IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Scaling Infeasibility Proofs via Concurrent, Codimension-one, Locally-updated Coxeter Triangulation

Sihui Li[1] and Neil T. Dantam[1]

***Abstract*—Achieving a complete motion planner that guarantees a plan or infeasibility proof in finite time is challenging, especially in high-dimensional spaces. Previous efforts have introduced asymptotically complete motion planners capable of providing a plan or infeasibility proof given long enough time. The algorithm trains a manifold using configuration space samples as data and triangulates the manifold to ensure its existence in the obstacle region of the configuration space. In this paper, we extend the construction of infeasibility proofs to higher dimensions by adapting Coxeter triangulation's manifold tracing and cell construction procedures to concurrently triangulate the configuration space codimension-one manifold, and we apply a local elastic update to fix the triangulation when part of it is in the free space. We perform experiments on 4-DOF and 5-DOF serial manipulators. Infeasibility proofs in 4D are two orders of magnitude faster than previous results. Infeasibility proofs in 5D complete within minutes.**

***Index Terms*—Motion and Path Planning, Computational Geometry**

I. Introduction

A Complete motion planner must either return a path from the start to the goal or report path non-existence in finite time. Complete motion planning would benefit many high-level planning problems where motion planning is a sub-problem [1]–[4]. However, achieving completeness is challenging when the configuration spaces are continuous, high-dimensional, and implicitly defined. Many sampling-based motion planners are *probabilistically complete* [5]–[10], meaning that if a path exists in the configuration space, the planner returns a plan given long enough time. Recent work [11] defined the notion of *asymptotic completeness* in which the planner returns a path or reports path non-existence given long enough time. Compared to probabilistic completeness, asymptotic completeness offers the additional capability of producing path non-existence or infeasibility proofs. In this paper, we address scalability challenges of asymptotically complete motion planning.

Previous work [12], [13] proposed an asymptotically complete sampling and learning based motion planning framework. The constructed infeasibility proof is a closed manifold or polytope that is fully in the obstacle region and separates the start and goal. The algorithm runs in parallel with sampling-based motion planners and uses the sampled configurations as data to learn a manifold. A triangulation of this manifold then provides the infeasibility proof. Each facet of the triangulation is checked to ensure containment in the obstacle region of the configuration space. This prior algorithm scaled to 4D configuration spaces. The major scalability limit is the triangulation step, which takes the majority of the runtime. This paper presents an approach to scale the triangulation step by adapting and integrating a new computational geometry tool, *Coxeter triangulation* [14].

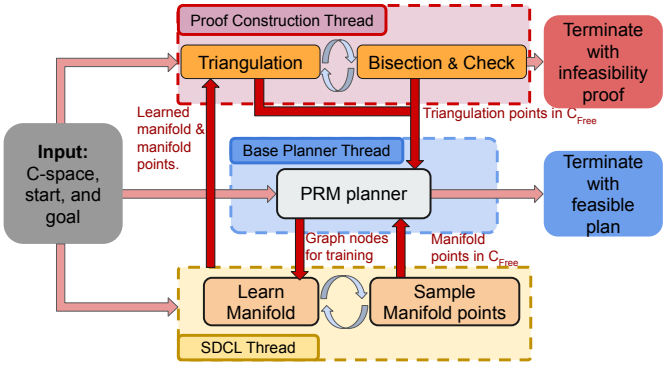


Fig. 1: Algorithm overview. Three threads run in parallel. The algorithm terminates with a plan or an infeasibility proof.

The same training and sampling framework can also solve narrow passage motion planning problems, which is called *sample-driven connectivity learning* (SDCL) [15]. There is a natural integration of SDCL and infeasibility proof construction since both use the same manifold learned from configuration samples. This paper employs a new algorithmic framework that combines SDCL and infeasibility proof construction (Figure 1).

In this work, we present an asymptotically complete motion planner that generates infeasibility proofs using an adaptation of Coxeter triangulation and simultaneously improve narrow passage motion planning. The infeasibility proof construction is two orders of magnitude faster than previous work. The major contributions are listed here. 1) We adapt Coxeter triangulation [14] for codimention-one manifold with elastic updates to locally fix the triangulation to resolve the scalability limit. 2) We adapt the overall algorithm structure to facilitate narrow passage motion planning. 3) We utilize parallel computing techniques for triangulation, which is enabled by our adaptation of Coxeter triangulation and overall framework. 4) We propose a new bisection and checking step for facets in the triangulation

Manuscript received: June, 13, 2023; Accepted September, 23, 2023. This paper was recommended for publication by Editor Hanna Kurniawati upon evaluation of the Associate Editor and Reviewers' comments. This work was supported by NSF IIS-1849348, NSF CCF-2124010, ONR N00014-21-1-2418, and ARL TBAM-CRP [W911NF- 22-2-0235].

[1] Sihui Li and Neil T. Dantam are with the Department of Computer Science, Colorado School of Mines, USA. {li, ndantam}@mines.edu

Digital Object Identifier (DOI): 10.1109/LRA.2023.3327655

when configuration space penetration depth cannot be robustly calculated.

We evaluate this algorithm on 4-DOF and 5-DOF manipulators. Constructing infeasibility proofs in 4D is two orders of magnitude faster than [13], and in 5D, we can construct infeasibility proofs within a few minutes, which was not previously possible. We further discuss scalability issues in Sec. VII. Finally, in evaluated scenes with feasible plans through narrow passages, our framework found plans two orders of magnitude faster than a baseline of RRT-Connect.

II. Related Work

A. Completeness and Sampling-based motion planning

Achieving completeness in motion planning is desired but difficult since we typically do not have an explicitly defined configuration space, especially in high dimensions. Sampling-based motion planning is an effective and widely applicable strategy that incorporates typically random sampling to search a metric configuration space [5]–[8], [10], [16]–[23], and thus only requires a configuration validity checker. However, sampling-based planners have traditionally only offered probabilistic completeness. If the planner times out without returning a plan, there is no guarantee of plan non-existence, since the planner may just need more time to find the path. This uncertainty about plan existence poses challenges when motion planning is a sub-routine in a higher-level planning problem [1]–[4]. Conversely, an asymptotically complete planner [11], will eventually return a plan or an infeasibility proof. The infeasibility proof provides an exact proof of plan non-existence, which could help eliminate some of the search branches of a higher-level planning problem. Asymptotic completeness brings us one step closer to completeness but does not guarantee termination in finite time.

B. Infeasibility Proofs

Some previous works construct exact path non-existence guarantees. [24] proves path non-existence for single, rigid objects in a 2D or 3D workspace to guarantee stable grasp. [25] considers the specific problem of a rigid body passing through a narrow gate. [26] constructs alpha-shapes in the obstacle region to query the connectivity of two configurations, which works for up to 3-dimensional configuration spaces. These methods do not apply to general manipulators' configuration spaces. [27] proposed a method to construct infeasibility proofs by growing facets in the obstacle region and then identifying closed polytopes from the set of facets. This method is computationally expensive due to the combinatorial identification step, scaling only to 3-DOF manipulators within reasonable time limits.

There are also methods that provide approximate path non-existence guarantees. Visibility [28] and sparsity [29] based planners achieve high coverage of the free space, so if no plan is found when the algorithm terminates, the problem may be considered infeasible [30]. Deterministic sampling-based motion planning also provides certain guarantees on plan non-existence [31]–[34]. If no plan is found, then either no solution exists or a solution exists only through some narrow passages. Previous works have also applied learning-based methods to predict infeasible plans [35]–[39]. However, these methods do not provide definitive plan nonexistence guarantees.

This paper uses an overall structure similar to previous work in [12], [13], in which we proposed a sampling and learning based infeasibility proof construction algorithm. The algorithm runs in parallel with a sampling-based motion planner. First, using the base planner's search tree or graph, the algorithm learns and samples a manifold. Then, the manifold is triangulated and each facet of the resulting triangulation is checked to ensure the triangulation is entirely in $\mathcal{C}_{\text{obs}}$. This paper uses a similar strategy and adapts new computational geometry tools [14] in the triangulation step for faster computation to extend the algorithm to higher dimensions.

C. Narrow Passage Motion Planning

Narrow passages pose challenges for sampling-based motion planners due to low sampling probabilities. Various strategies aim to address this issue [40]–[42]. The extreme case of narrow passages is when obstacles in the workspace cause the configuration space narrow passage to become occluded (making the problem infeasible), which sampling-based motion planning has traditionally not addressed. This work integrates infeasibility proof construction with a narrow passage motion planner SDCL [15] so that when plans exist in narrow passages, they are quickly found, and when the plans become infeasible, the algorithm switches to constructing the infeasibility proof.

III. Problem Definition

We focus on kinematic motion planning and aim to achieve asymptotic completeness—that is, to return a plan when one exists and to return an infeasibility proof when no plan exists. A motion planning problem [43] consists of a configuration space $\mathcal{C}$ of dimension n, a start configuration $\boldsymbol{q}_{\text{start}}$, and a goal configuration $\boldsymbol{q}_{\text{goal}}$. The configuration space $\mathcal{C}$ is the union of the disjoint obstacle region $\mathcal{C}_{\text{obs}}$ and free space $\mathcal{C}_{\text{free}}$. For high-DOF manipulators, $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ are implicitly defined using a validity checking function that takes a configuration as input, returns false if the configuration collides with obstacles, and returns true otherwise. Both $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are in $\mathcal{C}_{\text{free}}$. When a plan exists, the output is a plan σ such that $\sigma[0,1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \boldsymbol{q}_{\text{start}}$, $\sigma[1] = \boldsymbol{q}_{\text{goal}}$. When there is no feasible plan, the output is an infeasibility proof $\mathcal{M}$. An infeasibility proof $\mathcal{M}$ is a closed manifold that lies entirely in the obstacle region and separates $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ [13].

A. Requirements and Assumptions

We have additional requirements on the configuration space. First, the obstacle region of the configuration space must be entirely ε-blocked, meaning the obstacle region cannot be infinitesimal. Also, we require a Euclidean configuration space instead of a *metric space* (e.g. $\mathcal{SE}(3)$) since the triangulation of the manifold requires Euclidean space. We further define virtual obstacles at the configuration space boundaries so that we can treat boundaries the same as obstacle regions. More details on these requirements are in [11]. Finally, the current algorithm applies to kinematic motion planning only.

IV. Background

We briefly summarize the key prior results in this section, including SDCL [15], infeasibility proof construction [13], and Coxeter triangulation [14], [44].

A. Sample-Driven Connectivity Learning from Roadmaps

SDCL integrates sampling-based planning and machine learning to effectively solve difficult motion planning problems with narrow passages [15]. There are two main steps in SDCL: learning a manifold and sampling the manifold.

In the learning step, SDCL uses a partially-constructed probabilistic roadmap G as training data for a binary classifier. All samples in G that are connectable to $\boldsymbol{q}_{\text{goal}}$ are one class, and all other samples are the other class. The result of learning is a configuration space manifold $F(\boldsymbol{q})$ ($\boldsymbol{q} \in \mathcal{C}$), i.e., the decision boundary of the classifier, that separates the start and goal.

In the sampling step, SDCL finds points on the manifold by solving a non-linear optimization problem to minimize the manifold function's absolute value, $|F(\boldsymbol{q})|$. Sampled manifold points in $\mathcal{C}_{\text{free}}$ offer potential connections between the start and goal components, providing a strong heuristic when planning in configuration spaces with narrow passages [15].

B. Infeasibility Proofs

The algorithm in [13] generates infeasibility proofs in four main steps. The first two steps are similar to learning and sampling a manifold in SDCL (see Sec. IV-A). After training the manifold, the algorithm verifies that the manifold is completely contained in $\mathcal{C}_{\text{obs}}$ to confirm it is an infeasibility proof. Directly checking the manifold itself poses challenges since $\mathcal{C}_{\text{obs}}$ is often implicitly defined. Instead, the algorithm generates a triangulation of the manifold using tangential Delaunay complexes [45], [46] and checks the facets of the triangulation using configuration space penetration depth. When all facets are in the obstacle region, the triangulation is an infeasibility proof according to its definition.

C. Coxeter Triangulation

The authors in [14] introduce the manifold tracing algorithm which uses $\mathbb{R}^n$ Coxeter triangulation (call it $\mathbb{CTR}$ in the following) to triangulate a manifold. The inputs to manifold tracing are an m-dimensional smooth manifold in $\mathbb{R}^n$ and seed points on the manifold. The result is a set of k-dimensional simplices that intersect the manifold (where $k = n - m$, known as the codimension), forming a manifold triangulation later on.

To avoid explicit construction of every point of $\mathbb{CTR}$ (which is infinite), the algorithm uses a *permutahedral* representation [14]. This permutahedral representation supports queries for vertices, faces, and cofaces of a simplex, and locating points on simplices ($\texttt{locate}(\boldsymbol{q})$). Faces are lower-dimensional simplices contained within a simplex. For example, $s.\texttt{face}(1)$ returns all the 1-simplices of a simplex s, which are edges. Cofaces are all simplices that contain the given simplex. For example, $s.\texttt{coface}(n)$ returns all the n-simplices in which simplex s is a face. The $\mathbb{R}^n$ triangulation size is adjustable with a parameter λ_T, which also determines the final manifold triangulation size. In Figure 2, λ_T is each triangle's height. Smaller triangles produce intersection points that are closer to each other, i.e., a finer manifold triangulation.

V. Algorithm

In this section, we discuss the integration of SDCL, Coxeter Triangulation, and infeasibility proof construction. Our overall algorithm operates by integrating SDCL and infeasibility proof construction (see Figure 1). The algorithm has three components and correspondingly three parallel threads. One thread runs a PRM, which provides the samples we use as data for training the manifold. The SDCL thread uses these samples to train the manifold and then samples points on the manifold. If we sample any $\mathcal{C}_{\text{free}}$ points on the manifold, we add these points back to the PRM. Lastly, the infeasibility proof construction thread uses the manifold and manifold points from the SDCL thread for triangulation (see Sec. V-A and Sec. V-B) and checking (see Sec. V-C). If the triangulation passes the check, then we have an infeasibility proof. If the PRM finds a plan, we return the plan. The algorithm terminates with either a plan or an infeasibility proof. This algorithm structure ensures quick narrow passage motion planning by running infeasibility proof construction and SDCL in separate threads.

A. Codimension One Coxeter Triangulation

The proof construction thread takes the most recent manifold and triangulates it. Previous work [12] used tangential Delaunay complexes [45], [46] to triangulate the manifold, However, triangulation with tangential Delaunay complexes requires post-processing to fix inconsistencies, which scales poorly to higher dimensions, and tangential Delaunay complexes does not parallelize well. In this work, we employ and adapt the Coxeter triangulation algorithm [14]. In the manifold tracing algorithm, we implement a new method to calculate intersection points between line segments and the manifold. We also parallelize the manifold tracing iterations. Then, we use the output from manifold tracing to concurrently construct the triangulation and at the same time locally fix the triangulation with elastic updates when there are parts of it in $\mathcal{C}_{\text{free}}$.

1) Manifold Tracing: We collect all points sampled on the manifold into a set Q_{seeds} in the SDCL thread. The inputs to manifold tracing are the learned manifold and Q_{seeds}. In our case, the learned manifold is always (n-1)-dimensional for $\mathcal{C}$ in $\mathbb{R}^n$. This means the codimension is always one and the output simplices are always 1-simplices, i.e., line segments. Algorithm 1 describes the parallelized manifold tracing.

Two features enable concurrent implementation of manifold tracing. First, the final collection of intersecting line segments produced by manifold tracing is unaffected by the graph traversal order because all neighboring line segments are interconnected in $\mathbb{CTR}$ (see Figure 2b) and would be visited regardless of the ordering. Second, we have a set of manifold points Q_{seeds} from the SDCL thread. Consequently, we parallelize manifold tracing by starting from multiple seeds Q_{seeds} in different threads. This parallelization facilitates triangulation in higher dimensions.

Algorithm 1: Parallel Manifold Tracing

Input: $\mathbb{CTR}$; // $\mathbb{R}^n$ Coxeter triangulation
Input: $F(\boldsymbol{q})$; // Up-to-Date manifold
Input: Q_{seeds}; // Seed points on manifold
Output: Ls; // Intersecting line segs
1 $Q \leftarrow \emptyset$; // Empty queue for line segs
#pragma parallel for
2 **foreach** $\boldsymbol{q} \in Q_{\text{seeds}}$ **do**
3 $ls \leftarrow \texttt{locate}(\boldsymbol{q})$; // locate starting simplex, return its line segments
4 $Q.\texttt{add}(ls)$;
5 $Ls.\texttt{add}(ls)$;
#pragma parallel for
6 **repeat**
7 $ls \leftarrow Q.\texttt{pop}()$;
8 **foreach** $c \in ls.\texttt{coface}(2)$ **do**
9 **foreach** $l \in c.\texttt{face}(1)$ **do**
10 **if** $l \notin Ls$ & l intersects F **then**
11 $Q.\texttt{add}(l)$;
12 $Ls.\texttt{add}(l)$;
13 **until** $Q = \emptyset$;

While finding intersecting line segments in Algorithm 1, we must also calculate the intersection points. We find intersection points between line segments and the manifold using two-point bracketing [47]. A line segment intersects the manifold if and only if its endpoints are on opposite sides and thus produce manifold function values with opposite signs. Given our manifold function $F(\boldsymbol{q})$ and line segment endpoints $\mathbf{v}_1$ and $\mathbf{v}_2$, an intersection exists only when $F(\mathbf{v}_1) * F(\mathbf{v}_2) < 0$. We find the intersection by updating bracket $\mathbf{v}_1, \mathbf{v}_2$ using the *false-position* method [47], [48] (see Algorithm 2). Figure 2b shows the $\mathbb{CTR}$ in the 2D scene, and Figure 2c shows the result of manifold tracing, including the line segments and their intersection points with the manifold.

Algorithm 2: Calculate Intersection Point

Input: $F(\boldsymbol{q})$; // Manifold function
Input: $\mathbf{v}_1, \mathbf{v}_2$; // Line segment vertices
Output: $\mathbf{p}$; // Intersection Point
1 **repeat** // False-Position [47], [48]
2 $\mathbf{p} \leftarrow \frac{\mathbf{v}_1 F(\mathbf{v}_2) - \mathbf{v}_2 F(\mathbf{v}_1)}{F(\mathbf{v}_2) - F(\mathbf{v}_1)}$;
3 **if** $F(\mathbf{p}) * F(\mathbf{v}_1) > 0$ **then** $\mathbf{v}_2 \leftarrow p$;
4 **else** $\mathbf{v}_1 \leftarrow \mathbf{p}$;
5 **until** $|F(\mathbf{p})| < \tau$; // within tolerance τ

When manifold tracing returns, the manifold must be closed. We do not provide any boundaries of the manifold to do manifold tracing, so if the manifold is not closed, the graph traversal would not terminate because there would always be neighboring line segments in $\mathbb{CTR}$ that intersect the infinitely extending manifold. In this sense, the triangulation step also proves that the manifold is closed, which is one of the requirements in the definition of an infeasibility proof.

Algorithm 3: Construct Triangulation

Input: Ls; // Line segments
Input: Ps; // Line to intersection map
Output: T; // Triangulation of manifold
1 $M \leftarrow \{\}$; // Concurrent hash table
#pragma parallel for
2 **foreach** $l \in Ls$ **do**
3 **foreach** $c \in l.\texttt{coface}(n)$ **do**
4 $M[c].\texttt{add}(Ps[l])$;
5 $T \leftarrow \texttt{collect-values-for-each-key}(M)$;

2) Construct triangulation: We next construct the triangulation using the line segments and corresponding manifold intersection points output from manifold tracing. Since we always have codimension-one manifolds, we develop a faster routine to construct the triangulation composed of (n-1)-simplices. The algorithm supports efficient parallel construction of the triangulation by incorporating a concurrent hash table [49], [50] (line 1) for data shared between threads. We also perform *elastic updates* (see Sec. V-B) to fix areas on the triangulation that leave the obstacle region.

Algorithm 3 describes the triangulation construction. We start by iterating through the set of line segments. For each line segment, we find its n-dimensional cofaces by querying the permutahedral representation of $\mathbb{CTR}$, which is an n-simplex in $\mathbb{CTR}$ for $\mathbb{R}^n$ configuration space. Different line segments may have the same n-dimensional coface, since an n-simplex has $n * (n-1)$ lines segments. We save each n-simplex and its corresponding line segments' intersection points to the concurrent hash table (line 4). The intersection points of an n-simplex's line segments with the manifold form facets of the triangulation. By grouping all the facets, Algorithm 3 constructs the manifold triangulation. Figure 2d shows the manifold triangulation in the 2D scene. Next, we discuss the elastic update procedure to correct $\mathcal{C}_{\text{free}}$ triangulation vertices.

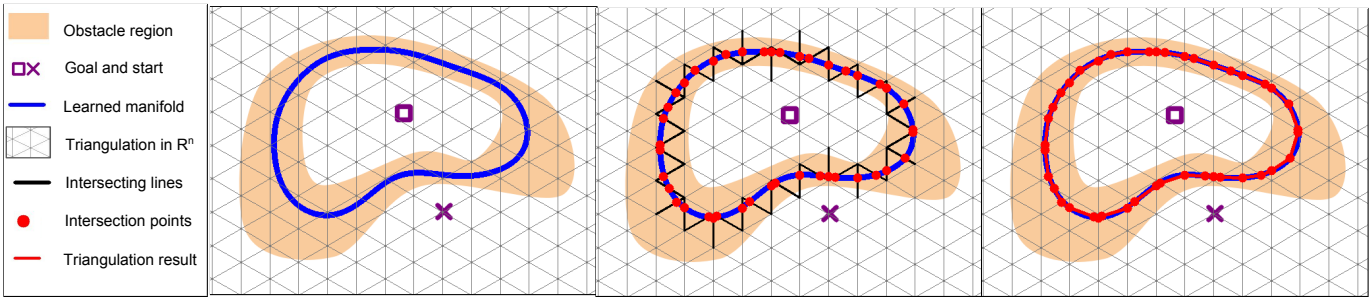
B. Elastic Triangulation Updates

We locally update the triangulation to ensure containment in $\mathcal{C}_{\text{obs}}$. When iterating through line segments, we check whether manifold intersection points are in $\mathcal{C}_{\text{obs}}$. An intersection point in $\mathcal{C}_{\text{free}}$ means that part of the manifold is in $\mathcal{C}_{\text{free}}$. Generally, we could retrain a new manifold and re-triangulate. However, to save the cost of another triangulation, we locally update the triangulation to correct intersection points in $\mathcal{C}_{\text{free}}$. Conceptually, these updates "stretch" the vertices of the existing triangulation to fit a newly trained manifold as if the edges are elastic, so we call this procedure an *elastic update*.

Algorithm 4 summarizes the elastic update, and Figure 3 illustrates a 2D case. For an intersection point in $\mathcal{C}_{\text{free}}$, the elastic update adds the point to roadmap G (line 1), then projects the intersection point onto the most recent manifold by solving the following optimization problem (line 3),

$$\begin{aligned} \min_{\boldsymbol{q}_m} \quad & \text{abs}\left(F(\boldsymbol{q}_m)\right) \\ \text{s.t.} \quad & \boldsymbol{q}_m \in \mathcal{C} \ , \end{aligned} \tag{1}$$

where F is the function of the learned manifold, and $\boldsymbol{q}_m$ is the projected manifold point. This formulation is the same as that



(a) Legend (b) Coxeter triangulation of $\mathbb{R}^2$. (c) Manifold tracing results. (d) Constructed manifold triangulation.

Fig. 2: 2D demonstration shows manifold tracing results and triangulation construction. Line segments of the $\mathbb{R}^n$ triangulation intersect the manifold, and the intersecting points form a triangulation of the manifold.

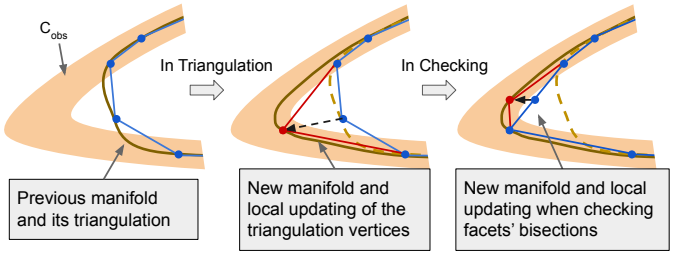


Fig. 3: Elastic triangulation in a 2D configuration space. In the first transition, the triangulation of the previous manifold is fixed using the projection point on the new manifold; In the second transition, a vertex of a facet's bisection is fixed.

used to sample manifold points in SDCL. If we successfully project the point $\boldsymbol{q}_m$ and it is in $\mathcal{C}_{\text{obs}}$, we use this projected point to replace the original intersection point in the final triangulation. If $\boldsymbol{q}_m$ is instead in $\mathcal{C}_{\text{free}}$, we add this projected point to roadmap G (line 5), retrain the manifold (line 6, line 7), and re-project the point until we find a projected manifold point in $\mathcal{C}_{\text{obs}}$ or the projection fails. If projection fails, which means the optimization problem in Equation 1 fails to solve, then $\boldsymbol{q}$ is $\emptyset$ and we re-triangulate. The updated triangulation remains valid since we only change the positions of vertices and not the connections of the edges. The resulting triangulation may have crossings but it is not a concern as long as all facets are in $\mathcal{C}_{\text{obs}}$, which we discuss next.

Algorithm 4: Elastic-Update

Input: G; // Planning Graph
Input: $\boldsymbol{q}$; // Point in $\mathcal{C}_{\text{free}}$
Input: $F(\boldsymbol{q})$; // Up-to-Date Manifold
Output: G, F, $\boldsymbol{q}$; /* Planning Graph, New Manifold, New Point */
1 add-sample$(G, \boldsymbol{q})$;
2 **loop**
3 $\quad \boldsymbol{q} \leftarrow$ project-to-manifold$(\boldsymbol{q}, F)$;
4 $\quad$ **if** $\boldsymbol{q} \in \mathcal{C}_{\text{obs}}$ *or* $\boldsymbol{q} = \emptyset$ **then return**;
5 $\quad$ add-sample$(G, \boldsymbol{q})$;
6 $\quad P_{\text{rest}}, P_{\text{goal}} \leftarrow$ Acquire-Input-Data(G);
7 $\quad F \leftarrow$ train-SVM$(P_{\text{rest}}, P_{\text{goal}})$;

C. Facets Bisection and Checking

We need to check if the manifold triangulation is contained in $\mathcal{C}_{\text{obs}}$. The triangulation produces a set of $(n\text{-}1)$-simplices (facets). Previous work [13] used configuration space penetration depth to check each facet, which is computationally expensive and not always possible to robustly determine, especially in high dimensions. In this work, we propose a simplex bisection method that recursively checks all vertices of decomposed simplices until all vertices are enclosed in a hyper-ball with a small radius ε_b. This method is analogous to the local planner interpolation used to add new samples to, e.g., PRMs and RRTs [7], [51]. Here, we extend line segment checks to simplex checks.

We divide a simplex into smaller simplices with recursive subdivision [52]. Recursive subdivision picks two vertices of the simplex each time and cuts through the midpoint of the two vertices and all other vertices, which bisects a simplex. We pick the longest edge to cut each time, which provides better quality simplices and requires fewer checks [53].

While bisecting and checking simplices, we locally update bisected simplices' vertices in the same way as the elastic update during triangulation (Algorithm 4, Figure 3). If the elastic update succeeds, then we avoid another iteration of triangulation and checking facets. If elastic update fails, then we re-triangulate with a smaller λ_T (smaller $\mathbb{R}^n$ triangulation size) and check the facets again. If all simplices are checked and are in $\mathcal{C}_{\text{obs}}$, then we have an infeasibility proof.

To summarize, we have three threads running in parallel and exchanging data. The SDCL thread takes the planning graph from the PRM, learns a manifold, and samples the manifold. The proof construction thread takes the manifold and samples, generates a triangulation, and checks the triangulation. Also, both the SDCL thread and the proof construction thread provide samples to the planning graph which helps planning in narrow passages. The algorithm outputs a plan or an infeasibility proof given long enough time.

VI. Experiments

In this section, we show the experimental results of the algorithm in multiple serial manipulator scenes. We run experiments in 4-DOF scenes to compare the current algorithm with previous work. We perform experiments in 5-DOF scenes to show how the algorithm scales in higher dimensions. We

also show how the algorithm performs in feasible plan scenes. We run 30 trials for each experiment.

To leverage parallelism in several parts of our algorithm, we run our experiment on a multi-core system with NVIDIA TU102 GPU and a dual CPU AMD EPYC 7402 with 24 cores per CPU. We adapt PRM [7] in OMPL [9] to run in parallel with our infeasibility proof construction thread and SDCL thread. We solve the nonlinear optimization problems using sequential least-squares quadratic programming (SLSQP) [54], [55] in NLopt [56]. We adapt the Coxeter triangulation module in GUDHI [57] for triangulation. We train the RBF-kernel SVM using ThunderSVM [58], which supports GPU-accelerated SVM training. We use the concurrent hash table in libcuckoo [49], [50] when parallelizing the triangulation step. We check collisions using the Flexible Collision Library [59]. We use Miniball to calculate the smallest enclosing ball when dividing and checking simplices [60]. We model robot kinematics using Amino [61].

A. 4-DOF Experiments

The 4-DOF experiments setup is the same as in [13] for us to compare the results. We have two scenes, one with a shoulder-elbow robot and the other with a SCARA robot [62]. Figure 4a and Figure 4c show these scenes. Experimental results are in Table I. In both scenes, we use $\lambda_T = 0.1$ for $\mathbb{R}^n$ triangulation. The mean runtime is two orders of magnitude faster than the result in [13].

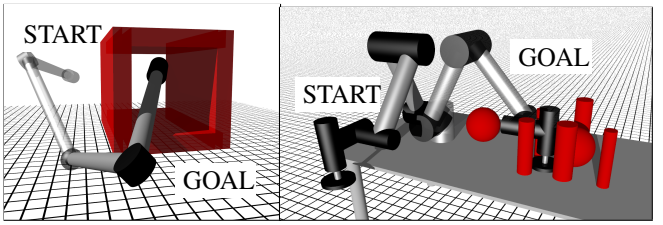
B. 5-DOF Experiments

We setup two 5-DOF experimental scenes. The first uses a manipulator structure similar to the PackBot [63] (see Figure 4d). The goal in the scene is to reach inside the shelf. In this scene, we use $\lambda_T = 0.1$ for $\mathbb{R}^n$ triangulation. The second scene uses a manipulator similar to the Universal robot [64] (see Figure 4b). Since the Universal robot is 6-DOF, we use a round end-effector and fix the last joint to make it 5-DOF. The goal is to reach the target position in a clustered tabletop environment. In this scene, we use $\lambda_T = 0.08$ for $\mathbb{R}^n$ triangulation. Both scenes' runtime results are in Table I.

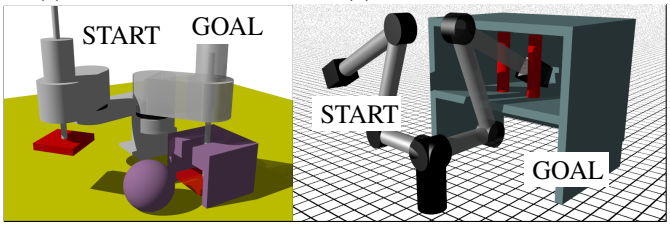
C. Feasible Plan Experiments

We also run experimental scenes where plans exist in narrow passages. The goal is not to thoroughly compare with other motion planners, since these results would be similar to the results of using SDCL with PRM without the infeasibility proof part, which is shown in previous work [15]. Instead, we wish to demonstrate that this new algorithm structure maintains the capability to efficiently solve these problems.

We modify the 5-DOF infeasible scenes to make narrow passages such that plans exist in the scene but are still hard to find. For the tabletop environment in Figure 4b, we move the cylinders to create more room for the end-effector to pass. For the shelf environment in Figure 4d, we move the two obstacles further away from each other and the arm. We also use the same modified Figure 4b scene with a square-end effector for setting up a 6-DOF narrow passage problem.



(a) 4 DoF robot scene. (b) Universal robot scene.



(c) SCARA robot scene. (d) PackBot scene.

Fig. 4: 4-DOF and 5-DOF Experiment Scenes.

	Total (s)	*Cox (s)*	*Check (s)*
4-DOF	2.60 ±0.75	0.95 ±0.76	1.00 ±0.31
SCARA	6.19 ±4.05	1.76 ±1.25	1.64 ±0.70
Packbot	344.64 ±139.60	88.25 ±53.14	251.85 ±92.04
Universal	186.47 ±53.44	22.33 ±8.25	162.42 ±52.36

TABLE I: Runtime results for 4-DOF and 5-DOF manipulators, mean (s) ±STD, averaged over 30 trials. "Cox" is for Coxeter triangulation, including manifold tracing and construction of triangulation. "Check" is for the bisection and checking.

We run 30 trials for each scene. Our algorithm successfully terminates with a plan for each scene and each trial. Table II shows the results. We compare our planner with RRT-Connect as a baseline to show these are actually difficult, narrow passage motion planning problems. We give a 300 seconds time limit to run RRT-Connect with the three scenes. Our results are similar to [15], significantly outperforming the baseline planner. This experiment also shows that attempting to construct an infeasibility proof does not significantly impact planning performance in scenes with feasible plans.

VII. Discussion, Analysis and Future Work

A. Parallel algorithm analysis

As computational hardware continues to scale primarily through parallel rather than serial performance, algorithmic designs leveraging parallelism are increasingly critical. Prior work has used multi-core CPUs to parallelize sampling [23], [65] or nearest neighbor search [66], GPUs for nearest neighbor search [67] and collision checking [68], and FPGAs for collision checking [69]. In our algorithm, we have parallelized all the main components. This includes manifold learning, utilizing off-the-shelf GPU-based SVM training [58]; manifold sampling; the triangulation step, facilitated by adapting Coxeter triangulation; and the facet bisection and checking step.

Feasible Plan Experiments (30 trials) mean/std (s)			
	5-DOF Tabletop	*5-DOF Shelf*	*6-DOF Tabletop*
Our Algorithm	1.74 ±0.94	2.00 ±0.87	1.63 ±0.68
RRT-Connect	187.51 ±104.64	240.34 ±79.58	236.32 ±92.30
RRTC Solved	20/30	15/30	14/30

TABLE II: Experimental results for feasible plan experiments.

Speedup with Increasing Parallel Threads (50 trials) mean±std (s)			
	12 vs 6	*24 vs 12*	*48 vs 24*
Algorithm 1	1.68 ±0.09	1.59 ±0.11	1.25 ±0.16
Algorithm 3	1.86 ±0.25	1.59 ±0.22	1.40 ±0.29

TABLE III: Parallel algorithm speedup test over 50 trials.

Our key development is a concurrent variation of Coxeter triangulation to leverage available hardware parallelism. The geometric structure of Coxeter triangulation enables concurrency. Compared to tangential Delaunay complexes, the calculation of Coxeter triangulation is primarily element-wise, leveraging the manifold's information effectively without relying on (and synchronizing with) neighboring points.

We analyze our parallel Coxeter triangulation using the conventional *work/span* approach—where *work* represents time to execute if on a single processor, and *span* represents time to execute if given unlimited processors [70]—to determine speedup Sp as the ratio of sequential T_s and parallel T_p execution $Sp = T_s/T_p$. The main iterations in Algorithm 1 and Algorithm 3 are all parallel, meaning span is the length of a single iteration and offering a theoretical linear speedup in number of processors, $T_s/T_p = p/s$.

However, the speedup is usually less than linear in practice due to the critical sections, including those in the concurrent hash tables [49], [50]. We empirically test speedup using 6, 12, 24, and 48 cores (see Table III). The algorithm is faster with more cores but the speedup is less than linear. As the number of threads increases, contention intensifies due to shared data structures, leading to a decrease in speedup.

B. Hyperparameters

An important parameter in the triangulation step is λ_T, which controls the size of the $\mathbb{R}^n$ triangulation. Smaller λ_T produces a finer triangulation of the manifold, which also takes longer to complete. λ_T is adjusted online after each dividing and checking step. If the bisection and checking step fails, it means there are failed elastic updates, and the manifold needs a finer triangulation, so we need to triangulate the manifold again with a smaller λ_T. We multiply λ_T with a constant between 0 and 1 after each failed dividing and checking step. In all experiments, we use the value 0.9. With an iteratively smaller λ_T, the algorithm finishes the infeasibility proof construction eventually. Choosing a proper value for λ_T is also important and affects the total runtime. If λ_T is too small, then it takes longer for the triangulation step to complete. If λ_T is too large, then the algorithm needs to run many more iterations of triangulation and checking to reduce λ_T before it returns, which also increases overall runtime.

Other hyperparameters are ε_b, the facets' smallest enclosing ball radius, and τ, in Algorithm 2. ε_b must be small enough to capture the "thinnest" obstacle region, which is related to the definition of ε-blocked, but not too small since it would increase the number of bisections. If ε_b is too large, the algorithm may produce false positive results. The value of τ determines how close the triangulation vertices are to the manifold. Using a τ too large may cause discrepancies between the triangulation and the manifold. We use 0.05 in all experiments.

C. Scalability

Our current algorithm and implementation scales to find infeasibility proofs for 5-DOF manipulators, though it also is capable of quickly finding plans in feasible scenes for higher-dimensional manipulators. In the 5-DOF experimental results, checking the triangulation's facets takes a large portion of the total runtime. We also run preliminary tests on 6-DOF scenes, with a similar setup in Figure 4b. The algorithm completes learning, sampling, and triangulation until all vertices of the triangulation are in $\mathcal{C}_{\text{obs}}$, which on average takes less than 40 seconds with $\lambda_T = 0.08$. However, our current approach to bisect and check facets dominates running time and did not complete within 120 minutes. Further scaling of infeasibility proof construction will require a faster collision checking backend [71] or alternative approaches to facets checking, which remains an area of future work. Another potential way to scale this method is by using sub-space decomposition [30], [72], [73], where infeasibility proofs can be constructed in lower dimensions and remain valid in higher dimensions.

VIII. Conclusion

In this work, we introduced an asymptotically complete motion planner that combines SDCL and infeasibility proof construction. We enhance the Coxeter triangulation with elastic updates during triangulation and checking, utilizing parallel computing for faster processing. Our experiments in 4-DOF and 5-DOF scenarios demonstrate the algorithm's efficiency. For future work, infeasibility proofs can be applied in a multi-query setting for addressing higher-level planning problems [1]–[4].

References

[1] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *T-RO*, vol. 14, no. 4, pp. 549–565, 1998.
[2] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *IJRR*, vol. 28, no. 1, pp. 104–126, 2009.
[3] J. Ota, "Rearrangement of multiple movable objects-integration of global and local planning methodology," in *ICRA*, vol. 2, 2004, pp. 1962–1967.
[4] G. Wilfong, "Motion planning in the presence of movable obstacles," *Annals of Mathematics and Artificial Intelligence*, vol. 3, no. 1, pp. 131–150, 1991.
[5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Deptartment, Iowa State University, Tech. Rep. TR-98-11, October 1998.
[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, vol. 30, no. 7, pp. 846–894, 2011.
[7] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *T-RO*, vol. 12, no. 4, pp. 566–580, 1996.
[8] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *ICRA*, vol. 2, 2000, pp. 995–1001.
[9] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *RAM*, vol. 19, no. 4, pp. 72–82, 2012.
[10] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *arXiv preprint arXiv:1109.3145*, 2011.
[11] S. Li and N. T. Dantam, "Exponential convergence of infeasibility proofs for kinematic motion planning," in *Algorithmic Foundations of Robotics XV*, S. M. LaValle, J. M. O'Kane, M. Otte, D. Sadigh, and P. Tokekar, Eds. Cham: Springer International Publishing, 2023, pp. 294–311.
[12] ——, "Learning proofs of motion planning infeasibility," in *RSS*, 2021.
[13] ——, "A sampling and learning framework to prove motion planning infeasibility," *IJRR*, 2023.
[14] S. Kachanovich, "Meshing submanifolds using coxeter triangulations," Ph.D. dissertation, COMUE Université Côte d'Azur (2015-2019), 2019.

[15] S. Li and N. T. Dantam, "Sample-driven connectivity learning for motion planning in narrow passages," in *ICRA*, 2023.
[16] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *ICRA*, vol. 1, 1996, pp. 113–120.
[17] I. A. Şucan and L. E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *T-RO*, vol. 28, no. 1, pp. 116–131, February 2012.
[18] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *IJRR*, vol. 21, no. 3, pp. 233–255, 2002.
[19] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *IJRR*, vol. 34, no. 7, pp. 883–921, 2015.
[20] A. M. Ladd and L. E. Kavraki, "Fast tree-based exploration of state space for robots with dynamics," in *Algorithmic Foundations of Robotics VI*. Springer, 2004, pp. 297–312.
[21] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *IJRR*, vol. 35, no. 5, pp. 528–564, 2016.
[22] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *IJRR*, vol. 35, no. 7, pp. 797–822, 2016.
[23] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, and L. E. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *T-RO*, vol. 21, no. 4, pp. 597–608, 2005.
[24] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic, "Caging and path non-existence: A deterministic sampling-based verification algorithm," in *Robotics Research*, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds. Cham: Springer International Publishing, 2020, pp. 589–604.
[25] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen, "Disconnection proofs for motion planning," in *ICRA*, vol. 2. IEEE, 2001, pp. 1765–1772.
[26] Z. McCarthy, T. Bretl, and S. Hutchinson, "Proving path non-existence using sampling and alpha shapes," in *ICRA*. IEEE, 2012, pp. 2563–2569.
[27] S. Li and N. T. Dantam, "Towards general infeasibility proofs in motion planning," in *IROS*, 2020, pp. 6704–6710.
[28] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.
[29] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *IJRR*, vol. 33, no. 1, pp. 18–47, 2014.
[30] A. Orthey and M. Toussaint, "Sparse multilevel roadmaps for high-dimensional robotic motion planning," in *ICRA*, 2021, pp. 7851–7857.
[31] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, "Quasi-randomized path planning," in *ICRA*, vol. 2. IEEE, 2001, pp. 1481–1487.
[32] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *IJRR*, vol. 37, no. 1, pp. 46–61, 2018.
[33] M. Tsao, K. Solovey, and M. Pavone, "Sample complexity of probabilistic roadmaps via ε-nets," in *ICRA*. IEEE, 2020, pp. 2196–2202.
[34] D. Dayan, K. Solovey, M. Pavone, and D. Halperin, "Near-optimal multi-robot motion planning with finite sampling," *T-RO*, 2023.
[35] A. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *RAM*, vol. 4, no. 2, pp. 1255–1262, 2019.
[36] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, "Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning," in *ICRA*, 2020, pp. 9563–9569.
[37] D. Driess, J.-S. Ha, R. Tedrake, and M. Toussaint, "Learning geometric reasoning and control for long-horizon tasks from visual input," in *ICRA*, 2021, pp. 14 298–14 305.
[38] D. Driess, J.-S. Ha, and M. Toussaint, "Learning to solve sequential physical reasoning problems from a scene image," *IJRR*, vol. 40, no. 12-14, pp. 1435–1466, 2021.
[39] S. A. Bouhsain, R. Alami, and T. Simeon, "Learning to predict action feasibility for task and motion planning in 3D environments," in *ICRA*, 2023.
[40] H.-Y. Yeh, S. Thomas, D. Eppstein, and N. M. Amato, "UOBPRM: A uniformly distributed obstacle-based prm," in *IROS*. IEEE, 2012, pp. 2655–2662.
[41] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *ICRA*, vol. 3. IEEE, 2003, pp. 4420–4426.
[42] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif, "Narrow passage sampling for probabilistic roadmap planning," *T-RO*, vol. 21, no. 6, pp. 1105–1115, 2005.
[43] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.
[44] H. S. Coxeter, "Discrete groups generated by reflections," *Annals of Mathematics*, pp. 588–621, 1934.
[45] J.-D. Boissonnat and A. Ghosh, "Manifold reconstruction using tangential delaunay complexes," *Discrete & Computational Geometry*, vol. 51, no. 1, pp. 221–267, 2014.
[46] J.-D. Boissonnat, F. Chazal, and M. Yvinec, *Geometric and Topological Inference*. Cambridge University Press, 2018, cambridge Texts in Applied Mathematics. [Online]. Available: https://hal.inria.fr/hal-01615863
[47] S. D. Conte and C. De Boor, *Elementary numerical analysis: an algorithmic approach*. SIAM, 2017.
[48] Anonymous, *The Nine Chapters on the Mathematical Art*. Commentary by Liu Hui, 263.
[49] B. Fan, D. G. Andersen, and M. Kaminsky, "Memc3: Compact and concurrent memcache with dumber caching and smarter hashing," in *USENIX Symposium on Networked Systems Design and Implementation*, 2013, pp. 371–384.
[50] X. Li, D. G. Andersen, M. Kaminsky, and M. J. Freedman, "Algorithmic improvements for fast concurrent cuckoo hashing," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 1–14.
[51] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *IJRR*, vol. 20, no. 5, pp. 378–400, 2001.
[52] D. W. Moore, *Simplicial mesh generation with applications*. Cornell University, 1992.
[53] C. S. Petersen, B. R. Piper, and A. J. Worsey, "Adaptive contouring of a trivariate interpolant," *Geometric Modeling: Algorithms and New Trends, GE Farin, ed., SIAM*, pp. 385–395, 1987.
[54] D. Kraft, "A software package for sequential quadratic programming," Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Tech. Rep. DFVLR-FB 88-28, July 1988.
[55] ——, "Algorithm 733: TOMP–fortran modules for optimal control calculations," *Transactions on Mathematical Software*, vol. 20, no. 3, pp. 262–281, 1994.
[56] S. G. Johnson, "The NLopt nonlinear-optimization package," 2023. [Online]. Available: http://github.com/stevengj/nlopt
[57] The GUDHI Project, *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. [Online]. Available: http://gudhi.gforge.inria.fr/doc/latest/
[58] Z. Wen, J. Shi, Q. Li, B. He, and J. Chen, "ThunderSVM: A fast SVM library on GPUs and CPUs," *Journal of Machine Learning Research*, vol. 19, pp. 797–801, 2018.
[59] J. Pan, S. Chitta, and D. Manocha, "FCL: A general purpose library for collision and proximity queries," in *ICRA*, 2012, pp. 3859–3866.
[60] B. Gärtner, "Fast and robust smallest enclosing balls," in *7th Annual European Symposium on Algorithms (ESA)*. Springer, 1999, pp. 325–338.
[61] N. T. Dantam, "Robust and efficient forward, differential, and inverse kinematics using dual quaternions," *IJRR*, vol. 40, no. 10-11, pp. 1087–1105, 2021.
[62] H. Makino, "Assembly robot," July 1982, US Patent 4,341,502.
[63] "Packbot 510." [Online]. Available: https://www.flir.com/products/packbot/?vertical=ugs&segment=uis
[64] UniversalRobots, "UR5 collaborative robot arm: Flexible and lightweight cobot." [Online]. Available: https://www.universal-robots.com/products/ur5-robot/
[65] J. Ichnowski and R. Alterovitz, "Parallel sampling-based motion planning with superlinear speedup," in *IROS*, 2012.
[66] ——, "Concurrent nearest-neighbor searching for parallel sampling-based motion planning in SO(3), SE(3), and Euclidean spaces," in *Algorithmic Foundations of Robotics XIII*, M. Morales, L. Tapia, G. Sánchez-Ante, and S. Hutchinson, Eds. Cham: Springer International Publishing, 2020, pp. 69–85.
[67] J. Pan, C. Lauterbach, and D. Manocha, "Efficient nearest-neighbor computation for GPU-based motion planning," in *IROS*. IEEE, 2010, pp. 2243–2248.
[68] J. Pan and D. Manocha, "GPU-based parallel collision detection for fast motion planning," *IJRR*, vol. 31, no. 2, pp. 187–200, 2012.
[69] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot motion planning on a chip," in *RSS*, 2016.
[70] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.
[71] L. Montaut, Q. L. Lidec, J. Sivic, and J. Carpentier, "Collision detection accelerated: An optimization perspective," in *RSS*, 2022.
[72] O. Salzman, M. Hemmer, B. Raveh, and D. Halperin, "Motion planning via manifold samples," *Algorithmica*, vol. 67, no. 4, pp. 547–565, 2013.
[73] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris, "dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning," *Autonomous Robots*, vol. 44, no. 3-4, pp. 443–467, 2020.