2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

A Fast n-Dimensional Ray-Shooting Algorithm for Grasping Force Optimization

Yu Zheng, Ming C. Lin, and Dinesh Manocha*

Abstract

We present an efficient algorithm for solving the ray-shooting problem on high dimensional sets. Our algorithm computes the intersection of the boundary of a compact convex set with a ray emanating from an interior point of the set and represents the intersection point as a convex combination of a set of affinely independent points. We use our intersection algorithm to compute two types of optimal grasping forces, where either the sum or the maximum of normal force components is minimized. In our simulation, the algorithm converges well and performs the computations in tens of milliseconds on a laptop.

1. INTRODUCTION

Given a collection of surfaces and objects in $\mathbb{R}^n$, the ray-shooting problem deals with computing the first intersection point on the boundary of the objects by a query ray. This problem has been well studied in computational geometry and computer graphics over the last four decades [1]–[7]. More recently, there has been considerable interest in ray shooting and related problems in terms of designing efficient grasping algorithms, such as grasping force optimization [8], test and synthesis of force-closure grasps [8], [9] as well as some problems in fixturing [10]. In grasping, the ray-shooting problem needs to handle a high-dimensional compact convex set specified by implicit non-linear functions, rather than low-dimensional polytopes with given vertices or facets. Some problems must be solved in real time; then the computational efficiency is a challenging issue.

*The authors are with the Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599 USA {yuzheng,lin,dm}@cs.unc.edu. This work was supported by ARO Contract W911NF-04-1-0088, NSF awards 0636208, 0917040 and 0904990, DARPA/RDECOM Contract WR91CRB-08-C-0137, and Intel.

1.1. Related Work

Some ray-shooting algorithms were designed to determine the first facet of a polytope hit by a ray [1]. Matoušek and Schwarzkopf [3] presented efficient data structures to reduce the ray-shooting query time and avoid the parametric search used in [2]. This work was later improved by Chan [4]. Szirmay-Kalos *et al.* [5] compared various ray-shooting acceleration schemes used in computer graphics. Though the ray-shooting problem has been investigated for years, most practical algorithms are limited to 3-D polytopes or low-dimensional objects. van den Bergen [6], Zheng and Chew [7] independently proposed a ray-shooting algorithm applicable to general convex sets and apply it to collision detection and grasping problems, respectively.

Grasping: In grasping research, Liu *et al.* [8]–[10] formulated many problems in terms of the ray-shooting problem between the convex hull of primitive contact wrenches and a vector in 6-D wrench space. Using the linearized friction model and the duality principle, they cast the ray-shooting problem as a 6-dimensional linear programming problem and solved it by the simplex method [8]. This approach can be efficient when the problem size or the number of primitive contact wrenches is small. However, using fewer primitive contact wrenches reduces the solution accuracy. In grasping force optimization, linearizing the friction model can also result in a discontinuous solution. In addition, if the maximum contact force needs to be minimized, one may need to calculate the Minkowski sum of the sets of primitive contact wrenches for different contacts, which can exponentially increase the problem size with the number of contacts [11]. There have been attempts to solve the ray-shooting problem in 6-D wrench space without linearizing the friction model [7], [12], but the computation times can be considerably high.

1.2. Main Results

In this paper, we present a novel algorithm for solving the ray-shooting problem. Similar to [6], [7], our al-

gorithm is applicable to general convex sets, including polytopes, in n-dimensional space as long as their support functions and support mappings can be calculated. By contrast, our algorithm does not perform GJK-based distance computation [13] at the iteration. Instead, we seek a facet of a simplex inside the convex set intersected with the ray and construct a new simplex using this facet and the support mapping of the convex set along the facet normal. We show that this iterative process can quickly approach the intersection point of the boundary of the convex set with the ray, as the support function of the convex set along the facet normal decreases to unity. The computational complexity for determining the facet of a simplex intersected with a ray depends only on the dimension of space. With the aid of useful properties given in [14], calculating the support function and mapping of a convex set often requires just a few basic operations. Therefore, in each iteration of our algorithm, the computation cost is much lower than [6], [7]. To demonstrate its performance, we use our ray-shooting algorithm with respect to two different convex sets to minimize the sum or the maximum of normal forces in grasping force optimization. Taking only few milliseconds on a laptop, it can solve this problem an order of magnitude faster than the earlier algorithm [7].

The rest of the paper is organized as follows. Section 2 presents our ray-shooting algorithm. Section 3 discusses its application to grasping force optimization. Section 4 describes its implementation and performance.

2. THE RAY-SHOOTING ALGORITHM

2.1. Notations and Problem Statement

Let A be a compact set in $\mathbb{R}^n$ and co$A$ the convex hull of A. The set A is given in specific problems (see Section 3 for an example). Let $\boldsymbol{r}$ be a point in $\mathbb{R}^n$ other than the origin $\mathbf{0}$ and R the ray from $\mathbf{0}$ through $\boldsymbol{r}$:

$$R = \{\lambda \boldsymbol{r} \in \mathbb{R}^n \mid \lambda \geq 0\}. \tag{1}$$

The ray-shooting problem is to determine the intersection point of the boundary of co$A$ and R, denoted by $\boldsymbol{s}$, and a set of affinely independent points in A, denoted by S_A, such that $\boldsymbol{s}$ can be written as its convex combination. We assume that co$A$ contains the origin $\mathbf{0}$ of $\mathbb{R}^n$ as an interior point, so that co$A$ and R have an intersection.

To solve the ray-shooting problem, we will use the support function of A, $h_A : \mathbb{R}^n \to \mathbb{R}$, which is defined by

$$h_A(\boldsymbol{u}) = \max_{\boldsymbol{a} \in A} \boldsymbol{u}^T \boldsymbol{a} \tag{2}$$

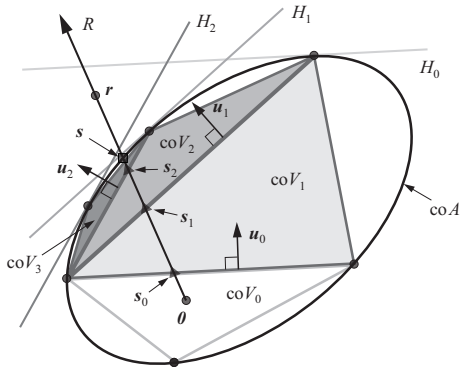

Fig. 1: Principle of the ray-shooting algorithm. R is the ray starting from the origin $\mathbf{0}$ inside the given convex hull co$A$ through a given point $\boldsymbol{r}$. $\boldsymbol{s}$ is the intersection point on the boundary of co$A$ with R to be determined. H_0, H_1, H_2 are the hyperplanes supporting co$A$ with normal $\boldsymbol{n}_1, \boldsymbol{n}_2, \boldsymbol{n}_3$, which are the outward normals to the facets of simplices coV_0, coV_1, coV_2 hit by R.

where $\boldsymbol{u} \in \mathbb{R}^n$. The support mapping, $s_A : \mathbb{R}^n \to A$, finds a point $s_A(\boldsymbol{u}) \in A$ such that $h_A(\boldsymbol{u}) = \boldsymbol{u}^T s_A(\boldsymbol{u})$. The support function h_A together with $\boldsymbol{u}$ defines a hyperplane H with normal $\boldsymbol{u}$ supporting co$A$ at the point $s_A(\boldsymbol{u})$ [15]:

$$H = \{\boldsymbol{a} \in \mathbb{R}^n \mid \boldsymbol{u}^T \boldsymbol{a} = h_A(\boldsymbol{u})\}. \tag{3}$$

2.2. Iterations of the Ray-Shooting Algorithm

Here, we denote by subscript l the iteration number. First, let $l = 0$ and V_0 be an initial set, consisting of $n+1$ affinely independent points in co$A$, denoted by $\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{n+1}$, such that co$V_0$ is a simplex containing $\mathbf{0}$ in its interior. Let $F_i = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}, \boldsymbol{a}_{i+1}, \ldots, \boldsymbol{a}_{n+1}\}$. Then co$F_i$ is the i-th facet of coV_0. Let

$$\boldsymbol{c} = \boldsymbol{W}_i^{-1} \boldsymbol{r} \tag{4}$$

where $\boldsymbol{W}_i = \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_{i-1} & \boldsymbol{a}_{i+1} & \cdots & \boldsymbol{a}_{n+1} \end{bmatrix} \in \mathbb{R}^{n \times n}$. Let $\mu(\boldsymbol{c})$ and $\sigma(\boldsymbol{c})$ denote the minimum and the sum of components of $\boldsymbol{c}$, respectively. It can be proved that R intersects coF_i if and only if $\mu(\boldsymbol{c}) \geq 0$. Furthermore, the intersection point is given by $\boldsymbol{r}/\sigma(\boldsymbol{c})$.

The normal $\boldsymbol{u}_i$ to coF_i can be calculated by

$$\boldsymbol{u}_i = \boldsymbol{W}_i^{-T} \boldsymbol{u}_0 \tag{5}$$

where $\boldsymbol{u}_0 = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T \in \mathbb{R}^n$. Let $\bar{i}$ designate a facet co$F_{\bar{i}}$ intersected with R. We update the set V_l by

$$V_l = F_{\bar{i}} \bigcup \{s_A(\boldsymbol{u}_{\bar{i}})\}. \tag{6}$$

The iteration can stop when $h_A(\boldsymbol{u}_{\bar{i}}) = \boldsymbol{u}_{\bar{i}}^T \boldsymbol{a}$, where $\boldsymbol{a}$ is any element of $F_{\bar{i}}$. Indeed, $(h_A(\boldsymbol{u}_{\bar{i}}) - \boldsymbol{u}_{\bar{i}}^T \boldsymbol{a})/\|\boldsymbol{u}_{\bar{i}}\|$ is the distance between the facet co$F_{\bar{i}}$ and the supporting hyperplane H_l to co$A$ with normal $\boldsymbol{u}_{\bar{i}}$, which is parallel

Algorithm 1 Ray-Shooting Algorithm

Input: A point set A with $\mathbf{0}$ inside co$A$ and a point $\boldsymbol{r}$
Output: Intersection point $\boldsymbol{s}$ of the boundary of co$A$ and R, and an affinely independent subset S_A of A such that $\boldsymbol{s} \in \text{co}S_A$

$\boldsymbol{u}_0 \leftarrow \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$
$V \leftarrow$ a set of $n+1$ points in co$A$ such that co$V$ is a simplex containing $\mathbf{0}$ as an interior point
repeat
  $i \leftarrow 0$
  repeat
    $i \leftarrow i+1$
    $\boldsymbol{W} \leftarrow$ the matrix comprising all points in V except the i-th one
    $\boldsymbol{c} \leftarrow \boldsymbol{W}^{-1}\boldsymbol{r}$
  until $\mu(\boldsymbol{c}) \geq 0$
  remove the i-th point from V
  $\boldsymbol{u} \leftarrow \boldsymbol{W}^{-T}\boldsymbol{u}_0$
  add $s_A(\boldsymbol{u})$ to V as its last point
until $h_A(\boldsymbol{u}) - 1 < \varepsilon$
return $\boldsymbol{r}/\sigma(\boldsymbol{c})$ and V

to coF_i. If $h_A(\boldsymbol{u}_{\bar{i}}) = \boldsymbol{u}_{\bar{i}}^T\boldsymbol{a}$, then H_l contains co$F_{\bar{i}}$, which implies that co$F_{\bar{i}}$ is on the boundary of co$A$ and its intersection point $\bar{\boldsymbol{r}}_l$ with R is just the intersection point $\boldsymbol{s}$ we are seeking, as illustrated in Fig. 1. In addition, from (5) it follows that $\boldsymbol{u}_{\bar{i}}^T\boldsymbol{a} = 1$ for all $\boldsymbol{a} \in F_{\bar{i}}$. In practice, therefore, we may adopt $h_A(\boldsymbol{u}_{\bar{i}}) - 1 < \varepsilon$ as the termination condition, which ε is the termination tolerance.

2.3. Computation Cost in Each Iteration

In every iteration the algorithm needs to determine the facet of a simplex intersected with R. In the worst case, all $n+1$ facets of the initial simplex coV_0 are checked, while later only the newly formed n facets are necessary. For each facet, $\boldsymbol{c}$ is calculated by (4) and the condition $\mu(\boldsymbol{c}) \geq 0$ is verified. By the Gaussian elimination, the inverse $\boldsymbol{W}_i^{-1}$ of $\boldsymbol{W}_i$ can be computed in $O(n^3)$ complexity; then the computational complexity for checking a facet is $O(n^3)$. Thus for a simplex, the facet intersected with R can be found in $O(n^4)$ in the worst case. Since $\boldsymbol{W}_i^{-1}$ is obtained in computing (4), calculating $\boldsymbol{u}_{\bar{i}}$ by (5) requires only $O(n^2)$ operations.

Also, the algorithm calculates $h_A(\boldsymbol{u}_{\bar{i}})$ and $s_A(\boldsymbol{u}_{\bar{i}})$ for checking the termination condition and constructing a new simplex in each iteration. Their complexities depend on the complexity of the given set A in the problems. With the aid of useful properties indicated in [14], the computation can be straightforward and simple.

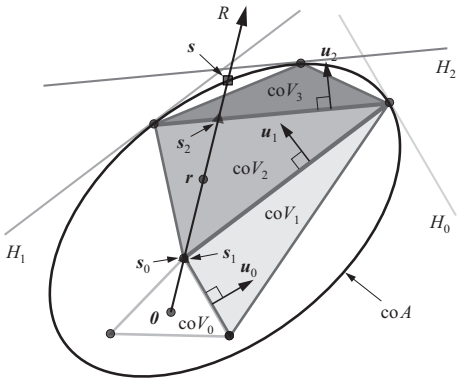

Fig. 2: Singular case in the iteration. R passes through a vertex of coV_0 such that that $\boldsymbol{s}_1 = \boldsymbol{s}_0$. $\boldsymbol{s}_2$ in the 2-nd iteration steps towards $\boldsymbol{s}$.

2.4. Discussion on the Convergence

Next we discuss the convergence of the algorithm. We classify the computation into two cases.

Case 1 (regular case): R intersects the relative interior of a facet of coV_l in each iteration (see Fig. 2). In this case, the intersection point $\bar{\boldsymbol{r}}_{l+1}$ in the next iteration must lie on a new facet and step further from $\mathbf{0}$. Then the value $\sigma(\boldsymbol{c}_l) = \|\boldsymbol{r}\|/\|\bar{\boldsymbol{r}}_l\|$ is strictly decreasing. As $\bar{\boldsymbol{r}}_l \in \text{co}A$, eventually $\bar{\boldsymbol{r}}_l$ will converge to the intersection point $\boldsymbol{s}$ and $h_A(\boldsymbol{u}_{\bar{i}})$ will converge to unity; simultaneously $\sigma(\boldsymbol{c}_l)$ will converge to its minimum value. Hence, the convergence is well guaranteed in this case.

Case 2 (singular case): Occasionally, R does not pass through the relative interior of any facet of coV_l but through a face with dimension lower than $n-1$. For example, R passes through a point in V_0, i.e., a vertex of coV_0 (see Fig. 2). Then $\boldsymbol{c}$ determined by (4) satisfies $\mu(\boldsymbol{c}) \geq 0$ for more than one facet. Currently, the proposed algorithm just adopts the first one found in each iteration. As described in Fig. 2, the intersection point $\bar{\boldsymbol{r}}_l$ on the boundary of coV_l with R does not change in the next few iterations. However, the iteration can recover to the regular case automatically, as tested in Section 4.

3. GRASPING FORCE OPTIMIZATION

In this section, we apply the ray-shooting algorithm in 6-D wrench space to grasping force optimization.

3.1. Preliminaries in Grasping

Assume that an object is grasped by m contacts, which can be frictionless point contacts (FPC), point contacts with friction (PCwF), and/or soft finger contacts (SFC). Let $\boldsymbol{p}_i$ $(i = 1, 2, \ldots, m)$ be the position vector of contact i, $\boldsymbol{n}_i$ the unit inward normal, and $\boldsymbol{o}_i$ and $\boldsymbol{t}_i$

two unit tangent vectors satisfying $\boldsymbol{n}_i = \boldsymbol{o}_i \times \boldsymbol{t}_i$. Then the contact force $\boldsymbol{f}_i$ is expressed as $\boldsymbol{f}_i = \begin{bmatrix} f_{i1} & f_{i2} & f_{i3} & f_{i4} \end{bmatrix}^T$, where f_{i1}, f_{i2}, f_{i3} are the force components along $\boldsymbol{n}_i$, $\boldsymbol{o}_i$, $\boldsymbol{t}_i$, respectively, and f_{i4} is the spin moment about $\boldsymbol{n}_i$.

To maintain contact and avoid slippage, $\boldsymbol{f}_i$ must be within the following set [16], [17]:

$$F_i = \left\{ \lambda \boldsymbol{f}_i \mid \lambda \geq 0,\ \boldsymbol{f}_i \in U_i \right\}$$

where U_i is the primitive contact force set [7], given by

$$\text{FPC} : U_i = \left\{ \boldsymbol{f}_i \mid f_{i1} = 1,\ f_{i2} = f_{i3} = f_{i4} = 0 \right\}$$

$$\text{PCwF} : U_i = \left\{ \boldsymbol{f}_i \mid f_{i1} = 1,\ \sqrt{f_{i2}^2 + f_{i3}^2} = \mu_i,\ f_{i4} = 0 \right\}$$

$$\text{SFC} : U_i = \left\{ \boldsymbol{f}_i \mid f_{i1} = 1,\ \sqrt{\frac{f_{i2}^2 + f_{i3}^2}{\mu_i^2} + \frac{f_{i4}^2}{\mu_{si}^2}} = 1 \right\}$$

where μ_i is the tangential friction coefficient and μ_{si} is the torsional friction coefficient for elliptic SFC model [17]. Then the primitive contact wrench set is $W_i = \boldsymbol{G}_i(U_i)$, where $\boldsymbol{G}_i = \begin{bmatrix} \boldsymbol{n}_i & \boldsymbol{o}_i & \boldsymbol{t}_i & \boldsymbol{0} \\ \boldsymbol{p}_i \times \boldsymbol{n}_i & \boldsymbol{p}_i \times \boldsymbol{o}_i & \boldsymbol{p}_i \times \boldsymbol{t}_i & \boldsymbol{n}_i \end{bmatrix}$ is the grasp matrix. Two grasp wrench sets are widely used in grasping research [18], [11], which are defined as

$$W_{L_1} = \text{co}\left\{ \bigcup_{i=1}^{m} W_i \right\}$$

$$W_{L_\infty} = \text{co}\left\{ \bigcup_{k=1}^{m} \bigcup_{i_1 < i_2 < \ldots < i_k = 1}^{m} (W_{i_1} \oplus W_{i_2} \oplus \cdots \oplus W_{i_k}) \right\}.$$

Both W_{L_1} and W_{L_∞} are compact convex sets in $\mathbb{R}^6$ with non-linear boundaries and often contain the origin $\boldsymbol{0}$ in their interiors, as grasps are usually force-closure.

To maintain the grasped object in equilibrium, the resultant wrench $\boldsymbol{w}_{\text{res}}$ from contacts and the sum of the other wrenches, denoted by $\boldsymbol{w}_{\text{ext}}$, must satisfy [19]

$$\boldsymbol{w}_{\text{res}} = \sum_{i=1}^{m} \boldsymbol{G}_i \boldsymbol{f}_i = -\boldsymbol{w}_{\text{ext}}. \tag{7}$$

Contact force optimization is to compute the minimum contact forces $\boldsymbol{f}_i \in F_i$, $i = 1, 2, \ldots, m$ satisfying (7). In what follows, we will use our ray-shooting algorithm with W_{L_1} and W_{L_∞} to yield two kinds of minimum contact forces. As required in the algorithm, the computation of the support function and mapping of W_{L_1} or W_{L_∞} has been derived in [7], [20], [21].

3.2. Contact Force Minimization

According to [8], [11], [16], the overall contact force magnitude can be measured by

$$\sigma_{L_1} = \sum_{i=1}^{m} f_{i1} \quad \text{or} \quad \sigma_{L_\infty} = \max_{1 \leq i \leq m} f_{i1}.$$

Using our ray-shooting algorithm with W_{L_1} (or W_{L_∞}) and the ray along $\boldsymbol{w}_{\text{res}}$, we obtain a set of points $\boldsymbol{w}_k \in W_{L_1}$ (or $\boldsymbol{w}_k \in W_{L_\infty}$), $k = 1, 2, \ldots, K$ and the non-negative coefficients $\boldsymbol{c} = \begin{bmatrix} c_1 & c_2 & \cdots & c_K \end{bmatrix}^T$ such that $\boldsymbol{w}_{\text{res}} = \sum_{k=1}^{K} c_k \boldsymbol{w}_k$, where $\boldsymbol{w}_k$ is the support mapping $s_{W_{L_1}}$ (or $s_{W_{L_\infty}}$) in a certain direction obtained in an iteration. We also attain an index $\hat{i}_k$ (or an index set $\hat{I}_k$) and a point $\boldsymbol{s}_{\hat{i}_k} \in U_{\hat{i}_k}$ (or points $\boldsymbol{s}_{i_k} \in U_{i_k}$, $i_k \in \hat{I}_k$) such that $\boldsymbol{w}_k = \boldsymbol{G}_{\hat{i}_k} \boldsymbol{s}_{\hat{i}_k}$ (or $\boldsymbol{w}_k = \sum_{i_k \in \hat{I}_k} \boldsymbol{G}_{i_k} \boldsymbol{s}_{i_k}$). Then

$$\boldsymbol{w}_{\text{res}} = \sum_{k=1}^{K} c_k \boldsymbol{G}_{\hat{i}_k} \boldsymbol{s}_{\hat{i}_k} \quad \text{or} \quad \boldsymbol{w}_{\text{res}} = \sum_{k=1}^{K} \left(c_k \sum_{i_k \in \hat{I}_k} \boldsymbol{G}_{i_k} \boldsymbol{s}_{i_k} \right),$$

which implies that $\boldsymbol{f}_i$ can be expressed by

$$\boldsymbol{f}_i = \sum_{k=1}^{K} c_k \boldsymbol{s}_k \text{ with } \boldsymbol{s}_k = \begin{cases} \boldsymbol{s}_{\hat{i}_k} \text{ (or } \boldsymbol{s}_{i_k}) & \text{if } i = \hat{i}_k \text{ (or } i = i_k \in \hat{I}_k) \\ \boldsymbol{0} & \text{otherwise} \end{cases}.$$

It can be proved that such $\boldsymbol{f}_i$, $i = 1, 2, \ldots, m$ have the minimum σ_{L_1} (or σ_{L_∞}).

4. NUMERICAL EXAMPLES

The proposed algorithm is implemented in MATLAB on a laptop with Pentium-M 1.86GHz CPU and 512MB RAM, and tested with numerical examples.

4.1. Grasp and Algorithm Parameters

We borrow a grasp example used in [7], where the contact positions and normals are as below:

$$\boldsymbol{r}_1 = \begin{bmatrix} 0 & -3/2 & \sqrt{3}/2 \end{bmatrix}^T,\ \boldsymbol{n}_1 = \begin{bmatrix} 0 & \sqrt{3}/2 & 1/2 \end{bmatrix}^T;$$
$$\boldsymbol{r}_2 = \begin{bmatrix} 0 & 3/2 & \sqrt{3}/2 \end{bmatrix}^T,\ \boldsymbol{n}_1 = \begin{bmatrix} 0 & -\sqrt{3}/2 & 1/2 \end{bmatrix}^T;$$
$$\boldsymbol{r}_3 = \begin{bmatrix} 3\sqrt{3}/8 & 0 & 3\sqrt{3}/4 \end{bmatrix}^T,\ \boldsymbol{n}_1 = \begin{bmatrix} -3/5 & 0 & -4/5 \end{bmatrix}^T;$$
$$\boldsymbol{r}_4 = \begin{bmatrix} -3\sqrt{3}/8 & 0 & 3\sqrt{3}/4 \end{bmatrix}^T,\ \boldsymbol{n}_1 = \begin{bmatrix} 3/5 & 0 & -4/5 \end{bmatrix}^T.$$

The contact is SFC, where $\mu_i = 0.2$ and $\mu_{si} = 0.2$ mm.

The initial set V_0 for our ray-shooting algorithm is taken to consist of the vertices of a 6-D regular simplex centered at the origin of $\mathbb{R}^6$, which are given by

$$\boldsymbol{a}_v = \left(\boldsymbol{a}_v^0 - \frac{1}{7} \sum_{v=1}^{7} \boldsymbol{a}_v^0 \right) / 10 \| \boldsymbol{a}_v^0 - \frac{1}{7} \sum_{v=1}^{7} \boldsymbol{a}_v^0 \|,\ v = 1, 2, \ldots, 7$$

where $\boldsymbol{a}_1^0 = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_2^0 = \begin{bmatrix} -1 & -1 & 1 & 0 & 0 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_3^0 = \begin{bmatrix} -1 & 1 & -1 & 0 & 0 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_4^0 = \begin{bmatrix} 1 & -1 & -1 & 0 & 0 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_5^0 = \begin{bmatrix} 0 & 0 & 0 & \sqrt{5} & 0 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_6^0 = \begin{bmatrix} 0 & 0 & 0 & \sqrt{5}/5 & 2\sqrt{30}/5 & 0 \end{bmatrix}^T$, $\boldsymbol{a}_7^0 = \begin{bmatrix} 0 & 0 & 0 & \sqrt{5}/5 & 2/\sqrt{30} & \sqrt{42}/3 \end{bmatrix}^T$. We set the termination tolerance $\varepsilon = 10^{-5}$.

TABLE 1: RESULTS OF METHODS (i)-(iii) TO MINIMIZE σ_{L_1}

$\boldsymbol{w}_{\text{res}}$	(i) ours				(ii)	(iii)
	σ_{L_1}	σ_{L_∞}	N	time (ms)	time (ms)	time (ms)
1	3.5725	1.5027	46	10.75	105.90	210.42
2	4.0356	1.5394	40	8.18	77.83	122.74
3	3.3020	1.2421	32	7.28	95.96	184.87
4	5.2178	1.8060	47	10.05	89.28	106.91
5	3.7799	1.3472	46	9.36	80.03	87.53
$6^{(a)}$	4.9394	1.5591	48	12.23	97.38	126.44
7	3.6473	1.1326	39	8.55	26.78	100.53
$8^{(b)}$	22.1498	10.8535	39	7.49	101.44	88.80

N—the number of required iterations
(i)—the ray-shooting algorithm presented in this paper
(ii)—the ray-shooting algorithm in [7]
(iii)—the function **fmincon** of MATLAB to solve an optimization problem

TABLE 2: RESULTS OF METHODS (i)-(iii) TO MINIMIZE σ_{L_∞}

$\boldsymbol{w}_{\text{res}}$	(i) ours				(ii)	(iii)
	σ_{L_1}	σ_{L_∞}	N	time (ms)	time (ms)	time (ms)
1	3.6944	1.4513	69	17.70	154.59	150.79
2	4.1380	1.5005	63	17.47	256.53	103.34
3	3.6321	1.1688	68	19.26	478.42	118.84
4	5.7457	1.6753	75	21.17	300.84	112.20
5	3.9094	1.3091	52	12.78	254.48	85.44
$6^{(c)}$	5.0119	1.5360	75	21.81	180.72	108.56
7	3.7149	1.1118	67	17.94	2.828	105.53
$8^{(d)}$	25.8958	10.4439	41	9.86	186.266	117.28

4.2. Numerical Results

First, we respectively assign $\boldsymbol{w}_{\text{res}} = 10\boldsymbol{a}_v$ for $v = 1,2,\ldots,7$ and $\boldsymbol{w}^8_{\text{res}} = \begin{bmatrix}3 & 2 & -10 & 0 & 0 & 1\end{bmatrix}^T$. Tables 1 and 2 exhibit the results and CPU times of our ray-shooting algorithm used to determine the two minimum grasping forces with respect to W_{L_1} and W_{L_∞}, respectively. It can be seen that the CPU times for minimizing σ_{L_1} and σ_{L_∞} are not much different, since the operation counts for computing the support functions and mapping of W_{L_1} and W_{L_∞} are similar when the number of contacts, m, is not large. The iterations of our ray-shooting algorithm in cases (a)-(d) indicated in Tables 1 and 2 are plotted in Fig. 3. In cases (a) and (c), since the ray R through $\boldsymbol{w}_{\text{res}}$ passes the vertex $\boldsymbol{a}_6$ of the initial simplex coV_0, σ_{L_1} or σ_{L_∞} does not change in the next few iterations. But, later they quickly decrease and converge to their minima. In cases (b) and (d), no singular situation occurs.

Next, we assume that $\boldsymbol{w}_{\text{res}}$ is time-varying:

$$\boldsymbol{w}_{\text{res}} = \begin{bmatrix}-\cos(\pi t/5)\cos(\pi/4) & -0.5\sin(\pi t/5)\cos(\pi/4) \\ 3\sin(\pi/4) & -0.2\cos(\pi t/5) & -0.2\sin(\pi t/5) & 0\end{bmatrix}^T.$$

The external wrench is periodical with the period of 10 s. In each period we take 501 sampling points with interval of 20 ms. The average CPU times for minimizing σ_{L_1} and σ_{L_∞} by our ray-shooting algorithm at a sampling point are listed in Table 3, which are below 20 ms and meet the requirement for real-time applications.

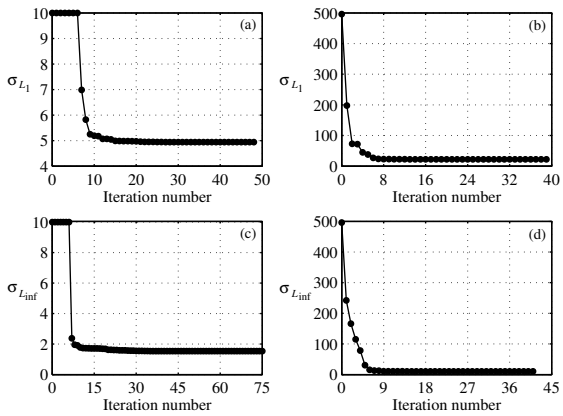

Fig. 3: Iteration of our ray-shooting algorithm. σ_{L_1} in (a), (b) or σ_{L_∞} in (c), (d) equals $\sigma(\boldsymbol{c})$ in the iteration of the ray-shooting algorithm with respect to W_{L_1} or W_{L_∞}.

TABLE 3: AVERAGE CPU TIMES OF METHODS (i)-(iii)

objective function	(i) ours	(ii)	(iii)
σ_{L_1}	10.96 ms	112.50 ms	230.74 ms
σ_{L_∞}	18.75 ms	382.78 ms	276.93 ms

4.3. Comparisons and Discussions

Besides our ray-shooting algorithm, numbered (i), we also implement other two methods for determining the minimum contact forces in these tests:

(ii) the ray-shooting algorithm given in [7];

(iii) the function **fmincon** provided by the Optimization Toolbox of MATLAB to solve the problem:

$$\text{minimize } \sigma_{L_1} \text{ or } \sigma_{L_\infty}$$
$$\text{s.t. } \boldsymbol{w}_{\text{res}} = \sum_{i=1}^{m} \boldsymbol{G}_i \boldsymbol{f}_i \text{ and } \boldsymbol{f}_i \in F_1,\ i = 1,2,\ldots,m.$$

The above optimization problem is more straightforward than the one solved in [7] using **fmincon**. The problem in [7] has more variables and constraints, so that the function **fmincon** requires more time to compute the minimum grasping forces.

The CPU times of methods (ii) and (iii) are also listed in Tables 1–3, which shows that our algorithm is about one order of magnitude faster than (ii) and (iii). In these tests, we also notice that the terminal values of σ_{L_1} or σ_{L_∞} obtained by methods (i) and (iii) are same, but those obtained by method (ii) are slightly bigger.

Our ray-shooting algorithm has higher solution accuracy and computational efficiency than the earlier work [7], because in every iteration the prior algorithm [7] needs to perform an iterative GJK-based distance computation. Since both W_{L_1} and W_{L_∞} here are convex sets with non-linear boundaries in 6-D space, the

GJK-based distance computation does not run as fast as it typically does for polytopes in 3-D space and must terminate according to a termination tolerance, called the "inner tolerance". The iteration of the earlier algorithm [7] stops based on another termination tolerance, called the "outer tolerance". To obtain a more accurate minimum value of σ_{L_1} or σ_{L_∞}, one must reduce both inner and outer tolerances. However, this necessarily increases the numbers of iterations required by the GJK-based distance computation and the algorithm [7], thereby increasing its overall computational cost. To reduce the running time, one may lower one or both termination tolerances at the cost of losing solution accuracy. In contrast, our ray-shooting algorithm performs only one-layer iteration controlled by a single termination tolerance and a few simple operations with constant complexity in each iteration (see Section 2.3), so it can reach the same level of solution accuracy much more quickly (see the last row of Table 1). Take the case of $\boldsymbol{w}_{\text{res}}^8$ as an example, which was also tested in [7]. The algorithm in [7] used larger tolerances to terminate the iteration and the GJK-based distance computation for the timing reported there, but $\sigma_{L_1} = 22.2794$. Here we set smaller termination tolerances for the algorithm in [7] to obtain comparable solution accuracy as ours; however, its running time increases significantly in such cases.

5. CONCLUSION AND FUTURE WORK

In this paper, we introduce a new n-dimensional ray-shooting algorithm. Running in 6-D wrench space, this algorithm provides a fast method to determine the minimum grasping forces and shows high computational efficiency, which makes it possible to perform optimization of grasping forces in real time.

There remain other possible directions to further improve this work. First, although the singular case may rarely occur, there could exist ways to ensure the iterations would transit from singular to regular cases, thereby enabling us to derive proof of convergence. The techniques for avoiding or recovering from the singular cases need to be investigated, such as proper selection of the initial set. Methods of accelerating the convergence of the algorithm would also be very useful. In addition to possible advances in the algorithmic front, the applications of n-dimensional ray-shooting algorithms to other areas, such as robotics and computer graphics, may provide new excitement and insight to solve this problem more efficiently. In practical applications, the direction of the ray can change continuously or incrementally. Therefore, temporal or spatial coherence may be utilized in designing n-dimensional ray-shooting algorithms to provide additional computational gain.

References

[1] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Englewood Cliffs, NJ: Prentice-Hall, 1994.

[2] P. K. Agarwal and J. Matoušek, "Ray shooting and parametric search," in *Proc. 24th ACM Symposium on Theory of Computing*, 1992, pp. 517–526.

[3] J. Matoušek and O. Schwarzkopf, "On ray shooting in convex polytopes," *Discrete Comput. Geom.*, vol. 10, no. 1, 215–232, 1993.

[4] T. M. Chan, "Output-sensitive results on convex hulls, extreme points, and related problems," *Discrete Comput. Geom.*, vol. 16, no. 4, 369–387, 1996.

[5] L. Szirmay-Kalos, V. Havran, B. Balázs, and L. Szécsi, "On the efficiency of ray-shooting acceleration schemes," in *Proc. 18th Spring Conf. Computer Graphics.*, Budmerice, Slovakia, 2002, pp. 97–106.

[6] G. van den Bergen, "Ray casting against general convex objects with application to continuous collision detection," 2004, http://www.dtecta.com.

[7] Y. Zheng and C.-M. Chew, "A numerical solution to the ray-shooting problem and its applications in robotic grasping," in *Proc. IEEE Int. Conf. Robot. Automat.*, Kobe, Japan, May 2009, pp. 2080–2085.

[8] Y.-H. Liu, "Qualitative test and force optimization of 3-D frictional form-closure grasps using linear programming," *IEEE Trans. Robot. Automat.*, vol. 15, no. 1, pp. 163–173, 1999.

[9] Y.-H. Liu, M.-L. Lam, and D. Ding, "A complete and efficient algorithm for searching 3-D form-closure grasps in the discrete domain," *IEEE Trans. Robot.*, vol. 20, no. 5, pp. 805–816, 2004.

[10] D. Ding, Y.-H. Liu, Y. Wang, and S. G. Wang, "Automatic selection of fixturing surfaces and fixturing points for polyhedral workpieces," *IEEE Trans. Robot. Automat.*, vol. 17, no. 6, pp. 833–841, 2001.

[11] Y. Zheng and W.-H. Qian, "Limiting and minimizing the contact forces in multifingered grasping," *Mech. Mach. Theory*, vol. 41, no. 10, pp. 1243–1257, 2006.

[12] X.-Y. Zhu, H. Ding, and Y. Wang, "A numerical test for the closure properties of 3D grasps," *IEEE Trans. Robot. Automat.*, vol. 20, no. 3, pp. 543–549, 2004.

[13] 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 J. Robot. Automat.*, vol. 4, no. 2, pp. 193–203, 1988.

[14] E. G. Gilbert and C. P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Trans. Robot. Automat.*, vol. 6, no. 1, pp. 53–61, 1990.

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

[16] L. Han, J. C. Trinkle, and Z. X. Li, "Grasp analysis as linear matrix inequality problems," *IEEE Trans. Robot. Automat.*, vol. 16, no. 6, pp. 663–674, 2000.

[17] R. D. Howe, I. Kao, and M. R. Cutkosky, "The sliding of robot fingers under combined torsion and shear loading," in *Proc. IEEE Int. Conf. Robot. Automat.*, Apr. 1988, pp. 103–105.

[18] C. Ferrari and J. Canny, "Planning optimal grasps," in *Proc. IEEE Int. Conf. Robot. Automat.*, Nice, France, May 1992, pp. 2290–2295.

[19] R. M. Murray, Z.-X. Li, and S.S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, 1994.

[20] Y. Zheng and W.-H. Qian, "Improving grasp quality evaluation," *Robot. Auton. Syst.*, vol. 57, no. 6-7, pp. 665–673, 2009.

[21] Y. Zheng and C.-M. Chew, "Distance between a point and a convex cone in n-dimensional space: computation and applications," *IEEE Trans. Robot.*, vol. 25, no. 6, pp. 1397–1412, 2009.