The International Journal of Robotics Research
2023, Special Issue: RSS2021
© The Authors 2023
Article reuse guidelines:
sagepub.com/journal-permissions
DOI: 10.1177/02783649231154674
journals.sagepub.com/home/ijr
SAGE

A Sampling and Learning Framework to Prove Motion Planning Infeasibility

Sihui Li and Neil T. Dantam

Abstract
We present a learning-based approach to prove infeasibility of kinematic motion planning problems. Sampling-based motion planners are effective in high-dimensional spaces but are only probabilistically complete. Consequently, these planners cannot provide a definite answer if no plan exists, which is important for high-level scenarios, such as task-motion planning. We apply data generated during multi-directional sampling-based planning (such as PRM) to a machine learning approach to construct an infeasibility proof. An infeasibility proof is a closed manifold in the obstacle region of the configuration space that separates the start and goal into disconnected components of the free configuration space. We train the manifold using common machine learning techniques and then triangulate the manifold into a polytope to prove containment in the obstacle region. Under assumptions about the hyper-parameters and robustness of configuration space optimization, the output is either an infeasibility proof or a motion plan in the limit. We demonstrate proof construction for up to 4-DOF configuration spaces. A large part of the algorithm is parallelizable, which offers potential to address higher dimensional configuration spaces.

Keywords
motion planning, infeasibility proof

1 Introduction

Many high-level planning problems incorporate motion planning as a subproblem (Ben-Shahar and Rivlin (1998); Cambon et al. (2009); Ota (2004); Wilfong (1991)). A *complete* motion planner would support such high-level planning by not only finding plans when possible but also deciding whether or not a plan is feasible. Most previous motion planning work focused on finding a plan. In this work, we seek to solve the other—less explored—side of the completeness problem: finding infeasibility proofs in motion planning.

Complete motion planning is challenging, and many approaches aim for weaker notions of completeness. *Resolution complete* planners offer completeness up to a certain granularity or resolution level of the configuration space. Typical resolution complete planners use a cell decomposition, such as a grid-based approach. Resolution-complete methods (Zhang et al. (2007, 2008)) are effective in low-dimensional spaces, but decomposing and covering a high-dimensional configuration space is usually too expensive. Conversely, sampling-based motion planners (LaValle (1998); Karaman and Frazzoli (2011); Kavraki et al. (1996); Kuffner and LaValle (2000); Şucan et al. (2012); Shkolnik and Tedrake (2011)) are widely used for high-dimensional configuration spaces. These methods are *probabilistically complete*, meaning if a feasible plan exists, they find the plan given enough time. However, if a plan does not exist, a probabilistically complete planner will run forever or until a specified timeout (Karaman and Frazzoli 2011). A timeout is not a guarantee of plan non-existence, but the sampled points offer insight to generate such guarantees.

Previously, we proposed a general formulation to find motion planning infeasibility proofs as polytopes in the configuration space obstacle region that separate the start and goal (Li and Dantam 2020). The key idea is to avoid fully covering the configuration space and instead find only a boundary—the polytope—that disconnects the start and goal. However, combinatorial steps to directly construct such polytopes pose challenges to practically scale to high-dimensional manipulators. We now address this challenge by incorporating learning.

We propose a novel framework to prove motion planning infeasibility that integrates supervised learning and multi-directional, sampling-based motion planning. Compared to prior methods, this approach supports the configuration spaces of robot manipulators and demonstrates better empirical scalability to higher degree-of-freedom (DOF) motion planning problems. Figure 2 outlines the framework. The key insight is to use planning graph components as training data by assigning different classes depending whether a component includes the start point or the goal point. Then, we train a classifier on these planning graph components (see subsection 4.1). Geometrically, the decision boundary of the classifier is a manifold that separates the classes—i.e., planning graph components. Such a manifold proves infeasibility when it is (1) closed, (2) entirely in the obstacle region of the configuration space, and (3) separating the start and goal configurations. That is, such a manifold is a proof that part of the obstacle region completely separates the start

Department of Computer Science, Colorado School of Mines, USA

Corresponding author:
Sihui Li, 1500 Illinois St, Colorado School of Mines, Golden, Colorado, USA
Email: li@mines.edu

and goal, which means that no motion plan exists. To verify that the manifold is entirely in the obstacle region, we sample points on the manifold (see subsection 4.2) and construct a triangulation polytope to approximate the manifold using the sampled manifold points (see subsection 4.3). Finally, we prove that the approximating polytope is entirely in the obstacle region by checking each facet of the polytope (see subsection 4.4).

Figure 3 illustrates the two classes assigned to the planning graph vertices, the learned manifold from the two classes, points sampled on the manifold, and the polytope constructed using the manifold points in a 2D maze-like configuration space. The existence of such a manifold in the obstacle region means the start and the goal are in separate components of the free configuration space, thus proving the infeasibility for the motion planning problem.

We make certain assumptions in this approach. If all assumptions are met, the result is either a motion plan, when one exists, or an infeasibility proof in the form of the separating manifold. First, the current algorithm applies to kinematic motion planning problems where infeasibility is caused only by the geometric obstacle region in the configuration space. Infeasibility motions caused by differential constraints, such as the robot's dynamics, are not within the present scope and remain for future work. Second, the algorithm depends on two hyper-parameters. The first hyper-parameter is for training the manifold (see subsection 4.1) and the second hyper-parameter is used in the triangulation step (subsection 4.3). We must choose proper hyper-parameters for the approach to work. Assuming a kinematic motion planning problem and proper hyper-parameters, our algorithm produces a motion plan or infeasibility proof in the limit.

An initial version of this work appeared in Li and Dantam (2021). We now extend that work to improve generality and efficiency as follows:

- generalizing to configuration spaces with multiple, disconnected free space components by using a Probabilistic Roadmap (PRM) instead of a bidirectional rapidly-exploring random tree (RRT-connect);
- parallelization of parts of the algorithm to reduce overall running time;
- more efficient checking of polytope facets using a minimum enclosing ball;
- more efficient handling of triangulation and facet checking failures;
- additional experiments on up to 4-DOF configuration spaces to show scalability of our algorithm.

We demonstrate this approach on up-to four-dimensional configuration spaces of serial robot manipulators (see Figure 1 for one example). To the best of our knowledge, this is the first approach to construct motion planning infeasibility proofs for such serial manipulators in higher than three dimensions.

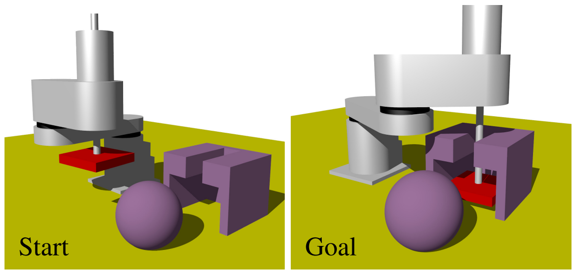

Figure 1. An infeasible motion planning problem for a four degree-of-freedom SCARA (Makino 1982) arm.

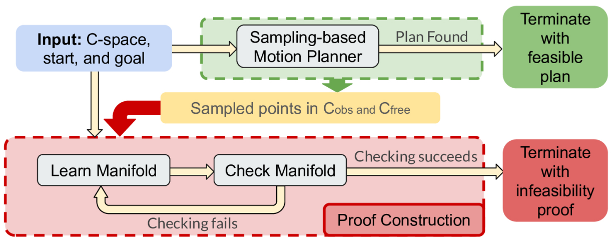
Input: C-space, start, and goal
Sampling-based Motion Planner
Plan Found
Terminate with feasible plan
Sampled points in Cobs and Cfree
Learn Manifold
Check Manifold
Checking succeeds
Terminate with infeasibility proof
Checking fails
Proof Construction

Figure 2. Overview of sampling and learning framework. The part in the red block is showing the infeasibility proof construction steps.

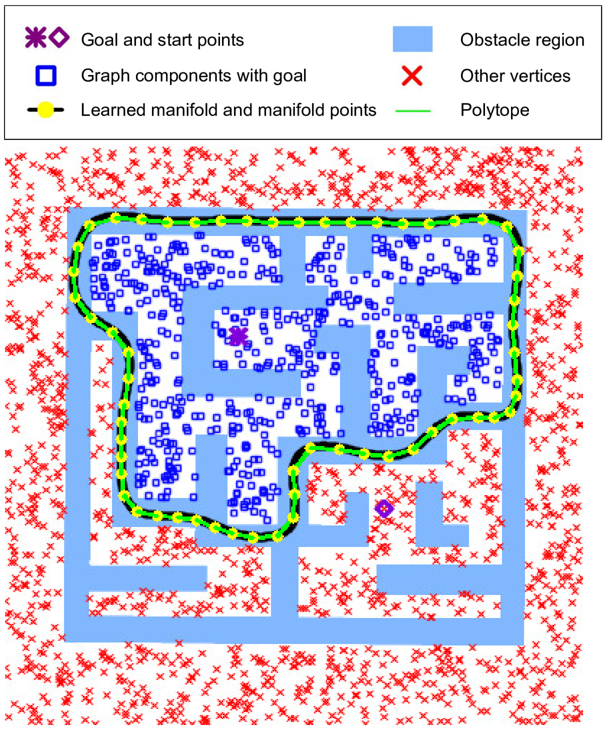

Figure 3. Visualization of learned manifold for a 2D motion planning problem. The learned proof manifold and the polytope constructed from it separates the start and the goal.

2 Related work

2.1 Completeness

Varying notions of completeness exist for motion planning algorithms (LaValle 2006). Generally, a *complete* planner returns a plan or guarantee of plan nonexistence in finite time. A *resolution complete* motion planner is complete up to a set resolution level of the configuration space. A *probabilistically complete* motion planner returns a plan, if one exists, in the limit. It is important to note that the failure of a probabilistically complete planner to find a plan—i.e., a timeout—is not a definite guarantee of infeasibility; such timeouts may occur because the problem is infeasible or simply because we need more time to find the plan. Compared to these forms of completeness, our approach is complete in the limit. That is, given enough time, our approach returns either a plan or an infeasibility proof.

2.2 Sampling-based planning

Sampling-based motion planning is widely used for high-dimensional motion planning (Amato and Wu 1996; Şucan and Kavraki 2012; Hsu et al. 2002; Janson et al. 2015; Karaman and Frazzoli 2011; Kavraki et al. 1996; Kuffner and LaValle 2000; Ladd and Kavraki 2004; LaValle 1998; Li et al. 2016; Otte and Frazzoli 2016; Plaku et al. 2005; Shkolnik and Tedrake 2011). Many sampling-based planners are probabilistically complete, but probabilistic completeness does not provide guarantees on path non-existence. The points sampled by those planners do, however, provide useful information about the configuration space. RRT-connect generate two trees, one rooted from the start configuration and the other rooted from the goal configuration, which can be used as training data to learn the free regions containing the start and the goal in the configuration spaces (Li and Dantam 2021). PRM planners generate a graph covering the entire configuration space, which is multi-directional. In this sense, the graph from the PRM planner offers information about the entire configuration space. In section subsection 4.1, we describe our use of the planning graph in more detail.

2.3 Surface Triangulation

Surface triangulation is an important step in our algorithm. We triangulate the manifold to prove containment in the obstacle region. Surface triangulation is also an important topic in the area of finite element analysis and computer graphics. Previous work has successfully triangulated 3D surfaces (Akkouche and Galin 2001; Hartmann 1998; Hilton et al. 1997; Karkanis and Stewart 2001). However, most of these algorithms are not scalable to higher dimensions, which is necessary for our use case when extending to higher DOF configuration spaces. In recent years, further development in high-dimensional computational geometry tools have enabled triangulation of high DOF manifolds (Boissonnat and Ghosh 2014; Fogel et al. 2012; Halperin et al. 2017). Particularly, Boissonnat and Ghosh (2014) provide an algorithm to reconstruct a closed, differentiable manifold using a set of sampled points on the manifold. This method scales to higher dimensions, and we use this algorithm for triangulation in subsection 4.3.

2.4 Plan Infeasibility

Some previous work addressed motion planning infeasibility proofs for single objects. Varava et al. (2020) prove path non-existence for single, rigid objects in a 2D or 3D workspace. They approximate the obstacle region with a decomposition into lower dimensional subsets and connected components of those subsets. Using these components, they construct a connectivity graph to query whether two configurations are connected. Basch et al. (2001) consider the simplified problem of a rigid body passing through a narrow gate. They discretize the object's orientation and test whether the object can pass through the gate for each discrete orientation region. These works focus on single objects in the Cartesian space rather than the configuration space of robot manipulators.

Other works offer complete motion planning based on space decomposition. McCarthy et al. (2012) decompose the obstacle region into alpha-shapes and then query the connectivity of two points; scalability to higher dimensions depends on the computation of high-dimensional alpha-shapes, which is still an open research question. Zhang et al. (2007, 2008) combine cell decomposition with a probabilistic roadmap (PRM), which offers resolution-completeness due to the underlying cell decomposition. However, decomposing the entire configuration space poses scalability challenges in higher dimensions.

Deterministic sampling-based motion planning provides certain guarantees on plan non-existence (Branicky et al. 2001; Janson et al. 2018). If such a planner using low-dispersion sampling strategies does not find a plan, then either no solution exists or a solution exists only through some narrow passage. However, low-dispersion sampling must largely cover the configuration space, and the result is similar to resolution-completeness where the infeasibility guarantee is not exact.

Visibility (Siméon et al. 2000) and sparsity (Dobson and Bekris 2014) based planners also provide some degree of infeasibility information. These methods add sampling points to a roadmap if the points are useful for coverage, connectivity, or path quality. Planning terminates when no further points can be added for a certain number (M) of consecutive samples, and the percentage of the free space not covered by the roadmap is estimated as $1/M$. Thus, these methods usually achieve high coverage of the free space. If no plan is found when the algorithm terminates, the problem may be considered to be infeasible (Orthey and Toussaint 2021). However, these methods do not definitely prove plan nonexistence since they are based on covering a portion of the free space. In contrast, our approach seeks to find definite, exact infeasibility proofs through geometric methods.

Learning has previously been applied to feasibility of planning. Wells et al. (2019) learn classifiers for motion planning feasibility. Driess et al. (2020, 2021a,b) predict feasibility information from images. Kuo et al. (2018) train models with previous planning information and use learned models in the steering function of RRT* (Karaman and Frazzoli 2011) to guide new samples towards more feasible directions. However, these prior approaches use learning only as a heuristic estimate of feasibility. In contrast, our approach tightly couples sampling-based planning and a geometric interpretation of machine learning to produce definitive infeasibility proofs.

To summarize, previous works on infeasibility proofs either limit analysis to single objects, require decomposition of the entire configuration space, or do not offer definitive plan non-existence guarantees. The algorithm in this paper applies to robot manipulators and only requires decomposition of a manifold in the configuration space. The result, under stated assumptions, is either a feasible plan or an infeasiblity proof.

Our previous approach for infeasibility proof construction in Li and Dantam (2020) proposed an algorithm to construct a polytope in the configuration space obstacle region. First, we generated a set of facets in the obstacle region. Then, we identify the facets that form a closed polytope separating the start and goal by solving a set of linear constraints. Facet generation is computationally expensive, limiting scalability to higher dimensions. Compared to Li and Dantam (2020), Li and Dantam (2021) and this paper use learning to generate an initial manifold. The learned manifold is a strong heuristic for constructing the infeasibility polytope, offering better scalability to higher dimensions.

2.5 Robot Planning and Motion Infeasibility

Motion planning feasibility is an important issue in many robot planning approaches. Navigation among movable obstacles (NAMO) must determine the feasibility of motions—typically in a planar navigation space—to select which obstacles to move (Stilman and Kuffner 2005, 2008; Wilfong 1991). Similarly, rearrangement planning must address feasibility to plan motions from one arrangement of objects to another (Ben-Shahar and Rivlin 1998; Han et al. 2018; Huang et al. 2019; King et al. 2015; Ota 2009). Hauser (2013, 2014) addresses a different aspect of infeasibility by identifying minimal object sets or displacements that ensure motion plan existence. Task and motion planning must also address motion feasibility, such as through feasibility checks (Cambon et al. 2009; Dantam et al. 2018; Dantam 2020b; Erdem et al. 2016; Garrett et al. 2021; Kaelbling and Lozano-Pérez 2013; Lagriffoul and Andres 2016; Srivastava et al. 2014; Thomason and Knepper 2019; Toussaint 2015; Vega-Brown and Roy 2020). Varying approaches to handling feasibility significantly impact the guarantees and performance of such algorithms. Some approaches interpret failure to find a motion plan—e.g., timeout of a sampling-based planner—to mean the motion planning problem is infeasible (Dornhege et al. 2012; Rodrıguez et al. 2019; Yalciner et al. 2017). Such assumed infeasibility may offer heuristic benefits for certain scenarios, but the probabilistic completeness of sampling-based planners means this assumption may impact completeness of the overall approach. Overall, these numerous planning approaches relating to the feasibility of motion planning indicate that proving infeasibility is a fundamental issue in robot planning. New techniques addressing motion planning feasibility may thus offer new insight into these areas.

We conduct the following motivating experiments to illustrate issues that arise from interpreting a planner timeout as plan infeasibility. We apply RRT-Connect (Kuffner and LaValle 2000) to two feasible plan scenes. The environment we use is similar to Figure 1, with the ball shaped obstacle further from the target box to create room for valid plans. Both scenes have narrow passages and are difficult to solve, but one is easier than the other. We run 50 trials for each scene and count the number of timeouts (interpreted as infeasibility) given time limits from 100 seconds to 1200 seconds.

Figure 4 illustrates the timeout results. Fewer timeouts occur as the time limit increases, as expected. However, even in the easier scene, interpreting timeouts as infeasibility produces incorrect results until a time limit of 1000 seconds. In the harder scene, 60% of the trials still return incorrect

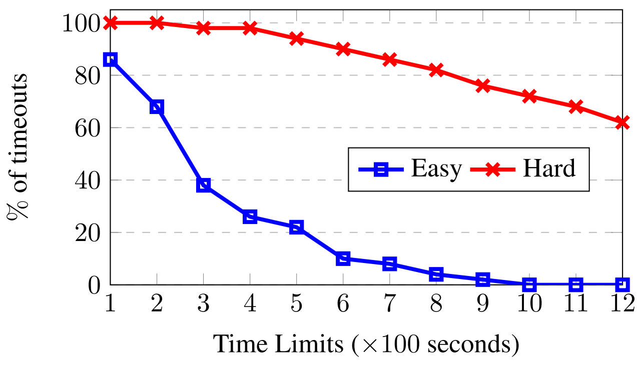

Figure 4. The number of false infeasibility results (timeouts) for different time limits on two feasible plan scenes, one easier and one harder, similar to Figure 1. We run 50 trials for each scene.

results with a time limit of 1200 seconds. Consequently, the accuracy of using a timeout to approximate plan infeasibility depends on the problem and does not provide guarantees. Choosing a proper time limit may also be challenging, even for relatively simple scenes.

2.6 Feasibility Checking for Dynamic Systems

Prior work has also addressed the feasibility problem for dynamic systems, often referred to as reachability analysis (Lygeros et al. 1999; Bajcsy et al. 2019; Bansal and Tomlin 2021). These works consider whether a state of a dynamic system can be reached. This question is closely related to safety verification, where the goal is to confirm that a trajectory of a system do not enter unsafe regions. Control barrier functions provide effective safety verification by separating the safe and unsafe regions such that no initial condition starting from a safe state can reach unsafe states (Prajna and Jadbabaie 2004). Our method is similar in spirit to control barrier functions in that we also provide a definitive proof by separating different parts of the free configuration space; however, our approach to construct infeasibility proofs differs from typical approaches to create control barrier functions. Previously, sampling-based motion planning algorithms were also used for safety verification—or rather falsification (Plaku et al. 2009). Our current algorithm is only applicable to kinematic problems, but a possible direction of future work would be to further address dynamic constraints.

3 Problem Definition

We find infeasibility proofs for kinematic motion planning problems. A motion planning problem consists of configuration space $\mathcal{C}$ of dimension n, start configuration $\boldsymbol{q}_{\text{start}}$, and goal configuration $\boldsymbol{q}_{\text{goal}}$ (LaValle 2006). Configuration space $\mathcal{C}$ is the union of the closed set obstacle region $\mathcal{C}_{\text{obs}}$ and the open set free space $\mathcal{C}_{\text{free}}$. A feasible plan is a path σ such that $\sigma[0, 1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \boldsymbol{q}_{\text{start}}$, $\sigma[1] = \boldsymbol{q}_{\text{goal}}$. The output of our approach is a feasible path if one exists, or an infeasible proof if no path exists.

We define an infeasibility proof as a closed manifold which is contained entirely in $\mathcal{C}_{\text{obs}}$ and that separates the start and the goal.

Definition 1. Infeasibility Proof. *A manifold $\mathcal{M}$ in $\mathcal{C}$ defined by a continuous function $f(\boldsymbol{q}) = 0$ is an* infeasibility proof *if and only if:*

(I) *$\mathcal{M}$ is a closed manifold;*

(II) *$\mathcal{M}$ is contained entirely in $\mathcal{C}_{\text{obs}}$, $\{q \mid f(q) = 0\} \subseteq \mathcal{C}_{obs}$;*

(III) *$\mathcal{M}$ separates the start and the goal configurations, $f(\boldsymbol{q}_{\text{start}})f(\boldsymbol{q}_{\text{goal}}) < 0$.*

We view the configuration space boundaries (e.g., joint limits) as a special case of $\mathcal{C}_{\text{obs}}$ so that we can handle a manifold that is partially outside of bounds in the same way as a manifold that is entirely in $\mathcal{C}_{\text{obs}}$. Specifically, we regard boundaries as a virtual obstacle region with positive ϵ thickness, and we define a region of virtual $\mathcal{C}_{\text{free}}$ to enclose the configuration space boundaries' virtual obstacle region (see Figure 5). The virtual obstacle region at the boundaries and the virtual $\mathcal{C}_{\text{free}}$ regions surrounding it are also sampled to generate training data to learn the manifold. With this special treatment to boundaries, now we can introduce the following proposition.

Proposition 1. *No plan σ exists if and only if there exists an infeasibility proof according to Definition 1.*

Proof. First, we prove by contradiction that if no plan σ exists, there must exists an infeasibility proof $\mathcal{M}$. Since we only consider kinematic motion planning problems, $\mathcal{C}_{\text{obs}}$ is the only reason that would result plan infeasibility. We perceive the configuration space boundaries as a special kind of obstacle region as described in the previous paragraph. Assuming no such $\mathcal{M}$ exists, then it means the $\mathcal{C}_{\text{free}}$ components of $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are not entirely separated by $\mathcal{C}_{\text{obs}}$, thus there would exist a plan σ.

Second, we prove that if there exists an infeasibility proof $\mathcal{M}$, no plan σ exists. According to Definition 1 (III), $f(\boldsymbol{q}_{\text{start}}) < 0$ (> 0) and $f(\boldsymbol{q}_{\text{goal}}) > 0$ (< 0), meaning $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are on different sides of $\mathcal{M}$. Also, because of Definition 1 (I) and Definition 1 (II), $\mathcal{M}$ separates $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ into disconnected components of $\mathcal{C}_{\text{free}}$, and there is no collision free path that connects $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$. Thus, no plan σ exists. □

We note that the proof manifold in Definition 1 may be either a smooth or piecewise manifold in the configuration space. In our algorithm, we first learn a smooth manifold. Then to check that the manifold is entirely in obstacle region, we use a polytope to approximate the learned manifold, which can be viewed as a piecewise manifold. The resulting infeasibility proof is actually the approximating polytope, since we do not directly prove that learned manifold is entirely in the obstacle region.

Requirements and Assumptions

To ensure successful construction of an infeasibility proof, we must add additional requirements to the obstacle region of the configuration space. Consider a 3-DOF configuration space where part of the obstacle region is a surface with zero thickness, e.g., an infinitesimal sheet of paper. The probability of finding samples to learn a manifold inside

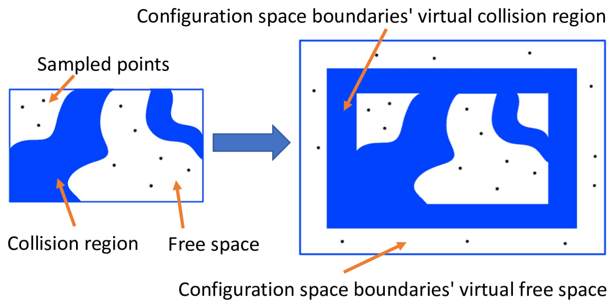
Configuration space boundaries' virtual collision region
Sampled points
Collision region
Free space
Configuration space boundaries' virtual free space

Figure 5. Changes made to the configuration space boundaries to unify joint limits and $\mathcal{C}_{\text{obs}}$. With these modifications, we can treat the joint limits as a special obstacle region.

this infinitesimal obstacle region is zero since that part of the obstacle region has zero volume. This counterexample leads to a requirement for $\mathcal{C}_{\text{obs}}$: everywhere in $\mathcal{C}_{\text{obs}}$ must have positive volume—i.e., the obstacle region must have some "thickness." Theoretically, the proof manifold could exist in infinitesimal, zero-volume regions, but for our algorithm to properly construct an infeasibility proof, we require $\mathcal{C}_{\text{obs}}$ to always have a non-zero volume. We claim that this is a reasonable requirement on $\mathcal{C}_{\text{obs}}$ since robot manipulators and real-world obstacles have non-zero volume.

Our approach treats configurations as real-valued vectors for the purpose of learning the manifold. Generally, many motion planning algorithms require only that the configuration space be a *metric space*, for example $\mathcal{SE}(3)$. Instead, we effectively require a Euclidean configuration space, e.g., a vector in $\mathbb{R}^n$ of a manipulator's n joint angles. We claim this Euclidean space requirement is reasonable for physical manipulators, since when considering inner hardware arrangements and outside physical environment, joint angles θ and $\theta + 2k\pi$ are rarely the same.

We assume for this work, kinematic motion planing in which infeasibility is caused only by the obstacle region, and we do not consider differential constraints. That is, we do not consider steering functions (Lafferriere and Sussmann 1991), dynamics (Donald et al. 1993), or implicit constraints (Berenson et al. 2009; Kingston et al. 2019). Though this assumption is valid for many manipulation scenarios, future work is needed to address more general cases.

Another important assumption of our algorithm is the ability to sample points on the manifold and the ability to obtain configuration space penetration depth. We provide in subsection 4.2 and subsection 4.4 empirically robust, optimization-based approaches to sample the manifold and calculate configuration space penetration depth for Cartesian obstacles, which is a typical case for robot manipulators. However, these results are not theoretical proofs that points on the manifold and the penetration depth are always available. Under the assumption that we can sample on the manifold and compute configuration space penetration depth, our algorithm will terminate. We demonstrate robust ability to find plans or infeasibility proofs for robot manipulators in the experiments (see section 5).

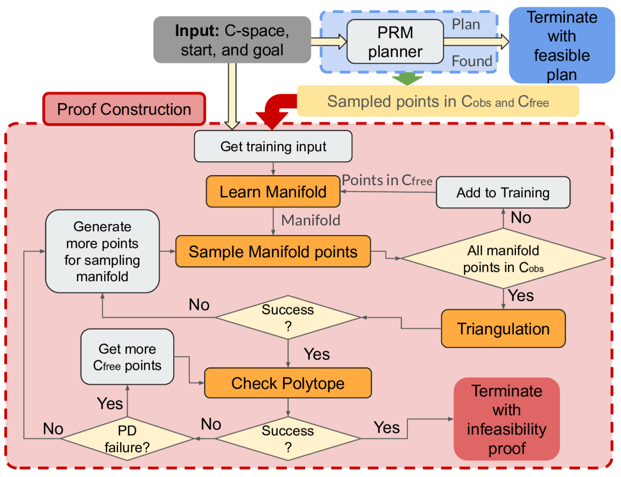

Figure 6. An algorithm overview. The part in the red block is showing the infeasibility proof construction steps.

4 Framework and Implementation

Our overall framework has three major components (see Figure 2). At the top level, a sampling-based motion planner and the infeasibility proof construction algorithm run in parallel, terminating with either a plan or an infeasibility proof. On the sampling-based motion planner side, we save all sampled configurations in $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$. On the infeasibility proof side, there are two major steps. First, we use configurations sampled in $\mathcal{C}_{\text{free}}$ to learn the manifold. Second, we check if the manifold is contained entirely in $\mathcal{C}_{\text{obs}}$. If we successfully check that the manifold is in $\mathcal{C}_{\text{obs}}$, then we have proven plan infeasibility. If this check fails, then we go back to re-learn the manifold. This overall framework can work with different approaches for sampling, learning the manifold, and checking the manifold. In the rest of this section, we describe our current implementation.

Our implementation of infeasibility proof construction has four important steps, indicated in Figure 6 with orange blocks. If all four steps complete successfully, then we have an infeasibility proof. Otherwise, the algorithm loops back to previous steps based on where the failure happens. Given $\boldsymbol{q}_{\text{start}}$, $\boldsymbol{q}_{\text{goal}}$, and saved $\mathcal{C}_{\text{free}}$ points, the first step is to learn a manifold (see subsection 4.1). Next, we sample points on the manifold by projecting the obstacle region points onto the manifold (see subsection 4.2). Then, we triangulate the manifold using the sampled manifold points with tangential Delaunay complex (see subsection 4.3). At last, we check that the polytope is entirely in $\mathcal{C}_{\text{obs}}$ by testing each facet (see subsection 4.4). Corresponding to the framework in Figure 2, the first step learns the manifold, and the remaining three steps check the manifold. In the following subsections, we discuss each of these four steps in detail.

4.1 Learning the manifold

In this step, we learn the manifold using sampled free space points. A successful learning result is the foundation of all following steps, since the rest of the three steps all depends closely on the learned manifold.

First, we specify our input training data. The PRM planner grows a planning graph and tries to connect sampled points in $\mathcal{C}_{\text{free}}$ to existing graph components. Disconnected regions in $\mathcal{C}_{\text{free}}$ result in disconnected components of the planing graph. We gather the $\mathcal{C}_{\text{free}}$ points from the PRM planner's graph. We take the graph component that contains the goal point as one of its vertices and mark all vertices (points) in this graph component as one class. This goal component is similar to marking the goal tree in RRT-connect. Lastly, we mark all vertices in other graph components not containing the goal as the other class. At this point, we have marked all the input data for training. We emphasize that the training input data consists of the two classes of $\mathcal{C}_{\text{free}}$ points, not $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$ points. The goal of the learning step is to construct a manifold that exists in $\mathcal{C}_{\text{obs}}$, not a manifold that separates $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$. This is different from the biased sampling algorithm proposed by Bialkowski et al. (2013) that iteratively refines obstacle representations.

Next, we train a classifier with the input data. Finding the right training method is important. There are several requirements for the training method. First, the training classifier should have enough flexibility to fit any boundary. We need to learn a classifier that completely separates the two classes we defined so that the classifier's decision boundary can fit into the obstacle region entirely. The obstacle region of the configuration space may have any curvature, and the classifier's decision boundary must fit into that curvature. Secondly, it is more convenient for the following steps if the classier directly provides a closed-form, differentiable function. This closed-from function is useful to sample points on the manifold (in subsection 4.2). Thirdly, the training method ideally has few hyper-parameters to tune, since we want the algorithm to be easily applicable to general configuration spaces.

The particular classifier we use is a support vector machine (SVM) with Radial Basis Function (RBF) kernel, which offers certain advantages in this application. RBF kernel SVM directly trains a classification function, in the following from,

$$\sum a_i K(\mathbf{x}_i, \mathbf{x}) + b = 0, K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma(\mathbf{x}_i - \mathbf{x}_j)^2}, \quad (1)$$

where each $\mathbf{x}_i$ is a support vectors, b is a known number, each a_i is a known coefficients, and K is the RBF kernel function. This function is differentiable. There is one hyper-parameter in the function to tune—the over-fitting parameter γ. By adjusting γ, the classifier may fit any curve. This γ is also one of the hyper-parameters in our algorithm, and we further discuss this hyper-parameter in the next paragraph. Compared to learning methods that may require substantial tuning, such as neutral networks, having only one hyper-parameter is helpful in this sub-step. Additionally, another advantage of the SVM is that it creates a margin to maximize separation between classes. This maximum margin increases penetration depth of the manifold points in the obstacle region, which helps with checking facets in subsection 4.4. While the RBF SVM meets our needs to learn a separating manifold, other classifiers that satisfy the above requirements may also fit within this overall framework.

While typical machine learning applications attempt to avoid over-fitting that fully separates the training data, our application actually requires such an over-fit manifold that separates the two classes without exception. When training the manifold, we increase the over-fitting parameter γ in the RBF kernel by a small amount $\Delta\gamma$ each time until the training data's accuracy is 1, that is, the manifold completely separates

Algorithm 1: Train-Manifold

```
  Input: $P_{\text{goal}}$ // Goal component vertices
  Input: $P_{\text{rest}}$ // Non-goal component
  Output: $f$ // Learned manifold
1 accuracy $\leftarrow 0$;
2 $\gamma \leftarrow 1$;
3 repeat // Ensure complete separation
4     $f \leftarrow$ train-SVM($P_{\text{rest}}, P_{\text{goal}}, \gamma$);
5     accuracy $\leftarrow$ test-SVM($f, P_{\text{rest}}, P_{\text{goal}}$);
6     $\gamma \leftarrow \gamma + \Delta\gamma$ ; // increases separation
7 until accuracy $= 1$;
```

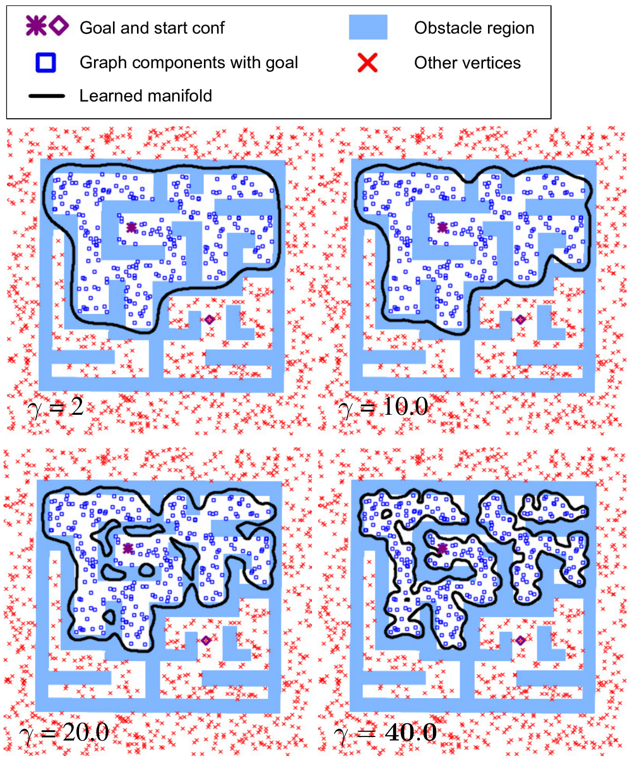

Figure 7. Effect of over-fitting parameter γ on RBF-kernel SVM training. Too large γ may produce discontinuous manifolds.

the two classes. This process is described in Algorithm 1. P_{goal} has all the points in the planning graph component containing $\boldsymbol{q}_{\text{goal}}$, and P_{rest} has the rest of the points in the planning graph. This step ensures that the manifold has enough over-fitting to separate the two classes and that it does not over-fit so much that the one class's region become discontinuous (see Figure 7). The γ parameter and the increment $\Delta\gamma$ are hyper-parameters of the algorithm. We use 1.0 and 0.1 in the experiments, which are robust across the tested robot scenes.

According to Definition 1, an infeasibility proof manifold must be closed, continuous, and entirely in $\mathcal{C}_{\text{obs}}$. In the following two paragraphs, We explain how learning the manifold from progressively larger trees results in such a manifold when there is no feasible plan.

First, if no plan exists, we eventually learn a manifold contained in $\mathcal{C}_{\text{obs}}$ given enough training points. If no plan exists, then there must be a closed obstacle region that separates the $\mathcal{C}_{\text{free}}$ region containing the goal point and the rest of $\mathcal{C}_{\text{free}}$. Since the obstacle region is closed, it has an outer boundary and an inner boundary. If we have enough points sampled close to both boundaries, these points as support vectors will force the manifold into the obstacle region. In Figure 8, as the number of points in the start and goal trees increases, the learned manifold fits more fully into $\mathcal{C}_{\text{obs}}$.

Second, using the training process that incrementally over-fits with a small enough over-fitting incremental value $\Delta\gamma$, the resulting manifold will eventually be closed and continuous. The manifold function from RBF-kernel SVM is a combination of Gaussian functions centered at the support vectors. The over-fitting parameter γ essentially influences the effecting range of the Gaussian functions. If the effecting range is too large, then there will be misclassifications. If the effecting range is too small, then the Gaussian functions will form separate regions at the support vectors (Figure 7). With the right effecting range or over-fitting parameter, the combination of Gaussian functions will be closed and continuous when the support vectors are sampled densely enough since the target obstacle region is closed. The training process that incrementally over-fits (Algorithm 1) will choose the largest effecting range (with the fixed increment $\Delta\gamma$ as changing steps) that is small enough to fit all the training data, so that the combination of Gaussian functions will not form separate regions, meaning the manifold is continuous.

To summarize, if no plan exist, given enough training points, Algorithm 1 eventually produces a manifold that is closed, continuous, and contained in $\mathcal{C}_{\text{obs}}$. At this point, we have learned a manifold that may be an infeasibility proof according to Definition 1. In the following steps, we triangulate the manifold to verify that it satisfies the three requirements of an infeasibility proof.

4.2 Sample the manifold

After learning the manifold, we must verify that it is entirely contained in the obstacle region $\mathcal{C}_{\text{obs}}$. However, directly proving that a high dimensional manifold is inside an implicitly defined space ($\mathcal{C}_{\text{obs}}$) is difficult. Instead, we triangulate the manifold into smaller pieces and then prove all the pieces are in $\mathcal{C}_{\text{obs}}$. The first step of triangulation is sampling points on the manifold.

We use the obstacle region points P_{obs} to sample points on the manifold. Typical sampling-based planners discard the P_{obs} points. In our algorithm, we save all the points sampled in obstacle region while constructing the PRM.

For each point in P_{obs}, we find the closest point on the manifold (line 5) by solving the following nonlinear constrained optimization problem,

$$\begin{aligned} \min_{\mathbf{q}_{\text{m}}} \quad & \text{dist}(\mathbf{q}_{\text{obs}}, \mathbf{q}_{\text{m}}) \\ \text{s.t.} \quad & f(\mathbf{q}_{\text{m}}) = 0 \ , \end{aligned} \tag{2}$$

where the $\mathbf{q}_{\text{obs}}$ is the given point in $\mathcal{C}_{\text{obs}}$, $\mathbf{q}_{\text{m}}$ is the manifold point we want to find, and f is the manifold's function learned in Algorithm 1. Solving this optimization problem for every point in P_{obs} produces a set of points on the manifold. Using P_{obs} as a seed to solve (2) offers an effective heuristic because these obstacle region points are likely to exist closer to the manifold in $\mathcal{C}_{\text{obs}}$. If solving this optimization problem fails for a point, we discard that point. Though (2) may not be robustly solvable for all possible configuration spaces, we are able to robustly solve this optimization problem for a

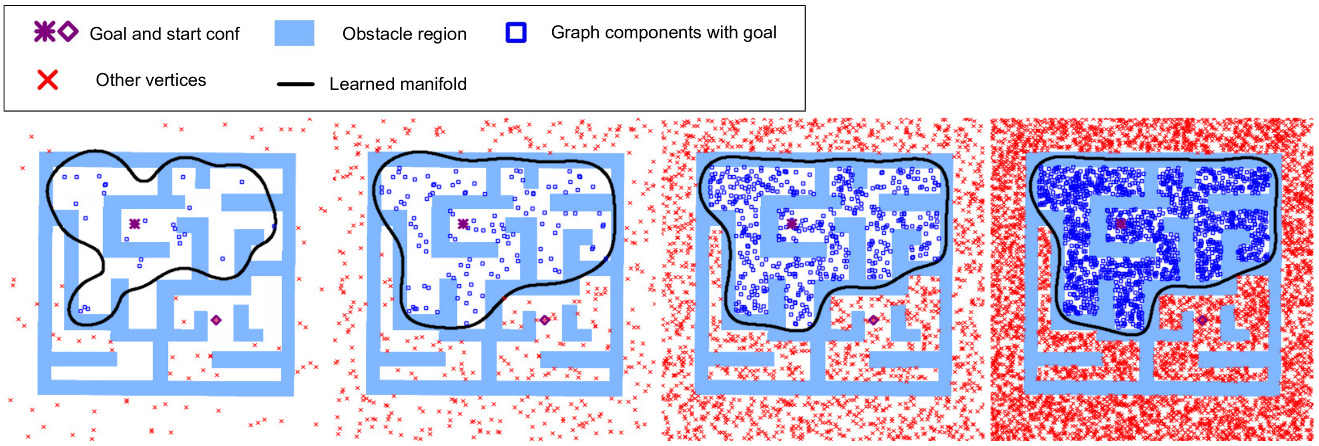

Figure 8. We treat the points connectable to the goal point as one class and all other points as the other class, and learn a classifier (the separating manifold). (a)–(d) Learned manifold fits into the obstacle region as more points are sampled (100, 500, 2000, 6000 from left to right), the blue circles are showing parts of the manifold outside of the obstacle region. The same applied to 4-DOF robot arms in section 5.

reasonable amount of manifold points in the experimental scenarios involving robot manipulators in section 5.

Algorithm 2: Sample-Manifold

```
   Input: $f$ // Manifold
   Input: $P_{\text{goal}}$ // Goal component vertices
   Input: $P_{\text{rest}}$ // Non-goal component
   Input: $P_{\text{obs}}$ // Obstacle region points
   Output: $P_{\text{m}}$ // Sampled manifold points
1  repeat
2      $P_{\text{m}} \leftarrow \emptyset$;
3      free-point ← false;
       #pragma parallel for
4      foreach $\mathbf{q}_{\text{obs}} \in P_{\text{obs}}$ do
5          $\mathbf{q}_{\text{m}} \leftarrow$ find-closest-point($\mathbf{q}_{\text{obs}}, f$);
6          if $\mathbf{q}_{\text{m}} \neq$ NAN then // opt succeed
7              if $\mathbf{q}_{\text{m}} \in \mathcal{C}_{\text{free}}$ then
8                  add-point($P_{\text{rest}}, P_{\text{goal}}, \mathbf{q}_{\text{m}}$);
9                  free-point ← true;
10             else $P_{\text{m}} \leftarrow P_{\text{m}} \cup \{\mathbf{q}_{\text{m}}\}$;
11     if free-point then // Retrain
12         $f \leftarrow$ Train-Manifold($P_{\text{rest}}, P_{\text{goal}}$);
13 until ¬free-point; // all samples in $\mathcal{C}_{\text{obs}}$
```

Algorithm 2 describes the process to sample the manifold. P_{obs} are the points sampled in $\mathcal{C}_{\text{obs}}$, P_{m} is the set of manifold points. We solve (2) in parallel for each point in P_{obs} using a thread pool. In line 5, we find the closest point on the manifold to the current point in $\mathcal{C}_{\text{obs}}$. If this optimization problem solves successfully, then we check if the manifold point is in $\mathcal{C}_{\text{free}}$. If the manifold point is in $\mathcal{C}_{\text{free}}$, then we add this point to either P_{rest} or P_{goal} with the `add-point` function (line 8). If any such sampled manifold point in $\mathcal{C}_{\text{free}}$, then the manifold cannot be fully in $\mathcal{C}_{\text{obs}}$, and we need to retrain the manifold (line 12). Before retraining, adding the violating points to the training data set can help the manifold converge into $\mathcal{C}_{\text{obs}}$ faster. After retraining, the function repeats to re-sample the points on the updated manifold. This process continues until all sampled manifold points are in $\mathcal{C}_{\text{obs}}$. We save all sampled manifold points in $\mathcal{C}_{\text{obs}}$ to P_{m}.

In the `add-point` function, we try to add the point in $\mathcal{C}_{\text{free}}$ to either P_{rest} or P_{goal} by interpolating a straight line between the point and the closest vertex on either P_{rest} or P_{goal} (line 8). Note that our implementation adds the point to a copy of the graph components for the infeasibility proof but not the graph used by the PRM planner to avoid modifying the underlying planner. Now that we have a set of points on the manifold, the next step uses these points to create a triangulation of the manifold.

4.3 Triangulation

In this step, we triangulate the manifold using the sampled manifold points from the previous step. The result is a polytope approximating the learned manifold. Proving a manifold is in $\mathcal{C}_{\text{obs}}$ directly is difficult. Instead, we construct the polytope using tangential Delaunay complexes (Boissonnat and Ghosh 2014; Boissonnat et al. 2018) to reconstruct a triangulation of a manifold from the set of sampled points on the manifold. Boissonnat and Ghosh (2014); Boissonnat et al. (2018) provide an algorithm to construct the tangential Delaunay complexes for triangulation of manifolds, which is implemented in Jamin (2020). We apply this algorithm to the sampled manifold points from subsection 4.2 to triangulate the learned SVM manifold from subsection 4.1. The triangulation of the manifold forms a polytope, which we use in later steps.

Here, we focus on the key requirements and results of the triangulation algorithm; please see (Boissonnat et al. 2018, Ch 7 and 8) for more details. Manifold triangulation using tangential Delaunay complexes (subsection 4.3) requires underlying the manifold to be a closed and differentiable submanifold of an n-dimensional Euclidean space (Boissonnat et al. 2018, Ch 8). In our case, the learned manifold from RBF kernel-SVM is differentiable since the resulting manifold function is a combination of Gaussian functions. At this point, even though we know the learned manifold eventually converges to be closed, we must still verify this. If the manifold is not closed, then the tangential Delaunay complexes triangulation will fail. In this sense, the triangulation step also validates that the manifold is closed. If triangulation is successful, then the manifold must be closed. If the triangulation step is not successful, we must retrain the manifold with more sampled points in $\mathcal{C}_{\text{free}}$. A non-closed manifold is not the only reason that triangulation may fail; the second reason—which we observed more frequently—depends on the sampled points on the manifold.

The second requirement for the tangential Delaunay complex triangulation is that the sampling points must be distributed *densely* and *evenly* on the manifold. Boissonnat et al. (2018) define densely and evenly as follows.

Definition 2. *A finite point set P is an (ϵ, η)-net of $\mathcal{M}$, if and only if it is:*

- *(ϵ-dense) for any point $x \in \mathcal{M}$, let p be the closest point to x in P, $\|p - x\| < \epsilon$;*
- *(η-separated) for any two points $p, q \in P$, $\|p - q\| > \epsilon\eta$.*

Definition 2 provides the second requirement on the set manifold points: they must be an (ϵ, η)-net of the manifold for a small enough ϵ.

We ensure the (ϵ, η)-net requirement using a subsampling process. This subsampling process has two parts. First, we subsample a fixed subset of the manifold points (half of all the manifold points) that are as far away from each other as possible. Next, we subsample again from the result of the first subsampling by ensuring a minimum distance $d_{\min}$ between two points. Together, these two subsampling steps ensure the resulting points set is an (ϵ, η)-net of the learned manifold for some ϵ and η. With a fixed $d_{\min}$, if we have more manifold points, ϵ would be smaller. We do not need to calculate ϵ and η exactly. If the triangulation step fails because of the (ϵ, η)-net requirement, it means we need a smaller ϵ, so we need to sample more points on the manifold.

The minimum distance $d_{\min}$ in the subsampling process is a hyper-parameter in our algorithm. We choose the $d_{\min}$ according to the obstacle region $\mathcal{C}_{\text{obs}}$ of the configuration space, with larger $d_{\min}$ if $\mathcal{C}_{\text{obs}}$ separating the goal or start is thick and smaller $d_{\min}$ if $\mathcal{C}_{\text{obs}}$ separating the goal or start is thin. In general, this value should be small enough to approximate any curves on the manifold. This parameter is empirical (see section 5 for the specific values for each robot scenes). Generally, one may choose a small value for $d_{\min}$ to support greater obstacle region curvature.

Boissonnat et al. (2018) prove that if the above requirements are satisfied and ϵ is small enough, the resulting triangulation from the algorithm approximates the manifold with bounded error. The algorithm is linear in the dimension of the Euclidean space (n), exponential in the dimension of the manifold ($n - 1$ in our case, where n is the dimension of $\mathcal{C}$), and quadratic in the number of sampled points (Boissonnat and Ghosh 2014).

If applying the algorithm on the subsampled points returns a triangulation of the manifold successfully, then we have constructed a closed polytope for the next step. Figure 3 shows a 2D polytope, and Figure 9 shows polytopes for a predefined 3D configuration space and the configuration space of a 3-DOF Jaco arm. If the triangulation is not successful, we go back to obtain more manifold points for a small ϵ.

The polytope constructed is an approximation of the learned manifold. Proving that the polytope is entirely in $\mathcal{C}_{\text{obs}}$ does not prove that the learned manifold is entirely in $\mathcal{C}_{\text{obs}}$.

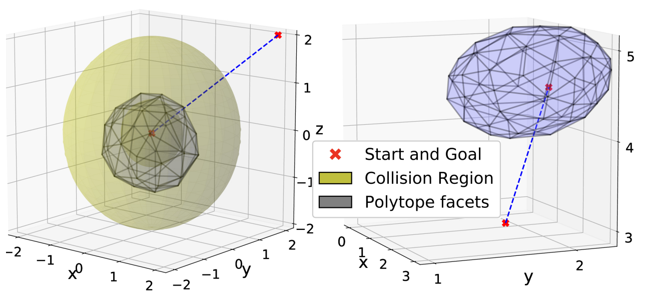

Figure 9. Example 3D polytopes constructed using tangential Delaunay complexes. (Left) The obstacle region is between two balls of radius 2.0 and 0.5. (Right) The polytope constructed in the 3 DOF Jaco arm experiment (no obstacle region shown).

Instead, the polytope is the infeasibility proof we have verified to satisfy all requirements in Definition 1, not the learned manifold itself. The learned manifold is a strong heuristic for constructing this polytope.

4.4 Checking the polytope

After constructing the polytope in the previous step, we check that the polytope is entirely in $\mathcal{C}_{\text{obs}}$ by checking each facet of the polytope. If all facets of the polytope are in $\mathcal{C}_{\text{obs}}$, the polytope is entirely in $\mathcal{C}_{\text{obs}}$ and satisfies the requirements of Definition 1—i.e., the polytope is an infeasibility proof.

Algorithm 3: Check-Polytope

```
Input: $F$ // Facets
Output: ok // All facets in $\mathcal{C}_{\text{obs}}$
1  function check-facet($\mathbf{f}_t$) is
2      $r, c \leftarrow$ Mini-Ball($\mathbf{f}_t$);
3      if $c \notin \mathcal{C}_{\text{obs}}$ then return false;
4      pd $\leftarrow$ cal-Pene-Dist($c$);
5      if pd = NAN then // PD not found
6          return false
7      else if $r \leq$ pd then// ball inside $\mathcal{C}_{\text{obs}}$
8          return true
9      else // ball outside $\mathcal{C}_{\text{obs}}$
10         res $\leftarrow$ true;
11         $\mathbf{f}_{td} \leftarrow$ decompose($\mathbf{f}_t$);
12         foreach $\mathbf{f}'_t \in \mathbf{f}_{td}$ do
13             if $\neg$check-facet($\mathbf{f}'_t$) then
14                 res $\leftarrow$ false;
15         return res;
16 ok $\leftarrow$ true;
   #pragma parallel for
17 foreach $\mathbf{f}_t \in F$ do
18     if $\neg$check-facet($\mathbf{f}_t$) then
19         ok $\leftarrow$ false;
```

Algorithm 3 shows the procedure to check the facets. We check each facet in the polytope iteratively, in parallel (line 17), while the check of each facet runs recursively (`check-facet`). In the `check-facet` function, $\mathbf{f}_t$ is the input facet to check, and pd stands for penetration depth. First, we calculate the minimum enclosing ball of the given facet (line 2). The minimum enclosing ball algorithm (Fischer et al. 2003) calculates the radius and center of the minimum enclosing ball in high dimension given a set of points. In our case, the set of points are the vertices of the facet $\mathbf{f}_t$. Note here the facets' vertices are n-dimensional, where n is the dimension of the configuration space. We want to find the minimum enclosing ball on the facet's hyper-plane to check the facet's existence in $\mathcal{C}_{\text{obs}}$. Because of this, we first project a facet's vertices to $n-1$ dimension with the Gram–Schmidt process, calculate the minimum enclosing ball, then project the minimum enclosing ball's center back to n dimension to calculate its penetration depth. If the center of the minimum enclosing ball is not in $\mathcal{C}_{\text{obs}}$, it means most likely the facet is not in $\mathcal{C}_{\text{obs}}$, and we need to make a finer triangulation.

If the center of the minimum enclosing ball is in $\mathcal{C}_{\text{obs}}$, then we calculate the center's configuration space penetration depth (line 4). Finding the *configuration space* penetration depth (CPD) is a key sub-routine. Existing libraries (Pan et al. (2012)) compute *Cartesian space* penetration depth. We find the configuration space penetration depth of a point in $\mathcal{C}_{\text{obs}}$ by solving the following nonlinear optimization problem,

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \text{dist}(\boldsymbol{q}_o, \boldsymbol{q}) \\ \text{s.t.} \quad & \text{Penetration-Depth}(x_l(\boldsymbol{q}), x_o) \le 0 \\ & l \in \{1, \ldots, n_{\text{link}}\}, o \in \{1, \ldots, n_{\text{obs}}\}, \end{aligned} \tag{3}$$

where $\boldsymbol{q}_o$ is the configuration in $\mathcal{C}_{\text{obs}}$, $\boldsymbol{q}$ is the point we want to find in $\mathcal{C}_{\text{free}}$, n_{link} is the number of manipulator links, n_{obs} is the number of obstacles, x_l is Cartesian point of maximum penetration on link l, x_o is the Cartesian point of maximum penetration on obstacle o. If link l and obstacle o are in collision, CPD between these two frames is positive; otherwise, CPD is negative. The optimization objective is to minimize configuration space distance, subject to $\boldsymbol{q}$ being a point in $\mathcal{C}_{\text{free}}$ (see Figure 10). In the constraints, we ensure $\boldsymbol{q}$ is in $\mathcal{C}_{\text{free}}$ by checking every robot link against every object. A point outside of $\mathcal{C}$ boundaries (joint limits) is also considered to be in $\mathcal{C}_{\text{obs}}$, and its CPD is the distance to its closest $\mathcal{C}_{\text{free}}$ point in the joint limits.

Whether we can solve this optimization problem successfully depends on a good initial value of $\boldsymbol{q}$. In our implementation, we find the closest point to $\boldsymbol{q}_o$ in the planning graph and use this point as the initial value of $\boldsymbol{q}$. If solving (3) fails with this initial value, then it means we need a initial value closer to $\boldsymbol{q}_o$. To get a better initial value, we sampled randomly around a neighborhood of $\boldsymbol{q}_o$ for a $\boldsymbol{q}$ in $\mathcal{C}_{\text{free}}$ that is closer to $\boldsymbol{q}_o$ than the first initial value from the planning graph and re-calculate its penetration depth for a fixed number of attempts. This is an empirically robust way to compute penetration depth. If the optimization problem is not solved for a point in $\mathcal{C}_{\text{obs}}$, then it means we need a denser sampling in $\mathcal{C}_{\text{free}}$, so we loop back to get more $\mathcal{C}_{\text{free}}$ configurations from the buffer.

If the penetration depth of the center of the minimum enclosing ball is larger than its radius, then the facet is entirely in $\mathcal{C}_{\text{obs}}$ and passes the check since the facet is enclosed inside the ball (line 7). If not, then we decompose the facet into smaller facets and recursively check each of its decompositions (line 12). We keep recursing on smaller pieces until we have checked the entire facet or we have found a failing piece of facet with its minimum enclosing ball's center in $\mathcal{C}_{\text{free}}$.

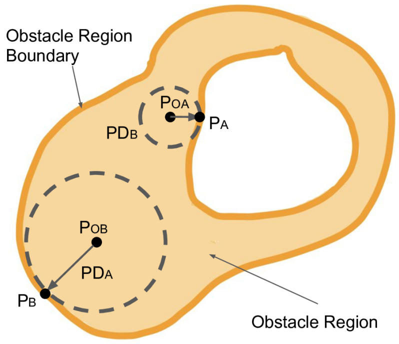

Figure 10. Configuration space penetration depth of two points P_{OA} and P_{OB} in $\mathcal{C}_{\text{obs}}$. P_A and P_B are their corresponding closest points that satisfy the constraints.

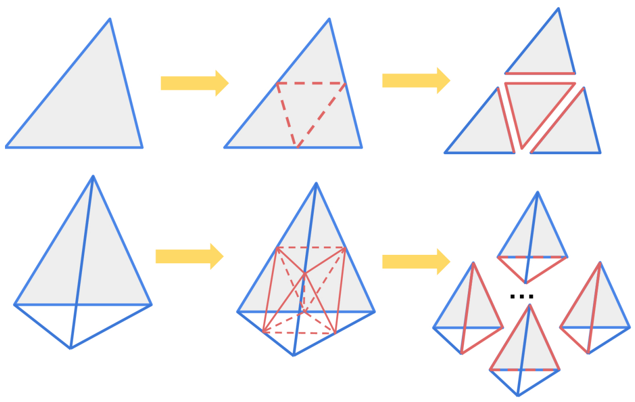

Figure 11. Decomposition of a 2D facet (top) and a 3D facet (bottom).

The `decompose` function (line 11) takes a facet and returns a set of smaller pieces of the facet whose union is the original facet. The decomposition depends on the dimension of facets. For example, in a 3-DOF configuration space, the facets are 2D triangles. Using the triangle edges' middle point, we can decompose a triangle into 4 pieces. In a 4-DoF configuration space, the facets are 3D, and the decomposition has 8 smaller pieces. Figure 11 shows this decomposition for a 2D facet and a 3D facet.

This facet checking process may fail for two reason. If the checking fail because one or more points penetration depth is not solved (line 5), then we need more sampled points in $\mathcal{C}_{\text{free}}$ to serve as initial value for the optimization problem (3). If the checking process fails because points on the facet may be in $\mathcal{C}_{\text{free}}$ (line 9), then either we need to retrain the manifold (the manifold is not entirely in $\mathcal{C}_{\text{obs}}$, so the facets are not entirely in $\mathcal{C}_{\text{obs}}$) or we need a denser sampling on the manifold (the manifold is entirely in $\mathcal{C}_{\text{obs}}$, but the facets are not entirely in $\mathcal{C}_{\text{obs}}$). In this case, we go back to retrain and sample more points on the manifold.

If all facets pass the check, it means the polytope is entirely in $\mathcal{C}_{\text{obs}}$ and we have found an infeasibility proof. As stated before, the final infeasibility proof is actually the polytope, not the manifold itself.

4.5 Algorithm Summary

In this section, we summarize how the infeasibility proof construction works as a whole. In previous sections, we described the four important steps: (1) learning a manifold, (2) sampling points on the manifold, (3) triangulation of the manifold, and (4) checking the polytope. The four steps run sequentially, and each step depends on the result of the previous step. If all steps are successful, the result is proven infeasibility. Although we know the algorithm would return in the limit, we may need to obtain additional samples and re-run the steps. Algorithm 4 describes the flow between the steps when constructing the infeasibility proof.

Algorithm 4: Find-InfProof (P_{rest}, P_{goal}, P_{obs})

```
1  TC-res ← false;
2  facets-res ← false;
3  $f$ ← Train-Manifold($P_{\text{rest}}$, $P_{\text{goal}}$);
4  repeat
5      Sample-Manifold($f$, $P_{\text{obs}}$, $P_{\text{rest}}$, $P_{\text{goal}}$);
6      if ¬TC-res then
7          TC-res ← generate-TC($P_{\text{m}}$);
8      if ¬TC-res then
9          generate-mpoints();
10         continue;
11     facets-res ← check-polytope(F);
12     if ¬facets-res then
13         if PD fail then
14             get-free-point();
15         else
16             generate-mpoints();
17             TC-res ← false;
18         continue;
19 until TC-res ∧ facets-res;
```

After we train the manifold (line 3), we perform an initial check to make sure the manifold is largely converged into $\mathcal{C}_{\text{obs}}$ by verifying that all manifold points are in $\mathcal{C}_{\text{obs}}$ (in `Sample-Manifold`). Then we try to triangulate the manifold. If the triangulation fails, we need a denser sampling. The function `generate-mpoint` helps generating more points on the manifold. In this function, we take two randomly picked manifold points and interpolate their middle point. Then we apply the middle point to (2) to calculate a new manifold points. We continue generating more manifold points until the triangulation is successful.

After we have a successful triangulation of the manifold, the last step is to check each facet in the triangulation. This step can fail for two reasons as stated in subsection 4.4. If calculating penetration depth fails, we need to get more $\mathcal{C}_{\text{free}}$ points (line 14) from the planning graph. Otherwise, we make a finer triangulation using `generate-mpoint`. If all steps run successfully, this algorithm terminates with an infeasibility proof of the given configuration space.

5 Experiments

We run experiments on manipulator motion planning problems to show how the algorithm works in different dimensions and how it behaves with and without a feasible plan. In scenarios with no feasible plan, we show distributions of running time over many trials and profile to find computational bottlenecks. These experiments indicate that triangulating the manifold is the current bottleneck limiting scaling to higher DOF. In scenarios with feasible plans, we compare running times with and without the infeasibility proof construction to show the overhead.

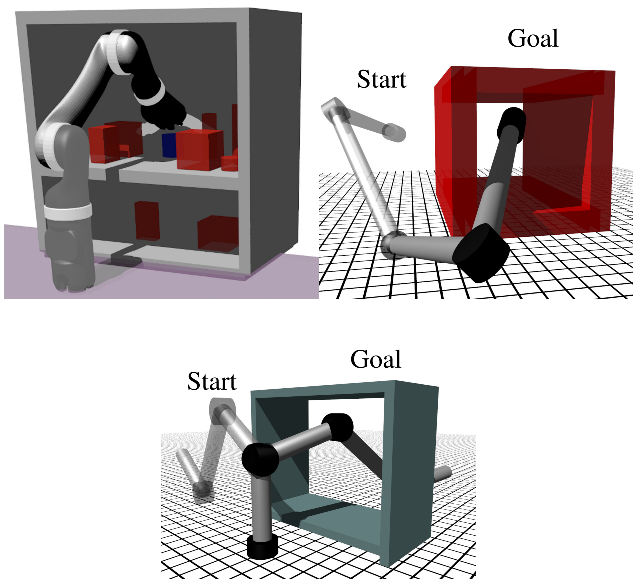

Figure 12. Experiment scenes. Top-left: Jaco arm trying to reach inside a shelf. Top-right: 4-DOF shoulder-elbow robot trying to reach inside a box. Bottom: ZYYY arm trying to reach inside the frame.

The algorithm terminated in every experimental trial, returning either an infeasibility proof when the motion planning problem was infeasible or a motion plan when the problem was feasible. We run experiments on one 3-DOF manipulator scene and three 4-DOF manipulator scenes. The following subsections describe the experiments in more detail.

To use the inherent parallelism in several parts of infeasibility proof construction, we run our experiments on a multi-core system, a dual CPU AMD EPYC 7402 with 24 cores per CPU. We adapt PRM (Kavraki et al. 1996) in OMPL (Şucan et al. 2012) to run in parallel with our infeasibility proof construction. We solve the nonlinear optimization problems using sequential least-squares quadratic programming (SLSQP) (Kraft 1988). We construct the polytope using the tangential complex module in GUDHI package (Jamin 2020). We train the manifold using LIBSVM (Chang and Lin 2011). We check collisions and penetration depth using the Flexible Collision Library (Pan et al. 2012), and we model robot kinematics using Amino (Dantam 2020a). We determine ground truth for plan non-existence to high-confidence for a scene by running RRT-connect continuously for greater than 20 minutes.

5.1 3-DOF Infeasible Experiments

The first experiment uses the same Jaco arm scenario as Li and Dantam (2020). The goal in this scene is to reach the blue block at the back of the shelf from a position outside of the shelf (see Figure 12, top-left). We run the experiments for 50 trials and calculate the mean running time and standard deviation. Table 1 shows the mean running time of the entire infeasibility proof construction, the time taken by the triangulation step, time taken by the sampling manifold points step and the time taken by the checking facets step. The table also shows the number of manifold points sampled and the number of facets in the triangulation polytope. The running time distribution among the 50 trials is shown in Figure 13, on the top-left.

In this experiment, we use a primitive shape collision geometry model of the Jaco arm for computing configuration space penetration depth. This is why the experiment takes significantly less time than in Li and Dantam (2020) and Li and Dantam (2021) (with average running time of 457.5s and 177.36s respectively). The use of parallel computing also greatly benefits running time. In this implementation, with an average running time of 8.00 seconds among 50 trials, and a running time distribution that is mostly under 20 seconds, we can say that the algorithm is practically applicable to 3-DOF manipulators.

5.2 4-DOF Infeasible Experiments

We also run experiments on three 4-DOF scenes to demonstrate the improved scalability and analyze running time distributions. The first scene uses a 4-DOF shoulder-elbow robot (a three DOF spherical joint and a one DOF revolute joint, see Figure 12 top-right). The goal in this scene is to reach inside the red box. The second scene uses the 4-DOF SCARA arm, and the scene is shown in Figure 1. The SCARA arm has three co-planar revolute joints and one prismatic joint (Makino 1982). The goal in this scene is to reach inside the purple box with the red block attached at the end-effector. The third scene use a ZYYY robot (see Figure 12, bottom). The goal in this scene is to reach the inside of the frame from a position outside of the frame. Table 1 lists running times, and Figure 13 shows the distributions.

In the 4-DOF shoulder-elbow robot scene, we use a subsampling minimum distance of 0.002, in the SCARA arm scene, we use a subsampling mininum distance of 0.006, and in the ZYYY arm scene, we use a subsampling minimum distance of 0.007. Intuitively, these minimum distance values comes from the smallest "curve" on the learned manifold. This minimum distance should be small enough to form triangulation at smallest curves on the manifold. In experiments, we determine these numbers empirically. We start with a larger value and the if triangulation with tangential complex does not work, then decrease the value gradually. A small value would always work but needs more manifold points and creates more facets, increasing running time.

Infeasibility proof construction for the 4-DOF shoulder-elbow robot and the ZYYY arm takes around two minutes on average, which is about half of the average running time in Li and Dantam (2021) for the shoulder-elbow robot. For the SCARA robot scene, the average running time is around four minutes, which is also about half of the average running time in Li and Dantam (2021). The running time improvements come primarily from parallelization. Table 1 shows that the parallel parts of the algorithm—sampling manifold points and checking facets—now consume a minor amount of the total running time. Triangulation, which remains serial in the current implementation, takes the significant majority ($> 90\%$) of the time.

We also see from Table 1 that all the rest of algorithm, including learning, takes a small portion of running time. Notably, learning the manifold is fast. Finding this manifold is the first and most important step in the infeasibility proof construction, since the rest of the algorithm is only to check the manifold's containment in $\mathcal{C}_{\text{obs}}$. Improvements to this check could greatly reduce overall running time.

Figure 13 shows distributions of running times for the four scenes' infeasibility proof construction. The majority of trials are faster than the average, with a few slow outliers. We also see that the distributions depend on the complexity of scenes. For the simpler scenes, the distributions are more concentrated (the 3-DOF jaco arm scene, the 4-DOF shoulder-elbow robot scene and the ZYYY arm scene). The SCARA arm scene has a more scattered distribution since it is more complicated.

5.3 Experiments with Feasible Plans

For the above four scenes, we modify the scenes to make plans feasible but still difficult to find. For the Jaco arm, we move the shelf further away from the robot base to make the blue box's position reachable. For the 4-DOF shoulder-elbow robot, we move the red box away from the robot base to make the inside reachable. For the SCARA arm, we move the ball away from the box to make room for the red block attached at the end-effector to pass. For the ZYYY arm, we move the frame further away from the robot's base to create room for the arm to get inside. We test these scenes building a PRM with and without the infeasibility proof construction for 50 trials each. Table 2 shows the results.

When there is a feasible plan, building a PRM with infeasibility proof construction introduces minor absolute overhead if the plan can be found in a relatively short time. If the scene is more complicated, infeasibility proof construction introduces more overhead. The overhead is mainly due to saving all the sampling configurations, since infeasibility proof and PRM construction run in parallel.

Another purpose of these feasible plan experiments is to show that the algorithm correctly terminates. With the 200 trials of feasible plan experiments we run, the algorithm always terminates with a plan.

6 Discussion and Future Work

Our algorithm has two hyper-parameters, and termination depends on appropriate selection of these hyper-parameters. The training over-fitting parameter γ effects learning the manifold. In the experiments, we use $\gamma = 1.0$ and $\Delta\gamma = 0.1$ for all the scenes. Qualitatively, these hyper-parameters were not sensitive to the tested scenes, and these given values worked across many scenes. Another hyper-parameter $d_{\min}$ does depend on the scene, since it must be small enough to capture any sharp curves on the learned manifold, which depends on curvature of the obstacle region. We acknowledge the algorithm's reliance on these hyper-parameters. However, we note that such dependence upon hyper-parameters exists in many algorithms. For example, selecting an RRT step size that is too large may cause failure (Wang and Meng 2016).

Our current algorithm applies to kinematic motion planning without differential constraints. We learn the manifold and check containment in $\mathcal{C}_{\text{obs}}$ based on geometries of the configuration space and specifically assume a Euclidean

Manipulator Experimental Results in Infeasible Scenes							
Scenes		*Total (s)*	*TC (s)*	*samp-mf (s)*	*checking (s)*	*# of mpoins*	*# of Facets*
Jaco Arm	*Mean*	8.00	4.43	1.43	1.86	20910.86	547.96
	STD	15.57	10.70	1.72	2.53	13236.44	536.45
Shoulder-Elbow	*Mean*	110.49	108.14	0.43	1.78	4432.52	12383.66
	STD	113.23	111.23	0.42	3.61	5087.12	8528.07
SCARA arm	*Mean*	245.07	221.00	3.34	17.73	174711.12	15096.52
	STD	168.34	159.62	1.85	14.50	45897.44	289.47
ZYYY arm	*Mean*	131.51	107.71	3.73	18.21	124019.48	96862.10
	STD	125.47	98.07	2.14	32.97	36671.50	14263.18

Table 1. Running time profiling results of infeasibility proof construction in four scenes, averaged over 50 trials. The jaco arm scene is 3-DoF, the rests are 4-DoF. "TC" is for triangulation with tangential complex, "samp-mp" is for sampling of manifold points, "checking" is for checking of facets.

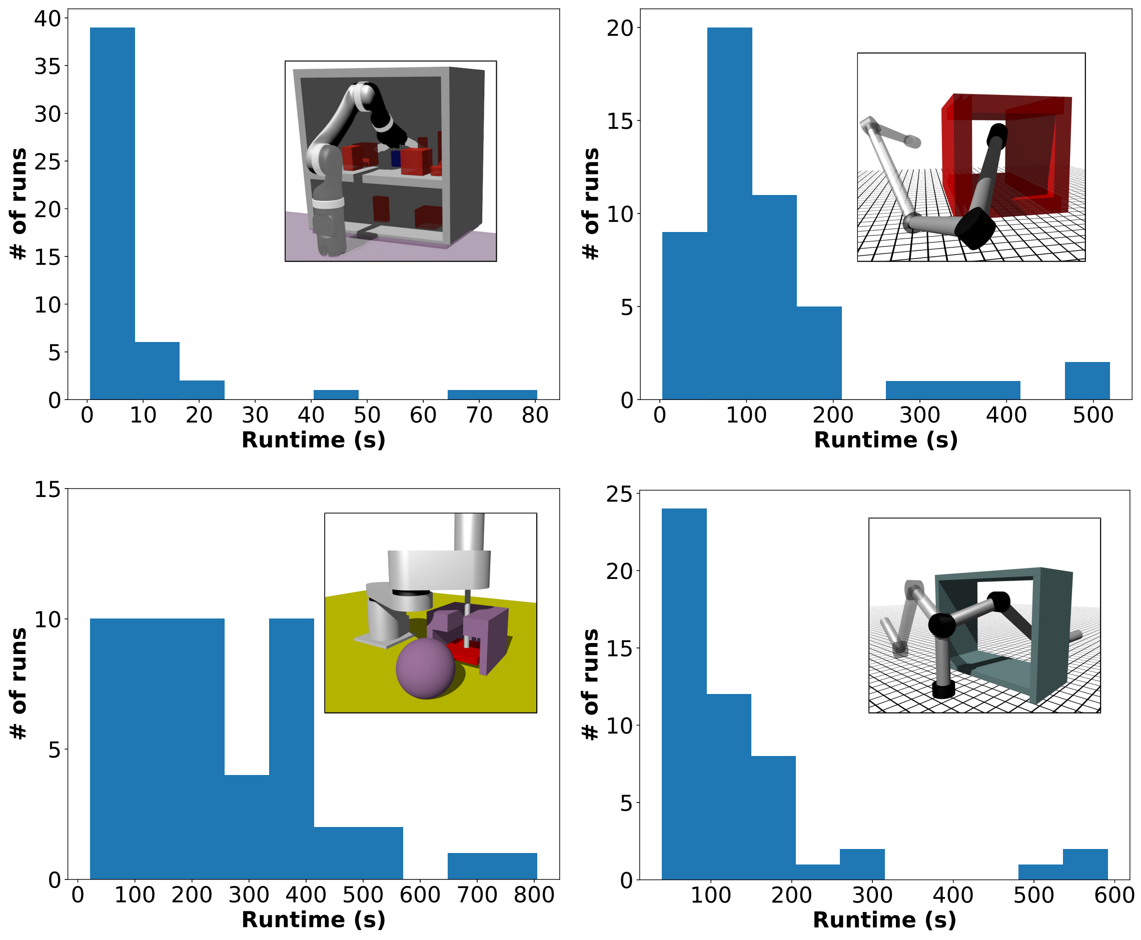

Figure 13. The running time distribution of 50 trials in the three scenes. Top-left: 3-DOF Jaco arm. Top-right: 4-DOF shoulder-elbow robot. Bottom-left: SCARA arm. Bottom-right: ZYYY arm.

Feasible Plan Experiments mean/std (s)				
	Jaco Arm	*Shoulder-Elbow Arm*	*SCARA Arm*	*ZYYY Arm*
PRM only	3.95 /5.73	0.70/0.45	15.01/12.87	1.21/1.02
PRM w/ IF	3.77/3.61	1.09/3.22	37.04/27.44	1.29/1.11

Table 2. Experimental results for plan feasible experiments, averaged over 50 trials, running PRM only vs. running PRM with infeasibility proof.

configuration space. In contrast, differential constraints need not follow such metrics of a space. Handling differential constraints in motion planning infeasibility proofs remains a potential direction for future work.

6.1 Scaling to higher dimensions

Our current algorithm works for up to 4-DOF configuration spaces. Critically, learning the manifold consumes a negligible part of running time. Rather, the bottleneck is checking that manifold is in the obstacle region. Scaling to a higher dimension may employ a similar learning step but requires greater efficiency in the checking step. Table 1 shows that the triangulation in the checking step takes a large part of the total runtime in all the scenes. Roughly, further optimization of other parts of the algorithm would improve running times by at most 20%, while optimizing the triangulation could yield 80% speedups (according to Amdahl's law). We propose two directions for future work to scale to higher dimensions: to explore and apply other triangulation methods and to further leverage parallelism.

For the first option, we anticipate that the runtime for triangulation will significantly decrease for more advanced triangulation methods. For example, Kachanovich (2019) proposed the Coxeter triangulation algorithm that in some cases is an order of magnitude faster than our current triangulation method using tangential Delaunay complexes. Our algorithm could be modified to incorporate this Coxeter triangulation.

At the same time, we want to explore methods to parallelize the triangulation step. Exploiting the parallelism of modern CPUs and GPUs offers benefits for planning problems (Pan and Manocha 2012; Chrétien et al. 2016). In our current algorithm, the sampling manifold and checking facets steps both apply to lists of independent elements, which is *embarrassingly parallel*, i.e., little data is shared between threads, enabling near linear speedup with number of cores. This useful property comes from the structure of our algorithm. In contrast, algorithms that must share more data between threads can pose challenges for parallelization and offer less potential speedup. In particular, the current triangulation step is not easily parallelizable since each element of the triangulation, i.e., the facets on a polytope, are connected with one another in the final structure. To efficiently parallelize the triangulation step, we propose to investigate methods that grow facet on the manifold in parallel and fix gaps between the facets in a later step (Akkouche and Galin 2001).

To summarize, exploring other triangulation methods and/or adapting the triangulation to be parallelizable offer potential directions for future work to scale this framework to higher dimensions.

6.2 Completeness

Given a Euclidean configuration space with non-zero volume obstacle regions and the ability to obtain manifold points and configuration space penetration depth (assumptions in section 3), the algorithm eventually terminates with either an infeasibility proof or a plan. To find an infeasibility proof, the learned manifold must exist entirely in $\mathcal{C}_{\text{obs}}$, which occurs as more $\mathcal{C}_{\text{free}}$ points are sampled in the configuration space. If there is a part of the manifold in $\mathcal{C}_{\text{free}}$, the checking steps would generate $\mathcal{C}_{\text{free}}$ manifold points on that part of the manifold. These $\mathcal{C}_{\text{free}}$ manifold points are added back to the training data set to re-learn the manifold and "push" the $\mathcal{C}_{\text{free}}$ part of the manifold into $\mathcal{C}_{\text{obs}}$. This iterative process causes the learned manifold to eventually converge into $\mathcal{C}_{\text{obs}}$. Then, the following checking step can use this manifold to generate an infeasibility proof, which is the triangulation of the manifold. Concurrently, the PRM planner runs in another thread to find a plan. The result of either an infeasibility proof or a plan terminates the algorithm. However, the termination of the algorithm depends on sampling in $\mathcal{C}_{\text{free}}$ and sampling on the manifold, which is not guaranteed in finite time. We say our algorithm is complete *in the limit* (given long enough time), which is stronger than probabilistic completeness with the additional infeasibility proof and weaker than the definition of completeness since the algorithm cannot guarantee return in finite time.

7 Conclusion

In this paper, we introduce a framework and algorithm to construct infeasibility proofs alongside sampling-based motion planners by using sampled configurations as training data for learning. This framework produces a plan or an infeasibility proof. The infeasibility proof construction has two major steps, learning a manifold and checking the manifold's containment in $\mathcal{C}_{\text{obs}}$. Our implementation based on this framework leverages parallel computing capabilities to reduce overall running time. Many experimental trials demonstrate improved performance over previous work. Potential approaches to improve efficiency of checking the manifold may offer a viable future route to apply this technique to higher DOF problems, with applications to many practical manipulators.

Acknowledgements

We thank Dr. Mehmet Belviranli for providing access to the multi-core server used for evaluating the parallel implementation of the presented algorithm.

Declaration of conflicting interests

The Authors declare that there is no conflict of interest.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Science Foundation [Grant No. IIS-1849348].

References

Akkouche S and Galin E (2001) Adaptive implicit surface polygonization using marching triangles. In: *Computer Graphics Forum*, volume 20. Wiley Online Library, pp. 67–80.

Amato NM and Wu Y (1996) A randomized roadmap method for path and manipulation planning. In: *1996 IEEE International Conference on Robotics and Automation (ICRA)*, volume 1. pp. 113–120.

Bajcsy A, Bansal S, Bronstein E, Tolani V and Tomlin CJ (2019) An efficient reachability-based framework for provably safe autonomous navigation in unknown environments. In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, pp. 1758–1765.

Bansal S and Tomlin CJ (2021) Deepreach: A deep learning approach to high-dimensional reachability. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 1817–1824.

Basch J, Guibas LJ, Hsu D and Nguyen AT (2001) Disconnection proofs for motion planning. In: *2001 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 1765–1772.

Ben-Shahar O and Rivlin E (1998) Practical pushing planning for rearrangement tasks. *IEEE Transactions on Robotics* 14(4): 549–565.

Berenson D, Srinivasa SS, Ferguson D and Kuffner JJ (2009) Manipulation planning on constraint manifolds. In: *2009 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 625–632.

Bialkowski J, Otte M and Frazzoli E (2013) Free-configuration biased sampling for motion planning. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1272–1279.

Boissonnat JD, Chazal F and Yvinec M (2018) *Geometric and Topological Inference*. Cambridge University Press. URL https://hal.inria.fr/hal-01615863. Cambridge Texts in Applied Mathematics.

Boissonnat JD and Ghosh A (2014) Manifold reconstruction using tangential delaunay complexes. *Discrete & Computational Geometry* 51(1): 221–267.

Branicky MS, LaValle SM, Olson K and Yang L (2001) Quasi-randomized path planning. In: *2001 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 1481–1487.

Cambon S, Alami R and Gravot F (2009) A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research* 28(1): 104–126.

Chang CC and Lin CJ (2011) LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology* 2: 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chrétien B, Escande A and Kheddar A (2016) Gpu robot motion planning using semi-infinite nonlinear programming. *IEEE Transactions on Parallel and Distributed Systems* 27(10): 2926–2939. DOI:10.1109/TPDS.2016.2521373.

Şucan IA and Kavraki LE (2012) A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics* 28(1): 116–131. DOI:10.1109/tro.2011.2160466. URL http://dx.doi.org/10.1109/tro.2011.2160466.

Şucan IA, Moll M and Kavraki LE (2012) The open motion planning library. *IEEE Robotics & Automation Magazine* 19(4): 72–82.

Dantam NT (2020a) Robust and efficient forward, differential, and inverse kinematics using dual quaternions. *The International Journal of Robotics Research* .

Dantam NT (2020b) *Task and Motion Planning*. Springer Berlin Heidelberg.

Dantam NT, Kingston ZK, Chaudhuri S and Kavraki LE (2018) An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research* 37(10): 1134–1151.

Dobson A and Bekris KE (2014) Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research* 33(1): 18–47.

Donald B, Xavier P, Canny J and Reif J (1993) Kinodynamic motion planning. *Journal of the ACM (JACM)* 40(5): 1048–1066.

Dornhege C, Eyerich P, Keller T, Trüg S, Brenner M and Nebel B (2012) Semantic attachments for domain-independent planning systems. In: *Towards Service Robots for Everyday Environments*. Springer, pp. 99–115.

Driess D, Ha JS, Tedrake R and Toussaint M (2021a) Learning geometric reasoning and control for long-horizon tasks from visual input. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 14298–14305.

Driess D, Ha JS and Toussaint M (2021b) Learning to solve sequential physical reasoning problems from a scene image. *The International Journal of Robotics Research* 40(12-14): 1435–1466.

Driess D, Oguz O, Ha JS and Toussaint M (2020) Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 9563–9569. DOI: 10.1109/ICRA40945.2020.9197291.

Erdem E, Patoglu V and Schüller P (2016) A systematic analysis of levels of integration between high-level task planning and low-level feasibility checks. *AI Communications* 29(2): 319–349.

Fischer K, Gärtner B and Kutz M (2003) Fast smallest-enclosing-ball computation in high dimensions. In: *European Symposium on Algorithms*. Springer, pp. 630–641.

Fogel E, Halperin D and Wein R (2012) *CGAL arrangements and their applications: A step-by-step guide*, volume 7. Springer Science & Business Media.

Garrett CR, Chitnis R, Holladay R, Kim B, Silver T, Kaelbling LP and Lozano-Pérez T (2021) Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems* 4: 265–293.

Halperin D, Salzman O and Sharir M (2017) Algorithmic motion planning. In: *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, pp. 1311–1342.

Han SD, Stiffler NM, Krontiris A, Bekris KE and Yu J (2018) Complexity results and fast methods for optimal tabletop rearrangement with overhand grasps. *The International Journal of Robotics Research* 37(13-14): 1775–1795.

Hartmann E (1998) A marching method for the triangulation of surfaces. *The Visual Computer* 14(3): 95–108.

Hauser K (2014) The minimum constraint removal problem with three robotics applications. *The International Journal of Robotics Research* 33(1): 5–17.

Hauser KK (2013) Minimum constraint displacement motion planning. In: *Proceedings of Robotics: Science and Systems*, volume 6. p. 2.

Hilton A, Illingworth J et al. (1997) Marching triangles: Delaunay implicit surface triangulation. *University of Surrey* .

Hsu D, Kindel R, Latombe JC and Rock S (2002) Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research* 21(3): 233–255.

Huang E, Jia Z and Mason MT (2019) Large-scale multi-object rearrangement. In: *2019 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 211–218.

Jamin C (2020) Tangential complex. In: *GUDHI User and Reference Manual*, 3.4.0 edition. GUDHI Editorial Board. URL https://gudhi.inria.fr/doc/3.4.0/group__tangential__complex.html.

Janson L, Ichter B and Pavone M (2018) Deterministic sampling-based motion planning: Optimality, complexity, and performance. *The International Journal of Robotics Research* 37(1): 46–61.

Janson L, Schmerling E, Clark A and Pavone M (2015) Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research* 34(7): 883–921.

Kachanovich S (2019) *Meshing submanifolds using Coxeter triangulations*. PhD Thesis, COMUE Université Côte d'Azur (2015-2019).

Kaelbling LP and Lozano-Pérez T (2013) Integrated task and motion planning in belief space. *The International Journal of Robotics Research* 32(9-10): 1194–1227.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7): 846–894.

Karkanis T and Stewart AJ (2001) Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications* 21(2): 60–69.

Kavraki LE, Svestka P, Latombe JC and Overmars MH (1996) Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics* 12(4): 566–580.

King JE, Haustein JA, Srinivasa SS and Asfour T (2015) Nonprehensile whole arm rearrangement planning on physics manifolds. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2508–2515.

Kingston Z, Moll M and Kavraki LE (2019) Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research* 38(10-11): 1151–1178.

Kraft D (1988) A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, Institut für Dynamik der Flugsysteme, Oberpfaffenhofen.

Kuffner JJ and LaValle SM (2000) RRT-connect: An efficient approach to single-query path planning. In: *2000 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 995–1001.

Kuo YL, Barbu A and Katz B (2018) Deep sequential models for sampling-based planning. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6490–6497.

Ladd AM and Kavraki LE (2004) Fast tree-based exploration of state space for robots with dynamics. In: *Algorithmic Foundations of Robotics VI*. Springer, pp. 297–312.

Lafferriere G and Sussmann H (1991) Motion planning for controllable systems without drift. In: *1991 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 1148–1149.

Lagriffoul F and Andres B (2016) Combining task and motion planning: A culprit detection problem. *The International Journal of Robotics Research* 35(8): 890–927.

LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. Technical Report TR-98-11, Computer Science Deptartment, Iowa State University.

LaValle SM (2006) *Planning algorithms*. Cambridge university press.

Li S and Dantam NT (2020) Towards general infeasibility proofs in motion planning. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 6704–6710. DOI: 10.1109/IROS45743.2020.9340804.

Li S and Dantam NT (2021) Learning proofs of motion planning infeasibility. In: *Proceedings of Robotics: Science and Systems*.

Li Y, Littlefield Z and Bekris KE (2016) Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research* 35(5): 528–564.

Lygeros J, Tomlin C and Sastry S (1999) Controllers for reachability specifications for hybrid systems. *Automatica* 35(3): 349–370.

Makino H (1982) Assembly robot. US Patent 4,341,502.

McCarthy Z, Bretl T and Hutchinson S (2012) Proving path non-existence using sampling and alpha shapes. In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2563–2569.

Orthey A and Toussaint M (2021) Sparse multilevel roadmaps for high-dimensional robotic motion planning. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 7851–7857. DOI:10.1109/ICRA48506.2021.9562053.

Ota J (2004) Rearrangement of multiple movable objects-integration of global and local planning methodology. In: *2004 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 1962–1967.

Ota J (2009) Rearrangement planning of multiple movable objects by a mobile robot. *Advanced Robotics* 23(1-2): 1–18.

Otte M and Frazzoli E (2016) RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research* 35(7): 797–822.

Pan J, Chitta S and Manocha D (2012) FCL: A general purpose library for collision and proximity queries. In: *2012 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 3859–3866.

Pan J and Manocha D (2012) Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research* 31(2): 187–200.

Plaku E, Bekris KE, Chen BY, Ladd AM and Kavraki LE (2005) Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics* 21(4): 597–608.

Plaku E, Kavraki LE and Vardi MY (2009) Hybrid systems: from verification to falsification by combining motion planning and discrete search. *Formal Methods in System Design* 34(2): 157–182.

Prajna S and Jadbabaie A (2004) Safety verification of hybrid systems using barrier certificates. In: *International Workshop on Hybrid Systems: Computation and Control*. Springer, pp. 477–492.

Rodrıguez I, Nottensteiner K, Leidner D, Kaßecker M, Stulp F and Albu-Schäffer A (2019) Iteratively refined feasibility checks in robotic assembly sequence planning. *IEEE Robotics and Automation Letters* 4(2): 1416–1423. DOI:10.1109/LRA.2019.2895845.

Shkolnik A and Tedrake R (2011) Sample-based planning with volumes in configuration space. *arXiv preprint arXiv:1109.3145* .

Siméon T, Laumond JP and Nissoux C (2000) Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics* 14(6): 477–493.

Srivastava S, Fang E, Riano L, Chitnis R, Russell S and Abbeel P (2014) Combined task and motion planning through an extensible planner-independent interface layer. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 639–646.

Stilman M and Kuffner JJ (2005) Navigation among movable obstacles: Real-time reasoning in complex environments.

International Journal of Humanoid Robotics 2(04): 479–503.

Stilman M and Kuffner JJ (2008) Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research* 27(11-12): 1295–1307.

Thomason W and Knepper RA (2019) A unified sampling-based approach to integrated task and motion planning. In: *International Symposium on Robotics Research*.

Toussaint M (2015) Logic-geometric programming: An optimization-based approach to combined task and motion planning. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Varava A, Carvalho JF, Pokorny FT and Kragic D (2020) Caging and path non-existence: a deterministic sampling-based verification algorithm. In: *Robotics Research*. Springer, pp. 589–604.

Vega-Brown W and Roy N (2020) Asymptotically optimal planning under piecewise-analytic constraints. In: *Algorithmic Foundations of Robotics XII*. Springer, pp. 528–543.

Wang C and Meng MQH (2016) Variant step size rrt: An efficient path planner for uav in complex environments. In: *IEEE International Conference on Real-time Computing and Robotics*. pp. 555–560.

Wells A, Dantam NT, Shrivastava A and Kavraki LE (2019) Learning feasibility for task and motion planning in tabletop environments. *IEEE Robotics & Automation Magazine* 4(2): 1255–1262.

Wilfong G (1991) Motion planning in the presence of movable obstacles. *Annals of Mathematics and Artificial Intelligence* 3(1): 131–150.

Yalciner IF, Nouman A, Patoglu V and Erdem E (2017) Hybrid conditional planning using answer set programming. *Theory and Practice of Logic Programming* 17(5-6): 1027–1047. DOI: 10.1017/S1471068417000321.

Zhang L, Kim YJ and Manocha D (2007) A hybrid approach for complete motion planning. In: *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 7–14.

Zhang L, Kim YJ and Manocha D (2008) A simple path non-existence algorithm using c-obstacle query. In: *Algorithmic Foundation of Robotics VII*. Springer, pp. 269–284.