combines these ideas with large-scale dynamic road maps (DRMs). The pipeline first leverages the DRM to rapidly find a collision-free PWL path from a starting configuration to a goal configuration. Next, this path is inflated with EI-ZO to produce a SCS, which is then used to recover high-quality motion plans using DBMPs.

We demonstrate the efficacy of our approach in simulation benchmarks in two and seven dimensions and validate the approach on hardware using a KUKA iiwa with perception in the loop, as shown in Fig. 1. Relative to the nonlinear trajectory optimization baseline, our approach produces slightly higher-cost trajectories, but increases the success rate from around 72% to 100% on our simulation benchmark, while computing trajectories around 17 times faster.

The remainder of this paper is structured as follows. In §II we outline the assumptions and required inputs. Next, we review prior work on constructing convex sets in configuration space for motion planning in §III. Our algorithm, EI-ZO, for inflating line segments in configuration space is then presented in §IV. We outline how we employ DRMs in §V. We then present our full motion planning pipeline combining DRMs, EI-ZO, and DBMPs in §VI. Our experiments are presented in §VII. In §VIII we discuss the limitations and in §IX draw a conclusion.

II. Assumptions

We make similar assumptions as in prior work on computing approximate convex decompositions of $\mathcal{C}^{\text{free}}$, e.g. [21, §2]. That is, we assume that we are given a description of the considered robot system which contains information about the kinematics and collision geometries of the system sufficient for collision checking, e.g. as provided in URDFs [23] or SDFs [24]. If we are dealing with additional obstacles entering and leaving the system, then we assume that (1) these obstacles remain static during the brief planning and execution time of the motion plan and (2) we are given observations of the obstacles sufficient to perform (potentially conservative) collision checks against them. Specifically, we assume that our collision checker never mislabels a configuration as safe when it is actually in collision.

III. Background

A. Constructing Safe Sets in Robot Configuration Space

The construction of convex safe sets in robot configuration space has been studied in [19–21]. The algorithms in [19] compute provably collision-free polytopes in a rational reparametrization of the configuration space using polynomial optimization. Due to the high computational cost of the approaches in [19], we direct our focus towards the algorithm

IRIS-NP, initially proposed in [20], and later improved and extended to IRIS-NP2 and IRIS-ZO in [21]. These algorithms leverage nonlinear optimization to compute probabilistically collision-free polytopes in robot configuration space in substantially less time. IRIS-NP2 and IRIS-ZO provide a probabilistic guarantee for a produced polytope $\mathcal{P}$ of the form

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

which controls what fraction of the volume of $\mathcal{P}$ is allowed to be in collision. In (1), λ denotes the Lebesgue measure in $\mathcal{C}^{\text{free}}$, ε is an admissible fraction of the volume of $\mathcal{P}$ that is allowed to be in collision, and δ is the admissible uncertainty.

Here we introduce an algorithm for constructing safe sets around line segments that primarily builds on IRIS-ZO [21, §5.2]. IRIS-ZO constructs probabilistically collision-free polytopes by iteratively removing collisions from the current region via hyperplanes. It optimizes the locations of these hyperplanes using sampling and collision checking. We build on IRIS-ZO because it only relies on collision checking and simple sample updates, making it massively parallelizeable, and amenable to GPU acceleration.

B. Positioning Safe Sets for Effective Motion Planning

There have been two approaches to position individual convex sets for motion planning. The first approach is to find a collection of convex sets that efficiently cover a large fraction of the entire configuration space. The second is to only cover an individual collision-free (but potentially suboptimal) path.

For example, in [22, 25], visibility graphs are used to construct approximate convex covers which try to minimize the number of polytopes required to meet a coverage threshold. The approach in [25] uses densely sampled visibility graphs and visibility kernels in two and three dimensions to construct a cover of polytopes represented as convex hulls of samples. The Visibility Clique Cover algorithm in [22] works in higher dimensions by expanding cliques on visibility graphs to probabilistically collision-free polytopes using an algorithm similar to [20].

Unfortunately, in relatively simple environments, the required number of sets is already prohibitively large for online cover generation. Instead, we focus on the second approach: covering an individual collision-free (but potentially suboptimal) path with collision-free convex sets.

This strategy has been investigated in two and three-dimensional task spaces to generate safe flight corridors on the fly [12–14, 26], where either a sequence of boxes is computed [12], or the IRIS algorithm [27] is modified to include line segments of a collision-free path [13, 14, 26] in the produced polytopes. In [13, 14], this is accomplished by careful selection of the initial ellipsoid, which defines the distance metric in the IRIS algorithm for optimizing hyperplane placement. A more involved formulation is used in [26] that admits constraining the inclusion of line segments directly in the optimization of the hyperplanes.

In general, this second strategy has the advantage of only generating regions relevant for the current planning problem, and thus creates substantially fewer sets. In this paper, we propose a method for applying this idea in higher-dimensional configuration spaces. Our approach proposes a simple modification to the IRIS algorithms: using the distance to a line segment as a metric instead of one defined by an ellipsoid. As discussed in IV, this guarantees the inclusion of the line segment (provided it is collision-free), skips checking the ellipsoids for collisions, and avoids the more involved optimization problems in [26], which would not be amenable for GPU implementation.

IV. Rapidly Inflating Line Segments

In this section, we discuss our algorithm, Edge Inflation Zero-Order (EI-ZO), for rapidly inflating line segments to full-dimensional, probabilistically collision-free polytopes in robot configuration space using only zero-th order information about the system. First, we observe that the distance to a convex set, in particular a line segment, is a convex function. This leads to a natural way of extending IRIS algorithms to guarantee the containment of line segments (and convex sets more generally) provided they are collision-free. Given our focus on real-time set generation, we specifically focus on adapting the IRIS-ZO algorithm. The proposed modifications, however, can be applied to both IRIS [27] and IRIS-NP2 [21] as well.

A. Preliminaries

Let $\mathcal{A} \subseteq \mathbb{R}^n$ be a convex set, and

$$\mathbf{dist}_{\mathcal{A}}(x) = \min_{z \in \mathcal{A}} ||x - z||_2 \tag{2}$$

be the distance of a point $x \in \mathbb{R}^n$ to $\mathcal{A}$. We verify in Appendix A that the function $\mathbf{dist}_{\mathcal{A}}(x)$ is convex in x. Given that a line segment $\mathcal{L} = \mathbf{conv}\{v_1, v_2\}$, with $v_1, v_2 \in \mathbb{R}^n$, is a convex set by construction, it follows that the sub-level sets

$$\mathcal{B}_{\mathcal{L}}(t) = \{x | \mathbf{dist}_{\mathcal{L}}(x) \leq t\} \tag{3}$$

are convex [28, §3.1.6]. In Fig. 2, the level sets for five different distances are indicated by the dashed lines. The convexity of these sub-level sets implies that the tangent plane

$$a = \nabla(\text{dist}_{\mathcal{L}})|_c, \; b = a^T c, \tag{4a}$$

$$\mathcal{T} = \{x | a^T x - b = 0\}, \tag{4b}$$

passing through any point $c \in \mathbb{R}^n$, with $\mathbf{dist}_{\mathcal{L}}(c) = t > 0$, is supporting to $\mathcal{B}_{\mathcal{L}}(t)$, and cannot intersect $\mathcal{L}$. Note that the gradient of the distance function to a line segment has a simple closed-form solution. The gradient at c reads

$$\nabla(\text{dist}_{\mathcal{L}})|_c = \frac{c - c_{\text{proj}}}{||c - c_{\text{proj}}||_2}, \tag{5}$$

where c_{proj} is the projection of c onto $\mathcal{L}$ given by

$$c_{\text{proj}} = \begin{cases} v_1 & \text{if } v_1 = v_2, \\ (1 - \alpha)\, v_1 + \alpha v_2, & \text{otherwise.} \end{cases} \tag{6}$$

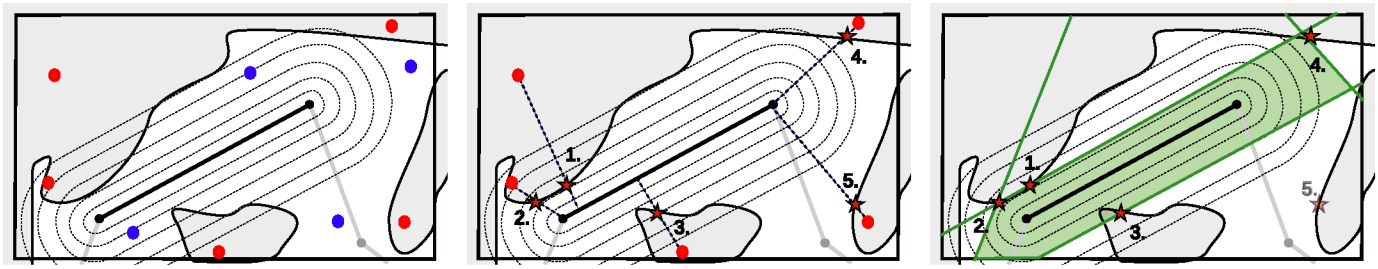

Fig. 2: A single iteration of adding hyperplanes to the polytope in the EI-ZO algorithm. The configuration obstacles are shaded in grey and the line segment seeding EI-ZO is shown in black. Starting from the left, first, a batch of configurations is sampled in the current polytope given by the black rectangle. These samples are checked for collisions. The colliding configurations, shown in red, are then used to seed a bisection search to find collisions that minimize the distance to the line segment. The resulting candidates, indicated by the red stars (center frame), are ranked by their distance to the line segment in ascending order, and used to position the tangent planes (final frame) that separate the collisions from the line segment.

with

$$\alpha = \min\left(\max\left(\frac{(c-v_1)^T(v_2-v_1)}{||(v_2-v_1)||_2^2}, 0\right), 1\right). \tag{7}$$

Next, consider the following procedure, which we formalize in §IV-B. Given a collision-free line segment $\mathcal{L} \subseteq \mathcal{C}^{\text{free}}$, repeatedly search for points c that are in collision, and place a hyperplane through point c according to (4) for each found collision. By construction, these hyperplanes separate each collision c from $\mathcal{L}$. Therefore, if we repeat this for all collisions in $\mathcal{C}$, then the polytope constructed by intersecting the halfspaces of each hyperplane that contains $\mathcal{L}$ is collision-free and, crucially, contains $\mathcal{L}$.

Unfortunately, this procedure could potentially require infinitely many hyperplanes and is therefore not practical in the general case. In order to obtain a practical algorithm, we construct our polytope $\mathcal{P}$ by iteratively placing hyperplanes until $\mathcal{P}$ is sufficiently collision-free and meets the criterion (1). First, we optimistically assume a large polytope $\mathcal{P}$ that contains $\mathcal{L}$ is collision-free. We then proceed to search for collisions inside of $\mathcal{P}$. Every time we find a collision, we place a separating hyperplane according to (4) that separates the found collision from $\mathcal{P}$, while still ensuring $\mathcal{L} \subseteq \mathcal{P}$. Note that placing a separating plane at a collision with a small distance to $\mathcal{L}$ can simultaneously also separate collisions at larger distances from $\mathcal{L}$. As a result, optimizing the hyperplane locations by attempting to find the collisions in $\mathcal{P}$ that minimize the distance to $\mathcal{L}$ tends to produce polytopes with substantially fewer hyperplanes. Therefore, a core subroutine in EI-ZO is optimizing hyperplane locations by finding good (although suboptimal) solutions to

$$\underset{x}{\textbf{minimize}} \quad \text{dist}_{\mathcal{L}}(x), \tag{8a}$$

$$\textbf{subject to} \quad x \notin \mathcal{C}^{\text{free}}, \quad x \in \mathcal{P}, \tag{8b}$$

in order to reduce the number of hyperplanes that are required for $\mathcal{P}$ to meet a termination condition that ensures (1).

Finally, in order to separate $\mathcal{L}$ from obstacles with concave boundary segments using a finite number of hyperplanes, we use a small, finite step back $\Delta > 0$ from the found collisions before placing the hyperplanes, similarly to [20].

In summary, this procedure constructs a large probabilistically collision-free polytope that contains the collision-free convex set (in this case a line segment) used to seed the polytope construction. We accomplish this by separating collisions from this seed set, by placing hyperplanes tangent to the sub-level sets of the distance function to the set. In the following section, we describe how EI-ZO implements these ideas to inflate collision-free line segments. It uses a zero-order optimization strategy on (8) to search for close collisions and uses a probabilistic termination condition (the 'unadaptive test' in [22, §5.1]) that checks if (1) is met and the polytope is sufficiently collision-free.

B. The EI-ZO Algorithm

The Edge Inflation Zero-Order (EI-ZO) algorithm is summarized in Alg. 1 and illustrated in Fig. 2.

Given a collision-free line segment in configuration space, $\mathcal{L} = \textbf{conv}\{v_1, v_2\} \subseteq \mathcal{C}^{\text{free}}$, an initial polytope describing the domain $\mathcal{D} = \{x | \mathcal{A}_{\mathcal{D}} x \leq b_{\mathcal{D}}\}$[1], such that $\mathcal{L} \subset \mathcal{D}$, EI-ZO computes a probabilistically collision-free polytope $\mathcal{P}$ that contains $\mathcal{L}$, and with probability larger than $1-\delta$, is colliding with no more than an ε-fraction of its volume. The algorithm closely follows IRIS-ZO [22], and proceeds as follows.

We initialize $\mathcal{P}$ with the domain $\mathcal{D}$. We then repeat the following until the termination criterion from the unadaptive test is met. First, we uniformly sample a batch of configurations $\mathcal{S}$ inside the current polytope $\mathcal{P}$ using hit-and-run sampling [29]. The batch size $|\mathcal{S}|$ is selected as $|\mathcal{S}| = \max\{N_p, M\}$, where $M = \lceil 2\log(1/\delta_k)/(\varepsilon\tau^2)\rceil$ and N_p is the maximum number of samples to update. Here, ε is the admissible fraction of the volume allowed to be in collision, $\tau \in (0,1)$ is a decision threshold [2], and δ_k is the admissible uncertainty at the k-th iteration. The value of δ_k is decayed according to

$$\delta_k = \frac{6\delta}{\pi^2 k^2}. \tag{9}$$

[1]Typically, a box describing the joint limits of the system is chosen.

[2]The parameter τ serves as a decision threshold, that trades off the power and cost of the unadaptive test. We typically choose $\tau = 0.5$.

Algorithm 1: EI-ZO

Input: Domain $\mathcal{D} \subseteq \mathcal{C}$, collision-free line segment $\mathcal{L}$ with $\mathcal{L} \subset \mathcal{D}$, test parameters $(\delta, \varepsilon, \tau)$, maximum step back Δ_{max}, and optimizer parameters (N_p, N_f, N_b).

Output: Polytope $\mathcal{P} \subseteq \mathcal{D}$ satisfying (1) for (ε, δ) with $\mathcal{L} \subseteq \mathcal{P}$.

Algorithm: $k \leftarrow 1$, $\mathcal{P} \leftarrow \mathcal{D}$

while TRUE **do**

- $\mathcal{S} \sim \text{UNIFORMSAMPLE}(\mathcal{P}, M)$
- $\mathcal{S}_{\text{col}} \leftarrow \text{POINTSINCOLLISION}(\mathcal{S})$
- **If** $\text{UNADAPTIVETEST}(|\mathcal{S}_{\text{col}}^M|, \delta_k, \varepsilon, \tau)$ returns `accept` **then** *break*.
- $\mathcal{S}_{\text{col}}^{\text{proj}} \leftarrow \text{PROJECTPOINTS}(\mathcal{S}_{\text{col}}, \mathcal{L})$
- $\mathcal{S}_{\text{col}}^{\star} \leftarrow \text{UPDATEPOINTSVIABISECTION}(\mathcal{S}_{\text{col}}, \mathcal{S}_{\text{col}}^{\text{proj}})$
- $\mathcal{P} \leftarrow \text{ORDERANDPLACEHYPERPLANES}(\mathcal{P}, \mathcal{S}_{\text{col}}^{\text{proj}}, \mathcal{S}_{\text{col}}^{\star}, \Delta)$
- $k \leftarrow k + 1$

end

return $\mathcal{P}$

More details of this statistical test are given in [21, §5.1].

Next, we check the configurations in $\mathcal{S}$ for collisions, collecting the colliding configurations in the set $\mathcal{S}_{\text{col}}$. We terminate and accept $\mathcal{P}$ if $|\mathcal{S}_{\text{col}}^M| \leq M(1-\tau)\varepsilon$, where $|\mathcal{S}_{\text{col}}^M|$ denotes the number of collisions in the first M samples in $\mathcal{S}$.

If the test fails, we modify $\mathcal{P}$ by adding more hyperplanes in an attempt to reduce the fraction in collision. To this end, we produce candidate solutions to (8) by performing gradient descent on $\mathbf{dist}_{\mathcal{L}}$, with the first N_p samples in $\mathcal{S}_{\text{col}}$ as feasible initial guesses, while ensuring that the candidates remain in collision.

First, we compute the projection of each sample in $\mathcal{S}_{\text{col}}$ onto $\mathcal{L}$ using the closed-form solution (6) in order to determine the gradient direction given by (5). We then perform a bisection search along the negative gradient direction, with a fixed number of bisection steps, N_b, in order to find a point close to the boundary of a configuration obstacle that is still in collision. The resulting points $\mathcal{S}_{\text{col}}^{\star}$ are our candidate local solutions to (8). These steps correspond to the left and the center frame in Fig. 2.

In a final step, we sort the candidates by their objective value (8a), the distance to $\mathcal{L}$, and place up to N_f hyperplanes. This is done by iteratively intersecting the halfspace

$$a_i = \nabla(\text{dist}_{\mathcal{L}})|_{c_i}, \; b_i = a_i^T c_i, \tag{10a}$$

$$\mathcal{H}_i = \{x | a_i^T x \leq b_i - \Delta\}, \tag{10b}$$

with $\mathcal{P}$, where $c_i \in \mathcal{S}_{\text{col}}^{\star}$ is the closest remaining candidate, and discarding any other candidates that now lie outside of the updated polytope. The step back Δ ensures that nonconvex obstacles can be excluded from the polytope with only a finite number of hyperplanes and needs to be computed for every hyperplane separately. How to compute δ is covered in §IV-C. Fig. 2 illustrates placing the hyperplanes in the rightmost frame. In this frame, the collision with the largest distance is walled off by the collision that is third-closest, and therefore no separating plane needs to be constructed for this collision.

This entire procedure of sampling inside of the current polytope, evaluating the statistical test, determining the gradient directions, updating candidates through bisection search, and placing non-redundant hyperplanes is repeated until the statistical test is passed.

C. Practical Algorithmic Considerations

In this section, we discuss three practical considerations: setting a maximum number of iterations N_{it}, computing the step back Δ, choosing the number of bisection steps N_b.

Maximum Iterations If one is concerned with generating regions quickly, it can be helpful to set a maximum number of iterations of adding hyperplanes to $\mathcal{P}$ to upper-bound the run time. This will void the probabilistic guarantee, but can still be useful in combination with mechanisms for recovering from collisions, as described in §VI-B.

Computing the Step Back While there are methods that can certify PWL paths or trajectories to be collision-free, e.g. [30–33], we check trajectories and paths for collisions by finely discretizing them because it is more amenable to our GPU-accelerated collision checker. In practice, this means that the line segments of the PWL path may be very close to or in collision. We handle these cases as follows.

If a candidate $c_i \in \mathcal{S}_{\text{col}}^{\star}$ has a distance that is smaller than a user-specified collision tolerance t_{col} (i.e. if $\mathbf{dist}_{\mathcal{L}}(c_i) \leq t_{\text{col}}$), then we throw an error stating that the line segment is likely in collision and do not attempt to construct the region. In order to 'fail fast' it can also make sense to check the projection for collisions at the start of the bisection search.

If a candidate $c_i \in \mathcal{S}_{\text{col}}^{\star}$ has a distance larger than the collision tolerance t_{col} but smaller than the maximum step back Δ_{max}, then applying the maximum step back to the hyperplane may exclude a part of $\mathcal{L}$. In this case, we relax the step back so $\mathcal{L}$ remains contained in the updated polytope. More precisely, first we find the vertex that is closest or the strongest violator to the new hyperplane and compute the corresponding value r as

$$r = \mathbf{max}\left\{a_i^T v_1, a_i^T v_2\right\} - b_i + \Delta_{\text{max}}, \tag{11}$$

where a_i and b_i are computed using (10). If $r > 0$, then the step back Δ_{max} excludes at least one of the vertices that span the line segment. Therefore, we compute the step back as follows

$$\Delta = \begin{cases} \Delta_{\text{max}} - r, & \text{if } r > 0, \\ \Delta_{\text{max}}, & \text{otherwise.} \end{cases} \tag{12}$$

This forces the inclusion of $\mathcal{L}$ in $\mathcal{P}$ at the potential cost of requiring more hyperplanes to meet the termination criterion.

Choosing the Number of Bisection Steps We use a fixed number of bisection steps N_b in order to simplify the parallelization of the algorithm. In order to choose N_b, we

suggest using the following rule of thumb:

$$N_b \approx \log_2\left(\frac{L}{\Delta_{\max}}\right), \tag{13}$$

where L is a guess for the maximum distance the bisection search would have to move a collision in order to get to a boundary of an obstacle. Note that choosing a small N_b does not impact the correctness of the regions. Instead, it might simply increase the runtime because more rounds of adding hyperplanes are necessary.

D. CUDA Implementation of EI-ZO

Accompanying this paper, we provide a Python library that implements EI-ZO using C++ and CUDA [34]. The software CSDecomp, short for configuration space decomposition toolbox, can be found at `https://github.com/wernerpe/csdecomp`.

We implement EI-ZO directly on the GPU to minimize costly data transfer between the CPU and the GPU. To this end, we employ thread-level parallelization to implement hit-and-run sampling [29], by assigning one CUDA thread per random walk where each walk accepts after a fixed number of mixing steps, forward kinematics by assigning one CUDA thread per configuration, collision checking by assigning one CUDA thread per collision pair, and the projection as well as the bisection updates by assigning one CUDA thread per configuration. To sort and place the non-redundant hyperplanes, we use a single CUDA thread.

As a result, at run time, we only need to send the seed line segment to the GPU and transfer the final polytope back from the GPU. Crucially, the samples never leave the GPU memory.

Currently, our collision-checker supports box and sphere collision geometries, but could readily be extended to support more geometries.

V. Generating Collision-Free Polygonal Paths using Dynamic Roadmaps

Our motion planning pipeline requires finding at least one collision-free PWL path in order to construct a sequence of convex sets that connects the starting configuration to the goal via EI-ZO. To this end, we employ dynamic roadmaps (DRMs) which are designed to rapidly find these paths in changing environments.

DRMs [35] approximate $\mathcal{C}^{\text{free}}$ with a probabilistic roadmap (PRM) [36], discretize the robot workspace into voxels, and build large lookup tables that map activated voxels to nodes and edges of the PRM that are in collision. For reference, see Fig. 3. The key idea is that approximating the current perceived environment by activating voxels, and subsequently pruning the PRM with lookup tables, is faster and leads to better motion plans than running a single-query motion planner [37]. Hence, DRMs enable us to offload much of the computational burden of motion planning (e.g. collision checks) to the offline construction of the roadmap, and thus enable fast planning in changing environments.

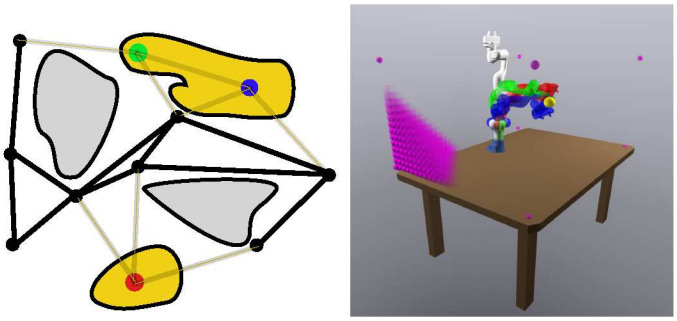

Fig. 3: A DRM is a PRM with additional lookup tables that allow the PRM to rapidly be updated provided task space observations of obstacles. To this end, the task space is discretized into voxels. We associate a circumscribing sphere with each voxel as collision geometry for collision checking. A portion of these spheres are shown in purple in the right figure. Next, a lookup table is constructed, $\mathcal{M}_c$, that maps each of these spheres to nodes in the road map that would collide with the sphere. Above, a single sphere is shown in yellow on the right, along with three configurations, shown in red, green, and blue, that collide with it and are stored in $\mathcal{M}_c$. On the left, a cartoon of the configuration space is shown with the underlying PRM and configuration obstacles. The three colliding configurations are highlighted again in red, blue, and green. The configuration obstacle corresponding to the yellow sphere is highlighted in yellow.

We closely follow the implementation in [38], which shows that GPU-accelerated collision checking both enables the construction of large-scale DRMs and effective online planning for a mobile bimanual manipulator. Similarly, we skip constructing the edge collision map, and use lazy collision checking online to invalidate colliding edges [39, 40].

A. Offline Roadmap Construction

Before any online planning, we construct our DRMs offline using a similar procedure [38]. Our DRMs consist of 4 data structures:

- A node map $\mathcal{M}_n : \mathcal{I} \to \mathcal{C}$ that maps a node identification number (node id) $i \in \mathcal{I}$ to its configuration,
- a node adjacency map $\mathcal{M}_a : \mathcal{I} \to \mathfrak{P}(\mathcal{I})$, where $\mathfrak{P}$ denotes the power set of $\mathcal{I}$, which maps a node id to all its neighbors,
- a collision map $\mathcal{M}_c : \mathcal{J} \to \mathfrak{P}(\mathcal{I})$ that maps a voxel identification number (voxel id) $j \in \mathcal{J}$ to all node ids that are in collision if the voxel j is active,
- a pose map $\mathcal{M}_p : \mathcal{I} \to SE(3)$ that maps each node id to its corresponding end-effector pose.

Fig. 3 depicts a high-level visual description of the roadmap and collision map. We do not detail the construction procedure here, but the interested reader can refer to Appendix B.

B. Online Planning given Perception

Once we have our DRM formed by these four data structures, we can use it to rapidly find collision-free PWL

paths online given perception. Online, we assume we are given a voxel map representation of the world, a starting configuration and a goal pose $P_{\text{g}} \in SE(3)$. The planning then proceeds in three phases.

1) Build the collision set $CS \in \mathfrak{P}(\mathcal{I})$ that contains the ids of all nodes that are colliding with the voxel map. Given a point cloud, this is done by activating all voxels containing points and subsequently using the collision map $\mathcal{M}_c$ to find all nodes in collision.
2) Determine a goal configuration g given our goal pose P_g by solving an inverse kinematics (IK) problem. We use the pose map $\mathcal{M}_p$ and the collision set to find collision-free configurations that lie near P_{g}. The closest configurations are used to warmstart a small IK problem to reach the final goal pose.
3) Connect the start s and goal g configurations up to the roadmap, and use A^* [41] combined with lazy collision checking and greedy short cutting to find the collision-free PWL path.

The right side of Figure 3 depicts the collision map being used to quickly prune out nodes in collision, and the left side of Figure 3 shows the resulting collision-free roadmap that can be efficiently used for path planning. We compute the obstacle representations for collision checking by voxelizing a point cloud observation of the environment and representing the collision geometries of the voxels as spheres.

VI. The Motion Planning Pipeline

Our motion planning pipeline takes a goal end-effector pose, the current configuration, and a voxel map of the perceived obstacles as input, and produces a collision-free trajectory as output. First, the DRM is leveraged to find a target configuration by solving an IK problem, and subsequently to find a collision-free PWL path. The details on how this is done are covered in §V. Next, the path is inflated using EI-ZO (§VI-A). In a last step, we optimize our motion plan by using DBMPs and check the result for collisions using fine-grained sampling along the trajectory. If collisions are present, we use the found collisions to refine the convex sets and re-solve the DBMP trajectory optimization until no collisions are found in the path (§VI-B). We illustrate the entire motion planning pipeline in Fig. 4.

Finally, we propose an approach for inflating multiple paths (§VI-C). Certain DBMPs are not restricted to sequences of convex sets and can handle more general covers of $\mathcal{C}^{\text{free}}$, e.g. [10, 11, 16–18]. Therefore, it can be beneficial to inflate multiple PWL paths in the hope that the safe sets intersect in ways that reveal new paths around obstacles that are not captured by the DRM.

A. Inflating a Single Path

Given a collision-free PWL path with K segments, $v_{0\leq k\leq K}$ such that $\mathcal{L}_k := \mathbf{conv}\{v_k, v_{k+1}\} \subseteq \mathcal{C}^{\text{free}}$ for $k = 0, \dots, K-1$, where v_0 is the start and v_K is the goal configuration, we inflate the path in sequence. Starting at the first line segment with $k = 0$, we construct the probabilistically collision-free polytope $\mathcal{P}_k$, which is guaranteed to contain $\mathcal{L}_k$, and subsequently check if $\mathcal{L}_{k+1}$ is already contained in one of the previously generated polytopes. We only inflate $\mathcal{L}_{k+1}$ if it is not yet contained and increment k. This is repeated iteratively until $k = K-1$.

B. Rapidly Recovering from Detected Collisions

The computed regions are only probabilistically collision-free. This means at planning time, it is possible that the optimized trajectory contains collisions. Given that the PWL path used to generate the polytopes is assumed collision-free, we have a natural way of iteratively refining the sets until the trajectory is collision-free. Given a trajectory, we rapidly check it for collisions by finely discretizing it. If collisions are found, we treat them as candidate points to warm start our search in (8) to modify the corresponding sets. We then re-solve the trajectory optimization and repeat the procedure until the trajectory is collision-free.

In detail, the procedure is the following. For each set, we aggregate all the found collisions inside of the set in $\mathcal{S}_{\text{col}}$. Next, we repeat the last three steps of Alg. 1 in order to exclude these collisions by updating the set. After the found collisions have been excluded, we still need to verify that the new regions still include the entire PWL path. While the line segments that seed the sets remain in the sets by construction, our path inflation approach in §VI-A may have attempted to cover additional line segments with a single set for efficiency. There is no guarantee that these other line segments remain contained in the edited set as well. Therefore, we check that each segment of the path, $\mathcal{L}_{0:K-1}$, is contained in at least one of the convex sets. If not, we inflate the corresponding segment. This procedure is summarized by the flowchart in Fig. 5 and is repeated until the found path is collision-free.

C. Inflating Multiple Paths

Some DBMPs can handle graphs of safe sets rather than just sequences [10, 11, 16–18]. If we inflate more than just the sequence of safe sets associated with the shortest path in the DRM, these approaches can potentially recover higher-quality trajectories. In particular, the convex sets associated with a longer, suboptimal path in the DRM could yield a better optimized trajectory. In an attempt to capture these benefits, we propose a method to inflate additional distinct paths through the DRM, besides the shortest, in order to add more safe sets.

Our heuristic for adding additional paths assumes there exists at least one safe set from a previous path that contains neither the start nor the goal. The heuristic is to select a random convex set that has not yet been selected, add all nodes of the DRM that lie inside of the selected polytope to the collision set CS (see §V-B), rerun A^* on the DRM, and inflate the resulting path according to §VI-A. If at any point the procedure fails, we simply terminate.

The idea behind this heuristic is to use the previously created sets to push the PWL path outside of the currently inflated one.

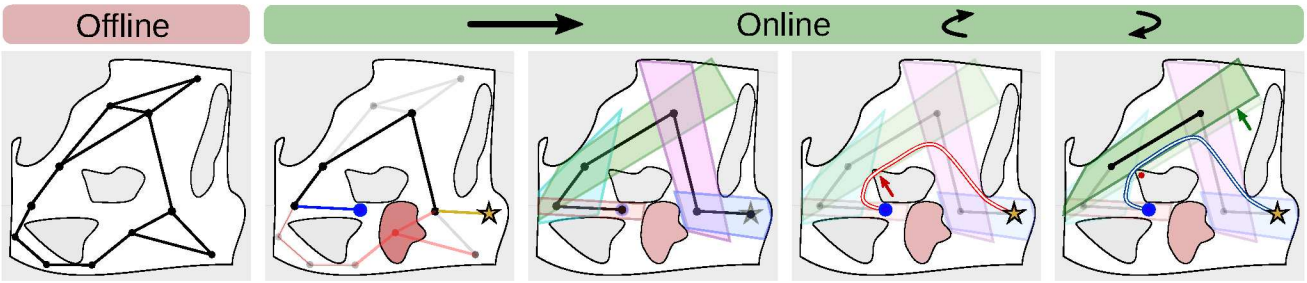

Fig. 4: This figure depicts our motion planning pipeline. Our motion planning pipeline is split into an offline phase in which a DRM is constructed, and an online phase where we use the DRM to generate a collision-free PWL paath, inflate the path, and use a DBMP to optimize a trajectory. Starting from the left, in the first figure the DRM is constructed. In the second, we observe a new obstacle (red blob), and build the collision set (red dot). Using the updated DRM, we then solve the IK problem for the goal (yellow star), and find a collision-free path connecting the start (blue dot) to the goal. We inflate this path to an SCS using EI-ZO in the third figure. In the fourth figure, we solve the motion planning problem using DBMPs and check the trajectory for collisions, finding one inside of the green set, indicated by the red arrow. In the fifth figure, we use the recovery mechanism from §VI-B to modify the green set and produce a collision-free path. In general, these last two steps need to be repeated multiple times.

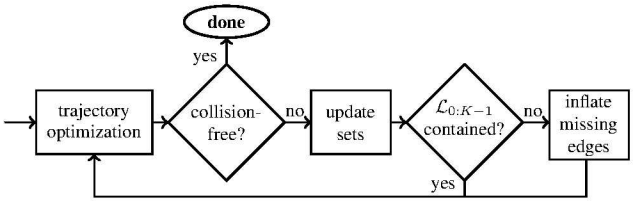

Fig. 5: Flowchart for recovering from collisions found in the optimized trajectory.

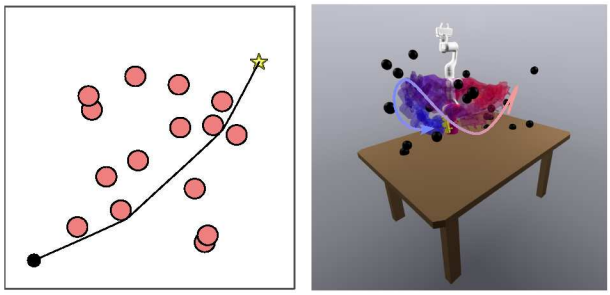

Fig. 6: Our simulation benchmark consists of the Forest environment, shown on the left, and the Franka environment, shown on the right. A collision-free motion plan starting at the red shaded robot and ending at the blue is shown in the Franka environment.

VII. Experimental Evaluation

In this section, we evaluate our proposed motion planning pipeline experimentally. This section is split into two parts. In the first part, §VII-A we discuss our simulation benchmark along with the baseline approach we compare against. In the second part, §VII-B, we validate our approach on our hardware setup.

A. Simulation Benchmark

This section covers the simulation benchmark, which consists of two environments used to generate randomized motion planning problems in two and seven dimensions. For each environment, we solve two types of problems: (1) minimum distance problems, where we seek to find the shortest collision-free PWL path, and (2) minimum time problems, in which we seek a smooth, velocity- and acceleration-constrained curve that minimizes the time needed to traverse between the start and goal.

1) Environments: Our benchmark environments are depicted in Fig. 6. In the "Forest" environment, 15 circular obstacles are scattered randomly in the center portion of a square-shaped domain with a side length of 10. The center portion corresponds to a square with a side length of 7, and the obstacles have a radius of 0.35. In this environment, shown on the left in Fig. 6, the goal is to find a collision-free path from the bottom left to the top right of the environment.

In the "Franka" environment, 20 spherical obstacles are randomly scattered in front of a seven degree-of-freedom Franka Research 3 (FR3) robot manipulator. The collision geometry of the FR3 is approximated with 33 spheres, and the table is modeled as a box. As a result, 1187 collision pairs need to be checked, of which 527 are associated with self-collisions. In this environment, the robot always starts at a random configuration on the right side of the table and needs to plan to a random pose on the left side of the table. The Franka environment is shown on the right in Fig. 6.

2) Problem Types: For each environment, we consider two problem types: minimum distance and minimum time problems. For the minimum distance problem, we seek the shortest collision-free PWL path p, with length $L(p)$, that connects the start s to the goal g configuration,

$$\underset{p}{\textbf{minimize}} \;\; L(p), \tag{14a}$$

$$\textbf{subject to} \;\; p \subseteq \mathcal{C}^{\text{free}}, \tag{14b}$$

$$p_{\text{init}} = s, \;\; p_{\text{term}} = g, \tag{14c}$$

where p_{init} denotes the start, and p_{term} the end of p.

For the minimum time problem, we seek a time-parametrized curve p, that obeys velocity and acceleration constraints, and connects the start s to the goal g in the shortest amount of time T, while starting and stopping at a standstill. The problem reads:

$$\begin{aligned}
\underset{p,T}{\textbf{minimize}} \quad & T, && \text{(15a)}\\
\textbf{subject to} \quad & p \subseteq \mathcal{C}^{\text{free}}, && \text{(15b)}\\
& p(0) = s,\ p(T) = g, && \text{(15c)}\\
& \dot{p}(0) = 0,\ \dot{p}(T) = 0, && \text{(15d)}\\
& v_{\text{min}} \leq \dot{p}(t) \leq v_{\text{max}}\ \forall t \in [0, T], && \text{(15e)}\\
& a_{\text{min}} \leq \ddot{p}(t) \leq a_{\text{max}}\ \forall t \in [0, T], && \text{(15f)}
\end{aligned}$$

where v_{min}, and v_{max} denote the minimum and maximum velocity, and a_{min}, and a_{max} denote the minimum and maximum acceleration.

3) Baseline Approach: We use nonlinear trajectory optimization (NTO), which we warm start from the path found by the DRM, as our baseline approach. To this end, we employ the KinematicTrajectoryOptimization class in Drake [42, link] to transcribe the NTO problems and solve them with SNOPT or IPOPT [43, 44]. For both problem types, (14) and (15), we discretize the trajectory and enforce the collision-avoidance constraint point-wise via [42, link] at N_c points.

For the minimum distance problems, we warm-start the search with the PWL path found by the DRM. For the minimum time problems, we warm-start the search by setting the control points of a 4th order B-Spline to the nodes of the PWL path from the DRM.

4) Employed DBMPs: For the minimum distance problem, given an SCS, the problem is convex and has a straightforward transcription as a second-order cone program, see Appendix C. We solve these problems directly using MOSEK [45] and call the approach LSCS standing for PWL shortest path through an SCS. In the second approach, we inflate two paths using §VI-C, and solve the mixed-integer convex program formulation of the Euclidean distance shortest path problem on a graph of convex sets problem [46, Eqn. 5.5] using Gurobi [47]. We abbreviate this approach as LGCS standing for PWL shortest path through a graph of convex sets.

We choose these two approaches for the minimum distance problem to evaluate the speed and reliability of the simple LSCS approach, and explore the value that is added by inflating multiple paths, by using the more expensive LGCS.

For the minimum time problem, we use the recently developed SCSTrajopt [15], which supports the minimum time objective and rapidly finds smooth, velocity-, and acceleration-constrained trajectories using alternations between two convex optimizations. We solve these problems with Clarabel [48], and abbreviate this approach as SCSTO.

5) Results: The simulation experiments are run on an Intel i9-10850K CPU and a NVIDIA GeForce RTX 3090 using the 560.28.03 graphics driver and CUDA 12.6.

We solve each motion planning problem for 10 random seeds of the environment, 10 random seeds for the DRM construction, and 4 different road map sizes, resulting in 400 planning problems for each trajectory optimization approach. The DRM sizes for the Forest environment are 200, 400, 800, and 1600, and for the Franka environment, the sizes are 3000, 6000, 9000, and 12000 nodes. We tuned NTO individually for both environments and problem types. Please refer to Appendix D for more specifics on the used parameters. The benchmark results are summarized in Tab. I.

In our experiments, the SCS approaches provide a substantial speedup and boost in the reliability over NTO. We find that, provided DRM success, the SCS approaches never failed, and computed solutions 18.5 times faster in the forest and 15.7 times faster in the franka environment, on average. Given DRM success, NTO produced a collision-free trajectory in 71.5% of the instances in the Forest environment and 72.8% of the instances in the Franka environment. We found tuning NTO for the minimum time problem instances in the Forest environment particularly challenging, only achieving a success rate of 45.9%.

Provided NTO succeeded, it produced equivalent cost trajectories on the Forest minimum distance. On the minimum time instances, it improved on the cost by 7.7%, provided the optimizer found a solution. For the Franka minimum distance and minimum time instances, it reduced the cost to the SCS approaches by 33.7%, and 28%, respectively.

Inflating two paths and solving the minimum distance problem using LGCS only yielded marginal improvements in the cost of the trajectory, while dramatically increasing the computation time.

For the Forest environment, the recovery mechanism described in §VI-B, was triggered in 5.6% of the minimum distance and 0.46% minimum time problems. In the Franka environment, it was triggered in 40.22% and 21.36% of the minimum distance and minimum time instances, respectively.

In summary, our pipeline affords a slight increase in trajectory cost, at the benefit of substantially increasing reliability and reducing computation times.

B. Validation on Hardware

In this section, we validate our pipeline on our system that consists of a KUKA LBR iiwa 7 R800 and three Intel RealSense D415 depth cameras for perceiving changes to the table-top environment shown in Fig. 1 and Fig. 7. For this setup, we only inflate the shortest path in the DRM and use SCSTrajopt [15] with Clarabel [48] to find good solutions to (15). We constrain the velocity between -1 and 1 [$\frac{rad}{s}$] and the acceleration between -1 and 1 [$\frac{rad}{s^2}$] for all joints.

In our hardware experiments, we alternate between pose targets on the right and the left of the table. Between plans, we modify the environment, as shown in Fig. 7, by placing or moving obstacles in the center of the table for the robot manipulator to plan around. We approximate the robot collision geometries with 11 boxes, resulting in 105 self-collision pairs. For our hardware experiments, we use an

Problem	Minimum Distance						Minimum Time			
Environment	Forest			Franka			Forest		Franka	
DRM SR	1.00			0.961			1.00		0.961	
DRM num. LS	3.45±0.86			4.49±1.24			3.45±0.86		4.49±1.24	
DRM path len.	10.90±0.40 [m]			8.55±2.28 [rad]			10.90±0.40 [m]		8.55±2.28 [rad]	
TO Approach	NTO	LSCS	LGCS	NTO	LSCS	LGCS	NTO	SCSTO	NTO	SCSTO
cost (S only)	**10.727±0.3**	10.770±0.3	10.732±0.3	**5.519±1.1**	7.382±1.7	7.110±1.6	**9.363±0.2**	10.096±0.7	**5.057±1.2**	6.473±1.5
time total [ms]	357.8±2520.3	**18.2±9.8**	96.8±142.1	3084.6±2938.0	**152.5±89.1**	4441.6±10194	335.1±167.2	**19.3±7.8**	4147.8±2674.6	**307.9±226.3**
time TO [ms]	357.8±2520.3	**13.9±5.7**	91.3±138.6	3084.6±2938.0	**17.4±14.4**	4236.5±10135	335.1±167.2	**15.9±6.1**	4147.8±2674.6	**189.1±170.7**
time CS [ms]	-	4.2±4.7	5.5±5.5	-	135.1±76.0	205.2±107.8	-	3.4±2.6	-	118.8±61.0
num. CS	-	2.45±0.9	3.67±2.3	-	3.49±1.2	6.18±2.8	-	2.45±0.9	-	3.49±1.2
TO SR	0.993	**1.000**	**1.000**	0.863	**1.000**	0.993	0.459	**1.000**	0.735	**1.000**
CFR	0.970	**1.000**	**1.000**	0.725	**1.000**	0.993	0.459	**1.000**	0.732	**1.000**
CFR given TO S	0.977	**1.000**	**1.000**	0.840	**1.000**	**1.000**	**1.000**	**1.000**	0.995	**1.000**

abbreviation	S	SR	CFR	TO	LS	CS
expanded	success	success rate	collision-free rate	trajectory optimization	line segments	convex sets

TABLE I: Simulation results of solving the minimum distance and minimum time problems on both the Forest and Franka environments. The table indicates the averages and empirical standard deviations of the statistics across the different road map sizes, and random seeds for the environment and road map construction provided the DRM found a solution.

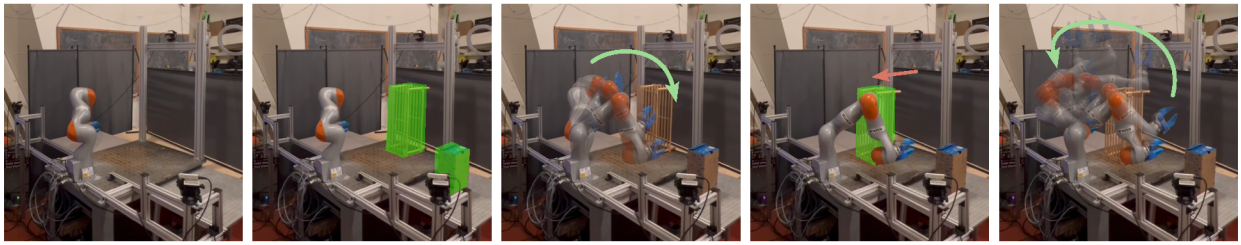

Fig. 7: In our hardware experiments, we alternate between planning to the right and the left side of the table. Between plans we place, move and remove obstacles. Starting from the left, first we place the shoe rack and box, highlighted in green, then we run our planner and move the robot to the right side of the table. Next, we slide the shoe rack closer to the robot and run our planner again. The shoe rack forces the arm to move in a higher arc to reach the left side of the table.

Intel i9-7900X CPU, a Nvidia GeForce 2080Ti GPU using the 555.42.06 driver and CUDA 12.5.

We implement a simple perception pipeline that uses the Python interface to the depth cameras and down-samples the point clouds to voxels with a side length of 8 cm. We associate a sphere collision geometry that circumscribes each voxel as shown in the center frame of Fig. 1.

The timing and auxiliary statistics of 15 trajectories with changing obstacle setups are summarized in Tab. II. For additional details please refer to Appendix D and the video accompanying this paper that demonstrates the setup.

We find that our entire pipeline ran in 0.82 seconds on average, of which only around 0.12 seconds were spent on constructing and editing the safe sets, and 0.08 seconds were spent running SCSTrajOpt. The planner did not fail during our experiments in this setup.

Component	Time [ms]	Auxiliary Stats.	Value
perception	369.7 ± 16.7	num. line seg.	2.9 ± 0.6
IK	130.1 ± 79.5	num. safe sets	1.93 ± 0.57
DRM	116.8 ± 34.8	col. checks / set	121018 ± 24262
convex sets	122.5 ± 43.1	hyperplanes / set	60.7 ± 14.4
SCSTO	82.8 ± 67.9	voxels	359.6 ± 76.4
overhead	1.5 ± 0.6	col. pairs	4431.2 ± 916.7
total	823.4 ± 122.6	self-col. pairs	105

TABLE II: Statistics (the mean and standard deviation) for the hardware experiments. For the 15 collected trajectories we aggregate the timing information on the left, and auxiliary information on the right. Between each of the trajectories, the obstacle setup was modified similarly to Fig. 7.

VIII. Limitations

We discuss four current limitations of our approach.

A. Local and Global Improvements with Nonlinear Solvers

DBMPs are restricted to searching for trajectories inside of the union of the convex sets. If our method seeds the set construction with a bad path, or our approach for finding solutions to (8) results in small sets, it is possible that no high-quality trajectories are covered; the result will potentially have a high cost. For nonlinear trajectory optimization, on the other hand, one can get lucky, and, starting from a bad initial guess, the solver can escape to a better local minimum and still find a good trajectory. We believe that this is reflected in the lower trajectory costs in Tab. I when the solver succeeded. We hoped to recover this behavior for the minimum distance problem instances by inflating multiple paths, which indeed created more intricately connected graphs of convex sets between the individual SCS. Unfortunately, resulting improvements on path length were only marginal, even when solving the trajectory optimization to global optimality with the LGCS approach.

The low trajectory costs of NTO, although, demand scrutiny. Depending on the use case, a system failure could have a high cost (e.g. human intervention) which could make the expected cost of NTO substantially worse than that of the SCS approaches.

B. GPU Memory

Currently, the resolution of the voxel maps we can handle is quite limited by the GPU memory. The layout of our collision checker requires checking the robot for collisions against each voxel for all configurations in parallel. Hence, increasing the resolution of the voxel map increases the memory requirements for the collision checker cubically. We aim to mitigate this in the future by representing the environment using a signed distance field [49–51].

C. Scaling to Higher Dimensions

Our experiments only investigate systems up to 7 degrees of freedom. We have yet to investigate how our method works in higher dimensions, e.g. in mobile or bimanual systems. In particular, it remains to be seen up to what dimension our approaches for constructing dynamic roadmaps and configuration-space sets remain effective.

D. Inverse Kinematics

Our hardware experiments only featured simple inverse kinematics problems. The obstacles were placed such that they would not overlap with the target end effector pose. We found our inverse kinematics approach to be a weak point of the pipeline, and have not tested its limits in this paper. In the future, we aim to improve this with an approach that is more tailored to the individual robot we are running our pipeline on, e.g. by using analytic inverse kinematics [52].

IX. Conclusion

We have proposed a massively parallelizeable algorithm, EI-ZO, for inflating collision-free line segments to probabilistically collision-free polytopes in robot configuration space. We have demonstrated this algorithm in a pipeline that enables motion planning in changing environments with perception in the loop. The motion planning pipeline leverages DRMs to rapidly find collision-free piecewise linear paths, inflates the paths with EI-ZO, and leverages DBMPs to optimize the final trajectory while eliminating collisions from the sets with candidate trajectories where necessary.

Our pipeline generates the configuration-space convex sets fast enough such that it enables using DBMPs in changing configuration spaces for the first time. As a result, the user can encode nontrivial costs, such as traversal time, and constraints, such as velocity and acceleration bounds, and still profit from the speed and reliability of these approaches.

Our experimental evaluation demonstrates that generating sequences of convex sets and using corresponding trajectory optimization approaches is not only fast but also reliable. On our simulation benchmark, this approach did not fail provided the DRM found a solution, which resulted in a 27.9% increase in reliability and a 17.1 times speedup over the nonlinear trajectory optimization baseline. Currently, this comes with the trade-off that this baseline tends to produce trajectories with a lower cost, provided the solver finds a solution.

In conclusion, our procedure is complementary to advances in sampling-based motion planning and advances in DBMPs. We hope to leverage the former to improve path generation, e.g. via [5, 53, 54], and the latter to expand the toolbox of supported costs and constraints for the user. In the future, we seek to leverage these advances and deploy our pipeline in practical pick-and-place and higher degree-of-freedom applications such as mobile manipulation.

Acknowledgments

We thank Tobia Marcucci for his insights and for providing the code for SCSTrajopt. We further thank Rebecca H. Jiang, Alexandre Amice, Nicholas Pfaff, Evelyn Fu, Thomas Cohn, Hongkai Dai, Xuchen Han, and the Mobile Manipulation Team at the Toyota Research Institute for many fruitful discussions, help with the software development, and assistance with the hardware experiments. We are grateful for the funding provided by the Toyota Research Institute.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] B. Siciliano and O. Khatib, "Robotics and the handbook," in *Springer Handbook of Robotics*. Springer, 2016, pp. 1–6.

[3] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[4] J. Pan and D. Manocha, "Gpu-based parallel collision detection for fast motion planning," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[5] W. Thomason, Z. Kingston, and L. E. Kavraki, "Motions in microseconds via vectorized sampling-based planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 8749–8756.

[6] A. Orthey, C. Chamzas, and L. E. Kavraki, "Sampling-based motion planning: A comparative review," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2023.

[7] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE international conference on robotics and automation*. IEEE, 2009, pp. 489–494.

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

[10] 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.

[11] T. Marcucci, P. Nobel, R. Tedrake, and S. Boyd, "Fast path planning through large collections of safe boxes," *IEEE Transactions on Robotics*, 2024.

[12] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1476–1483.

[13] 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.

[14] 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.

[15] T. Marcucci, M. Halm, W. Yang, D. Lee, and A. D. Marchese, "A biconvex method for minimum-time motion planning through sequences of convex sets," *Accepted for publication in Robotics: Science and Systems Foundation*, 2025.

[16] R. Natarajan, C. Liu, H. Choset, and M. Likhachev, "Implicit graph search for planning on graphs of convex sets," *arXiv preprint arXiv:2410.08909*, 2024.

[17] S. Y. C. Chia, R. H. Jiang, B. P. Graesdal, L. P. Kaelbling, and R. Tedrake, "Gcs*: Forward heuristic search on implicit graphs of convex sets," *arXiv preprint arXiv:2407.08848*, 2024.

[18] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 42–49.

[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*, vol. 43, no. 9, pp. 1322–1341, 2024.

[20] M. Petersen and R. Tedrake, "Growing convex collision-free regions in configuration space using nonlinear programming," *arXiv preprint arXiv:2303.14737*, 2023.

[21] P. Werner, T. Cohn, R. H. Jiang, T. Seyde, M. Simchowitz, R. Tedrake, and D. Rus, "Faster algorithms for growing collision-free convex polytopes in robot configuration space," *arXiv preprint arXiv:2410.12649*, 2024.

[22] 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," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 10 359–10 365.

[23] ROS.org, "Urdf - unified robot description format," 2024, accessed: 2024-06-09. [Online]. Available: http://wiki.ros.org/urdf

[24] O. S. R. Foundation, "Sdf specification," 2024, accessed: 2024-06-09. [Online]. Available: http://sdformat.org

[25] 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 International Conference on Robotics and Automation*. IEEE, 2005, pp. 3486–3491.

[26] Q. Wang, Z. Wang, M. Wang, J. Ji, Z. Han, T. Wu, R. Jin, Y. Gao, C. Xu, and F. Gao, "Fast iterative region inflation for computing large 2-d/3-d convex regions of obstacle-free space," *arXiv preprint arXiv:2403.02977*, 2024.

[27] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015, pp. 109–124.

[28] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[29] L. Lovász, "Hit-and-run mixes fast," *Mathematical programming*, vol. 86, 1999.

[30] A. Amice, P. Werner, and R. Tedrake, "Certifying bimanual rrt motion plans in a second," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 9293–9299.

[31] M. Tang, Y. J. Kim, and D. Manocha, "Ccq: Efficient local planning using connection collision query," in *Algorithmic Foundations of Robotics IX: Selected Contributions of the Ninth International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2011, pp. 229–247.

[32] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact collision checking of robot paths," *Algorithmic foundations of robotics V*, pp. 25–41, 2004.

[33] J. Pan, L. Zhang, and D. Manocha, "Collision-free and curvature-continuous path smoothing in cluttered environments," 2012.

[34] NVIDIA Corporation, *NVIDIA CUDA Toolkit Documentation*, NVIDIA Corporation, 2024. [Online]. Available: https://docs.nvidia.com/cuda/

[35] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[36] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[37] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4399–4404.

[38] R. Cheng, J. Petersen, J. Borders, D. Helmick, L. Kaul, D. Kruse, J. Leichty, C. Matl, C. Papazov, K. Shankar *et al.*, "Motion planning to cartesian targets leveraging large-scale dynamic roadmaps," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2023, pp. 1–7.

[39] H. Liu, X. Deng, H. Zha, and D. Ding, "A path planner in changing environments by using wc nodes mapping coupled with lazy edges evaluation," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2006, pp. 4078–4083.

[40] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, vol. 1. IEEE, 2000, pp. 521–528.

[41] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[42] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[43] 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.

[44] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, 2006.

[45] M. ApS, "MOSEK Optimization Suite," 2019.

[46] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[47] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com

[48] P. J. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," *arXiv preprint arXiv:2405.12762*, 2024.

[49] T.-T. Cao, K. Tang, A. Mohamed, and T.-S. Tan, "Parallel banding algorithm to compute exact distance transform with the gpu," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, 2010, pp. 83–90.

[50] Y. Chen, S. Lai, J. Cui, B. Wang, and B. M. Chen, "Gpu-accelerated incremental euclidean distance transform for online motion planning of mobile robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6894–6901, 2022.

[51] A. Millane, H. Oleynikova, E. Wirbel, R. Steiner, V. Ramasamy, D. Tingdahl, and R. Siegwart, "nvblox: Gpu-accelerated incremental signed distance field mapping," 2024.

[52] C. Faria, F. Ferreira, W. Erlhagen, S. Monteiro, and E. Bicho, "Position-based kinematics for 7-dof serial manipulators with global configuration control, joint limit and singularity avoidance," *Mechanism and Machine Theory*, vol. 121, pp. 317–334, 2018.

[53] T. S. Wilson, W. Thomason, Z. Kingston, L. E. Kavraki, and J. D. Gammell, "Nearest-neighbourless asymptotically optimal motion planning with fully connected informed trees (fcit*)," *arXiv preprint arXiv:2411.17902*, 2024.

[54] C. W. Ramsey, Z. Kingston, W. Thomason, and L. E. Kavraki, "Collision-affording point trees: Simd-amenable nearest neighbors for fast collision checking," *arXiv preprint arXiv:2406.02807*, 2024.

APPENDIX

A. Distance Function Convexity

Proof: Let $\mathcal{C} \subseteq \mathbb{R}^n$ be a convex set, $\lambda \in [0,1]$, and let $x_1, x_2 \in \mathbb{R}^n$. Then, we directly confirm that Jensen's inequality holds.

$$\begin{aligned} &\lambda \mathbf{dist}_{\mathcal{C}}(x_1) + (1-\lambda)\mathbf{dist}_{\mathcal{C}}(x_2) \\ &= \lambda||x_1 - z_1|| + (1-\lambda)||x_2 - z_2|| \end{aligned} \tag{16}$$

where $z_1, z_2 \in \mathcal{C}$ are any minimizers of the respective distances to x_1 and x_2. Observe that

$$\begin{aligned} &\lambda||x_1 - z_1|| + (1-\lambda)||x_2 - z_2|| \\ &\geq ||\lambda x_1 + (1-\lambda)x_2 - (\lambda z_1 + (1-\lambda)z_2)||, \end{aligned} \tag{17}$$

by the triangle inequality, and due to the convexity of $\mathcal{C}$, we have $(\lambda z_1 + (1-\lambda)z_2) \in \mathcal{C}$. Hence, it holds

$$\begin{aligned} &||\lambda x_1 + (1-\lambda)x_2 - (\lambda z_1 + (1-\lambda)z_2)|| \\ &\geq \mathbf{dist}_{\mathcal{C}}(\lambda x_1 + (1-\lambda)x_2), \end{aligned} \tag{18}$$

showing that Jensen's inequality is satisfied. □

B. DRM Construction

Our offline construction of the DRM closely follows [38]. First, we sample a batch of configurations uniformly in the collision-free configuration space of our system to build up our node map $\mathcal{M}_n$. Next, we construct our node adjacency map $\mathcal{M}_a$ from the sampled nodes. We limit the number of neighbors for each node to k, and only add edges between two configurations if both their L_2 distance in configuration space is smaller than d_{CS} and the distance in task space of a selected frame is less than d_{TS}. Then we ensure that the adjacency matrix is symmetric by adding missing edges.

To build the collision map $\mathcal{M}_c$, we discretize the task space of the robot into voxels. We use a sphere with radius $\frac{\sqrt{3}}{2}s$, where s is the side length of a voxel, that circumscibes each voxel for collision checking. For each voxel, we then check for collision with *each* node of the roadmap, and store the ids of all the nodes in the roadmap that this voxel is in collision with. The resultant mapping between voxels to nodes in collision then serves as our collision map, and this allows for fast look-up of collisions during online planning.

C. Polygonal Shortest Paths through Sequences of Convex Sets

Given a sequence of convex sets $\mathcal{P}_{1:M}$ such that successive sets intersect, $\mathcal{P}_i \cap \mathcal{P}_{i+1} \neq \emptyset$, then the LSCS second-order cone program is

$$\begin{aligned}
\underset{v_0 \leq i \leq M+1}{\textbf{minimize}} \quad & \sum_{i=1}^{M} ||v_i - v_{i+1}||_2, && \text{(19a)}\\
\textbf{subject to} \quad & v_1 = s, && \text{(19b)}\\
& v_{M+1} = g, && \text{(19c)}\\
& v_i = v_{i+1} \text{ for } i = 1, \ldots, M, && \text{(19d)}\\
& v_i \in \mathcal{P}_i, \text{ for } i = 1, \ldots, M, && \text{(19e)}\\
& v_i \in \mathcal{P}_{i-1}, \text{ for } i = 2, \ldots, M, && \text{(19f)}
\end{aligned}$$

where v_i are the knot points of the polygonal path.

D. Supplementary Information on Experimental Evaluation

In this section, we collect additional statistics and parameter values that we used in our experiments. This section is split into two parts. First, we cover the simulation experiments, and then the hardware experiments.

Simulation The simulation experiments are run on a PC running Ubuntu 22.04 with an Intel i9-10850K CPU and an NVIDIA GeForce RTX 3090 using the 560.28.03 graphics driver and CUDA 12.6.

With this setup, our collision checker has a throughput of around 19.9 million configurations per second in the Forest environment and 8.6 million configurations per second in the Franka environment when using a batch size of two million configurations.

We found the greatest success solving all NTO instances in the Forest environment using SNOPT. We employ $N_c = 20$ and $N_c = 100$ for the minimum distance and minimum time problems, respectively. In the Franka environment, we found the greatest success solving the minimum distance problems with SNOPT and the minimum time instances with IPOPT. In both cases $N_c = 150$ yielded a good trade-off between computation time and the resulting trajectories being collision-free when the solver reported success. The parameters employed in the DRM construction for both the Forest and the Franka environments are summarized in Tab. V. The employed EI-ZO parameters are summarized in Tab. III for the Forest environment, and in Tab. IV for the Franka environment. With these settings, EI-ZO would typically pass the unadaptive test in less than 10 iterations. For the lazy collision checking, we used a step size of 0.1 [m] or 0.1 [rad] in both environments, respectively.

Hardware For our hardware experiments, we use an Intel i9-7900X CPU, a Nvidia GeForce 2080Ti GPU using the 555.42.06 driver and CUDA 12.5. With this setup, our collision checker has a throughput of around 6.9 million configurations per second when using a batch size of 2 million. To generate the PWL paths, we employed a DRM with 35000 nodes. We used the same EI-ZO parameters as in Tab. IV, except for N_f, which was increased to 20. In those experiments, the recovery mechanism from §VI-B was never triggered. See Tab. VI for the employed parameters during the DRM construction. For the lazy collision checking, we used a step size of 0.1 [rad].

Parameter	Symbol	Value
admissible uncertainty	δ	0.05
admissible fraction in collisions	ε	0.01
number of samples to update	N_p	1000
max number of faces per iteration	N_f	10
number of mixing steps for sampling	N_{ms}	30
step back	Δ_{max}	0.01

TABLE III: Employed parameters in the Forest environment for EI-ZO.

Parameter	Symbol	Value
admissible uncertainty	δ	0.005
admissible fraction in collision	ε	0.005
number of samples to update	N_p	10000
max number of faces per iteration	N_f	10
number of mixing steps for sampling	N_{ms}	60
step back	Δ_{max}	0.01

TABLE IV: Employed parameters in the Franka environment for EI-ZO.

Parameter	Symbol	Value
max. task space distance	d_{TS}	10 [m]
max. configuration space distance	d_{CS}	10 [m]
max. neighbors	k	10
offline voxel length	s	0.06 [m]

TABLE V: Employed parameters in the Franka and Forest environments for constructing the DRMs.

Parameter	Symbol	Value
max. task space distance	d_{TS}	0.45 [m]
max. configuration space distance	d_{CS}	4.5 [rad]
max. neighbors	k	10
offline voxel length	s	0.06 [m]

TABLE VI: Employed DRM construction parameters for the hardware expermients.