Robotics: Science and Systems 2021
Held Virtually, July 12–16, 2021

Learning Proofs of Motion Planning Infeasibility

Sihui Li
Department of Computer Science
Colorado School of Mines
Golden, Colorado 80401

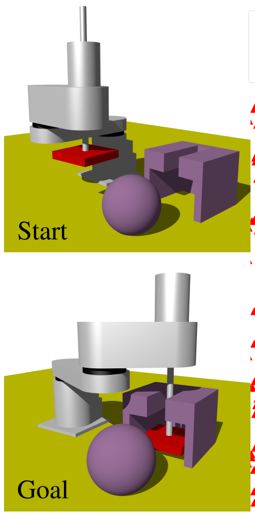
Neil T. Dantam
Department of Computer Science
Colorado School of Mines
Golden, Colorado 80401

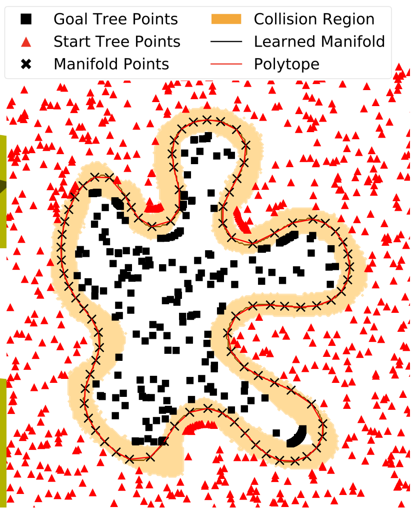
***Abstract*—We present a learning-based approach to prove infeasibility of kinematic motion planning problems. Sampling-based motion planners are effective in high-dimensional spaces but are only probabilistically complete. Consequently, these planners cannot provide a definite answer if no plan exists, which is important for high-level scenarios, such as task-motion planning. We propose a combination of bidirectional sampling-based planning (such as RRT-connect) and machine learning to construct an infeasibility proof alongside the two search trees. 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 learning hyper-parameters and robustness of configuration space optimization, the output is either an infeasibility proof or a motion plan. We demonstrate proof construction for 3-DOF and 4-DOF manipulators and show improvement over a previous algorithm.**

I. Introduction

Completeness is an important property for motion planning. A complete planner will always terminate with a plan when one exists or will report non-existence if no plan exists. Higher-level planning problems, such as rearrangement planning [1], [2] and task and motion planning [3], would benefit from complete motion planning. However, obtaining complete solutions in high-dimensional motion planning is an ongoing challenge, and significant effort has been devoted to working around the lack of full completeness [4]–[8].

Prior work has partially addressed the challenge of complete motion planning. Complete or resolution-complete methods such as cell decomposition [9], [10] are effective in low-dimensional spaces, but decomposing a high-dimensional configuration space is too expensive. On the other hand, sampling-based motion planners [11], [12] are widely used for high-dimensional spaces but are only *probabilistically complete*. If a plan with positive clearance exists, a probabilistically complete planner will find it in the limit (given enough time), but if no plan exists, the planner will not terminate (or will run until a timeout) [13]. Previously, we proposed a general approach to find motion planning infeasibility proofs by constructing a polytope in the configuration space obstacle region to separate the start and goal [14]. However, combinatorial steps to construct the polytope posed challenges to practically scale to high-dimensional manipulators. We address this challenge with a new, learning-based approach.





(a) 4 DoF scenes. (b) Learned manifold for a 2D example.

Fig. 1: (a) An infeasible motion planning problem for a four degree-of-freedom SCARA arm. (b) The learned proof manifold and the polytope constructed from it, separating the start and goal for a 2D scene.

We propose a novel approach to learn proofs of motion planning infeasibility and demonstrate better empirical scalability than prior methods for proving motion planning infeasibility. Our approach integrates learning and bidirectional sampling-based motion planning. We consider kinematic motion planning problems without differential constraints. The key insight is that the start and goal trees of the bidirectional search form two separate classes in the configuration space. We learn a manifold to separate these two classes (see Sec. IV-A). Then, we sample points on the manifold (see Sec. IV-B) and construct a closed polytope to approximate the manifold using a tangential Delaunay complex algorithm on the sampled manifold points (see Sec. IV-C). We prove that the polytope separates the start and goal by checking that each facet of the closed polytope is in the obstacle region (see Sec. IV-D). Figure 1b illustrates a trained manifold and the polytope constructed from it in a 2D configuration space. The existence of such a manifold or polytope in the obstacle region means the start and the goal are in separate components of the free configuration space. Thus, the manifold or polytope is an infeasibility proof.

Under certain assumptions, the result of this approach is either a motion plan, when one exists, or an infeasibility proof in the form of the separating manifold or polytope. First, we assume a kinematic motion planning problem where infeasibility is caused only by the obstacle region rather than the case of differential constraints. Second, we assume hyper-parameters are properly chosen to incrementally fit the SVM that will result in a closed and continuous manifold and a successful triangulation of the manifold (see Sec. IV-A and Sec. IV-C). Third, we assume the ability to sample points on the manifold and find penetration depths in configuration space; we present empirically robust, optimization-based approaches in (1) to sample points on the manifold and (2) to find such penetration depths for serial manipulators and workspace (Cartesian) obstacles. When these assumptions are satisfied, our algorithm results in a complete planner.

We demonstrate this approach on up-to four-dimensional configuration spaces of serial robot manipulators (see Figure 1a and Figure 7b). 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.

II. Related Work

Sampling-based motion planning is an efficient and widely-used approach for high-dimensional motion planning [11]–[13], [15]–[17]. However, these approaches are only probabilistically complete. Recent developments in tools for high-dimensional computational geometry have made complete motion planning more feasible [18]–[20].

Some previous work addressed motion planning infeasibility proofs for single objects. [21] proves 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. [22] considers 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. In [23], the authors 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. In [9] and [10], the authors combine cell decomposition with a probabilistic roadmap (PRM). The algorithms proposed in [9], [10] are resolution-complete 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 [24], [25]. Using low-dispersion sampling strategies, if such a planner does not find a plan, then either no solution exists or a solution exists only through some narrow passage. However, the 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 [26] and sparsity [27] 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 can usually cover a high percentage of the free space. If no plan is found when the algorithm terminates, the problem may be considered to be infeasible [28]. 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.

To summarize, previous works on infeasibility proofs either limit analysis to single objects, require decomposition of the entire configuration space, or provide semi-definite plan non-existence guarantee. 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 [14] 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 [14], this paper uses machine learning to generate an initial manifold, offering better scalability to higher dimensions.

III. Problem Description

This work finds infeasibility proofs for kinematic motion planning problems. A motion planning problem [29] consists of a configuration space $\mathcal{C}$ of dimension n, a start configuration $\boldsymbol{q}_{\text{start}}$, and a goal configuration $\boldsymbol{q}_{\text{goal}}$. The configuration space $\mathcal{C}$ is the union of the disjoint obstacle region $\mathcal{C}_{\text{obs}}$ and free space $\mathcal{C}_{\text{free}}$. Both $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are in $\mathcal{C}_{\text{free}}$. When a feasible plan exists, the output is a plan σ such that $\sigma[0,1] \in \mathcal{C}_{\text{free}}$, $\sigma[0] = \boldsymbol{q}_{\text{start}}$, $\sigma[1] = \boldsymbol{q}_{\text{goal}}$. When there is no feasible plan, the output is a proof of infeasibility.

We consider infeasibility proofs in the form of a closed manifold $\mathcal{M}$ which lies in the obstacle region and which separates the start $\boldsymbol{q}_{\text{start}}$ and the goal $\boldsymbol{q}_{\text{goal}}$. We define $\mathcal{M}$ as a level set with $f(\boldsymbol{q}) = 0$ where f is a continuous function in $\mathcal{C}$. To separate $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$, $\mathcal{M}$ must be closed and f must be positive for one and negative for the other.

Definition 1 (Infeasibility Proof)**.** *An infeasibility proof is a closed manifold $\mathcal{M}$, where,*

$$\begin{aligned} &\mathcal{M} = \{\boldsymbol{q} \mid f(\boldsymbol{q}) = 0 \wedge \boldsymbol{q} \in \mathcal{C}_{\text{obs}}\}, \\ &\textit{s.t.} \quad f(\boldsymbol{q}_{\text{start}}) f(\boldsymbol{q}_{\text{goal}}) < 0 \ . \end{aligned}$$

Proposition 1. *No plan σ exists if and only if there exists a manifold $\mathcal{M}$ according to Definition 1.*

Proof: First, we prove that if $\mathcal{M}$ exists, no plan σ exists. $\mathcal{M}$ is entirely in $\mathcal{C}_{\text{obs}}$, $f(\boldsymbol{q}_{\text{start}}) < 0$ (> 0) and $f(\boldsymbol{q}_{\text{goal}}) > 0$ (< 0), meaning $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are on different sides of $\mathcal{M}$. Thus, $\mathcal{M}$ separates $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ into disconnected components of $\mathcal{C}_{\text{free}}$, so no plan σ exists.

Second, we prove by contradiction that if no plan σ exists, $\mathcal{M}$ must exist. Assume no plan σ exists and no $\mathcal{M}$ exists. The $\mathcal{C}_{\text{free}}$ components of $\boldsymbol{q}_{\text{start}}$ and $\boldsymbol{q}_{\text{goal}}$ are not separated by $\mathcal{C}_{\text{obs}}$, thus there would exist a plan σ. Contradiction. ■

The configuration space boundaries (e.g. joint limits) are a special case. We treat the boundaries as fixed obstacle regions with ϵ thickness to ensure the existence of $\mathcal{M}$ in general.

We note that the proof manifold in Definition 1 may be either a smooth or piecewise manifold in the configuration space. In the rest of the paper, we will use the term "manifold" to refer to the learned function from the kernel-SVM, and "polytope" to refer to its triangulation.

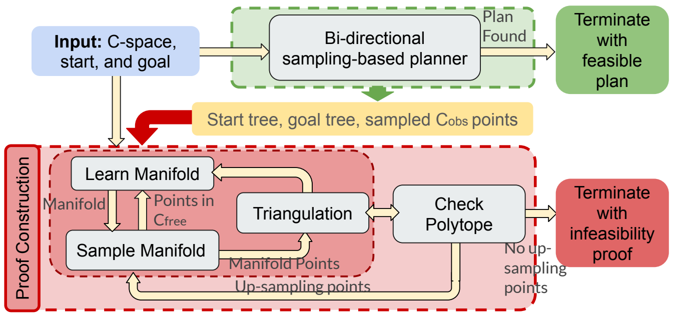
We assume, for this work, motion planing in which infeasibility is caused only by the obstacle region, and we do not consider differential constraints. That is, we do not consider steering functions [30], dynamics [31], or implicit constraints [32]. Though this assumption is valid for many manipulation 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 (1) and (2) 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 Sec. V).

IV. Algorithm

Our algorithm will generate either a motion plan or an infeasibility proof in the form of a polytope separating the start and goal configurations (Definition 1). Figure 2 illustrates the structure of our algorithm, and algorithm 1 describes the proof construction in pseudocode. We combine bidirectional sampling-based motion planning and machine learning of the separating manifold. Our algorithm runs the bidirectional search and manifold construction in parallel, terminating with either a plan or an infeasibility proof.

Our infeasibility proof construction follows two broad steps. First, we construct a potential infeasibility proof that separates the start and the goal via learning and manifold triangulation. Then, we check that the potential infeasibility proof is entirely



Input: C-space, start, and goal
Bi-directional sampling-based planner
Plan Found
Terminate with feasible plan
Start tree, goal tree, sampled Cobs points
Proof Construction
Learn Manifold
Manifold
Points in Cfree
Triangulation
Sample Manifold
Manifold Points
Check Polytope
Up-sampling points
No up-sampling points
Terminate with infeasibility proof

Fig. 2: An algorithm overview. The red block is showing the infeasibility proof construction, which runs in parallel with a bidirectional sampling-based planner.

in the obstacle region. If both steps are successful, we have an infeasibility proof as defined in Definition 1. If not, we up-sample and go back to the first step.

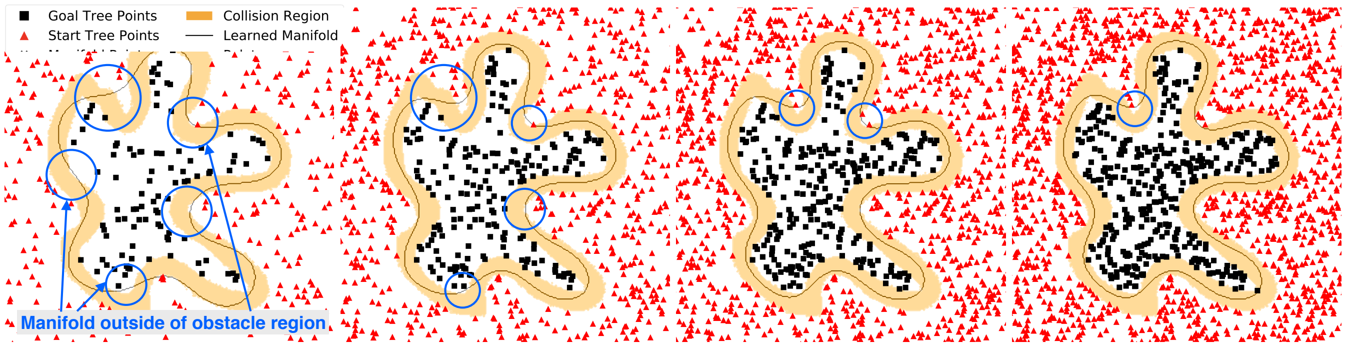
The input to the first broad step consists of the start and goal trees from the bidirectional planner and a set of points sampled in the obstacle region by the planner. First, we learn a manifold that separates the two trees into two different classes (see Sec. IV-A). Then, we sample points on the manifold by projecting the obstacle region points onto the manifold (see Sec. IV-B). Lastly, we triangulate the manifold using the sampled manifold points with tangential Delaunay complex (see Sec. IV-C). The output of this first broad step is a potential infeasibility proof in the form of a polytope.

The input to the second broad step is the polytope from the first step. We check that the polytope is entirely in $\mathcal{C}_{\text{obs}}$ by testing each facet (see Sec. IV-D). If part of a facet is outside the obstacle region, we may need to check a smaller cell that more-closely approximates the manifold, so we attempt to up-sample the manifold and recreate the polytope. If up-sampling fails, it means the manifold we learned in the first broad step is not entirely in $\mathcal{C}_{\text{obs}}$, so we repeat the first step with updated inputs. If all facets are in the obstacle region, then the polytope is a valid infeasibility proof.

A. Learn a Manifold

First, we learn a manifold. We treat the points in the two search trees from the bidirectional planner as two classes; then, we learn a classifier (line 15). The particular classifier we use is a support vector machine (SVM) with Radial Basis Function (RBF) kernel. Other classifiers may fit within the framework of our approach, but the SVM has three notable benefits. First, the SVM directly provides a closed-form, differentiable function for the separating manifold, which we use to check if the manifold is in the obstacle region. Second, the SVM has fewer hyper-parameters than other methods such as Neural Networks. Finally, the SVM creates a margin to maximize separation between classes. In our use case, this maximum margin increases penetration depth of the manifold points in the obstacle region, which helps us check facets in Sec. IV-D.

Unlike typical machine learning applications, over-fitting is



(a) 1000 sampling points (b) 2000 sampling points (c) 3000 sampling points (d) 4000 sampling points

Fig. 3: We treat the points in the two search trees as two classes and learn a classifier (the separating manifold). (a)-(d) Learned manifold fits into the obstacle region as more points are sampled (1000 points - 4000 points), the blue circles are showing parts of the manifold outside of the obstacle region. The same applied to 4 DoF robot arms in Sec. V.

Algorithm 1: Construct Infeasibility Proof

```
function samp-mf($f, P_{\text{start}}, P_{\text{goal}}, P_{\text{obs}}$) is
    $P_{\text{m}} \leftarrow \emptyset$; flag $\leftarrow$ false;
    foreach $\mathbf{q}_{\text{obs}} \in P_{\text{obs}}$ do
        $\mathbf{q}_{\text{m}} \leftarrow$ find-closest-point($\mathbf{q}_{\text{obs}}, f$);
        if $\mathbf{q}_{\text{m}} \in \mathcal{C}_{\text{free}}$ then
            add-point($P_{\text{start}}, P_{\text{goal}}, \mathbf{q}_{\text{m}}$);
            flag $\leftarrow$ true;
        else $P_{\text{m}} \leftarrow P_{\text{m}} \cup \{\mathbf{q}_{\text{m}}\}$;
    if flag then return $\emptyset$;
    else return $P_{\text{m}}$;
$P_{\text{start}}, P_{\text{goal}}, P_{\text{obs}} = \emptyset$;
repeat
    repeat /* Sec. IV-A and Sec. IV-B */
        obtain-points($P_{\text{start}}, P_{\text{goal}}, P_{\text{obs}}$);
        $f \leftarrow$ train-manifold($P_{\text{start}}, P_{\text{goal}}$);
        $P_{\text{m}} \leftarrow$ samp-mf($f, P_{\text{start}}, P_{\text{goal}}, P_{\text{obs}}$);
    until $P_{\text{m}} \neq \emptyset$;
    repeat /* Sec. IV-C and Sec. IV-D */
        $P_{\text{up}} \leftarrow \emptyset$;
        $P_{\text{subm}} \leftarrow$ subsampling($P_{\text{m}}$);
        $F \leftarrow$ tangential-complex($P_{\text{subm}}$);
        if $F = \emptyset$ then // Triangulation fails
            $P_{\text{up}} = \{0\}$;
            break;
        foreach $\mathbf{f}_{\text{t}} \in F$ do // Check polytope
            $\mathbf{q}_{\text{up}} \leftarrow$ check-facet($\mathbf{f}_{\text{t}}$);
            $P_{\text{up}} \leftarrow P_{\text{up}} \cup \{\mathbf{q}_{\text{up}}\}$;
        if $P_{\text{up}} \neq \emptyset$ then // Upsample Manifold
            $P_{\text{upm}} \leftarrow$ samp-mf($f, P_{\text{start}}, P_{\text{goal}}, P_{\text{up}}$);
            if $P_{\text{upm}} = \emptyset$ then $P_{\text{m}} \leftarrow \emptyset$;
            else $P_{\text{m}} \leftarrow P_{\text{m}} \cup P_{\text{upm}}$;
    until $(P_{\text{up}} = \emptyset) \vee (P_{\text{m}} = \emptyset)$;
until $P_{\text{up}} = \emptyset$;
return $F$;
```

actually desired in our case since we want to learn a manifold that separates the start and goal trees without exception. When training the manifold, we increase the over-fitting parameter γ in the RBF kernel by a small amount of $\Delta\gamma$ each time until the training set's score is 1. This step ensures that the manifold does not over-fit too much and become discontinuous (see Appendix A for a discontinuous example). The γ parameter and the small increasing value $\Delta\gamma$ are hyper-parameters of the algorithm. We use 1.0 and 0.1 in the experiments, which are robust for the tested robot scenes. We use a fixed regularization parameter for all the training; we do not need to adjust the regularization parameter since we do not allow any misclassifications.

According to Definition 1, the 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 start tree and the goal tree. Since the obstacle region is closed, it has an outer boundary and an inner boundary. We first consider the case of two free space components, where the outer boundary is in contact with the region of the start (goal) tree, and the inner boundary is in contact with the region of the goal (start) tree. For the case of multiple free space components, we must grow additional trees in the additional components similar to a PRM, though our current implementation addresses only the case of two components. If we have enough points sampled close to both boundaries, the points of the trees as support vectors will force the manifold into the obstacle region. In Figure 3, 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. 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 will choose the largest effecting range (with the fixed incremental value $\Delta\gamma$ 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, using the training process that incrementally over-fits with a small enough over-fitting incremental value, the resulting manifold will eventually be closed, continuous, and contained in $\mathcal{C}_{\text{obs}}$.

Manifold triangulation using tangential Delaunay complexes (Sec. IV-C) requires the manifold to be a closed, continuous, and differentiable submanifold of n-dimensional Euclidean space [33]. In our case, the learned manifold from RBF kernel-SVM is differentiable since the resulting manifold function is a combination of Gaussian functions. Since we do not know how many sample points are necessary to ensure manifold closure and continuity, the triangulation step also serves as a validation step. If the triangulation is successful, then the manifold must be closed and continuous. If the triangulation step is not successful, we need to retrain the manifold with more sampled points from the two trees.

B. Sample Manifold Points

After learning a manifold, we use the obstacle region points P_{obs} to sample points on the manifold, which we will then use to construct the polytope in the next step. In most sampling-based methods, P_{obs} are discarded, but in our algorithm, we save all the points sampled in obstacle region.

For each point in P_{obs}, we find the closest point on the manifold (line 4) which is the solution of the following nonlinear constrained optimization problem,

$$\begin{aligned} \min_{\mathbf{q}_{\text{m}}} \quad & \text{dist}(\mathbf{q}_{\text{obs}}, \mathbf{q}_{\text{m}}) \\ \text{s.t.} \quad & f(\mathbf{q}_{\text{m}}) = 0 \ , \end{aligned} \tag{1}$$

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 defined in Definition 1, which is the training result of the previous step. Solving this optimization problem for every point in P_{obs} produces a set of points on the manifold. We use P_{obs} to sample manifold points because they are more likely to exist closer to the manifold since the manifold is largely in $\mathcal{C}_{\text{obs}}$, thus solving the optimization problem faster. If solving this optimization problem fails for a point, we discard that point. Though (1) may not be robustly solvable for all possible configuration spaces, we are able to robustly solve this optimization problem for the experimental scenarios involving robot manipulators in Sec. V.

Because the manifold must be entirely in $\mathcal{C}_{\text{obs}}$, we check if any sampled manifold point is in $\mathcal{C}_{\text{free}}$ at this stage. If we find a 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. Before retraining, we try to add the point in $\mathcal{C}_{\text{free}}$ to either the start tree or the goal tree by interpolating a straight line between the point and the closest point to it on either of the two trees (line 6). Note that our implementation adds the point to a copy of the trees for the infeasibility proof but not the start tree and goal tree used by the bidirectional planner (e.g., RRT-connect), since we want this algorithm to work with any bidirectional motion planner without the need to modify the underlying planner. Now that we have a set of points on the manifold, the next step uses these points to create a polytope that approximates the manifold.

C. Manifold Triangulation

Since proving a manifold is in $\mathcal{C}_{\text{obs}}$ directly is difficult, we construct a polytope from the sampled manifold points using tangential Delaunay complexes [18], [33] and then prove that the polytope is in $\mathcal{C}_{\text{obs}}$. Tangential Delaunay complexes can be used to reconstruct a triangulation of a manifold given a set of points on the manifold. In [18], [33], the authors provide an algorithm to construct the tangential Delaunay complexes for triangulation of manifolds, which is implemented in [34]. We apply this algorithm to the sampled manifold points from Sec. IV-B to triangulate the SVM manifold. The triangulation of the manifold forms a polytope, which we use in later steps.

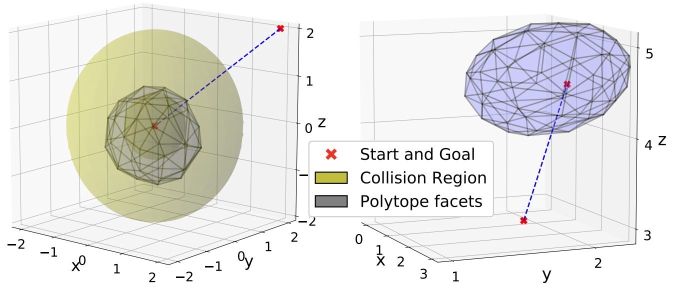
Here, we focus on the key requirements and results of the triangulation algorithm; please see [33] (chapter 7 and chapter 8) for a more complete explanation. To construct the triangulation successfully, the algorithm has several requirements. As stated in Sec. IV-A, the first requirement is that the manifold should be closed, continuous and differentiable, which is already satisfied. Secondly, the sampling points must cover the entire manifold and distribute evenly on the manifold. Stated precisely, the sampled set of points must be an $(\epsilon, \eta)-$net of the manifold.

Definition 2. *A finite point set P is an $(\epsilon, \eta)-$net of $\mathcal{M}$, iff*

- *(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 $p, q \in P$, $\|p - q\| > \epsilon\eta$.*

In algorithm 1, we ensure the $(\epsilon, \eta)-$net requirement by applying a subsampling process (line 20). 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 ensures the resulting points set is an $(\epsilon, \eta)-$net of $\mathcal{M}$ for some ϵ and η. If we have more manifold points, ϵ would be smaller. If we use smaller minimum distance in the second subsampling step, ϵ would also be smaller. We do not need to calculate ϵ and η exactly.

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 large and smaller $d_{\min}$ if $\mathcal{C}_{\text{obs}}$ separating the goal or start is small.



(a) Simple c-space polytope (b) 3 DoF Jaco arm's polytope.

Fig. 4: Example 3D polytopes constructed using tangential Delaunay complexes. (a) The obstacle region is between two balls of radius 2.0 and 0.5. (b) The polytope constructed in the 3 DoF Jaco arm experiment (no obstacle region shown).

In general, this value should be small enough to triangulate any curves on the manifold. This parameter is empirical (see Sec. V for the specific values for each robot scenes). A poor $d_{\min}$ may cause triangulation to fail. Automatic tuning of this hyper-parameter is an important future work.

[33] proves 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 [18].

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 1b shows a 2D polytope). If not, we go back to retrain the manifold and obtain more manifold points (line 22) for a small ϵ. After triangulation, we have completed the first broad step. The resulting polytope is a potential infeasibility proof. Figure 4 shows polytopes constructed using tangential Delaunay complexes in a predefined configuration space and the configuration space of a 3 DoF Jaco arm.

D. Check Polytope and Up-sampling

For the polytope from previous steps to be an a valid infeasibility proof, it must be entirely in $\mathcal{C}_{\text{obs}}$. We prove this by checking each facet of the polytope (line 26).

We check each facet by iteratively checking smaller cubes that enclose the facet. Starting from the smallest $(n-1)$-cube that encloses the facet, if the configuration space penetration depth of the center of the cube is larger than the center's distance to the cube's vertices, then the cube is entirely in $\mathcal{C}_{\text{obs}}$, so the enclosed facet is entirely in $\mathcal{C}_{\text{obs}}$. If this check fails, we decompose the cube into $2^{(n-1)}$ smaller cubes and go through the same checking process. We only need to check cubes that intersects with the facet. For each cube, we also check whether the center of the cube is inside the facet and in $\mathcal{C}_{\text{free}}$. If this is the case, it means part of the facet is in $\mathcal{C}_{\text{free}}$,

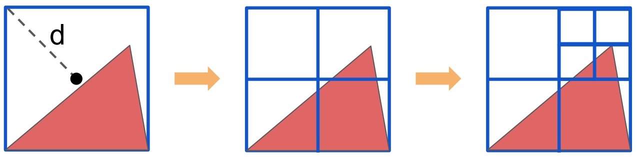


Fig. 5: This process ensures each facet is in the obstacle region by recursively checking smaller (n-1)-cubes.

then the center becomes an up-sampling point ($\mathbf{q}_{\text{up}}$ on line 26). Figure 5 shows the process of checking a 2D facet.

We terminate checking the current facet when either the entire facet is checked or we find up-sampling points. If there are upsampling points returned from checking facets (line 28), we use these upsampling points to sample more manifold points. If sampling manifold points is successful, we add these new manifold points to the existing set of manifold points, re-triangulate the manifold and re-check the new polytope. If sampling manifold points fails, we must retrain the manifold, so we go back to rerun the first broad step.

In this checking process, finding the *configuration space* penetration depth (PD) is a key sub-routine. Existing libraries compute [35] *Cartesian space* penetration depth. We find the configuration space penetration depth of point $\boldsymbol{q}$ by solving the following nonlinear optimization problem,

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \text{dist}(\boldsymbol{q}_o, \boldsymbol{q}) \\ \text{s.t.} \quad & \text{Penetration-Depth}(x_l(\boldsymbol{q}), x_o) \leq 0 \\ & l \in \{1, ..., n_{\text{link}}\}, o \in \{1, ..., n_{\text{obs}}\}\ , \end{aligned} \tag{2}$$

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 colliding with each other, Cartesian space PD between these two frames is positive, otherwise, it 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 6). 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 is also considered to be in $\mathcal{C}_{\text{obs}}$, and its PD is the distance to the closest point in $\mathcal{C}_{\text{free}}$.

If this optimization problem is not solvable for some points, we can discard the points and keep checking smaller cubes; however, we did not observe unsolved optimizations in the experiments, meaning this method is an empirically robust way to compute penetration depth.

E. Algorithm Summary

Our algorithm runs the bidirectional search and proof construction in parallel. Proof construction will use the latest available search trees and obstacle region points to learn and sample a manifold. We apply the tangential Delaunay complex algorithm to the subsampled manifold point to construct a polytope. Then, we check each facet of the polytope and collect

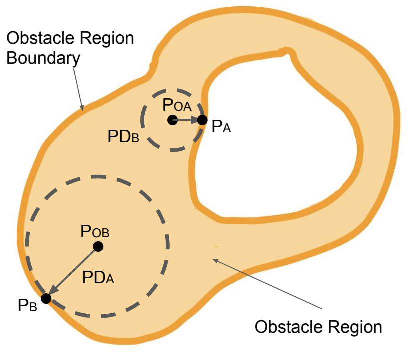
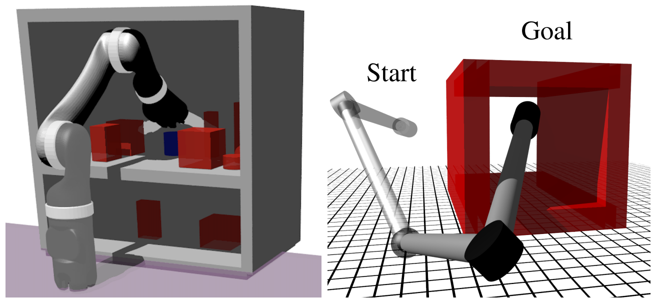


Fig. 6: Configuration space penetration depth of two points P_{OA} and P_{OB} in $\mathcal{C}_{\text{obs}}$. P_A and P_B are their corresponding closest points that satisfy the constraints.



(a) 3 DoF robot scene. (b) 4 DoF robot scene.

Fig. 7: Experiment scenes

up-sampling points. We use these up-sampling points to sample more manifold points, re-triangulate or retrain the manifold if necessary, until there are no up-sampling points. When there are no more up-sampling points, all facets of the closed polytope are in $\mathcal{C}_{\text{obs}}$, which means the entire polytope is in $\mathcal{C}_{\text{obs}}$ and thus constitutes an infeasibility proof according to Definition 1. To summarize, under the assumptions on configuration space optimization (Equation 1 and Equation 2) and good choices of hyper-parameters (γ, $\Delta\gamma$, and $d_{\min}$), the algorithm terminates with a plan or an infeasibility proof. Here we acknowledge that termination of our algorithm depends on good choices hyper-parameters. In Sec. VI, we discuss the three hyper-parameters in more detail.

V. Experiments

We run three sets of experiments to evaluate the performance of the algorithm. The first experiment uses the same Jaco arm scenario as [14] to directly compare with our previous algorithm. The next two experiments use two different four DoF arms—which could not be solved by [14]—to demonstrate the improved scalability.

In our experiments, we adapt RRT-Connect [15] in OMPL [16] to run in parallel with our infeasibility proof construction. We solve the nonlinear optimization problems using sequential least-squares quadratic programming (SLSQP) [36]. We construct the polytope using the tangential complex module in GUDHI package [34]. We train the manifold using Scikit-learn [37]. We check collisions and penetration depth using the Flexible Collision Library [35], and we model robot kinematics using Amino [38]. We determine ground truth for plan non-existence to high-confidence for a scene by running RRT-connect continuously for greater than 20 minutes.

A. Three DoF Robot Experiment

To compare with our previous approach, we apply the current method to the Jaco arm experiment in [14]. We test on three different scenes where no path exists. The goal in these scenes is to grasp the blue block at the back of the shelf from a position outside of the shelf (see Figure 7a).

Table I shows average time and profiling results of the current and prior algorithms. Compared with the previous method, the current algorithm's average running time is three times faster. Standard deviations of running time are also smaller with respect to the mean, indicating that the new algorithm has more consistent performance. In the previous algorithm, the large variance mainly comes from the generation of facets. Random sampling influences the number of facets needed to form the polytope significantly, which range from 53 to 332 in the 36 trials. In the current algorithm, the variance is much smaller because we learn the manifold to guide construction of the polytope, which reduces the impact of randomness.

When paths exist, the current algorithm is also better than the previous method. In the current algorithm, RRT-connect runs in parallel with the infeasibility proof construction. Assuming multiple cores to run both threads in parallel, proof construction will have less effect on wall clock running time when paths exist. With the previous algorithm, creating new facets was not separable from the sampling process, which increased running time due to the additional computation regardless of whether paths exist or not. A potential drawback of the current algorithm is that it only applies to bidirectional sampling-based planning, whereas the previous algorithm applies also to unidirectional sampling-based planning.

B. Shoulder-Elbow Robot Experiment

We apply our approach to the four DoF robot arm scene in Figure 7b. The robot has a shoulder joint (three DoF spherical) and an elbow joint (one DoF revolute). The goal is to reach inside the red box. We run 24 trials on this scene. Infeasibility proofs for this four DoF arm take less than 4 minutes on average, as shown in Table II. Table II also shows the size of P_{obs}, the number of polytope facets and the number of polytope vertices. In the experiments, we use a subsampling minimum distance of 0.002.

C. SCARA Robot Experiment

We apply the algorithm to the SCARA arm and the scene in Figure 1a. The SCARA arm has three co-planar revolute joints and one prismatic joint. The goal in this scene is to reach inside the purple box with the red block attached at the end-effector. Infeasibility proofs for this SCARA arm take about 7 minutes on average, as shown in Table III. In the experiments, we use a subsampling minimum distance of 0.01. Compared to the previous experiment, we see that the complexity of the workspace influences running time, since we need more points to train and sample the manifold.

D. Feasible Plan Experiments

For the above three 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 reachable. For the 4DOF 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 arm to pass. We run these scenes on RRT-connect with and without the infeasiblity proof for 24 trails each. Table IV shows the comparison.

When there is a feasible plan, running RRT-connect 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, running RRT-connect with infeasibility proof construction introduces more overhead, which is mainly caused by saving all the sampling points. A better-optimized implementation may reduce this overhead. The current implementation uses an interpreted language (Python) to save the additional data and construct the infeasibility proof, while RRT-connect is implemented in C++ [16].

We note that in all the trials we ran (84 trials of infeasible scenes and 72 trials of feasible scenes), the algorithm correctly found either the infeasibility proof or the plan.

Previous Algorithm [14] Runtime and Profiling (s)				
	Total	*LCSP*	*Colli-Check*	*PD*
Mean	457.50	46.62	51.24	326.79
STD	396.24	41.77	146.05	172.96
Current Algorithm Runtime and Profiling (s)				
	Total	*TC*	*samp-mf*	*PD*
Mean	177.36	15.15	51.69	89.23
STD	71.35	30.65	30.65	20.88

TABLE I: Runtime comparison with previous algorithm, averaged over 3 scenes and 12 trials each scene. Current algorithm runs about 3 times faster.

4 DoF Shoulder-Elbow Robot Experimental Results				
	Total (s)	*TC (s)*	*samp-mf (s)*	*PD (s)*
Mean	230.82	67.19	51.97	94.40
STD	90.95	47.82	39.36	27.89
		# of P_{obs}	*# of Facets*	*# of Vertices*
Mean		16 340.42	8188.38	1281.84
STD		4930.06	2353.53	361.61

TABLE II: Experimental results for 4 DoF shoulder-elbow robot, averaged over 24 trials. "TC" is for triangulation with tangential complex, "samp-mp" is for sampling of manifold points, "PD" is for calculating penetration depth.

SCARA Arm Experimental Results				
	Total (s)	*TC (s)*	*samp-mf (s)*	*PD (s)*
Mean	433.00	25.01	317.47	69.80
STD	220.43	29.97	184.78	10.95
		# of P_{obs}	*# of Facets*	*# of Vertices*
Mean		16 747.86	3094.17	503.80
STD		3104.73	365.65	57.46

TABLE III: Experimental results for SCARA arm, averaged over 24 trials. "TC" is for triangulation with tangential complex, "samp-mp" is for sampling of manifold points, "PD" is for calculating penetration depth.

Plan Feasible Experiments mean/std (s)			
	Jaco	*4 DoF*	*SCARA*
RRT-connect	4.21/1.69	0.025/0.010	42.20/22.43
RRT-connect w/ IF	5.18/2.56	0.109/0.027	84.34/48.30

TABLE IV: Experimental results for plan feasible experiments, averaged over 24 trials, running RRT-connect only vs. running RRT-connect with infeasibility proof.

VI. Discussion and Future Work

Our ongoing goal is to extend the infeasibility proof construction to higher dimensions. A current bottleneck is the triangulation of the manifold. Since calculating the tangential complex [34] is exponential in the dimension of the manifold space, we anticipate that it will take a larger percentage of runtime in higher dimensions. The other two time-consuming processes—sampling the manifold and calculating PD—apply nonlinear optimization, which we expect to scale well. Another issue is with random sampling of P_{obs}. An evenly distributed P_{obs} would potentially reduce the number of samples needed. Exploring deterministic sampling is a possible future direction.

There are three hyper-parameters in our algorithm, γ, $\Delta\gamma$, and $d_{\min}$. The first two hyper-parameters are set for learning the manifold. γ can be a small value since we will increase γ by $\Delta\gamma$ if the manifold does not fit all the training data. We use $\gamma = 1.0$ for all the experiments. $\Delta\gamma$ needs to be small enough so that the manifold would not go from under-fitting to having separate regions (see Appendix A) in one step's change. We use $\Delta\gamma = 0.1$ in all the experiments. $d_{\min}$ is a hyper-parameter in the subsampling process that controls the manifold triangulation step. $d_{\min}$ should be small enough to form triangulation at any small curves of the manifold. In the experiments, we choose small values of $d_{\min}$. Small $d_{\min}$ value will result large numbers of manifold points for triangulation after subsampling, which will produce more facets on the polytope and thus make triangulation slower.

Termination of our algorithm does depend upon the selected hyper-parameters. We acknowledge the algorithm's reliance on hyper-parameters, and sensitivity analyses and auto-tuning hyper-parameters is one area of future work. 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 it to fail [39].

Our current implementation uses only two search trees to learn the manifold. While we were able to construct proofs for the scenarios in Sec. V, general proof construction for multiple free space components requires creating and learning from multiple trees using a 1 vs. n classification.

Additionally, alternative approaches to check whether the learned manifold is in $\mathcal{C}_{\text{obs}}$ could benefit proof construction. In the experiments, we find that once we constructed a manifold with all *sampled* points in $\mathcal{C}_{\text{obs}}$, the later triangulation step rarely discovers parts of the manifold or polytope in $\mathcal{C}_{\text{free}}$. That is, after learning a manifold for the first time, we typically already have a manifold entirely in $\mathcal{C}_{\text{obs}}$. However, a large portion of runtime is expended to verify this fact.

VII. Conclusion

We have presented a novel method to learn proofs of motion planning infeasibility, and we demonstrated this method for three and four DoF robot manipulators. Our approach learns a manifold and constructs a polytope from manifold points that separates the start and goal into different components of the free configuration space. Under the assumptions on the configuration space components, the ability to obtain robust solutions to the optimization problems, and the suitability of hyper-parameters, our planner is complete. This work improves scalability compared to our previous method [14] and is, to our knowledge, the first approach to construct infeasibility proofs for robot manipulators in greater than three dimensions. Extensions of this work to higher dimensions, more free space components, and generalization to steering functions [30], dynamics [31], and additional constraints [32] are promising directions for future research.

VIII. Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. IIS-1849348.

References

[1] J. Ota, "Rearrangement of multiple movable objects-integration of global and local planning methodology," in *International Conference on Robotics and Automation*, vol. 2. IEEE, 2004, pp. 1962–1967.

[2] O. Ben-Shahar and E. Rivlin, "Practical pushing planning for rearrangement tasks," *Transactions on Robotics and Automation*, vol. 14, no. 4, pp. 549–565, 1998.

[3] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[4] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[5] K. Hauser, "The minimum constraint removal problem with three robotics applications," *International Journal of Robotics Research*, vol. 33, no. 1, pp. 5–17, 2014.

[6] F. Lagriffoul and B. Andres, "Combining task and motion planning: A culprit detection problem," *International Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

[7] M. Stilman and J. J. Kuffner, "Navigation among movable obstacles: Real-time reasoning in complex environments," *International Journal of Humanoid Robotics (IJHR)*, vol. 2, no. 04, pp. 479–503, 2005.

[8] A. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, "Learning feasibility for task and motion planning in tabletop environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1255–1262, 2019.

[9] L. Zhang, Y. J. Kim, and D. Manocha, "A hybrid approach for complete motion planning," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2007, pp. 7–14.

[10] ——, "A simple path non-existence algorithm using c-obstacle query," in *Algorithmic Foundation of Robotics VII*. Springer, 2008, pp. 269–284.

[11] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Deptartment, Iowa State University, Tech. Rep. TR-98-11, October 1998.

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

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[14] S. Li and N. T. Dantam, "Towards general infeasibility proofs in motion planning (accepted)," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2020.

[15] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Intl. Conference on Robotics and Automation*, vol. 2. IEEE, 2000, pp. 995–1001.

[16] I. A. Şucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *Robotics & Automation Magazine (RAM)*, vol. 19, no. 4, pp. 72–82, 2012.

[17] A. Shkolnik and R. Tedrake, "Sample-based planning with volumes in configuration space," *arXiv preprint arXiv:1109.3145*, 2011.

[18] J.-D. Boissonnat and A. Ghosh, "Manifold reconstruction using tangential delaunay complexes," *Discrete & Computational Geometry*, vol. 51, no. 1, pp. 221–267, 2014.

[19] E. Fogel, D. Halperin, and R. Wein, *CGAL arrangements and their applications: A step-by-step guide*. Springer Science & Business Media, 2012, vol. 7.

[20] D. Halperin, O. Salzman, and M. Sharir, "Algorithmic motion planning," in *Handbook of Discrete and Computational Geometry*. Chapman and Hall/CRC, 2017, pp. 1311–1342.

[21] A. Varava, J. F. Carvalho, F. T. Pokorny, and D. Kragic, "Caging and path non-existence: a deterministic sampling-based verification algorithm," in *Robotics Research*. Springer, 2020, pp. 589–604.

[22] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen, "Disconnection proofs for motion planning," in *International Conference on Robotics and Automation*, vol. 2. IEEE, 2001, pp. 1765–1772.

[23] Z. McCarthy, T. Bretl, and S. Hutchinson, "Proving path non-existence using sampling and alpha shapes," in *International Conference on Robotics and Automation*. IEEE, 2012, pp. 2563–2569.

[24] L. Janson, B. Ichter, and M. Pavone, "Deterministic sampling-based motion planning: Optimality, complexity, and performance," *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 46–61, 2018.

[25] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang, "Quasi-randomized path planning," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 2. IEEE, 2001, pp. 1481–1487.

[26] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility-based probabilistic roadmaps for motion planning," *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[27] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 18–47, 2014.

[28] A. Orthey and M. Toussaint, "Sparse multilevel roadmaps for high-dimensional robotic motion planning."

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

[30] G. Lafferriere and H. Sussmann, "Motion planning for controllable systems without drift," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. IEEE Computer Society, 1991, pp. 1148–1149.

[31] B. Donald, P. Xavier, J. Canny, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.

[32] Z. Kingston, M. Moll, and L. E. Kavraki, "Exploring implicit spaces for constrained sampling-based planning," *The International Journal of Robotics Research*, vol. 38, no. 10-11, pp. 1151–1178, 2019.

[33] J.-D. Boissonnat, F. Chazal, and M. Yvinec, *Geometric and Topological Inference*. Cambridge University Press, 2018, cambridge Texts in Applied Mathematics. [Online]. Available: https://hal.inria.fr/hal-01615863

[34] C. Jamin, "Tangential complex," in *GUDHI User and Reference Manual*, 3.4.0 ed. GUDHI Editorial Board, 2020. [Online]. Available: https://gudhi.inria.fr/doc/3.4.0/group__tangential__complex.html

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

[36] D. Kraft, “A software package for sequential quadratic programming,” Institut für Dynamik der Flugsysteme, Oberpfaffenhofen, Tech. Rep. DFVLR-FB 88-28, July 1988.

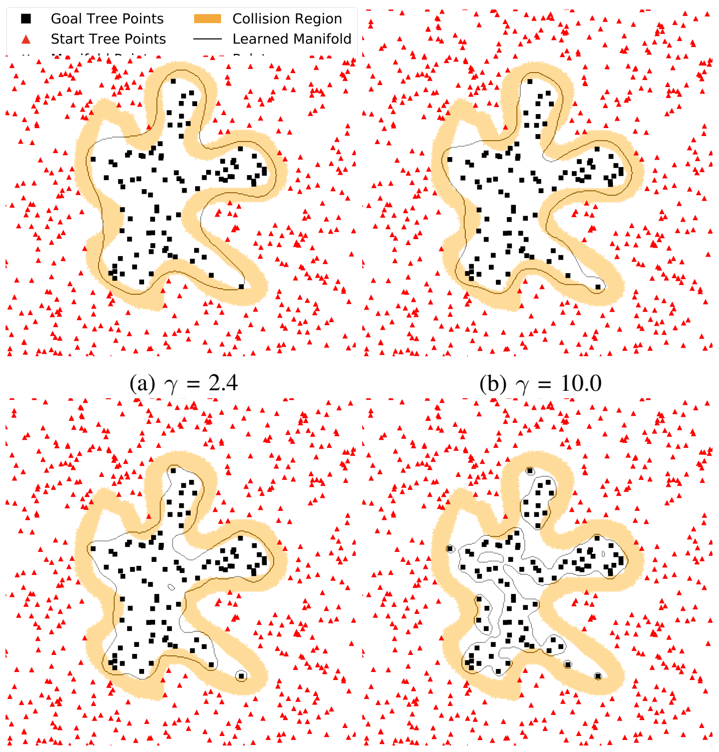
[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] N. T. Dantam, “Robust and efficient forward, differential, and inverse kinematics using dual quaternions,” *International Journal of Robotics Research*, 2020.

[39] C. Wang and M. Q.-H. Meng, “Variant step size rrt: An efficient path planner for uav in complex environments,” in *IEEE International Conference on Real-time Computing and Robotics*. IEEE, 2016, pp. 555–560.

APPENDIX A

When the over-fitting parameter γ is set too large, RBF-kernel SVM can produce manifolds that are separated for the different parts of the training data, which is undesired in our case. Figure 8 shows the effect of γ on a 2D example with the same 1000 training points. When γ is 4.0, the manifold is continuous and closed. When γ is 10.0, the manifold is still closed but more curved. When γ is 20.0, the manifold has two parts. When γ is 40.0, the manifold has multiple parts. In the algorithm, we increase the value of γ gradually by a small, fixed value to avoid the discontinuities (“overly” over-fitting) in Figure 8c and Figure 8d.



(a) γ = 2.4 (b) γ = 10.0

(c) γ = 20.0 (d) γ = 40.0

Fig. 8: Effect of over-fitting parameter γ on RBF-kernel SVM training. Too large γ may produce discontinuous manifolds.