An Efficient Algorithm for a Generalized Distance Measure

Yu Zheng, Katsu Yamane

***Abstract*— This paper presents an efficient algorithm for computing a distance measure between two compact convex sets Q and A, defined as the minimum scale factor such that the scaled Q is not disjoint from A. An important application of this algorithm in robotics is the computation of the minimum distance between two objects, which can be performed by taking A as the Minkowski difference of the objects and Q as a set containing the origin in its interior. In this generalized definition, the traditional Euclidean distance is a special case where Q is the unit ball. While this distance measure was proposed almost a decade ago, there has been no efficient algorithm to compute it in general cases. Our algorithm fills this void and we demonstrate its superior efficiency compared to approaches based on general-purpose optimization.**

I. Introduction

Computing the minimum distance between two point sets is a fundamental problem in many research fields, such as robot motion planning, computer-aided design, and physics simulation. The minimum distance between two separated sets is equivalent to the minimum distance from the origin to all points in their Minkowski difference. Traditionally, the distance is defined in terms of the 2-norm and can be interpreted as the minimum scale factor of the origin-centered unit ball such that the scaled ball and the Minkowski difference are not disjoint. Gilbert *et al.* [1], [2] first proposed an algorithm to compute this distance (GJK algorithm), which generates a sequence of simplices in the Minkowski difference such that their minimum distances to the origin converge to the globally minimum distance. Lin and Canny [3] developed another efficient algorithm to calculate the distance between polyhedra by finding their closest features (LC algorithm). For the past two decades, these algorithms have been improved by Cameron [4], Mirtich [5], and Ong and Gilbert [6], and have been widely used in motion planning, physics simulation, and computer animation.

More recently, Zhu et al. [7], [8] gave a more general distance measure: the minimum scale factor of a compact convex set Q such that the scaled Q is not disjoint from the other compact convex set A. When A is the Minkowski difference of two objects and Q is a set containing the origin in its interior, the distance measure gives a generalized distance between the objects in a metric defined by Q. For example, the traditional Euclidean distance is given by taking Q as the origin-centered unit ball. This distance measure has been successfully applied in grasping research to evaluating how far a grasp is from being stable and capable of generating required forces to fulfill a grasping task [7].

The authors are with Disney Research Pittsburgh, PA 15213, USA {yu.zheng, kyamane}@disneyresearch.com

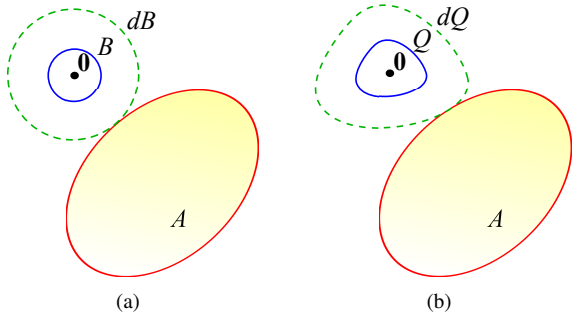


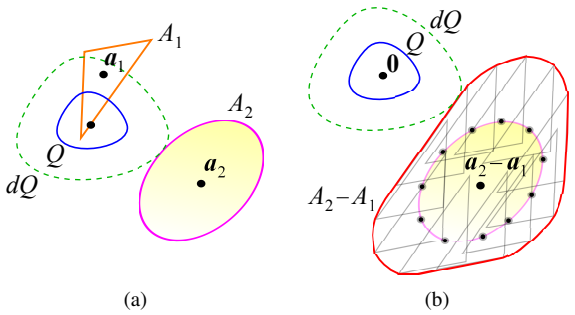
Fig. 1. Illustration in 2-dimensional (2-D) space of the distance defined in (a) the 2-norm and (b) a general metric. The distance d between the origin and a set A is equal to the minimum scale factor of (a) the origin-centered unit ball B or (b) a general compact convex set Q such that the scaled set dB or dQ is not disjoint from A.

However, there has been no efficient algorithm to calculate it for general convex sets A and Q. The algorithm used in [7], [8] reduced the computation to a linear program by approximating A and Q by polytopes and applying Simplex method. The only method applicable to general convex sets has been general-purpose numerical optimization, which is computationally expensive.

Another possible method for distance computation is support vector machines (SVMs). Given two sets of training data points, SVM training algorithms determine a hyperplane that separates the two sets as well as their minimum distance. If each of the two sets consist of the vertices of a convex polytope, an SVM that separates the sets gives the minimum distance between the polytopes. One prominent method for training SVMs is sequential minimal optimization [9], [10]. However, it works only for polytopes.

In this paper, we present a novel algorithm for computing the generalized distance proposed in [7], [8]. The algorithm is geometry-based and works for general compact convex sets represented as polytopes and parametric surfaces. Through a number of numerical examples in spaces with different dimensions, we demonstrate that the algorithm is several to tens of times faster than the optimization-based method to achieve the result at the same level of accuracy.

The rest of this paper is organized as follows. Section II introduces the distance definition and other mathematical concepts. Section III describes our distance algorithm, followed by numerical examples in Section IV that verify its performance quality. Conclusions and future work are given in Section V.



(a) (b)

Fig. 2. Equivalence of (a) the distance between two separated compact convex sets A_1 and A_2 to (b) the distance between the origin $\mathbf{0}$ and their Minkowski difference $A_2 - A_1$.

II. Definitions

A. A Generalized Distance Measure

Let A be a compact convex set with nonempty interior in $\mathbb{R}^n$ that does not contain the origin of $\mathbb{R}^n$. The traditional minimum distance between the origin and A is defined in terms of the 2-norm as [1]

$$d(A) \triangleq \min_{\boldsymbol{a}\in A} \|\boldsymbol{a}\|_2 \tag{1}$$

where $\|\cdot\|_2$ denotes the 2-norm of a vector. More generally, the distance can also be defined using by the p-norm given as

$$\|\boldsymbol{a}\|_p \triangleq \left(\sum_{i=1}^{n} |a_i|^p\right)^{\frac{1}{p}} \tag{2}$$

where $p \geq 1$ is a real number. We can define unit balls centered at the origin in different norms as

$$B \triangleq \{\boldsymbol{a} \in \mathbb{R}^n \mid \|\boldsymbol{a}\|_p = 1\}. \tag{3}$$

For $p \geq 1$, unit ball B is convex and centrally symmetric. Fig. 1a shows the case of using the traditional 2-norm. Using B, the minimum distance between the origin and A in terms of the p-norm can be rewritten as

$$d_B(A) = \min_{A\cap\lambda B\neq\emptyset,\lambda\geq 0} \lambda. \tag{4}$$

We can define even more general distance measures by replacing replace B with general compact convex set Q with nonempty interior containing the origin of $\mathbb{R}^n$, as depicted in Fig. 1b. Then, the distance $d_Q(A)$ between the origin and A with respect to Q is defined by [7], [8]

$$d_Q(A) \triangleq \min_{A\cap\lambda Q\neq\emptyset,\lambda\geq 0} \lambda = \min_{\boldsymbol{x}\in A,\boldsymbol{x}\in\lambda Q,\lambda\geq 0} \lambda. \tag{5}$$

In other words, $d_Q(A)$ is the minimum nonnegative scale factor λ such that A and λQ are not disjoint, and therefore generalizes the traditional Euclidean distance.

This distance definition can be easily extended to the distance between two separated compact convex sets A_1 and A_2, as illustrated in Fig. 2. Since A_1 and A_2 are separated, the interior of their Minkowski difference $A_2 - A_1$ does not contain the origin and the distance between them with respect to Q is equal to $d_Q(A_2 - A_1)$. Therefore, $d_Q(A)$ can also give a distance measure for separated sets in the metric represented by Q. Without loss of generality, this paper focuses on the distance $d_Q(A)$ between the origin and a single set A in following discussion.

We can also interpret $d_Q(A)$ from another perspective. Instead of using Q to define a general metric, Q can also represent an object like A. Then, $d_Q(A)$ is the minimum scale factor of one object with respect to one of its interior points such that the scaled model is not disjoint from the other object. The two objects are separated if $d_Q(A) > 1$, in contact if $d_Q(A) = 1$, and penetrating if $d_Q(A) < 1$. In this sense, $d_Q(A)$ provides a distance measure for not only separated but also penetrating objects. Also, $d_Q(A)$ is the maximum scale factor of Q such that $d_Q(A)Q$ does not penetrate A. This could be useful in some design or planning tasks that require a certain margin from contact. Basically, $d_Q(A)$ tells how big an object can be to fit into a given environment without interfering with other existing objects.

Since both A and Q are assumed to be convex and the interior of A does not contain the origin, the definition (5) of $d_Q(A)$ is a convex programming problem, which can be solved by general-purpose numerical optimization techniques. Zhu et al. [7], [8], for example, applied Simplex method by approximating A and Q by polytopes and formulating a linear program. In general, however, methods based on general-purpose optimization techniques are computationally expensive. The contribution of this paper is an efficient algorithm that can be applied to general convex sets A and Q without approximating them as polytopes.

B. Other Concepts

Before addressing our algorithm to compute $d_Q(A)$, we would like to briefly introduce some mathematical concepts to help readers understand the derivation of our algorithm. First, a hyperplane H with normal $\boldsymbol{n}$ passing through a point $\boldsymbol{p}$ in $\mathbb{R}^n$ is a set of points given by $H = \{\boldsymbol{a} \in \mathbb{R}^n | \boldsymbol{n}^T(\boldsymbol{a} - \boldsymbol{p}) = 0\}$. A supporting hyperplane of a set A at a point $\boldsymbol{p} \in A$ is a hyperplane that passes through $\boldsymbol{p}$ and bounds A to one side of it, i.e., $\boldsymbol{n}^T(\boldsymbol{a}-\boldsymbol{p}) \leq 0$ or $\boldsymbol{n}^T(\boldsymbol{a}-\boldsymbol{p}) \geq 0$ for $\forall \boldsymbol{a} \in A$ depending on the direction of $\boldsymbol{n}$. Two sets A_1 and A_2 are said to be separated if there exists a hyperplane that bounds A_1 and A_2 to different sides, i.e., $\boldsymbol{n}^T(\boldsymbol{a} - \boldsymbol{p}) \leq 0$ for $\forall \boldsymbol{a} \in A_1$ and $\boldsymbol{n}^T(\boldsymbol{a}-\boldsymbol{p}) \geq 0$ for $\forall \boldsymbol{a} \in A_2$, and the hyperplane is called a separating hyperplane. If both inequalities hold strictly, A_1 and A_2 are strictly separated or disjoint.

From [11] the support function h_A and the support mapping $\boldsymbol{s}_A$ of a set A are defined as

$$h_A(\boldsymbol{u}) = \max_{\boldsymbol{a}\in A} \boldsymbol{u}^T\boldsymbol{a}, \quad \boldsymbol{s}_A(\boldsymbol{u}) = \arg\max_{\boldsymbol{a}\in A} \boldsymbol{u}^T\boldsymbol{a} \tag{6}$$

where $\boldsymbol{u}$ is an arbitrary vector in $\mathbb{R}^n$. Readers are referred to [2], [11] for more detailed description on the properties of h_A and $\boldsymbol{s}_A$. Closed-form expressions of h_A and $\boldsymbol{s}_A$ can often be derived from those properties depending on the expression of A. In addition, if $\boldsymbol{u}$ is nonzero, then the hyperplane with normal $\boldsymbol{u}$ passing through the point $\boldsymbol{s}_A(\boldsymbol{u})$ is a supporting hyperplane of A at $\boldsymbol{s}_A(\boldsymbol{u})$ and its Euclidean distance from the origin is $h_A(\boldsymbol{u})/\|\boldsymbol{u}\|$.

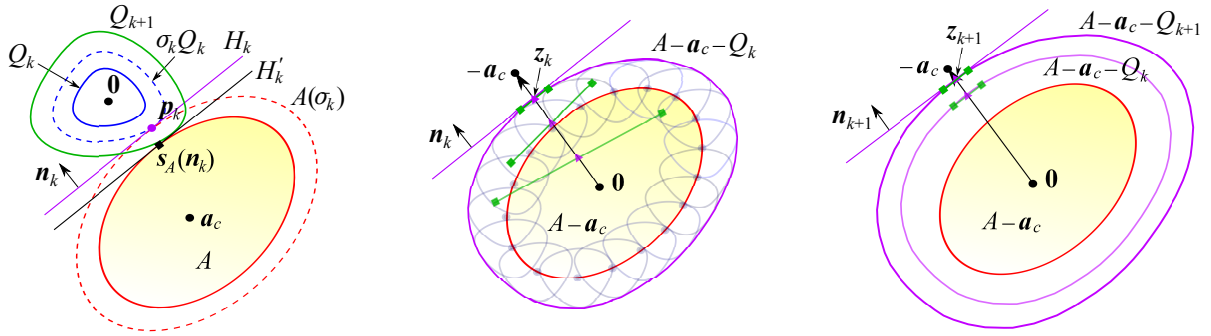


Fig. 3. Illustration in 2-D space of our algorithm for computing the generalized distance. (a) The sets A and Q_k are scaled by the same factor σ_k with respect to the point $\boldsymbol{a}_c$ and the origin, respectively, such that $A(\sigma_k)$ and $\sigma_k Q_k$ touches at $\boldsymbol{p}_k$. Hyperplane H_k with normal $\boldsymbol{n}_k$ passing through $\boldsymbol{p}_k$ separates $A(\sigma_k)$ from $\sigma_k Q_k$. Hyperplane H_k' with the same normal $\boldsymbol{n}_k$ passes through and supports A at $\boldsymbol{s}_A(\boldsymbol{n}_k)$. A scale factor of $\sigma_k Q_k$ is chosen such that Q_{k+1}, equal to the scaled $\sigma_k Q_k$, touches H_k' and is separated from A by H_k'. (b) Scale factor σ_k can be obtained by first computing the farthest intersection point $\boldsymbol{z}_k$ between set $A-\boldsymbol{a}_c-Q_k$ and the ray pointing $-\boldsymbol{a}_c$ from the origin, and then computing $\sigma_k = \|\boldsymbol{a}_c\|/\|\boldsymbol{z}_k\|$. Point $\boldsymbol{z}_k$ can be computed by an existing algorithm [12], which iteratively changes a finite subset of points (marked with green squares) in $A-\boldsymbol{a}_c-Q_k$ such that the intersection between its convex hull and the ray (marked with purple triangles) converges to $\boldsymbol{z}_k$. The algorithm also gives the normal $\boldsymbol{n}_k$ of the supporting hyperplane of $A-\boldsymbol{a}_c-Q_k$ at $\boldsymbol{z}_k$, which can then be used as $\boldsymbol{n}_k$ in (a). Moreover, a set of points in $A-\boldsymbol{a}_c-Q_k$ that contain $\boldsymbol{z}_k$ in its convex hull is computed. Those points can be used as the initial points for the algorithm [12] to speed up the computation of σ_{k+1} and $\boldsymbol{n}_{k+1}$ in the next iteration of our distance algorithm, as shown in (c).

III. An Algorithm for the Generalized Distance

A. Iteration of the Algorithm

Figure 3 and **Algorithm 1** summarizes the algorithm for computing $d_Q(A)$. This subsection gives a detailed description of the iteration process.

We denote by k the iteration number starting from 0. Let $\boldsymbol{a}_c$ be an arbitrary point in the interior of A and $A(\sigma)$ the set obtained by scaling A by a factor $\sigma \geq 0$ with respect to $\boldsymbol{a}_c$, i.e.,

$$A(\sigma) \triangleq \boldsymbol{a}_c + \sigma(A - \boldsymbol{a}_c). \tag{7}$$

Also let η_k ($\eta_0 = 1$) be another scaling factor for Q with respect to the origin, and define $Q_k = \eta_k Q$. We then define

$$\sigma_k \triangleq \min_{A(\sigma)\cap\sigma Q_k \neq \emptyset, \sigma \geq 0} \sigma \tag{8}$$

where σQ_k is the set obtained by scaling Q_k by the factor σ or scaling Q by the factor $\sigma\eta_k$ with respect to the origin. Equation (8) implies that σ_k is the minimum scale factor such that $A(\sigma_k)$ and $\sigma_k Q_k$ are not disjoint from each other, which means that $A(\sigma_k)$ and $\sigma_k Q_k$ share some boundary points but their interiors do not intersect, as depicted in Fig. 3a. If A and Q_k are separated from each other, then $\sigma_k \geq 1$; otherwise, $\sigma_k < 1$.

Since $A(\sigma_k) \cap \sigma_k Q_k \neq \emptyset$, let $\boldsymbol{p}_k$ be a point in $A(\sigma_k) \cap \sigma_k Q_k$. Then, there exists a hyperplane H_k, passing through $\boldsymbol{p}_k$, that separates $A(\sigma_k)$ from $\sigma_k Q_k$ [11]. Let $\boldsymbol{n}_k$ denote the normal of the separating hyperplane pointing to the side that $\sigma_k Q_k$ lies on. Then, $h_{\sigma_k Q_k}(-\boldsymbol{n}_k) = -\boldsymbol{n}_k^T \boldsymbol{p}_k$. With the same normal $\boldsymbol{n}_k$, we have a supporting hyperplane H_k' of A at $\boldsymbol{s}_A(\boldsymbol{n}_k)$, as shown in Fig. 3a. There could be a gap between H_k and H_k', which implies that $\sigma_k Q_k$ can be further scaled by $\frac{h_A(\boldsymbol{n}_k)}{\boldsymbol{n}_k^T \boldsymbol{p}_k}$ and the scaled set is separated by H_k' from A, where $\frac{h_A(\boldsymbol{n}_k)}{\boldsymbol{n}_k^T \boldsymbol{p}_k}$ is the ratio of the Euclidean distance of H_k' to that of H_k from the origin. Then, we set $Q_{k+1} = \eta_{k+1} Q$ with

$$\eta_{k+1} = \frac{h_A(\boldsymbol{n}_k)}{\boldsymbol{n}_k^T \boldsymbol{p}_k} \sigma_k \eta_k. \tag{9}$$

Since $h_{\sigma_k Q_k}(-\boldsymbol{n}_k) = -\boldsymbol{n}_k^T \boldsymbol{p}_k$ and $h_{\sigma_k Q_k}(-\boldsymbol{n}_k) = \sigma_k \eta_k h_Q(-\boldsymbol{n}_k)$, (9) can be rewritten as

$$\eta_{k+1} = -\frac{h_A(\boldsymbol{n}_k)}{h_Q(-\boldsymbol{n}_k)}. \tag{10}$$

At $k = 0$, we can have $\sigma_0 \geq 1$ or $\sigma_0 < 1$ because A and $Q_0 = Q$ may or may not be separated from each other. After the first iteration, however, A and Q_{k+1} will always be separated from each other from the above arguments. As a consequence, we have $\sigma_{k+1} \geq 1$ and $h_A(\boldsymbol{n}_{k+1}) \leq \boldsymbol{n}_{k+1}^T \boldsymbol{p}_{k+1} < 0$. Hence, from (9) we can deduce that $\eta_{k+2} \geq \eta_{k+1}$ and the equal sign holds only if $A \cap Q_{k+1} \neq \emptyset$, which only occurs when η_{k+1} is exactly the distance $d_Q(A)$ we are computing. If A and Q_{k+1} are strictly separated, then $\sigma_{k+1} > 1$ and $\eta_{k+2} > \eta_{k+1}$. In this case, Q_{k+2} contains Q_{k+1} in its interior, which implies $\sigma_{k+2} < \sigma_{k+1}$ from (8). If A and Q_{k+1} are always strictly separated in the iteration described by (8) and (9), then σ_{k+1} is strictly monotonically decreasing, while σ_{k+1} is bounded below by 1. By the monotone-convergence principle [13], therefore, σ_{k+1} must converge to 1 and η_{k+1} to $d_Q(A)$, and we can stop the iteration by the criterion $\sigma_{k+1} - 1 < \epsilon$, where $\epsilon > 0$ is the termination tolerance.

B. Computation Details

This subsection presents a few implementation details that affect the efficiency of the algorithm.

We first describe how to compute σ_k, $\boldsymbol{p}_k$, and $\boldsymbol{n}_k$. Since $A(\sigma) \cap \sigma Q_k \neq \emptyset$ is equivalent to $\mathbf{0} \in A(\sigma) - \sigma Q_k$, we can rewrite σ_k defined by (8) as

$$\sigma_k = \min_{-\frac{1}{\sigma}\boldsymbol{a}_c \in A - \boldsymbol{a}_c - Q_k, \sigma \geq 0} \sigma. \tag{11}$$

Algorithm 1 Algorithm for the Generalized Distance Computation

Input: compact convex sets A and Q
Output: the generalized distance $d_Q(A)$
1: $k \leftarrow 0$ and $\eta_0 \leftarrow 1$
2: compute σ_0 and $\boldsymbol{n}_0$ by a ray-shooting algorithm [12]
3: **while** $|\sigma_k - 1| > \epsilon$ **do**
4: set η_{k+1} by (10)
5: compute σ_{k+1} and $\boldsymbol{n}_{k+1}$ by a ray-shooting algorithm [12]
6: $k \leftarrow k + 1$
7: **end while**
8: **return** η_k

Equation (11) means that $-\frac{1}{\sigma_k}\boldsymbol{a}_c$ is the farthest intersection point, denoted by $\boldsymbol{z}_k$, of set $A - \boldsymbol{a}_c - Q_k$ with the ray pointing $-\boldsymbol{a}_c$ from the origin (see Fig. 3b). The computation of $\boldsymbol{z}_k$ is known as the ray-shooting problem, for which several algorithms have been developed [14], [15], [12]. As illustrated in Fig. 3b, some of the latest ray-shooting algorithms start with a finite set of points (green squares) in $A - \boldsymbol{a}_c - Q_k$, whose convex hull (green line segment) intersects with the ray, and then iteratively change the set such that the intersection point (purple triangle) converges to $\boldsymbol{z}_k$. From $\boldsymbol{z}_k = -\frac{1}{\sigma_k}\boldsymbol{a}_c$, it follows $\sigma_k = \|\boldsymbol{a}_c\|/\|\boldsymbol{z}_k\|$. The algorithm also gives a set of affinely independent points, denoted by $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_L$, in $A - \boldsymbol{a}_c - Q_k$ such that $\boldsymbol{z}_k$ can be written as their positive convex combination, i.e.,

$$-\frac{1}{\sigma_k}\boldsymbol{a}_c = \sum_{l=1}^{L} c_l \boldsymbol{s}_l \quad \text{with} \quad c_l > 0 \quad \text{and} \quad \sum_{l=1}^{L} c_l = 1. \tag{12}$$

Each point $\boldsymbol{s}_l$ is the support mapping of $A - \boldsymbol{a}_c - Q_k$ for a certain vector $\boldsymbol{u}_l$, i.e., $\boldsymbol{s}_l = \boldsymbol{s}_{A-\boldsymbol{a}_c-Q_k}(\boldsymbol{u}_l)$. From the definition (6) of support function and mapping, we can derive

$$h_{A-\boldsymbol{a}_c-Q_k}(\boldsymbol{u}_l) = h_{A-\boldsymbol{a}_c}(\boldsymbol{u}_l) + h_{Q_k}(-\boldsymbol{u}_l) \tag{13a}$$

$$\boldsymbol{s}_{A-\boldsymbol{a}_c-Q_k}(\boldsymbol{u}_l) = \boldsymbol{s}_{A-\boldsymbol{a}_c}(\boldsymbol{u}_l) - \boldsymbol{s}_{Q_k}(-\boldsymbol{u}_l). \tag{13b}$$

Substituting (13b) into (12), we can deduce

$$\boldsymbol{a}_c + \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A-\boldsymbol{a}_c}(\boldsymbol{u}_l) = \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{Q_k}(-\boldsymbol{u}_l). \tag{14}$$

From (7) and the convexity of A it follows that the left side of (14) is a point in $A(\sigma_k)$. Similarly, the right side of (14) gives a point in $\sigma_k Q_k$. Therefore, (14) implies that the point $\boldsymbol{p}_k$ in $A(\sigma_k) \cap \sigma_k Q_k$ used in (9) can be written as

$$\boldsymbol{p}_k = \boldsymbol{a}_c + \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A-\boldsymbol{a}_c}(\boldsymbol{u}_l). \tag{15}$$

Moreover, the ray-shooting algorithm determines the normal $\boldsymbol{n}$ of the supporting hyperplane of $A - \boldsymbol{a}_c - Q_k$ at $\boldsymbol{z}_k = -\frac{1}{\sigma_k}\boldsymbol{a}_c$, as depicted in Fig. 3b, which implies

$$h_{A-\boldsymbol{a}_c-Q_k}(\boldsymbol{n}) \leq -\frac{1}{\sigma_k}\boldsymbol{n}^T\boldsymbol{a}_c. \tag{16}$$

By substituting (13a) with $\boldsymbol{u}_l$ replaced by $\boldsymbol{n}$ into (16) and some mathematical manipulation, we obtain

$$h_{A(\sigma_k)}(\boldsymbol{n}) \leq -h_{\sigma_k Q_k}(-\boldsymbol{n}), \tag{17}$$

which implies that the hyperplane with normal $\boldsymbol{n}$ passing through $\boldsymbol{p}_k$ separates $A(\sigma_k)$ from $\sigma_k Q_k$ and $\boldsymbol{n}$ points to the side of the hyperplane that contains $\sigma_k Q_k$. Hence, $\boldsymbol{n}_k$ in (9) and (10) is just equal to $\boldsymbol{n}$. For the details of those ray-shooting algorithms, we refer readers to [12].

We may speed up the ray-shooting algorithm in the iteration of the proposed distance algorithm. From the discussion in Section III-A, we know that $\eta_{k+1} \geq \eta_k$ for $k \geq 1$ and it may also be true for $k = 0$, which implies that Q_{k+1} contains Q_k and $A - \boldsymbol{a}_c - Q_{k+1}$ contains $A - \boldsymbol{a}_c - Q_k$, as shown in Fig. 3c. Then, the points in $A - \boldsymbol{a}_c - Q_k$, computed by the ray-shooting algorithm, that contain the point $\boldsymbol{z}_k$ in their convex hull can be used as the initial points for the ray-shooting algorithm in the next iteration to compute the farthest intersection point $\boldsymbol{z}_{k+1}$ of $A - \boldsymbol{a}_c - Q_{k+1}$ with the ray along $-\boldsymbol{a}_c$. As the iteration proceeds, the increase of η_{k+1} from η_k is reducing. Then, the points obtained in the current iteration can be close to the result in the next iteration and give a good initialization for the ray-shooting algorithm.

Assume that our distance algorithm requires K iterations to reach the stopping criterion $|\sigma_k - 1| < \epsilon$ and in each iteration the ray-shooting algorithm takes N_k iterations to compute η_k and $\boldsymbol{n}_k$. In a space with fixed dimension, every iteration of the ray-shooting algorithm costs an almost constant number of basic operations. Hence, the computational complexity of our distance algorithm is simply $O(N_{\text{rs}})$, where $N_{\text{rs}} = \sum_{k=1}^{K} N_k$ is the total number of iterations of the chosen ray-shooting algorithm in computing $d_Q(A)$.

C. Other Results

In some applications, it is also required to determine the closest points in two sets in addition to their distance. Here we discuss how to calculate the closest points in terms of this generalized distance.

Recall that $\boldsymbol{p}_k$ in a point in $A(\sigma_k) \cap \sigma_k \eta_k Q$. As σ_k converges to 1 by the iteration, η_k converges to $d_Q(A)$ and then $\boldsymbol{p}_k$ converges to a point $\boldsymbol{v}$ in $A \cap d_Q(A)Q$. If Q is considered as an object like A, then $\boldsymbol{v}/d_Q(A)$ and $\boldsymbol{v}$ are the closest points in Q and A, respectively, in terms of the distance $d_Q(A)$. Particularly, if Q and A contact each other, then $d_Q(A) = 1$ and $\boldsymbol{v}$ is the contact point between them.

When A is the Minkowski difference $A_2 - A_1$ between sets A_1 and A_2 while Q is just used to define a metric, $d_Q(A)$ gives a distance between A_1 and A_2 that generalizes the Euclidean distance, as explained in Section II-A. The closest points in A_1 and A_2 in terms of the generalized distance $d_Q(A)$ can be determined as follows. From (15) with $A = A_2 - A_1$ and $\boldsymbol{a}_c = \boldsymbol{a}_2 - \boldsymbol{a}_1$, where $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are interior points of A_1 and A_2, respectively, we obtain

$$\boldsymbol{p}_k = \boldsymbol{a}_2 + \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A_2-\boldsymbol{a}_2}(\boldsymbol{u}_l) - \boldsymbol{a}_1 - \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A_1-\boldsymbol{a}_1}(-\boldsymbol{u}_l). \tag{18}$$

Since $c_l > 0$ for $\forall l$ and $\sum_{l=1}^{L} = 1$, $\boldsymbol{a}_{1k} \triangleq \boldsymbol{a}_1 + \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A_1-\boldsymbol{a}_1}(-\boldsymbol{u}_l)$ is a point in the set $\boldsymbol{a}_1 + \sigma_k(A_1 - \boldsymbol{a}_1)$, while $\boldsymbol{a}_{2k} \triangleq \boldsymbol{a}_2 + \sigma_k \sum_{l=1}^{L} c_l \boldsymbol{s}_{A_2-\boldsymbol{a}_2}(\boldsymbol{u}_l)$ is a point in the set $\boldsymbol{a}_2 + \sigma_k(A_2 - \boldsymbol{a}_2)$. As σ_k converges to 1, $\boldsymbol{p}_k$

TABLE I. RESULTS OF NUMERICAL TESTS

	our algorithm				active-set	
ϵ	error	t	K	$N_{\rm rs}$	error	t
10^{-2}	1.79×10^{-3}	1.81	1.78	15.03	5.78×10^{-3}	14.01
10^{-3}	1.86×10^{-4}	2.37	1.95	21.66	3.81×10^{-5}	14.29
10^{-4}	1.70×10^{-5}	2.95	2.07	28.63	1.21×10^{-6}	14.60
10^{-5}	1.75×10^{-6}	3.55	2.18	35.53	4.24×10^{-8}	14.69
10^{-6}	1.74×10^{-7}	4.13	2.26	42.67	6.39×10^{-9}	14.62
10^{-7}	1.70×10^{-8}	4.75	2.31	49.96	3.73×10^{-10}	15.12
10^{-8}	1.64×10^{-9}	5.41	2.39	57.47	7.77×10^{-11}	15.38
10^{-9}	1.57×10^{-10}	6.05	2.45	65.16	1.33×10^{-11}	15.50
10^{-10}	1.65×10^{-11}	6.62	2.51	73.02	7.67×10^{-12}	15.17

t — CPU running time (unit: millisecond);
K — average number of iterations of our distance algorithm;
$N_{\rm rs}$ — average number of total iterations of the ray-shooting algorithm [12] used in our algorithm.

TABLE II. RESULTS OF NUMERICAL TESTS

	our algorithm				active-set	
ϵ	error	t	K	$N_{\rm rs}$	error	t
10^{-2}	1.72×10^{-3}	1.84	1.78	15.18	2.10×10^{-3}	16.72
10^{-3}	1.67×10^{-4}	2.40	1.94	21.64	4.37×10^{-4}	17.61
10^{-4}	1.56×10^{-5}	2.99	2.06	28.57	1.75×10^{-4}	19.03
10^{-5}	1.52×10^{-6}	3.59	2.17	35.48	2.53×10^{-4}	20.20
10^{-6}	1.53×10^{-7}	4.20	2.25	42.65	1.47×10^{-4}	21.21
10^{-7}	1.52×10^{-8}	4.78	2.32	49.64	3.54×10^{-5}	22.48
10^{-8}	1.49×10^{-9}	5.40	2.38	57.05	3.78×10^{-6}	24.25
10^{-9}	1.44×10^{-10}	6.03	2.43	64.43	6.77×10^{-7}	24.95
10^{-10}	1.46×10^{-11}	6.76	2.49	72.40	5.93×10^{-8}	25.76

t — CPU running time (unit: millisecond);
K — average number of iterations of our distance algorithm;
$N_{\rm rs}$ — average number of total iterations of the ray-shooting algorithm [12] used in our algorithm.

converges to the closest point in $A_2 - A_1$ to the origin in terms of the distance $d_Q(A_2 - A_1)$, and $\boldsymbol{a}_{1k}$ and $\boldsymbol{a}_{2k}$ converge respectively to the closest points in A_1 and A_2 in terms of the generalized distance between them.

IV. NUMERICAL EXAMPLES

Here we report the results of some numerical tests to verify the performance of our algorithm. The distance algorithm is implemented in MATLAB on a laptop with an Intel Core i7 2.67GHz CPU and 3GB RAM.

Example 1. We first test our algorithm in 3-D space with A taken as a truncated cone and Q as an ellipsoid, as shown in Fig. 4. Their surfaces are specified by parametric functions with randomized sizes and relative positions and orientations. We place them to be in contact with each other (Fig. 4a) and shrink (Fig. 4b) or enlarge (Fig. 4c) Q by 2 times such that they become separated or penetrated. Thus, the ground truth of $d_Q(A)$ in the three cases is 1, 2, and 0.5, respectively. We generate 3000 pairs of ellipsoids and truncated cones for each case to collect the average error and CPU running time of our algorithm.

Table I lists the average absolute error between the computed $d_Q(A)$ and the ground truth, and the average CPU running time for different termination tolerance ϵ. Figure 5 plots the average error, the average CPU running time of our algorithm, and the total number of iterations of the ray-shooting algorithm for different ϵ. We also swap the models for A and Q as shown in Fig. 6, and perform the same tests. The results are summarized in Table II and plotted in Fig. 7.

As a comparison, we write $d_Q(A)$ as a convex program as in (5) with variables λ and $\boldsymbol{x}$ and compute it using the active-set algorithm provided by the Optimization Toolbox of MATLAB. To obtain a value of $d_Q(A)$ having comparable accuracy with our algorithm, we take the objective function value tolerance to be the same as the termination tolerance ϵ in our algorithm. To guarantee the satisfaction of the constraints, the tolerance on the constraint violation is set to 10^{-12}. Also, to save the computation cost of the active-set algorithm, we provide a closed-form expression of the gradient for the objective function, which is simply $[1 \;\; 0 \;\; \cdots \;\; 0]^T$. The results of the active-set algorithm are also include in Tables I and II.

From the results shown in Tables I and II it can be seen that our algorithm is several times faster than the active-set algorithm in MATLAB for computing $d_Q(A)$. The number K of iterations of our algorithm is small and increases slowly as ϵ decreases. Figs. 5 and 7 clearly show that the CPU running time of our algorithm is proportional to the total number $N_{\rm rs}$ of iterations required by the ray-shooting algorithm. Although the number $N_{\rm rs}$ is big, each iteration of the ray-shooting algorithm is straightforward and takes only a few basic operations, so that the overall efficiency of our algorithm is still high. In every iteration of the active-set algorithm, however, it is required to solve an optimization problem with equality constraints (i.e., active constraints) and update the active constraint set. Thus, the computation cost of its iteration depends on the complexity and the number of constraints. Here, some constraints are nonlinear due to the nonlinear surfaces of models. Therefore, the iteration of the active-set algorithm is much more time-consuming, which induces its relatively lower efficiency.

Note that, however, it is not straightforward to compare the computational cost because the errors of the active-set algorithm do not necessarily match the corresponding ϵ we give. In some cases, particularly in Table I, Active-set algorithm often takes more iterations to bring the constraint violation within the constraint tolerance, resulting in smaller objective function values. In other cases, on the other hand, the errors of the active-set algorithm can be larger than ϵ because the tolerance ϵ intended for the objective function is also used for other termination criteria, such as the predicted change in the objective function (i.e., the absolute value of the step length multiplied by the direction derivative of the objective function along a certain direction). Some termination criteria cannot guarantee that the error in the computed $d_Q(A)$ is less than ϵ. By contrast, it is easier to control the accuracy of our algorithm.

Example 2. We also test our algorithm in higher-dimensional spaces, where sets A and Q consist of N randomized points on two separated unit balls in n-D space, respectively. Figure 8 shows a 3-D example. Then, $d_Q(A)$ can be formulated as a linear program with lower bound constraints on $2N$ variables and $n+1$ equality constraints [7], which can be solved by the Simplex method provided by

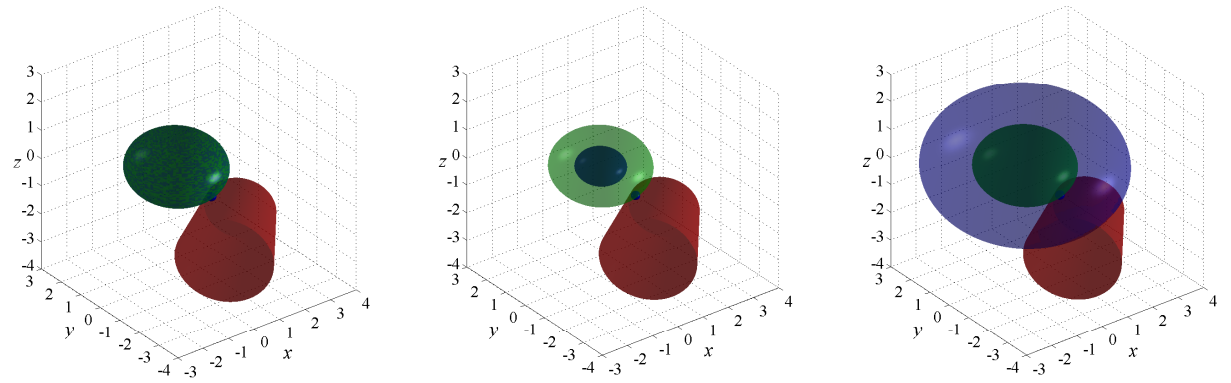


Fig. 4. 3-D models used in the tests. The red, blue, and green models represent the set A, Q, and the scaled set $d_Q(A)Q$, respectively. The black dot denotes the contact point between A and $d_Q(A)Q$. (a) A contacts Q so that the ground truth of $d_Q(A)$ is 1. (b) A and Q are separated and the ground truth of $d_Q(A)$ is 2. (c) A and Q penetrate each other and the ground truth of $d_Q(A)$ is 0.5.

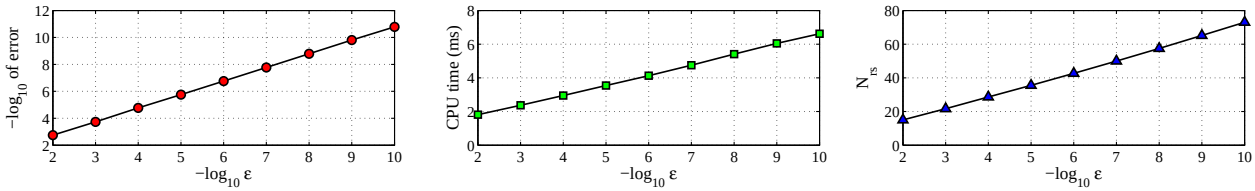


Fig. 5. Graphs showing (a) the error, (b) the CPU running time, and (c) the total number of iterations required by the ray-shooting algorithm with respect to the termination tolerance ϵ, where A is a truncated cone and Q is an ellipsoid, as shown in Fig. 4. The values are the average for 9000 random tests.

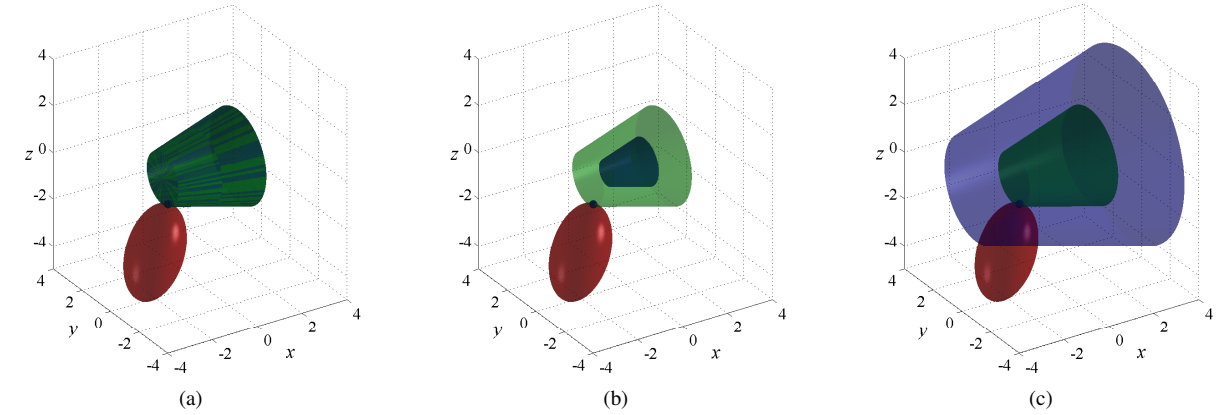


Fig. 6. 3-D models used in the tests. The red and blue models represent the sets A and Q, respectively, which can be (a) in contact with, (b) separated from, or (c) penetrating each other. The green model represents $d_Q(A)Q$ and the black dot indicates the contact point between A and $d_Q(A)Q$. In the three cases, the ground truth of $d_Q(A)$ is 1, 2, and 0.5, respectively.

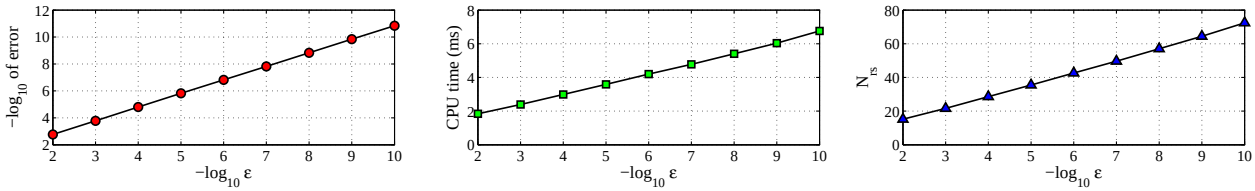
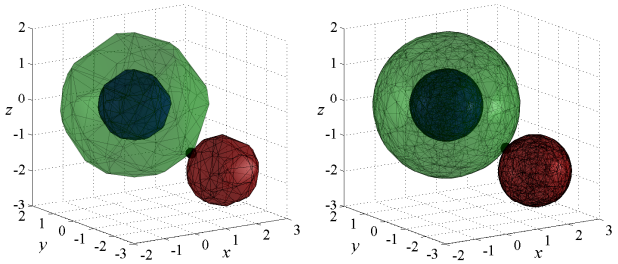


Fig. 7. Graphs showing (a) the error, (b) the CPU running time, and (c) the total number of iterations required by the ray-shooting algorithm with respect to the termination tolerance ϵ, where A is an ellipsoid and Q is a truncated cone, as shown in Fig. 6. The values are the average for 9000 random tests.

TABLE III. CPU RUNNING TIME OF OUR ALGORITHM AND COMPARISON WITH THE SIMPLEX METHOD (unit: millisecond)

	$N = 200$	400	600	800	1000	1200	1400	1600	1800	2000
$n = 3$	1.36 (0.19)	1.52 (0.15)	1.70 (0.12)	1.83 (0.11)	1.96 (0.09)	2.11 (0.08)	2.27 (0.08)	2.36 (0.07)	2.51 (0.07)	2.66 (0.06)
4	2.21 (0.27)	2.49 (0.20)	2.73 (0.16)	2.99 (0.14)	3.17 (0.12)	3.44 (0.11)	3.64 (0.11)	3.87 (0.10)	3.96 (0.09)	4.30 (0.09)
5	3.44 (0.37)	3.79 (0.27)	4.29 (0.22)	4.51 (0.18)	4.78 (0.16)	5.13 (0.15)	5.46 (0.14)	5.78 (0.13)	6.00 (0.12)	6.32 (0.11)
6	5.20 (0.48)	5.68 (0.34)	6.26 (0.28)	6.58 (0.23)	7.10 (0.20)	7.65 (0.19)	7.91 (0.17)	8.40 (0.16)	8.61 (0.15)	9.16 (0.14)
7	7.04 (0.59)	8.19 (0.42)	8.91 (0.34)	9.50 (0.29)	10.05 (0.25)	10.90 (0.23)	11.56 (0.21)	11.64 (0.19)	12.34 (0.18)	13.17 (0.17)
8	9.54 (0.69)	10.89 (0.50)	11.67 (0.40)	12.68 (0.34)	13.48 (0.29)	14.36 (0.27)	15.12 (0.24)	15.64 (0.22)	16.71 (0.22)	17.39 (0.20)
9	13.35 (0.87)	15.21 (0.62)	16.78 (0.50)	17.95 (0.43)	18.67 (0.36)	20.13 (0.33)	20.62 (0.30)	21.74 (0.28)	22.92 (0.25)	23.57 (0.24)
10	17.48 (0.99)	19.92 (0.72)	22.10 (0.59)	22.78 (0.48)	24.46 (0.42)	25.73 (0.37)	26.34 (0.33)	27.86 (0.31)	28.93 (0.29)	30.04 (0.27)

N — number of points in A (Q consists of the same number of points);
n — dimension of space;
The values between parentheses are the ratios of the CPU running time of our algorithm to that of the Simplex method.



(a) (b)

Fig. 8. A 3-D illustration of polytopes with vertices randomly picked from two separated unit balls. The red and blue polytopes are taken as A and Q, respectively. The green polytope depicts $d_Q(A)Q$ and the black dot indicates the contact point between A and $d_Q(A)Q$. The more vertices, the closer the polytopes are to the balls. (a) 200 vertices for each polytope. (b) 1000 vertices for each polytope.

MATLAB. Here, we set the same termination tolerance $\epsilon = 10^{-6}$ for our algorithm and the Simplex method and take the Simplex method as a reference to verify the performance of our algorithm for different N and n. The results are listed in Table III, where we can see that the running time of our algorithm is more sensitive to the space dimension n than the Simplex method but much less to the number of points in A and Q. Especially in low-dimensional (e.g., 3-D or 4-D) space with a large number (e.g., thousands) of points, our algorithm is much faster than the Simplex method.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a novel geometry-based algorithm for computing general distance measure between two compact convex sets. The measure was proposed almost a decade ago but no efficient algorithm has been known for its computation. Through numerical examples with models given by parametric functions or discrete points in spaces with 3 or more dimensions, we verified that our algorithm is faster than approaches based on general-purpose optimization techniques. Its efficiency can be further enhanced in a dynamic situation where the locations of sets are time-varying by using the frame-coherence property as some distance algorithms do [3]–[6]. Applications of the distance algorithm in robotics include motion planning and physics simulation. We would like to explore these applications and possibly improve the efficiency of the algorithm in a specific application in our future work.

REFERENCES

[1] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Transactions on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[2] E. G. Gilbert and C. P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 53–61, 1990.

[3] M. Lin and J. Canny, "A fast algorithm for incremental distance calculation," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Sacramento, CA, April 1991, pp. 1008–1014.

[4] S. A. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Albuquerque, NM, April 1997, pp. 3112–3117.

[5] B. Mirtich, "V-clip: Fast and robust polyhedral collision detection," *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177–208, 1998.

[6] C. J. Ong and E. G. Gilbert, "Fast versions of the Gilbert-Johnson-Keerthi distance algorithm: additional results and comparisons," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 531–539, 2001.

[7] X.-Y. Zhu and J. Wang, "Synthesis of force-closure grasps on 3-D objects based on the Q distance," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 4, pp. 669–679, 2003.

[8] X.-Y. Zhu, H. Ding, and S. K. Tso, "A pseudodistance function and its applications," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 2, pp. 344–352, 2004.

[9] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Machines*, B. Schölkopf, C. J. C. Burges, and A. Smola, Eds. MIT Press, 1998.

[10] S. S. Keerthi, S. K. Shevade, C. Bhattachayya, and K. R. K. Murth, "Improvements to platts SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, no. 4, pp. 637–649, 2001.

[11] S. R. Lay, *Convex Sets and their Applications*. New York, NY, USA: John Wiley & Sons, 1982.

[12] Y. Zheng and K. Yamane, "Ray-shooting algorithms for robotics," in *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, Cambridge, MA, June 2012, in press.

[13] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York, NY, USA: McGraw-Hill, 1976.

[14] Y. Zheng and C.-M. Chew, "A numerical solution to the ray-shooting problem and its applications in robotic grasping," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009, pp. 2080–2085.

[15] Y. Zheng, M. C. Lin, and D. Manocha, "A fast n-dimensional ray-shooting algorithm for grasping force optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, May 2010, pp. 1300–1305.