Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

PROXIMA: An Approach for Time or Accuracy Budgeted Collision Proximity Queries

Daniel Rakita
Department of Computer Sciences
University of Wisconsin-Madison
rakita@cs.wisc.edu

Bilge Mutlu
Department of Computer Sciences
University of Wisconsin-Madison
bilge@cs.wisc.edu

Michael Gleicher
Department of Computer Sciences
University of Wisconsin-Madison
gleicher@cs.wisc.edu

***Abstract*—Many applications in robotics require computing a robot manipulator's "proximity" to a collision state in a given configuration. This collision proximity is commonly framed as a summation over closest Euclidean distances between many pairs of rigid shapes in a scene. Computing many such pairwise distances is inefficient, while more efficient approximations of this procedure, such as through supervised learning, lack accuracy and robustness. In this work, we present an approach for computing a collision proximity function for robot manipulators that formalizes the trade-off between efficiency and accuracy and provides an algorithm that gives control over it. Our algorithm, called PROXIMA, works in one of two ways: (1) given a *time budget* as input, the algorithm returns an as-accurate-as-possible proximity approximation value in this time; or (2) given an *accuracy budget*, the algorithm returns an as-fast-as-possible proximity approximation value that is within the given accuracy bounds. We show the robustness of our approach through analytical investigation and simulation experiments on a wide set of robot models ranging from 6 to 132 degrees of freedom. We demonstrate that controlling the trade-off between efficiency and accuracy in proximity computations via our approach can enable safe and accurate real-time robot motion-optimization even on high-dimensional robot models.**

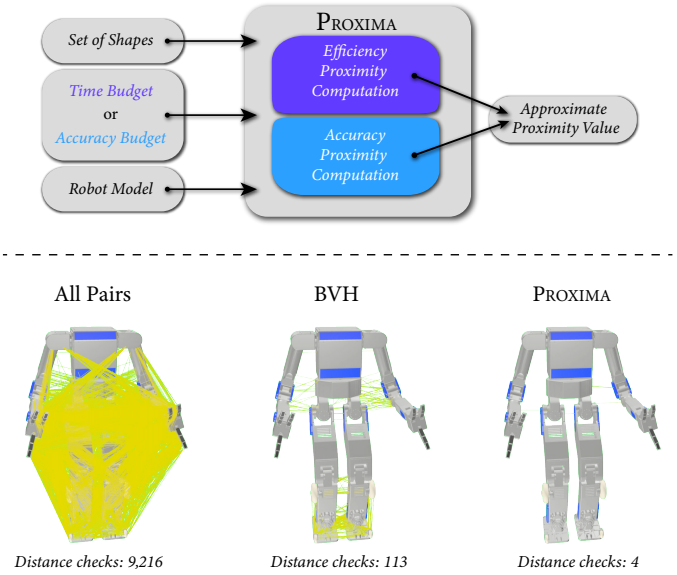

Fig. 1. We present an approach for computing a collision proximity function for robot manipulators that formalizes the trade-off between efficiency and accuracy and provides an algorithm that gives control over it. (Top) Our approach takes as input a set of shapes, robot model, and either a time or accuracy budget. The algorithm outputs an approximate proximity value that is either as-accurate-as-possible in the given time budget or as-fast-as-possible within the given accuracy bounds. (Bottom) Our approach strategically culls distance checks between any pairs of shapes whose pairwise distances can be tightly bounded in the first step of our algorithm. Thus, our approach often involves many fewer expensive distance checks between shapes compared to naïve iteration (left) and bounding volume hierarchies (middle).

I. INTRODUCTION

In order to safely execute tasks, a robot manipulator must exhibit motions that avoid or adapt to possible collisions with itself, other robots, and static or dynamic obstacles in its environment. Many robot motion generation strategies involve defining a scalar function, $c(\Theta)$, that characterizes how close a robot manipulator is to a collision state in a given configuration, Θ. A common form of this function, proposed by Schulman et al. [1], is some summation over distances between pairs of rigid shapes in the scene (*e.g.*, Equation 1). This function, along with its gradient, is used in optimization models to generate collision-free robot motions.

Computing the collision proximity function is often a computational bottleneck because it requires expensive *signed distance checks* between pairs of rigid shapes. Here, a signed distance check involves computing the closest Euclidean distance between a pair of non-intersecting shapes or computing the maximum penetration depth between two intersecting shapes. Naïvely iterating through all pairs of shapes is prohibitively expensive. To address this challenge, prior research has proposed, at a high level, three classes of solutions: (1) *shape approximation approaches* that replace the given set of shapes with approximating shapes that afford more efficient signed distance checks at run-time, *e.g.*, sets of spheres [2, 3]; (2) *supervised-learning approaches* that involve constructing some approximation of c from a large labeled dataset that will eliminate the need for any signed distance checks at runtime [4, 5, 6, 7, 8, 9]; and (3) *culling approaches* that use specialized data structures or geometric reasoning to ideally eliminate many signed distance checks at run-time. This approach can also utilize hierarchies of shape approximations, *e.g.*, bounding volume hierarchies (BVHs) [10, 11, 12].

The classes of approaches discussed above have contrasting strengths and weaknesses. For instance, shape approximation and supervised-learning approaches afford efficient estimates of c; however, these approximations exhibit errors with respect to the original function c that may be difficult to model,

bound, or control. This lack of control over accuracy and robustness makes this strategy impractical for performance-critical applications where collisions have a high cost such as remote homecare, robot surgery, space robotics, or nuclear materials handling. Conversely, culling methods often compute exact outputs of c and, thus, guarantee accuracy; however, their efficiency is not guaranteed and, in the worst case, $\frac{|S|^2}{2}$ distance checks (where S is the set of all shapes in the scene) may be needed for just one computation of c. This lack of control over efficiency makes this strategy impractical for time critical applications, such as shared-control or telemanipulation.

In this work, we introduce an approach for computing a collision proximity function for robot manipulators that formalizes the trade-off between efficiency and accuracy and provides control over it. Our algorithm, called PROXIMA, works in one of two ways. First, given a *time budget*, *i.e.*, an amount of time wherein a result should be returned, the algorithm returns an as-accurate-as-possible approximation value in this time. If the time budget is large enough, the "approximation" here will be exact, *i.e.*, $\hat{c} = c$. Second, given an *accuracy budget*, *i.e.*, a value ϵ such that the approximation is within this distance of the ground truth ($|c - \hat{c}| < \epsilon$), the algorithm returns an as-fast-as-possible approximation value that is within the accuracy bounds. Our algorithm also allows control over the accuracy budget computation such that it can always return a cautious, pessimistic approximation. Thus, if desired, a stronger accuracy budget bound of $c \leq \hat{c} \leq c+\epsilon$ can always be computed using our method to ensure the safety of the robot or surrounding obstacles in downstream applications. If the accuracy budget ϵ is zero, the "approximation" will be exact, *i.e.*, $\hat{c} = c$.

At a high level, PROXIMA operates in two phases: computing an error-bounded estimate, and then revising that estimate until either a time limit is reached, or the bounds on the estimate are sufficiently accurate. First, the algorithm computes guaranteed upper and lower bounds on the signed distances for all pairs of objects. These bounds are significantly less expensive to compute than ground truth signed distance checks, allowing the computation of all pairs even for hundreds of objects. The bounds also provide an estimate of the signed distance, along with a bound on the amount of error that the estimate may bring. Second, the algorithm replaces the estimates of pairs that have high potential error with the actual signed distance computations. These replacements are processed to reduce the overall error until it either falls within a required accuracy limit or fills the alloted time. By prioritizing the pairs with largest errors, the algorithm uses its time most effectively.

Through the steps above, our approach covers the strengths of both exact and approximation approaches by allowing users to dynamically determine how efficiency or accuracy should be prioritized. Our algorithm does not require any learning or pre-processing *a priori*, allows for dynamically changing kinematic structures or environments, and does not require any underlying shape properties such as convexity. Additionally, our approach enables the computation of an approximate gradient that can also accommodate time or accuracy budgets, meaning the approach as a whole can be incorporated into many non-linear optimization frameworks.

We show the robustness of our approach through analytical investigation and thorough simulation experiments on a wide set of robot models ranging from 6 to 132 DOF (§IV–V). We demonstrate through experiments that PROXIMA is able to compute proximity output values within these allotted time or accuracy budgets. Our experiments show that our algorithm computes results that are close approximations to the answers provided by exact approaches, with computation times that are competitive with learning and shape approximation methods. Additionally, we demonstrate that controlling the trade-off between efficiency and accuracy in proximity computations via our approach can enable safe and accurate real-time robot motion-optimization even on high-dimensional robot models. We provide open-source code for an implementation of our approach.[1]

II. RELATED WORKS

Our approach draws on many works in the areas of computational geometry, robot collision and proximity queries, robot motion optimization, and learning-based proximity function approximations. In this section, we highlight these works and discuss how they relate to our current work.

A. Object Distance Queries

A fundamental problem in computational geometry is determining the distance between shapes. The seminal GJK method developed by Gilbert, Johnson, and Keerthi finds the shortest distance between pairs of convex shapes by using a support function to iteratively converge on closest simplices [13]. Enhanced GJK methods also use edge information to speed up this convergence process [14]. The GJK family of approaches is still widely used, and is a common geometric primitive in many robotics and graphics software libraries [12, 15, 16].

Many distance computation approaches leverage a strategy called *coherence*, where the current geometric query tries to utilize information from previous queries in order to speed up computation. This strategy is particularly effective when consecutive moments ("frames" in graphics or "updates" in robotics) exhibit small, incremental changes. While GJK can take advantage of coherence, *e.g.*, initializing the algorithm at the previously found solution, other intersection and proximity query approaches more explicitly use this strategy. For instance, the Lin-Canny closest features algorithm tracks the closest combination of features between convex shapes (vertices, edges, or faces) from moment to moment and either quickly verifies that the previous closest points are still accurate or updates these closest points as necessary [17]. Using coherence and convexity, this algorithm achieves near constant time complexity in practice. The V-clip algorithm improves upon the Lin-Canny algorithm in terms of numerical

[1]Optima Toolbox: https://github.com/djrakita/optima_toolbox

stability, comprehensibility, and robustness [18]. We note that the coherence-based distance method proposed by Larsen et al. [19] using swept sphere volumes would be a particularly useful subroutine for our approach as the approximate distance afforded by the approach could be directly used in our accuracy budget computation.

Our work differs from the approaches above as we are not proposing a new way to compute a distance between a pair of geometric shapes. Instead, the goal of our approach is to efficiently and robustly bound and, when appropriate, approximate the distances between many pairs of shapes to be used in an aggregated scalar distance function for applications in robotics. Our approach is complementary to the class of approaches above and uses them as an important sub-routine (*e.g.*, our current implementation uses GJK for distance checks between pairs of shapes). While our approach is heavily inspired by the general strategy of coherence, it uses a new class of coherence that is not tied to a global coordinate system or time (explained in §IV).

B. Reducing Number of Signed Distance Checks

A common strategy when computing distances between many pairs of shapes is trying to eliminate as many distance checks at run-time as possible. For instance, a bounding volume hierarchy (BVH) is a data structure that attempts to reduce the number of distance checks between a set of shapes by traversing trees of nested approximating geometric shapes. Many BVH structures have been proposed and implemented for graphics and robotics applications [10, 11, 12]. Updating BVH data structures can be relatively expensive, especially for a large number of objects, though this update can use coherence to ease this issue [20, 21]. An alternative to a hierarchical data structure is a signed distance field (SDF) which discretizes a 3D environment and caches a distance to a closest surface at each voxel in space [2, 3]. SDFs allow for fast distance look-ups at run-time, though they are computationally expensive to construct and, thus, do not easily allow for dynamic updates in the environment making them non-ideal for robotics applications in practice.

Our approach shares similar goals with BVHs in terms of trying to cull as many distance checks as possible between pairs of shapes in a scene. However, instead of culling based on a pre-determined splitting strategy in the BVH tree and corresponding layers of nested approximating geometric shapes, our approach culls distance checks between non-salient pairs of shapes whose pairwise distances can be tightly approximated at the current proximity query between provable upper and lower distance bounds (explained in §III and IV).

C. Applications of Proximity Functions in Robotics

Intersection and distance checking is often an important sub-problem when generating robot motions. For instance, sampling-based motion planners often require a collision-checker to determine whether a C-space sample is feasible or infeasible [22, 23, 24]. While our approach may have benefits in motion planning frameworks to make the binary determination of whether a state is in collision or not, our approach is more tailored for optimization approaches that need to reason over the robot's continuous proximity to a collision state and the gradient of this proximity function. However, prior work that utilizes robot proximity computations for motion planners suggests interesting future work in this direction [25].

As mentioned above, our work is particularly designed for proximity and gradient computations for optimization frameworks. One such area where this is used is *trajectory optimization*. In prior work, trajectory optimization frameworks have used SDFs to compute distances and gradients [2, 3]. Work by Schulman et al. [26] improved on the robustness and generality of robot proximity checking and gradients.

Another recent area of attention in robot motion optimization has been collision-free inverse kinematics solvers. These approaches have one of two goals: (1) Optimize a sequence of independent robot configurations as fast as possible (typically in real-time) based on some primary objective (*e.g.*, end-effector pose matching) [4, 5, 9, 27]. This paradigm is often called *per-instant pose optimization*; or (2) Optimize a sequence of robot configurations in an off-line manner to match a pre-determined series of objectives (*e.g.*, a series of end-effector pose goals) [28, 29, 30]. This paradigm is often called *pathwise inverse kinematics*. In both cases, recent attention has turned towards achieving primary objectives while also maintaining other feasibility objectives and constraints such as avoiding self-collisions, kinematic singularities, or collisions with the environment. Because robot proximity checking is computationally expensive, these approaches often use supervised-learning approximations of proximity models to ease the computational burden [4, 5, 29, 6, 7, 8, 9]. For instance, Mirrazavi Salehian et al. [5] propose a support vector machine (SVM) classifier that learns a self-collision boundary for use in real-time motion optimization. Also, Kang et al. [9] present a method that also includes environment collision avoidance by using a collision-cost prediction network to output a robot's approximate proximity to obstacles given an input occupancy grid.

Our approach is also amenable for use in optimization-based IK solvers. Instead of using a supervised-learning-based proximity model to maintain efficiency, our approach uses a geometric approach that achieves a provably robust approximation through culling and bounded distance estimates. Our approach also does not involve pre-processing, it is more comprehensible and easier to prove robustness and accuracy of the output, and can account for dynamically updating kinematic structures or environments without retraining. We compare against a state-of-the-art learning-based method in our evaluation.

III. Technical Overview

In this section, we outline preliminary concepts and notation before overviewing our approach.

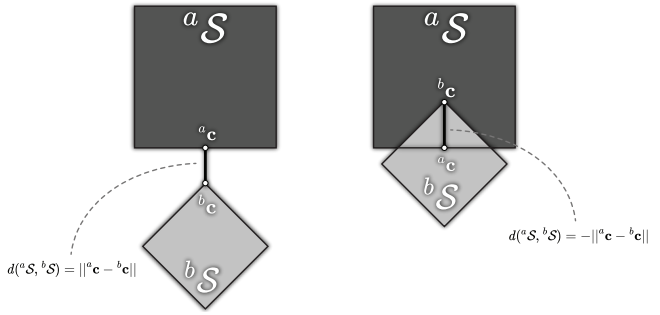

Fig. 2. Signed distance between a pair of shapes if they are not intersecting (left) or intersecting (right).

A. Geometric Preliminaries

Suppose we have a set of geometric shapes, S. An individual shape in S will be denoted as $^{\mathrm{a}}\mathcal{S}$, where the preceding superscript "a" is some reference label. The signed distance between a pair of shapes $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$ will be denoted as $d(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})$. If $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$ are not intersecting, $^{\mathrm{a}}\mathbf{c}$ and $^{\mathrm{b}}\mathbf{c}$ will denote the closest pair of points on $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$, respectively. Conversely, if $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$ are intersecting, $^{\mathrm{a}}\mathbf{c}$ and $^{\mathrm{b}}\mathbf{c}$ will denote the closest pair of points on $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$, respectively, that are both also contained in $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$. These concepts can be seen illustrated in Figure 2. Here, $d(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S}) = ||^{\mathrm{a}}\mathbf{c} - ^{\mathrm{b}}\mathbf{c}||$ if the shapes are not intersecting and $d(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S}) = -||^{\mathrm{a}}\mathbf{c} - ^{\mathrm{b}}\mathbf{c}||$ if the shapes are intersecting. A general function we are investigating in this work resembles the following:

$$\begin{aligned} \mathrm{c} = \sum_{\{^{\mathrm{a}}\mathcal{S}\in \mathrm{S}, ^{\mathrm{b}}\mathcal{S}\in \mathrm{S} \mid \mathrm{f}_1 \wedge \mathrm{f}_2\}} & \ell(d(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})) \\ \mathrm{f}_1 &:= \texttt{active}(^{\mathrm{a}}\mathcal{S}, \ ^{\mathrm{b}}\mathcal{S}), \\ \mathrm{f}_2 &:= d(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S}) < d_{\max} \end{aligned} \tag{1}$$

Here, $\ell(.)$ is some loss function that will map its scalar input to some output range (*e.g.*, it is standard for small signed distances to map to high values). The f logical rules specify which pairs of shapes in the scene will be filtered from the summation. For example, the `active` function indicates if a pair of shapes should be considered based on some pre-set rules (*e.g.*, $^{\mathrm{a}}\mathcal{S} \neq {}^{\mathrm{b}}\mathcal{S}$, etc.). The value $d_{\max}$ denotes some cut-off such that any greater distance will not be considered in the sum. In the following section, we adapt the abstract formulation in Equation 1 to a robot manipulation setting.

B. Robotics Preliminaries

In our work, the set S consists of 3-dimensional shapes that represent a robot's links or obstacles in a robot's environment. The robot model and objects in the environment may change poses over time, thus we will account for this possible motion in our notation. Now, an individual shape in S will be denoted as $^{\mathrm{a}}\mathcal{S}_t$, where the preceding superscript "a" is still some reference label of a shape, and the subscript "t" is some time label. A shape at time t, $^{\mathrm{a}}\mathcal{S}_t$, has a corresponding transform, $^{\mathrm{a}}\mathrm{T}_t \in \mathrm{SE}(3)$. We denote a robot's n-dimensional configuration at time t as Θ_t. Using a robot's forward kinematics model, we can map Θ_t to a transform T_t for any shape that is rigidly attached to a robot link. Note that the time parameter t on the robot configuration and underlying scene shapes does not necessarily imply smooth, continuous motion, it is solely a notational mechanism that allows the robot's links or objects in the environment to change poses from moment to moment. Using this notation, we update c in Equation 1 as follows:

$$\begin{aligned} \mathrm{c}(\Theta_t, t) = \sum_{\{^{\mathrm{a}}\mathcal{S}_t\in \mathrm{S}, ^{\mathrm{b}}\mathcal{S}_t\in \mathrm{S} \mid \mathrm{f}_1 \wedge \mathrm{f}_2 \wedge \mathrm{f}_3\}} & \ell\left(\frac{d(^{\mathrm{a}}\mathcal{S}_t, ^{\mathrm{b}}\mathcal{S}_t)}{a(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})}\right) \\ \mathrm{f}_1 &:= \texttt{active}(^{\mathrm{a}}\mathcal{S}_{\mathrm{t}}, \ ^{\mathrm{b}}\mathcal{S}_{\mathrm{t}}), \\ \mathrm{f}_2 &:= d(^{\mathrm{a}}\mathcal{S}_t, ^{\mathrm{b}}\mathcal{S}_t) < d_{\max} \\ \mathrm{f}_3 &:= \frac{d(^{\mathrm{a}}\mathcal{S}_t, ^{\mathrm{b}}\mathcal{S}_t)}{a(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})} < a_{\max} \end{aligned} \tag{2}$$

Here, $a(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})$ denotes the average signed distance between shapes $^{\mathrm{a}}\mathcal{S}$ and $^{\mathrm{b}}\mathcal{S}$. For example, if shapes represent robot links, average signed distances between all pairs of links could be computed off-line by random sampling many robot configurations. If an average signed distance between shapes is unknowable *a priori*, then $a(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S}) = 1$ by default. The $a_{\max}$ parameter is conceptually similar to $d_{\max}$; it serves as a threshold value such that shape pairs can be excluded or included in the proximity function summation based on how a shape pair's calculated signed distance at any moment compares to the shape pair's average signed distance.

The inclusion of $a(^{\mathrm{a}}\mathcal{S}, ^{\mathrm{b}}\mathcal{S})$ and $a_{\max}$ in Equation 2 is useful in a robotics context as it can correct for links that happen to be safely close together. For example, the distance between adjacent fingers on a dexterous hand may be naturally small, but concern should only be raised in a collision proximity model when this distance drops some ratio below their expected, average distance. The `active` function in a robotics context may filter shapes based on pre-computed criteria (*e.g.*, shapes that represent links on a robot may be considered inactive pairs if they are deemed never in collision or always in contact) or by object type (*e.g.*, distances should not be checked between two shapes representing static or dynamic obstacles in the environment).

At a high level, the c function in Equation 2 is similar to previously proposed proximity functions for robot manipulators [1]. The addition of the average distance between shapes is new and may help with stability in some cases, though this technique could be easily incorporated into prior frameworks. The novelty of our work relates to our procedure for computing an approximation of c in Equation 2 with time or accuracy guarantees based on user-defined budgets.

C. Proxima Overview

Our proposed algorithm involves six steps. Technical details and analyses regarding these steps can be found in §IV. We assume these steps begin with all pairs of shapes having a cached signed distance computed at some prior time. Thus, on the initial pass through these steps, ground truth signed distances must first be computed between all pairs of shapes.

1) Guaranteed upper and lower bounds on signed distance are computed between each active pair of shapes in the scene. In this work, these bounds are computed by assessing how relative transforms between shapes change over time, a novel technique we call *Pairwise Relative Spatial Coherence*;
2) Any pairs of shapes whose lower signed distance bound is greater than some cutoff (*e.g.*, f_2 or f_3 in Equation 2) are culled. Shape pairs remaining at this stage will be referred to as *summation eligible* pairs.
3) The upper and lower bounds computed in Step 1 are linearly interpolated based on a given parameter to create an estimated signed distance for each summation eligible shape pair;
4) The maximum loss function error that could possibly be induced by the estimated signed distance is calculated for each summation eligible shape pair;
5) All summation eligible shape pairs are sorted by their respective maximum possible loss function errors from highest to lowest;
6) Ground truth signed distance checks are processed in this sorted order until either time is up (in the case of a time budget) or until the maximum possible summation of remaining errors in the sorted list is less than a given ϵ (in the case of an accuracy budget)

IV. Technical Details and Analysis

In this section, we present additional mathematical and conceptual details on the steps above in §III-C and provide proofs for any guarantees.

A. Distance Bounds

A key aspect of our approach is computing guaranteed upper and lower bounds on signed distance between a pair of shapes without invoking a more expensive ground truth signed distance check. Our algorithm can accommodate any subroutine for computing these bounds; however, the approach as a whole will work best if these bounds are highly efficient to compute while still being reasonably tight. To achieve these goals, we developed a new technique that computes bounds by assessing how relative transforms between shapes change over time, a strategy we call Pairwise Relative Spatial Coherence.

We introduce the central strategy of Pairwise Relative Spatial Coherence using two arbitrary shapes, ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$. Suppose a ground truth signed distance check was last performed on ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ at a time point j. At this time, the approach has already cached the shapes' SE(3) transforms, ${}^{a}\mathrm{T}_j$ and ${}^{b}\mathrm{T}_j$, closest pair of points, ${}^{a}\mathbf{c}_j$ and ${}^{b}\mathbf{c}_j$, and signed distance, $d({}^{a}\mathcal{S}_j, {}^{b}\mathcal{S}_j) = \pm||{}^{a}\mathbf{c}_j - {}^{b}\mathbf{c}_j||$ (negative value denoting penetration depth if shapes are intersecting).

Now, suppose we are assessing the distance between ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ at the "current" time, k. Our key observation is if the relative transform between ${}^{a}\mathrm{T}_k$ and ${}^{b}\mathrm{T}_k$ is similar to the relative transform between ${}^{a}\mathrm{T}_j$ and ${}^{b}\mathrm{T}_j$, then $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k) \approx d({}^{a}\mathcal{S}_j, {}^{b}\mathcal{S}_j)$ (we formally define similar relative transforms below). Note that this effect holds even if ${}^{a}\mathrm{T}_k$ is far from ${}^{a}\mathrm{T}_j$ and ${}^{b}\mathrm{T}_k$ is far from ${}^{b}\mathrm{T}_j$ (*i.e.*, the shapes have both moved far in global space) or if k is much larger than j (*i.e.*, the proximity queries are far apart in time). We will leverage this general observation in order to bound the current distance, $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k)$.

The description above relies on the notion of "similar relative transforms". We will define relative transforms in this work with respect to transforms' translation and rotation components. Here, the translation component of an SE(3) transform ${}^{a}\mathrm{T}_t$ will be denoted as ${}^{a}\mathbf{t}_t \in \mathrm{R}^3$, and its rotation component will be denoted as ${}^{a}\mathbf{R}_t$. The rotation component is commonly either a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ or a unit quaternion $\mathbf{R} \in \mathrm{S}^3$. The math in this section will work with either choice of rotation representation. Between a time j and k, shapes ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ will exhibit the following relative change in translation, Δm:

Definition IV.1 (Relative Change in Translation)**.**

$$\Delta m = \left|\; ||{}^{a}\mathbf{t}_j - {}^{b}\mathbf{t}_j|| - ||{}^{a}\mathbf{t}_k - {}^{b}\mathbf{t}_k||\;\right|$$

Between a time j and k, shapes ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ will exhibit the following relative change in rotation, Δr:

Definition IV.2 (Relative Change in Rotation)**.**

$$\Delta r = \texttt{angle}(\texttt{disp}(\texttt{disp}({}^{a}\mathbf{R}_j, {}^{b}\mathbf{R}_j), \texttt{disp}({}^{a}\mathbf{R}_k, {}^{b}\mathbf{R}_k)))$$

Here, the `disp` function is the displacement function between rotations [31, 32], and the `angle` function specifies the angle in radians that a given rotation exhibits around its axis of rotation. For convenience, we provide `disp` and `angle` for unit quaternions and rotation matrices in Appendix A.

Using these definitions, if $\Delta m = 0$ and $\Delta r = 0$, then $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k) = d({}^{a}\mathcal{S}_j, {}^{b}\mathcal{S}_j)$ for rigid shapes, *i.e.*, if the relative translation and rotation between shapes ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ is equal between times j and k, then their closest points must be unchanged. The more interesting challenge in this work is reasoning about $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k)$ if Δm and Δr are non-zero. The following lemmas and definitions utilize these notions of relative transforms in order to prove a lower distance bound, $l({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k)$, and upper distance bound, $u({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k)$, between shapes ${}^{a}\mathcal{S}_k$ and ${}^{b}\mathcal{S}_k$ in Theorems IV.1 and IV.2, respectively.

Lemma IV.1. If $\Delta r = 0, \Delta m > 0$, *i.e.*, there is only relative translation change between ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ from times j and k without any relative orientation change, then $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k) \geq d({}^{a}\mathcal{S}_j, {}^{b}\mathcal{S}_j) - \Delta m$.

Proof: In Appendix B-A. ∎

Definition IV.3 (Translation of a point induced by rotation)**.** By the law of cosines, the translation distance of a point $\mathbf{p}$ on a rigid shape that is distance h from its local origin point $\mathbf{o}$ induced by rotation θ is

$$\Upsilon(h, \theta) = \sqrt{2h^2(1 - \cos(\theta))}\ .$$

Lemma IV.2. If $\Delta m = 0, \Delta r > 0$, *i.e.*, there is only relative orientation change between ${}^{a}\mathcal{S}$ and ${}^{b}\mathcal{S}$ between times j and k without any relative translation change, then $d({}^{a}\mathcal{S}_k, {}^{b}\mathcal{S}_k) \geq$

$d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$. Here, f refers to the maximal distance between any point on shape $\mathcal{S}$ and the shape's origin.

Proof: In Appendix B-B. ∎

Theorem IV.1 (Lower Signed Distance Bound)**.**

$$l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Delta m - \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$$

Proof: This result assumes the maximal change in distance due to both relative translation and orientation offsets from both Lemma IV.1 and Lemma IV.2. ∎

Theorem IV.2 (Upper Signed Distance Bound)**.**

$$u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = ||({}^{\mathrm{a}}\mathbf{R}_k({}^{\mathrm{a}}\mathbf{R}_j^{-1}\ ({}^{\mathrm{a}}\mathbf{c}_j - {}^{\mathrm{a}}\mathbf{t}_j)) + {}^{\mathrm{a}}\mathbf{t}_k) - ({}^{\mathrm{b}}\mathbf{R}_k({}^{\mathrm{b}}\mathbf{R}_j^{-1}\ ({}^{\mathrm{b}}\mathbf{c}_j - {}^{\mathrm{b}}\mathbf{t}_j)) + {}^{\mathrm{b}}\mathbf{t}_k)||$$

Proof: This equation transforms the closest points at time j, ${}^{\mathrm{a}}\mathbf{c}_j$ and ${}^{\mathrm{b}}\mathbf{c}_j$, to their new positions at time k and calculates their updated distance. Because this distance is exact for these two points, we know the ground truth distance between the two shapes cannot possibly be greater than this distance. ∎

B. Estimated Distance

An estimated distance for each summation eligible pair of shapes at a "current" time point k, denoted as $\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$, is computed by linearly interpolating between the upper and lower distance bounds specified in Theorems IV.1 and IV.2:

Definition IV.4 (Estimated Distance)**.**

$$\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = (1-r) * l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) + r * u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k), \quad r \in [0, 1]$$

Here, r is a scalar value. Any r value between 0 and 1 is a valid selection as this will guarantee that the estimated signed distance via interpolation is within the bounds. A value of $r = 0$ on all signed distance approximations will ensure that the proximity approximation output is a cautious, pessimistic estimate, *i.e.*, $\mathrm{c} \leq \hat{\mathrm{c}}$. Further, when used in an accuracy budgeted proximity computation, a value of $r = 0$ on all signed distance approximations will afford a strong guarantee that $\mathrm{c} \leq \hat{\mathrm{c}} \leq \mathrm{c} + \epsilon$ where ϵ is the given accuracy budget. Any other value of $r \in (0, 1]$ in an accuracy budgeted proximity computation will afford the weaker guarantee that $|\mathrm{c} - \hat{\mathrm{c}}| < \epsilon$. By default, the value $r = 0$ should be used in our algorithm as this will ensure that safety is enforced by any algorithm that uses our approach as a subroutine. Exceptions to this guideline are discussed below (*e.g.*, §IV-E). While a complete analysis of this r parameter is outside the scope of our current work, we empirically demonstrate some effects of this parameter in our simulation experiments below. Future extensions of our work could comprehensively characterize the effects of this parameter and, in turn, make informed, automatic selections of r per query that are better able to approximate the ground truth signed distance value between pairs of geometric shapes.

C. Maximum Loss Function Error

The estimated distance above is used to calculate a maximum possible loss function error associated with each summation eligible shape pair. Here, the loss function refers to $\ell(.)$ in Equation 2. This error, which we will denote as $\varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$, is calculated by comparing the loss function output at the estimated distance to loss function outputs at the bounds. We specify this error in Theorem IV.3 using the following definition.

Definition IV.5 (Loss with Cutoff)**.** Given a pair of shapes ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$, their *loss with cutoff*, which we will refer to as ℓ_{c}, is a modified, piecewise version of the loss function from Equation 2 that returns zero for any values that are either larger than $d_{\max}$ or larger than $a_{\max}$ when divided by the shapes' average distance:

$$\ell_{\mathrm{c}}(x) = \begin{cases} 0, & \text{if } x \geq d_{\max} \ \vee \ \frac{x}{a({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})} \geq a_{\max} \\ \ell(\frac{x}{a({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})}), & \text{otherwise} \end{cases}$$

This loss with cutoff function mathematically represents the f_2 and f_3 filter conditions in Equation 2, making it easier to account for corner cases where $l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) < d_{\max}$ but $u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \geq d_{\max}$ or $\frac{l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)}{a({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})} < a_{\max}$ but $\frac{u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)}{a({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})} \geq a_{\max}$ in Theorem IV.3.

Theorem IV.3 (Maximum Loss Function Error)**.** If the scalar loss function $\ell(.)$ is monotonically non-increasing and has a range of $\mathbb{R}_{\geq 0}$ (and, by extension, same applies for $\ell_{\mathrm{c}}(.)$), then the maximum loss function error given an estimated distance $\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ and bounds $l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ and $u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ will be:

$$\varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = \max(\ [\ell_{\mathrm{c}}(l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) - \ell_{\mathrm{c}}(\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))], [\ell_{\mathrm{c}}(\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) - \ell_{\mathrm{c}}(u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))]\ ) .$$

Proof: In Appendix B-C ∎

D. Budgeted Computations

All summation eligible shape pairs are sorted by their maximum possible loss function errors in highest to lowest order. Ground truth signed distance checks are then processed in this sorted order until either (1) a time limit is up in the case of a given time budget; or (2) the sum of the remaining maximum possible loss function errors is less than ϵ in the case of a given accuracy budget. Processing pairs of shapes in this order affords two useful qualities: (1) Subject to the given loss function errors $\varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ for all summation eligible shape pairs at time k, the proposed algorithm provides an as-accurate-as-possible approximation of c in less than $\mathcal{T}+\xi$ time, where $\mathcal{T}$ is a given time budget and ξ is a small time error to account for residual computation; and (2) Subject to the given loss function errors $\varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ for all summation eligible shape pairs at time k, the proposed algorithm provides an as-fast-as-possible approximation of c such that $|\mathrm{c} - \hat{\mathrm{c}}| < \epsilon$ for some

given accuracy budget ϵ. Further, if a value of $r = 0$ is used in the Estimated Distance step of our algorithm (§IV-B), the inequality above becomes the stronger guarantee $c \leq \hat{c} \leq c+\epsilon$.

E. Gradient Calculation

In this section, we specify how to compute an approximate gradient of Equation 2, *i.e.*, $\frac{\hat{\partial}c}{\partial\Theta_t}$, using either time or accuracy budgets. Our approach modifies the standard *finite difference* method as it allows for comprehensible, accurate, and easy to analyze results. As a quick review, finite differencing to compute a gradient of some function, c, involves the following steps: (1) Call c at the given input variable at time k, in our case, Θ_k. We will refer to the output value of this function call as x_0; (2) Slightly perturb the i-th element of the input variable vector for a given i by adding a small number υ. In our case, we will call this perturbed vector $\widetilde{\Theta_{ki}}$; (3) Call the original function at the perturbed input $\widetilde{\Theta_{ki}}$ and get an output value $\widetilde{x}_i$. The i-th element of the output gradient will be approximately $(-x_0+\widetilde{x}_i)/\upsilon$; and (4) Repeat Steps 2 and 3 for each input vector element $i \in \{1, ..., n\}$. We will refer to one pass through Steps 2 and 3 a *finite difference pass*.

Our PROXIMA gradient modifies the process above in order to account for a given time budget or accuracy budget. In the case of a time budget, the gradient takes a time parameter $\mathcal{T}_0$ as an allotted computation time for the initial function approximation (x_0) and a time parameter $\mathcal{T}$ as an allotted computation time for computing the gradient with respect to x_0. In the case of an accuracy budget, the gradient takes a value ϵ_0 as an allowable error on the initial function approximation (x_0), and a value ϵ as the total allowable L_∞-norm error on the approximated gradient with respect to x_0.

The four finite differencing steps above are adapted in the following ways: (1) Call the PROXIMA collision proximity function $\hat{c}$ at time k, *i.e.*, Θ_k, using either a time budget of $\mathcal{T}_0$ or an accuracy budget of ϵ_0. A linear interpolation value of $r = 1$ must be used for this step in order to keep it consistent with the r value in Step 3 (specified below); (2) Perturb the input variable using the same Step 2 specified above; (3) Call the PROXIMA collision proximity function at the perturbed input $\widetilde{\Theta_{ki}}$ using either a time budget of $\mathcal{T}/n$ or an accuracy budget of ϵ. We will consider each pass through this step as a new sub-time k' such that any distance checks taken at earlier parts of the same gradient calculation within the larger scope of "time" k can be used for subsequent distance bounds computations. It is important to use a linear interpolation value of $r = 1$ for estimated distances for each pass through Step 3 here. Because shape transforms are being perturbed a minuscule amount at each pass, the upper bound will be a much more accurate estimation of the updated function output than the lower bound. Also, if an appropriately small υ value is used, it is reasonable to assume that the set of summation eligible pairs will remain unchanged throughout this gradient computation. Thus, this set of shapes can be saved at the computation of x_0 for quick recall at each pass through this step; and (4) Repeat Steps 2 and 3 for each input vector element $i \in \{1, ..., n\}$.

V. EVALUATION

In this section, we overview several simulation experiments designed to demonstrate the robustness of our approach.

A. Implementation Details

Our prototype PROXIMA implementation is implemented in the Rust programming language. All distance queries are calculated using the `ncollide` Rust library. Using the same collision library throughout the evaluation ensures that conditions are not getting an unintended benefit with respect to each other due to faster underlying distance query implementations. Our prototype system uses the `NLopt` library, specifically using the SLSQP algorithm option, for all non-linear optimization routines. Experiments were run on a Lenovo Legion laptop with an i7-9750H processor and 32GB RAM.

In our prototype system, we select a piecewise Gaussian and linear function for the loss ℓ in Equation 2 as it is monotonically non-increasing and elicits smooth output values and gradients in practice:

$$\ell(x) := \begin{cases} \exp(\frac{-x^2}{2c^2}), c = 0.2 * a_{\max}, & \text{if } x > 0 \\ -x + 1, & \text{if } x \leq 0 \end{cases}$$

However, any reasonable monotonically non-increasing loss function is compatible with our approach. In all experiments, we use cutoff values of $d_{\max} = 0.3$ meters and $a_{\max} = 0.5$ in Equation 2, and all robot links are represented as best-fitting convex hulls.

B. Experimental Benchmark 1: C-Space Random Walks

Our first experimental benchmark is designed for Experiment 1 below. This benchmark involves taking many random walks in configuration space on numerous simulated robot models and assessing proximity metrics at each query. Specifically, Benchmark 1 involves the following steps: (1) Sample a collision-free robot configuration and set it as the robot's "current" configuration, $\mathbf{q}_{\text{curr}}$; (2) Sample a random direction to move in the robot's C-space by sampling n separate angle offset values (n being the number of robot DOF) from a Normal distribution for all i DOF $\in \{1, ...n\}$:

$$\begin{aligned} \mathbf{q}_{\text{curr}}[i] &= \mathbf{q}_{\text{curr}}[i] + z \\ z &\sim N(0, \sigma^2) \end{aligned} \tag{3}$$

Here, σ is the standard deviation of the normal distribution. Robot joint position limits are ignored at this step to allow unbiased steps through C-space; (3) Repeat Step 2 one thousand times and log any pertinent metrics at each configuration; (4) Repeat Steps 1–3 one hundred times; (5) Repeat Steps 1–4 for each experimental condition. All experimental conditions use the same random walks to mitigate unintended bias; and (6) Repeat Steps 1–5 for each robot model.

Experimental Benchmark 1 includes eleven simulated robot models in Step 6. These robot models were selected to span a diversity of kinematic structures and number of DOF.

Averages Over All Robot Models

	Time Budgeted Proxima						*Accuracy Budgeted Proxima*										
	t=150µs, r=1.0	t=150µs, r=0.0	t=100µs, r=1.0	t=100µs, r=0.0	t=50µs, r=1.0	t=50µs, r=0.0	a=0.001, r=1.0	a=0.001, r=0.0	a=0.1, r=1.0	a=0.1, r=0.0	a=0.5, r=1.0	a=0.5, r=0.0	Spheres	Neural Net	Sphere BVH	AABB BVH	Ground Truth
Time (µs)	*M*: 71.52 *SD*: 57.43	*M*: 72.00 *SD*: 57.45	*M*: 57.57 *SD*: 38.98	*M*: 57.21 *SD* : 39.05	*M*: 35.02 *SD*: 18.59	*M*: 35.96 *SD*: 18.68	*M*: 209.1 *SD*: 404.6	*M*: 208.9 *SD*: 404.0	*M*: 191.8 *SD*: 379.6	*M*:193.3 *SD*: 381.5	*M*: 167.9 *SD*: 345.2	*M*: 176.6 *SD*: 357.8	*M*: 97.18 *SD*: 123.27	*M*: 47.26 *SD*: 18.91	*M*: 527.5 *SD*: 700.4	*M*: 912.3 *SD*: 1094	*M*: 2900 *SD*: 4290
Accuracy	*M*: -2.366 *SD*: 2.249	*M*: 1.180 *SD*: 2.037	*M*: -2.542 *SD*: 5.223	*M*: 1.304 *SD*: 2.397	*M*: -2.639 *SD*: 5.418	*M*: 1.403 *SD*: 2.457	*M*: -1.2e-7 *SD*: 2.8e-6	*M*: 1.3e-7 *SD*: 2.8e-6	*M*: 0.001 *SD*: 0.006	*M*: 0.003 *SD*: 0.025	*M*: -0.011 *SD*: 0.046	*M*: 0.016 *SD*: 0.056	*M*: 18.52 *SD*: 19.03	*M*: 33.31 *SD*: 95.36	*M*: 0.0 *SD*: 0.0	*M*: 0.0 *SD*: 0.0	*M*: 0.0 *SD*: 0.0

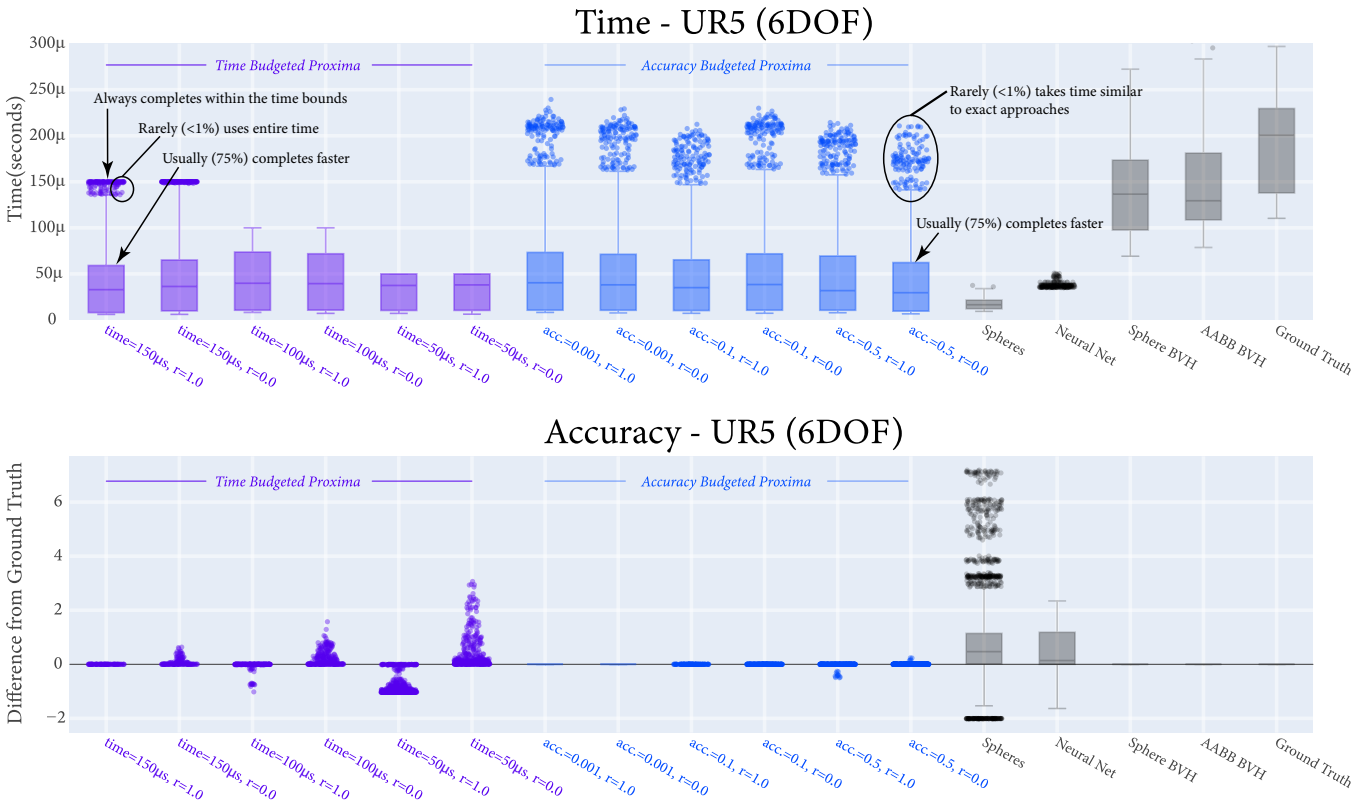

Fig. 3. Results for Experiment 1. (Top) Results that average over all robot models. Here, *M* denotes mean and *SD* denotes standard deviation. (Bottom) Tukey boxplots that show time and accuracy metric results for the UR5 robot. There are 100,000 data points in each box plot. Outliers (dots above or below plot whiskers) represent $< 1\%$ of all data points per condition. The UR5 robot is an easy test case due to its relatively low dimensionality (6 DOF) and small number of links (7). Thus, the times reflected here are lower than average. The results here indicate that the 150, 100, and 50 microsecond time budgets are always met (middle, left). The accuracy budgeted computation times are below 50 microseconds on average, but occasionally need similar time to the non-approximating conditions to meet their respective accuracy budgets (middle, middle). On average, time results from our approach are comparable to the *Neural Net* condition with much less error.

These robots are (1) Universal Robots UR5 (6 DOF)[2]; (2) Rethink Robotics Sawyer (8 DOF, including screen)[3]; (3) Franka Emika Panda (9 DOF, including gripper)[4]; (4) Kinova Jaco (10 DOF, including gripper)[5]; (5) Fetch Robotics Fetch (12 DOF, including head and gripper)[6]; (6) ABB Yumi (14 DOF)[7]; (7) PR2 (26 DOF, including head and grippers); (8) A Shadowhand gripper[8] on a Kuka IIWA (31 DOF, including all fingers)[9]; (8) NASA Valkyrie robot (64 DOF, including floating base and all fingers)[10]; (10) DRC Hubo+ (66 DOF, including floating base and grippers)[11]; and (11) NASA Robonaut2 (80 DOF, including floating base, all fingers, and leg grippers)[12].

C. Experiment 1: Comparing Proximity Model Approaches

In Experiment 1, we compare how efficiently and accurately various approaches compute (or approximate) Equation 2 through Experimental Benchmark 1. We compare five baseline conditions with several PROXIMA conditions. Baseline conditions are (1) *Spheres*, a shape approximation approach that represents each robot link as a union of spheres. This union of spheres is calculated by decomposing the link triangle mesh

[2]https://www.universal-robots.com/products/ur5-robot/
[3]https://www.rethinkrobotics.com/
[4]https://www.franka.de/
[5]https://www.kinovarobotics.com/
[6]https://fetchrobotics.com/
[7]https://new.abb.com
[8]https://www.shadowrobot.com/dexterous-hand-series/
[9]https://www.kuka.com/en-us
[10]https://www.nasa.gov/
[11]http://www.rainbow-robotics.com
[12]https://robonaut.jsc.nasa.gov/R2/

shapes into convex sub-components and wrapping each in a best fitting sphere; (2) *Neural Net*, which attempts to learn an approximation of Equation 2 using a neural network approach proposed in prior work [4]. This approach trains a multi-layer perceptron per robot, each using one million labeled input and output pairs. Because input and output pairs can only be feasibly collected for learning self-collision proximity outputs, we only consider robot models in empty environments when comparing with this approach; (3) *Sphere BVH*, which is a Bounding Volume Hierarchy approach where the hierarchical layers are approximating sphere shapes. This approach is designed to reduce the number of ground truth signed distance checks when computing Equation 2. The `ncollide` library implementation of a Bounding Volume Hierarchy was used for this condition, which follows the standard BVH principles specified in the work by Pan et al. [12]; (4) *AABB BVH*, which is also a Bounding Volume Hierarchy approach, but uses axis-aligned bounding boxes (AABBs) on each layer instead of spheres; and (5) *Ground Truth*, which computes Equation 2 by naïvely iterating through all pairs of shapes in the scene.

The baselines above are compared with twelve PROXIMA conditions, six time-budgeted variants and six accuracy budgeted variants. The time budgeted variants use maximum time values of 150, 100, or 50 microseconds. Each time budget was assessed using an r parameter value of 0.0 and 1.0 to see if the output approximation will indeed induce values such that $c \leq \hat{c}$ in the case of $r = 0$ and $c \geq \hat{c}$ in the case of $r = 1$. The accuracy budgeted variants use ϵ values of 0.001, 0.1, and 0.5. Each, accuracy budget was also assessed using r parameter values of 0 and 1.

Experiment 1 uses a random walk standard deviation $\sigma = 0.005$ in Equation 3. Recorded metrics for Experiment 1 are time per proximity query (in seconds) and the average difference between the output of a given approach from the ground truth output: $|\hat{c}-c|$ (if a condition is not an approximation, this will be zero). These metrics are averaged across all robots in this experiment.

1) Experiment 1 Results: Results from Experiment 1 can be seen in Figure 3. Figure 3 top displays the average results over all evaluated robot models, Figure 3 bottom more clearly displays just results for the 6DOF UR5 robot using box and whisker plots. First, we note that all PROXIMA conditions achieve their respective time or accuracy budgets. This can be clearly seen in Figure 3 (middle), where the times have hard cutoffs at 150, 100, and 50 microseconds and Figure 3 (bottom) where the accuracy values have hard accuracy cutoffs at 0.001, 0.1, and 0.5. We also see that the r value works exactly as intended: a value of $r = 0$ will result in output approximations such that $c \leq \hat{c}$ and a value of $r = 1$ will result in output approximations such that $c \geq \hat{c}$.

We also see that all PROXIMA conditions perform favorably compared to the baseline conditions. For example, *Spheres* and *Neural Net* have significantly higher errors than all other conditions, even compared to the PROXIMA conditions that run faster on average (*e.g.*, the variants with time budgets of 50 microseconds). This indicates that PROXIMA is wisely spending its given time budget, computing accurate-as-possible solutions even in short periods of time. Also, all PROXIMA conditions are significantly faster on average than *Sphere BVH*, *AABB BVH*, and *Ground Truth*, even when PROXIMA is given small accuracy budgets (*e.g.,* $\epsilon = 0.001$ or $\epsilon = 0.1$). This reflects the fact that PROXIMA is achieving its given accuracy budget as-fast-as-possible and stopping immediately after it guarantees that this target range has been met.

D. Experimental Benchmark 2: End-effector Curve Following

Our second experimental benchmark is designed for Experiment 2 below. This benchmark involves using various proximity function approaches for use in a robot motion optimization model. Specifically, this model is an optimization-based generalized inverse kinematics formulation based on prior work [4]. Our objectives and constraints in this work are taken directly from the proposed method by Rakita et al. [4], only switching out the self-collision avoidance objective for our various experimental conditions.

Benchmark 2 involves the following steps: (1) A cubic Bezier curve is procedurally generated by uniformly sampling spline knot points from a 1 meter $\times$ 1 meter $\times$ 1 meter domain in 3-dimensional space. These curves are arclength parameterized and broken up into 1,000 equally spaced points; (2) The cubic Bezier curve is lined up with a simulated robot model's end-effector such that the start of the curve matches the robot's end-effector in its given start configuration. If the robot model has more than one end-effector, Step 1 will involve generating a Bezier curve for each end-effector, and each is lined up accordingly in Step 2. (3) The optimization-based IK model is invoked 1,000 times. Each solve in the sequence updates its end-effector position-matching objective to be the given Bezier curve point in its respective sequence. The optimization model also includes a term that encourages the robot model to maintain the same end-effector orientation throughout the motion. Thus, the output of Step 3 will result in 1,000 individually optimized robot configurations that, in sequence, exhibit the robot tracing the Bezier curve (or curves) with its end-effector(s) while maintaining the same end-effector orientation as best as possible. The weighted sum objective model in the optimization model includes c or $\hat{c}$ as a term, thus the optimization will try to steer around self-collisions, subject to its given collision proximity model; (4) Steps 1–3 are repeated 50 times; (5) Steps 1–4 are repeated for each condition proximity model condition; (6) Steps 1–5 are repeated for two robot models: a 6 DOF UR5 and two 66 DOF Hubo robots standing side-by-side, resulting in a 132 DOF system overall. These robot models were selected as they are vastly different in structure, complexity, size, number of limbs, and number of DOF, so results should characterize proximity conditions over these broad differences. Example outputs of this procedure can be seen in Figure 4.

E. Experiment 2: Optimization-based Inverse Kinematics

In Experiment 2, we compare how various proximity models work as a collision avoidance objective in Experimental

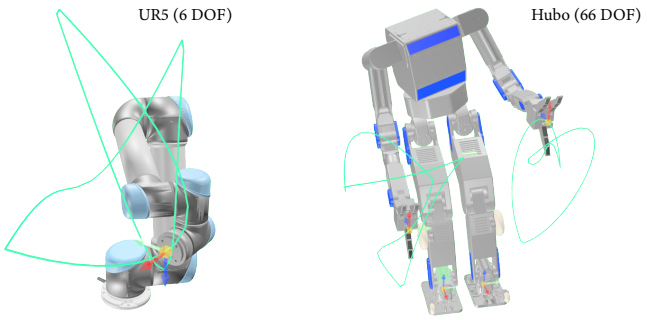

Fig. 4. Example motion outputs from the Bezier Curve Following procedure in Experimental Benchmark 2 on a UR5 and Hubo robot.

Benchmark 2. This experiment includes the 17 conditions from Experiment 1. In addition to these conditions, we add a condition where no self-collision avoidance term is included in the optimization objective function (referred to as *None*) as well as a condition that implements the collision avoidance and gradient procedures proposed by Schulman et al. [1] (referred to as *TrajOpt*. To summarize, this approach uses a BVH for distance check culling and a Jacobian-based procedure to approximate a gradient:

$$\mathbf{g} = \frac{\partial \hat{\mathrm{c}}}{\partial \Theta_t} = \mathrm{c}(\Theta_t) \ * \texttt{normalize}(\mathbf{g}) \sum_{\{{}^{\mathrm{a}}\mathcal{S}_t \in \mathrm{S}, {}^{\mathrm{b}}\mathcal{S}_t \in \mathrm{S} \mid \mathrm{f}_1 \wedge \mathrm{f}_2 \wedge \mathrm{f}_3\}} (\mathbf{n}^\top \mathbf{J}_{{}^{\mathrm{a}}\mathbf{c}_t}(\Theta_t))^\top - (\mathbf{n}^\top \mathbf{J}_{{}^{\mathrm{b}}\mathbf{c}_t}(\Theta_t))^\top \tag{4}$$

Here, $\mathbf{n}$ is a normalized vector that points in the direction from the closest point on ${}^{\mathrm{a}}\mathcal{S}$ (*i.e.*, ${}^{\mathrm{a}}\mathbf{c}_t$) to the closest point on ${}^{\mathrm{b}}\mathcal{S}$ (*i.e.*, ${}^{\mathrm{b}}\mathbf{c}_t$). The f logical statements are the same inclusion filters from Equation 2. A matrix $\mathbf{J}$ here denotes the $3 \times n$ translation Jacobian of a robot model, where n is the number of robot DOF. These Jacobians are calculated with respect to the closest points between shapes, either ${}^{\mathrm{a}}\mathbf{c}_t$ or ${}^{\mathrm{b}}\mathbf{c}_t$. If a particular point ${}^{\mathrm{p}}\mathbf{c}_t$ for any shape ${}^{\mathrm{p}}\mathcal{S}$ is not rigidly attached to the robot model, then $\mathbf{J}_{{}^{\mathrm{p}}\mathbf{c}_t}(\Theta_t)$ is a $3 \times n$ zero matrix. In order to get a more accurate scaling of the approximate gradient in Equation 4, we normalize the gradient direction, $\mathbf{g}$, and multiply it by the output of the original function. All other baseline conditions compute approximate gradients using standard finite differencing.

Recorded metrics for Experiment 2 are the number of solutions returned by the optimization solver per second (frequency); the percentage of configurations returned by the optimization that exhibit a self-collision; end-effector translation error (summed in the case of robots with multiple end-effectors); end-effector rotation error in radians (summed in the case of robots with multiple end-effectors); and the average joint velocity to indicate the smoothness of the motion.

1) Experiment 2 Results: Results from Experiment 2 can be seen in Figure 5. At a high level, we see that all PROXIMA conditions perform favorably on Benchmark 2. For instance, we see that PROXIMA avoids self-collisions in all conditions and successfully works alongside the end-effector pose matching and joint-smoothing objective terms.

Looking at results on the 6DOF UR5, we see that all PROXIMA conditions run at a faster frequency than all other conditions while still affording smooth and accurate motions. This is especially the case when using a low time budget of 50 microseconds or a loose accuracy budget of $\epsilon = 0.5$. While the frequency of PROXIMA drops accordingly when the time budget goes up or the accuracy budget becomes more strict, these conditions still run at highly interactive rates close to 1 KhZ. We speculate that this is due to the approximate gradient approach proposed in §IV as each finite differencing pass is able to quickly recall the summation eligible pairs and is able to eliminate almost all distance checks by using its tight upper bound on each perturbation.

Results on the 132DOF multi-robot Hubo system indicate that time budgeted PROXIMA affords reasonably interactive frequencies ranging from 42 Hz to 64 Hz depending on the time budget used. While the lower accuracy budgets for PROXIMA resulted in a drop-off in frequency, ranging from 5 Hz to 10 Hz, these results were still faster than all other collision-avoidance conditions with the exception of *TrajOpt*. Here, *TrajOpt* outperformed all other non-PROXIMA conditions in terms of frequency while still maintaining high quality motions. We speculate that this is because of its Jacobian-based approximation gradient, which avoids iterating through all 132 degrees of freedom using any finite differencing strategy. The *Spheres* and *Neural Net* conditions incur several collisions and erratic motions away from the end-effector paths, showcasing their lack of accuracy and robustness especially in this high-dimensional space. Lastly, we observe that *BVH* and *Grouth Truth* conditions are significantly slowed down by gradient calculations in this case, resulting in frequencies that would be too slow for most practical robotics applications.

VI. DISCUSSION

In this work, we have presented a flexible approach for computing a collision proximity function for robot manipulators that formalizes and allows for control over the trade-off between efficiency and accuracy. We showed the robustness of our approach through analytical investigation and thorough simulation experiments on a wide set of robot models ranging from 6 to 132 DOF. In the remainder of this section, we note limitations of our work, pointing towards potential avenues for future extensions or applications of our approach.

A. Limitations

Our implementation performs well at the scale of hundreds of objects. However, the method does consider the approximate signed distances between all object pairs. While these computations are fast, the quadratic complexity may become a problem for very large collections of objects. For such environments, some type of acceleration strategy, such as hierarchical culling, may be required. Our priority ordering step considers all distance checks to be equally expensive. In practice, different object pairs may have different costs for the

UR5 (6DOFs)

		Freq. (Hz)	% Collisions	EE Pos. Err.	EE Rot. Err.	Joint vel.
Time Budgeted Proxima	Time=150µs, r=1.0	981	0%	*M*: .046, *SD*: .05	*M*: .004, *SD*: .01	*M*: .004, *SD*: .00
	Time=150µs, r=0.0	977	0%	*M*: .045, *SD*: .05	*M*: .004, *SD*: .00	*M*: .004, *SD*: .00
	Time=100µs, r=1.0	999	0%	*M*: .048, *SD*: .07	*M*: .004, *SD*: .00	*M*: .004, *SD*: .00
	Time=100µs, r=0.0	982	0%	*M*: .049, *SD*: .07	*M*: .005, *SD*: .00	*M*: .004, *SD*: .00
	Time=50µs, r=1.0	1036	0%	*M*: .058, *SD*: .07	*M*: .005, *SD*: .00	*M*: .004, *SD*: .00
	Time=50µs, r=0.0	1027	0%	*M*: .057, *SD*: .07	*M*: .005, *SD*: .01	*M*: .004, *SD*: .00
Accuracy Budgeted Proxima	Acc. =0.001, r=1.0	850	0%	*M*: .043, *SD*: .04	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.001, r=0.0	849	0%	*M*: .044, *SD*: .03	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.1, r=1.0	969	0%	*M*: .049, *SD*: .07	*M*: .004, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.1, r=0.0	977	0%	*M*: .049, *SD*: .07	*M*: .004, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.5, r=1.0	1011	0%	*M*: .055, *SD*: .08	*M*: .004, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.5, r=0.0	1005	0%	*M*: .050, *SD*: .08	*M*: .004, *SD*: .00	*M*: .003, *SD*: .00
	TrajOpt	634	0%	*M*: .048, *SD*: .00	*M*: .005, *SD*: .01	*M*: .003, *SD*: .00
	Spheres	782	2%	*M*: .105, *SD*: .04	*M*: .009, *SD*: .02	*M*: .006, *SD*: .01
	Neural Net	756	0%	*M*: .08, *SD*: .10	*M*: .009, *SD*: .03	*M*: .003, *SD*: .01
	Sphere BVH	238	0%	*M*: .041, *SD*: .01	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	AABB BVH	227	0%	*M*: .042, *SD*: .01	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	Ground Truth	112	0%	*M*: .041, *SD*: .01	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	None	2392	8%	*M*: .031, *SD*: .03	*M*: .003, *SD*: .00	*M*: .001, *SD*: .00

Two Hubos (132DOFs)

		Freq. (Hz)	% Collisions	EE Pos. Err.	EE Rot. Err.	Joint vel.
Time Budgeted Proxima	Time=150µs, r=1.0	42.7	0%	*M*: .042, *SD*: .05	*M*: .002, *SD*: .01	*M*: .002, *SD*: .00
	Time=150µs, r=0.0	45.5	0%	*M*: .041, *SD*: .05	*M*: .002, *SD*: .00	*M*: .002, *SD*: .00
	Time=100µs, r=1.0	56.0	0%	*M*: .049, *SD*: .05	*M*: .002, *SD*: .00	*M*: .003, *SD*: .00
	Time=100µs, r=0.0	55.4	0%	*M*: .051, *SD*: .05	*M*: .002, *SD*: .00	*M*: .002, *SD*: .00
	Time=50µs, r=1.0	61.2	0%	*M*: .061, *SD*: .05	*M*: .002, *SD*: .01	*M*: .002, *SD*: .00
	Time=50µs, r=0.0	64.2	0%	*M*: .064, *SD*: .03	*M*: .002, *SD*: .01	*M*: .002, *SD*: .00
Accuracy Budgeted Proxima	Acc. =0.001, r=1.0	5.8	0%	*M*: .043, *SD*: .04	*M*: .002, *SD*: .00	*M*: .003, *SD*: .00
	Acc. =0.001, r=0.0	6.0	0%	*M*: .044, *SD*: .03	*M*: .002, *SD*: .00	*M*: .002, *SD*: .00
	Acc. =0.1, r=1.0	8.6	0%	*M*: .049, *SD*: .07	*M*: .003, *SD*: .00	*M*: .003, *SD*: .00
	Acc. =0.1, r=0.0	10.1	0%	*M*: .049, *SD*: .07	*M*: .003, *SD*: .00	*M*: .004, *SD*: .00
	Acc. =0.5, r=1.0	23.1	0%	*M*: .045, *SD*: .08	*M*: .002, *SD*: .01	*M*: .002, *SD*: .00
	Acc. =0.5, r=0.0	20.3	0%	*M*: .048, *SD*: .06	*M*: .002, *SD*: .00	*M*: .002, *SD*: .00
	TrajOpt	8.8	0%	*M*: .077, *SD*: .08	*M*: .005, *SD*: .01	*M*: .004, *SD*: .00
	Spheres	0.33	9%	*M*: .112, *SD*: .50	*M*: .005, *SD*: .00	*M*: .010, *SD*: .02
	Neural Net	2.1	7%	*M*: .134, *SD*: .24	*M*: .009, *SD*: .03	*M*: .018, *SD*: .03
	Sphere BVH	0.1	0%	*M*: .038, *SD*: .03	*M*: .003, *SD*: .00	*M*: .002, *SD*: .00
	AABB BVH	.06	0%	*M*: .038, *SD*: .03	*M*: .003, *SD*: .00	*M*: .002, *SD*: .00
	Ground Truth	.03	0%	*M*: .038, *SD*: .04	*M*: .003, *SD*: .00	*M*: .002, *SD*: .00
	None	112	33%	*M*: .009, *SD*: .01	*M*: .002, *SD*: .00	*M*: .001, *SD*: .00

Fig. 5. Results for Experiment 2. Reported metrics are number of solutions returned by the optimization solver per second (frequency); the percentage of configurations returned by the optimization that exhibit a self-collision; end-effector translation error (summed in the case of robots with multiple end-effectors); end-effector rotation error in radians (summed in the case of robots with multiple end-effectors); and the average joint velocity.

exact computation which could be considered in addition to the potential error.

To date, we have tested our approach for self-collisions in simulations, for both collision checking and within a collision-aware IK framework. In principle, our approach should be a drop-in replacement for the proximity functions used in numerical-optimization based trajectory optimizers and sampling-based planners. While we expect our method to perform well in such situations, there may be opportunities to exploit the repetitive structure of the queries made by such systems. Similarly, all of our usages to date have been in simulation. Because our approach makes no assumptions about the underlying geometric representations, it should extend to real-world sensed models. There may be ways to specialize the approach to particular model types such as point clouds, for example using the semi-infinite programming approach of Hauser [33].

Acknowledgments

This work was supported by a Cisco Graduate Student Fellowship, a Microsoft Research PhD Fellowship, National Science Foundation award 1830242, and a NASA University Leadership Initiative (ULI) grant awarded to the UW-Madison and The Boeing Company (Cooperative Agreement # 80NSSC19M0124). The Shadowhand gripper on a Kuka IIWA robot model was graciously provided by Andreas Orthey and is now publicly available in the robowflex_resources repository.

References

[1] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[2] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. 2009.

[3] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[4] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. RelaxedIK: Real-time Synthesis of Accurate and Feasible Robot Arm Motion. In *Proceedings of Robotics: Science and Systems*, Pittsburgh, Pennsylvania, June 2018. doi: 10.15607/RSS.2018.XIV.043.

[5] Seyed Sina Mirrazavi Salehian, Nadia Figueroa, and Aude Billard. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research*, 37(10):1205–1232, 2018.

[6] Nikhil Das and Michael Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[7] Mikhail Koptev, Nadia Figueroa, and Aude Billard. Real-time self-collision avoidance in joint space for humanoid robots. *IEEE Robotics and Automation Letters*, 6(2): 1240–1247, 2021.

[8] Yuheng Zhi, Nikhil Das, and Michael Yip. Diffco: Auto-differentiable proxy collision detection with multi-class labels for safety-aware trajectory optimization. *arXiv preprint arXiv:2102.07413*, 2021.

[9] Mincheul Kang, Yoonki Cho, and Sung-Eui Yoon. RCIK: Real-time collision-free inverse kinematics using a collision-cost prediction network. *IEEE Robotics and Automation Letters*, 7(1):610–617, 2021.

[10] Brian Mirtich. Efficient algorithms for two-phase collision detection. *Practical motion planning in robotics: current approaches and future directions*, pages 203–223, 1997.

[11] Thomas Larsson and Tomas Akenine-Möller. A dynamic bounding volume hierarchy for generalized collision detection. *Computers & Graphics*, 30(3):450–459, 2006.

[12] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, pages 3859–3866. IEEE, 2012.

[13] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, 1988.

[14] Stephen Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *Proceedings of international conference on robotics and automation*, volume 4, pages 3112–3117. IEEE, 1997.

[15] Erwin Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[16] Epic Games. Unreal engine. URL https://www. unrealengine.com.

[17] Ming C Lin. *Efficient collision detection for animation and robotics*. PhD thesis, PhD thesis, Department of Electrical Engineering and Computer Science . . . , 1993.

[18] Brian Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions On Graphics (TOG)*, 17(3):177–208, 1998.

[19] Eric Larsen, Stefan Gottschalk, Ming C Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of . . . , 1999.

[20] David Baraff. *Dynamic simulation of nonpenetrating rigid bodies*. PhD thesis, Cornell University, 1992.

[21] Jonathan D Cohen, Ming C Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff, 1995.

[22] James J Kuffner Jr and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *ICRA*, volume 2, 2000.

[23] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[24] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[25] James Kuffner, Koichi Nishiwaki, Satoshi Kagami, Yasuo Kuniyoshi, Masayuki Inaba, and Hirochika Inoue. Self-collision detection and prevention for humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 3, pages 2265–2270. IEEE, 2002.

[26] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[27] Daniel Rakita, Haochen Shi, Bilge Mutlu, and Michael Gleicher. Collisionik: A per-instant pose optimization method for generating robot motions with environment collision avoidance. *arXiv preprint arXiv:2102.13187*, 2021.

[28] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. Stampede: A discrete-optimization method for solving pathwise-inverse kinematics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3507–3513. IEEE, 2019.

[29] Pragathi Praveena, Daniel Rakita, Bilge Mutlu, and Michael Gleicher. User-guided offline synthesis of robot arm motion from 6-dof paths. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8825–8831. IEEE, 2019.

[30] Mincheul Kang, Heechan Shin, Donghyuk Kim, and Sung-Eui Yoon. Torm: Fast and accurate trajectory optimization of redundant manipulator given an end-effector path. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9417–9424. IEEE, 2020.

[31] Jehee Lee. Representing rotations and orientations in geometric computing. *IEEE Computer Graphics and Applications*, 28(2):75–83, 2008.

[32] Neil T Dantam. Robust and efficient forward, differential, and inverse kinematics using dual quaternions. *The International Journal of Robotics Research*, page 0278364920931948, 2020.

[33] Kris Hauser. Semi-infinite programming for trajectory optimization with non-convex obstacles. *The International Journal of Robotics Research*, 40(10-11):1106–1122, 2021.

Appendix A
Rotation Operators

A. Unit Quaternions

Our notation for unit quaternions will be $\mathbf{R} = (\mathrm{q_w}, \mathbf{q_v}) = [\mathrm{q}_w, i\mathrm{q}_x, j\mathrm{q}_y, k\mathrm{q}_z]$.

1) *Inverse:* $\mathbf{R}^{-1} = (\mathrm{q_w}, -\mathbf{q_v}) = [\mathrm{q}_w, -i\mathrm{q}_x, -j\mathrm{q}_y, -k\mathrm{q}_z]$
2) *Displacement:* $\texttt{disp}(\mathbf{R}_1, \mathbf{R}_2) = \mathbf{R}_1^{-1} * \mathbf{R}_2$
3) *Angle:* $\texttt{angle}(\mathbf{R}) = 2 * \mathrm{acos}(\mathrm{q}_w)$

B. Rotation Matrices

1) *Inverse:* $\mathbf{R}^{-1} = \mathrm{R}^\top$
2) *Displacement:* $\texttt{disp}(\mathbf{R}_1, \mathbf{R}_2) = \mathbf{R}_1^\top * \mathbf{R}_2$
3) *Angle:* $\texttt{angle}(\mathbf{R}) = \mathrm{acos}((\mathrm{trace}(\mathbf{R}) - 1)/2)$

Appendix B
Proofs for Lemmas IV.1 and IV.2

A. Proof for Lemma IV.1

Case 1: $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) > d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$
Trivial case as Δm is always non-negative.

Case 2: $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$
Assume $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) < d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Delta m$. This assumption can be rewritten as $d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) > \Delta m$. Now, consider $[{}^{\mathrm{a}}\mathbf{c}_k]_j$ and $[{}^{\mathrm{b}}\mathbf{c}_k]_j$ as the points at time j that will go on to be the closest points between ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$ at time k after ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$ reflect transforms ${}^{\mathrm{a}}\mathrm{T}_k$ and ${}^{\mathrm{b}}\mathrm{T}_k$, respectively. Because $\Delta r = 0$ and all change in distance between the shapes is due to translation, we can say that

$$d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = ||[{}^{\mathrm{a}}\mathbf{c}_k]_j - [{}^{\mathrm{b}}\mathbf{c}_k]_j|| - ||{}^{\mathrm{a}}\mathbf{c}_k - {}^{\mathrm{b}}\mathbf{c}_k||,$$

$$||[{}^{\mathrm{a}}\mathbf{c}_k]_j - [{}^{\mathrm{b}}\mathbf{c}_k]_j|| - ||{}^{\mathrm{a}}\mathbf{c}_k - {}^{\mathrm{b}}\mathbf{c}_k|| > \Delta m = |\ ||{}^{\mathrm{a}}\mathbf{t}_j - {}^{\mathrm{b}}\mathbf{t}_j|| - ||{}^{\mathrm{a}}\mathbf{t}_k - {}^{\mathrm{b}}\mathbf{t}_k||\ |$$

Because all transforms T are rigid transforms, we know that all points on ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$ are translated by the same vectors $({}^{\mathrm{a}}\mathbf{t}_k - {}^{\mathrm{a}}\mathbf{t}_j)$ or $({}^{\mathrm{b}}\mathbf{t}_k - {}^{\mathrm{b}}\mathbf{t}_j)$ respectively between time j and k. This contradicts the inequality above, which implies that the new closest points somehow moved farther than all other points on the shapes, thus the original assumption must be incorrect and $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \geq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Delta m$ when $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$. ■

B. Proof for Lemma IV.2

Case 1: $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) > d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$
Trivial case as $\Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$ is always non-negative.

Case 2: $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$
Assume $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) < d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$. This assumption can be rewritten as $d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) > \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$. Now, consider $[{}^{\mathrm{a}}\mathbf{c}_k]_j$ and $[{}^{\mathrm{b}}\mathbf{c}_k]_j$ as the points at time j that will go on to be the closest points between ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$ at time k after ${}^{\mathrm{a}}\mathcal{S}$ and ${}^{\mathrm{b}}\mathcal{S}$ reflect transforms ${}^{\mathrm{a}}\mathrm{T}_k$ and ${}^{\mathrm{b}}\mathrm{T}_k$, respectively. Because $\Delta m = 0$ and all change in distance between the shapes is due to angular motion, we can say that

$$d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = \Upsilon(||[{}^{\mathrm{a}}\mathbf{c}_k]_j - {}^{\mathrm{a}}\mathbf{o}_\mathrm{j}||, \mathrm{s} * \Delta r) + \Upsilon(||[{}^{\mathrm{b}}\mathbf{c}_k]_j - {}^{\mathrm{b}}\mathbf{o}_j||, (1 - \mathrm{s}) * \Delta r), \mathrm{s} \in [0, 1]$$

Here, s is a value between 0 and 1 that reflects how Δr is divvied between the two object's respective rotations. Based on our assumption, we can say

$$\Upsilon(||[{}^{\mathrm{a}}\mathbf{c}_k]_j - {}^{\mathrm{a}}\mathbf{o}_j||, \mathrm{s} * \Delta \mathrm{r}) + \Upsilon(||[{}^{\mathrm{b}}\mathbf{c}_k]_j - {}^{\mathrm{b}}\mathbf{o}_j||, (1 - \mathrm{s}) * \Delta r), \mathrm{s} \in [0, 1] > \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$$

We can maximize the left side of this equation by assigning s = 1 to the term in the sum with the greater distance between the rotating point and its origin, thus

$$\Upsilon(\max(||[{}^{\mathrm{a}}\mathbf{c}_k]_j - {}^{\mathrm{a}}\mathbf{o}_j||, ||[{}^{\mathrm{b}}\mathbf{c}_k]_j - {}^{\mathrm{b}}\mathbf{o}_j||), \Delta \mathrm{r}) \geq \Upsilon(||[{}^{\mathrm{a}}\mathbf{c}_k]_\mathrm{j} - {}^{\mathrm{a}}\mathbf{o}_\mathrm{j}||, \mathrm{s} * \Delta r) + \Upsilon(||[{}^{\mathrm{b}}\mathbf{c}_k]_j - {}^{\mathrm{b}}\mathbf{o}_j||, (1 - \mathrm{s}) * \Delta r), \mathrm{s} \in [0, 1] > \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$$

This implies that $\max(||[{}^{\mathrm{a}}\mathbf{c}_k]_j - {}^{\mathrm{a}}\mathbf{o}_j||, ||[{}^{\mathrm{b}}\mathbf{c}_k]_j - {}^{\mathrm{b}}\mathbf{o}_j|| > \max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f)$. However, this contradicts the definitions of ${}^{\mathrm{a}}f$ and ${}^{\mathrm{b}}f$, meaning the initial assumption in this case cannot be true and $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \geq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j) - \Upsilon(\max({}^{\mathrm{a}}f, {}^{\mathrm{b}}f), \Delta r)$ when $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq d({}^{\mathrm{a}}\mathcal{S}_j, {}^{\mathrm{b}}\mathcal{S}_j)$. ■

C. Proof for Theorem IV.3

We will assume there is some ground truth error term $\varepsilon^+({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})$ between the estimated distance and ground truth distance $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ such that the following is true:

$$\varepsilon^+({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S}) = |\ell_\mathrm{c}(\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) - \ell_\mathrm{c}(d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))| > \varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) = \max(\ [\ell_\mathrm{c}(l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) - \ell_\mathrm{c}(\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))], [\ell_\mathrm{c}(\hat{d}({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) - \ell_\mathrm{c}(u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))]\ )$$

Given this assumption, either $\ell_\mathrm{c}(d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) > \ell_\mathrm{c}(l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))$ or $\ell_\mathrm{c}(d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)) < \ell_\mathrm{c}(u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k))$ and, because $\ell_\mathrm{c}(.)$ is monotonically non-increasing, either $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) < l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ or $d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) > u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$. This observation presents a contradiction because $l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ and $u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ are proven bounds such that $l({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq d({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k) \leq u({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$ meaning our assumption above is false and there is no ground truth error $\varepsilon^+({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S})$ such that $\varepsilon^+({}^{\mathrm{a}}\mathcal{S}, {}^{\mathrm{b}}\mathcal{S}) > \varepsilon({}^{\mathrm{a}}\mathcal{S}_k, {}^{\mathrm{b}}\mathcal{S}_k)$.

■