



Calculating the Support Function of Complex Continuous Surfaces With Applications to Minimum Distance Computation and Optimal Grasp Planning

Yu Zheng , Senior Member, IEEE, and Kaiyu Hang , Member, IEEE

Abstract—The support function of a surface is a fundamental concept in mathematics and a crucial operation for algorithms in robotics, such as those for collision detection and grasp planning. It is possible to calculate the support function of a convex body in a closed form. For complex continuous, especially nonconvex, surfaces, however, this calculation can be far more difficult and no general solution is available so far, which limits the applicability of those related algorithms. This article first presents a branch-and-bound (B&B) algorithm to calculate the support function of complex continuous surfaces. An upper bound of the support function over a surface domain is derived. While a surface domain is divided into subdomains, the upper bound of the support function over any subdomain is proved to be not greater than the one over the original domain. Then, as the B&B algorithm sequentially divides the surface domain by dividing its subdomain having a greater upper bound than the others, the maximum upper bound over all subdomains is monotonically decreasing and converges to the exact value of the desired support function. Furthermore, with the aid of the B&B algorithm, this article derives new algorithms for the minimum distance between complex continuous surfaces and for globally optimal grasps on objects with continuous surfaces. A number of numerical examples are provided to demonstrate the effectiveness of the proposed algorithms.

Index Terms—Bounding volume, branch-and-bound, collision detection, computational geometry, distance, grasp planning, support function.

I. INTRODUCTION

THE support function of a surface along a vector is defined to be the maximum inner product of the vector with the points on the surface and such a point at which the maximum inner product is obtained is called the support mapping, where a surface can be described by parametric functions and is generally a set of infinite points in space. While the support function is a fundamental concept in mathematics, it is a crucial operation in algorithms for important problems in robotics, including

collision detection and grasp planning. In some cases where the surface is convex and the inner product of the vector with a point on the surface is a convex function of the surface parameters, the support function and mapping could be calculated by a convex optimization algorithm or occasionally in a closed form. In many more cases, however, a surface can be nonconvex, nonlinear, and have no explicit expression, which makes it hard to compute the global maximum value of the inner product and the exact value of the support function, not to mention a closed-form solution, and limits the application scopes of those important algorithms. In this article, we first present a branch-and-bound (B&B) algorithm to compute the support function and mapping of complex continuous surfaces. Based on this algorithm, we, then, propose not only extensions of existing algorithms but also new algorithms for minimum distance computation and globally optimal grasps on objects with continuous surfaces.

A. Related Work

The computing of support function and mapping is a key step in several algorithms in robotics. First of all, the well-known Gilbert-Johnson-Keerthi (GJK) algorithm [1], [2] calculates the support function and mapping of a compact convex set at every iteration to alter an inscribed simplex such that the minimum Euclidean distance between the origin and the simplex converges to the minimum Euclidean distance between the origin and the set. Since the minimum distance is a natural choice of index to determine if two sets or equivalently the origin and their Minkowski difference are separated, the GJK algorithm has been frequently used for collision detection, which is an essential component in many software packages, such as robot motion planners, simulators, and physics engines. Over the next decade since it was initially proposed, the GJK algorithm has continued to be improved to achieve higher computational efficiency on convex polyhedra and moving bodies [3]–[6]. While the GJK algorithm can calculate the minimum distance between two separated sets, it cannot tell how deeply two sets penetrate each other if they overlap. The penetration depth of two overlapping sets is often defined as the minimum translation required to separate them and also called the penetration distance [7]. For compact convex sets, the penetration distance can be calculated by the algorithm [8], which iteratively expands a polytope in the Minkowski difference such that the minimum Euclidean distance from the origin to the boundary of the Minkowski

Manuscript received August 1, 2019; accepted January 28, 2020. This paper was recommended for publication by Associate Editor F. Ficuciello and Editor M. Yim upon evaluation of the reviewers' comments. (Corresponding author: Yu Zheng.)

Y. Zheng is with the Tencent Robotics X, Shenzhen 518000, China (e-mail: petezheng@tencent.com).

K. Hang is with the Department of Mechanical Engineering and Material Science, Yale University, New Haven, CT 06511 USA (e-mail: kaiyu.hang@yale.edu).

Color versions of one or more of the figures in this article are available online at <https://ieeexplore.ieee.org>.

Digital Object Identifier 10.1109/TRO.2020.2971889

difference converges to the penetration distance. At every of its iteration, the support function and mapping is calculated to determine whether and how to continue expanding the polytope. It has been noticed that the Euclidean separation (resp. penetration) distance between two sets can also be expressed as the maximum scale factor of the origin-centered unit ball such that the scaled ball does not intersect (resp. exceed) the interior of the Minkowski difference of the sets. Referring to this, Zhu *et al.* [9]–[11] proposed a generalized distance function by replacing the unit ball with a compact convex set as the gauge set. Algorithms to compute the generalized distance for compact convex sets were proposed in the work [12], in which the computing of support function and mapping is still one of the key operations. As a special case of the generalized distance where the gauge set is taken to be a line segment, the ray-shooting problem determines the intersection of a ray with a set and can be solved by specialized algorithms, all of which calculate the support function and mapping of the set at every iteration [13]. In addition, computation and application of distance can be extended to the case of a convex cone and a point [14], which has been found useful in the equilibrium test and contact force distribution of multicontact robotic systems as well as whole-body locomotion generation and control of legged robots [15]–[18].

In addition to collision detection, another important application of distance functions, especially the penetration distance, is to provide a quality measure for grasps; that is, the minimum distance from the origin to the boundary of a so-called grasp wrench set can be used as a quantitative index of how capable a grasp is of applying wrenches to the grasped object [19]. During the past two decades, variants of this grasp quality measure were proposed to achieve metric invariance [10], [20], [21], reflect task requirements [22]–[26], and incorporate the structural information of a robot hand [27] or the uncertainty of friction coefficients [28], and more efficient and accurate algorithms were developed to compute them [8], [29]–[31]. These measures are often used in grasp planning to guide the computing of optimal contact locations [10], [26], [28], [31]–[43]. While some grasp planning algorithms are aimed at optimal grasps on 3-D objects with piecewise smooth surface [10], [28], [33], [34], it is hard to guarantee the actual optimality of the computed grasp because the problem is a highly nonlinear optimization problem with many local optima. Another approach is to discretize the object's surface and search contact locations within a set of preselected discrete points [31], [36], [37], [40], [42]. By doing this, the original nonlinear optimization problem is reduced to a combinatorial optimization problem, whose best solution is more attainable and can provide a reasonable approximation for the globally optimal grasp on theoretically any object.

B. Our Work

In the aforementioned algorithms, the computing of support function is currently limited to the case where a closed-form expression can be derived. However, there are situations where the computing of support function is a nonlinear optimization problem, for which an exact numerical solution is hard and a closed-form solution is even impossible to obtain. In this

article, we first present a B&B algorithm to calculate the support function and mapping of complex continuous surfaces. An upper bound of the support function over a surface domain is derived and proved to be monotonically decreasing; that is, when the domain is divided into subdomains, the upper bound of the support function over any subdomain will not increase. Then, in the sequential dividing of the surface domain as the B&B algorithm iterates, the overall upper bound successively decreases and converges to the value of the support function over the initial domain. Furthermore, we apply the algorithm to several selected problems including

- 1) *Building bounding polyhedra*: Since the support function of a surface along a nonzero vector defines a supporting plane of the surface, we use the proposed B&B algorithm in the algorithm [44] to compute a sequence of supporting planes enclosing and forming a convex bounding polyhedron of the surface. This computation can proceed and finally lead to the convex hull of the surface.
- 2) *Computing minimum distances*: With the help of the B&B algorithm for the support function, the aforementioned distance computation algorithms [1], [2], [8] is extended to separated or overlapping nonconvex sets such that the minimum distance between their convex hulls can be computed without explicitly calculating the convex hulls. Moreover, an algorithm to compute the true minimum distance between separated nonconvex sets is derived.
- 3) *Planning optimal grasps*: We use the B&B algorithm for the support function to compute an upper bound on the quality of grasps over a surface domain based on the Ferrari–Canny grasp quality measure [19]. With this upper bound, we then propose another B&B algorithm to compute the globally optimal grasps on objects with continuous surfaces, which is an unsolved problem in grasping research until now.

The rest of this article is organized as follows. Section II describes the algorithm to compute the support function with application to computing bounding polyhedra of complex continuous surfaces. Section III applies the algorithm to minimum distance computation between complex bodies. Section IV presents an algorithm for globally optimal grasps on objects with continuous surface. Numerical examples are provided in each of the three sections to verify the algorithm's performance. Section V concludes this article.

II. B&B ALGORITHM FOR SUPPORT FUNCTION

In this section, we present a B&B algorithm to compute the support function of a continuous surface.

A. Problem Definition

Let $S \triangleq \{f(\mathbf{x}) | \mathbf{x} \in X\}$ be a surface defined by a function $f : X \rightarrow S$, as depicted in Fig. 1, where $X \subset \mathbb{R}^m$ is the domain of f comprising the intervals for the components of $\mathbf{x} \in \mathbb{R}^m$ and $S \subset \mathbb{R}^n$ is the image of X under f . Then, X is an m -dimensional hyperrectangle in \mathbb{R}^m . Assume that f has Lipschitz continuity such that, for $\forall \mathbf{x}_1, \mathbf{x}_2 \in X$

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq L \|\mathbf{x}_1 - \mathbf{x}_2\| \quad (1)$$

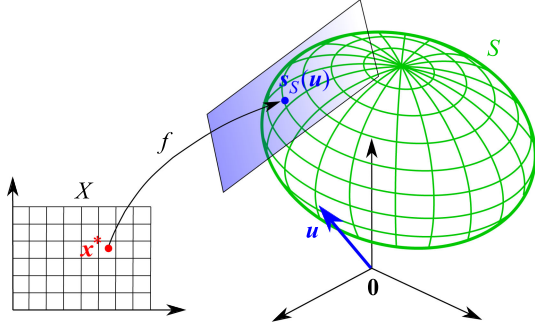


Fig. 1. Illustration of the support function $h_S(\mathbf{u})$ of a surface S along a vector \mathbf{u} . The surface S is described by a continuous function f over domain X and $\mathbf{s}_S(\mathbf{u})$ is a point on S called the support mapping such that $h_S(\mathbf{u}) = \mathbf{u}^T \mathbf{s}_S(\mathbf{u})$. \mathbf{x}^* consists of the variables such that $\mathbf{s}_S(\mathbf{u}) = f(\mathbf{x}^*)$.

where L is a Lipschitz constant and $\|\cdot\|$ denotes the 2-norm of vectors. In other words, S is a point set described by a Lipschitz continuous function f over domain X .

The support function of S is defined as [1], [45]

$$h_S(\mathbf{u}) \triangleq \max_{\mathbf{s} \in S} \mathbf{u}^T \mathbf{s} = \max_{\mathbf{x} \in X} \mathbf{u}^T f(\mathbf{x}) \quad (2)$$

where \mathbf{u} is a nonzero vector in \mathbb{R}^n . Any point on S where $h_S(\mathbf{u})$ is attained is called the support mapping [1], i.e.,

$$\mathbf{s}_S(\mathbf{u}) \triangleq \arg \max_{\mathbf{s} \in S} \mathbf{u}^T \mathbf{s}. \quad (3)$$

The hyperplane with normal \mathbf{u} passing through $\mathbf{s}_S(\mathbf{u})$ is a hyperplane of support¹ to S at $\mathbf{s}_S(\mathbf{u})$, as depicted in Fig. 1. If \mathbf{u} is a unit vector, then $h_S(\mathbf{u})$ gives the distance from the origin to the hyperplane. From (2), the computing of $h_S(\mathbf{u})$ is generally an optimization problem. Let \mathbf{x}^* denote the optimal solution, namely the value of $\mathbf{x} \in X$ such that $\mathbf{s}_S(\mathbf{u}) = f(\mathbf{x}^*)$ and $h_S(\mathbf{u}) = \mathbf{u}^T f(\mathbf{x}^*)$. When S is such a surface that $\mathbf{u}^T \mathbf{s}$ over S or $\mathbf{u}^T f(\mathbf{x})$ over X is convex, it is possible to calculate the values of $h_S(\mathbf{u})$ and $\mathbf{s}_S(\mathbf{u})$ as well as \mathbf{x}^* by an existing convex optimization algorithm or even in a closed form [2]. In the general case that S or $\mathbf{u}^T f(\mathbf{x})$ is nonconvex, however, the computation becomes much harder. While an analytic solution is often unattainable, optimization algorithms could fall into a local optimal solution, which is far from the global maximum. In the following, we present an algorithm for the exact values of $h_S(\mathbf{u})$, $\mathbf{s}_S(\mathbf{u})$, and \mathbf{x}^* in such a challenging but general situation.

B. Algorithm Description

B&B is known as an effective technique for solving nonlinear optimization problems. Here, we apply it to the computing of the support function and mapping.

¹The hyperplane of support to or the supporting hyperplane of a point set at one of its points is a hyperplane that passes through the point and bounds the entire set to one side [45]. It seems analogous to the tangent hyperplane but actually a different concept. The tangent hyperplane is defined only at a regular point on a surface and bounds the surface locally rather than globally, where a surface is deemed as a point set. If the tangent hyperplane at a regular point bounds the entire surface, then it is the unique supporting hyperplane of the surface at this point. At a singular point, the tangent hyperplane is not defined but there could exist nonunique hyperplanes supporting the surface.

We first derive an upper bound of $h_S(\mathbf{u})$ over X . From (1) it follows that the value of $\mathbf{u}^T \mathbf{s}$ has Lipschitz continuity over X as well, i.e.,

$$|\mathbf{u}^T \mathbf{s}_1 - \mathbf{u}^T \mathbf{s}_2| \leq \|\mathbf{u}\| \|\mathbf{s}_1 - \mathbf{s}_2\| \leq L_h \|\mathbf{x}_1 - \mathbf{x}_2\| \quad (4)$$

where $\mathbf{s}_1 = f(\mathbf{x}_1)$, $\mathbf{s}_2 = f(\mathbf{x}_2)$, and $L_h = L\|\mathbf{u}\|$. Let $\bar{\mathbf{x}}$ be the center of X and $\bar{\mathbf{s}} = f(\bar{\mathbf{x}})$. Substituting \mathbf{x}^* and $\bar{\mathbf{x}}$ into \mathbf{x}_1 and \mathbf{x}_2 in (4), respectively, yields

$$h_S(\mathbf{u}) - \mathbf{u}^T \bar{\mathbf{s}} \leq L_h \|\mathbf{x}^* - \bar{\mathbf{x}}\| \leq L_h \max_{\mathbf{x} \in X} \|\mathbf{x} - \bar{\mathbf{x}}\|. \quad (5)$$

From (5), we attain an upper bound of $h_S(\mathbf{u})$ over X as

$$\hat{h}_S(\mathbf{u}) \triangleq \mathbf{u}^T \bar{\mathbf{s}} + L_h \max_{\mathbf{x} \in X} \|\mathbf{x} - \bar{\mathbf{x}}\|. \quad (6)$$

We next prove that the upper bound given by (6) is monotonically decreasing in dividing X into smaller domains. Let X be divided into 2^m isometric subdomains X_j 's along its center $\bar{\mathbf{x}}$ by bisecting each interval constituting X . Let $\bar{\mathbf{x}}_j$ be the center of X_j . Then, $\max_{\mathbf{x} \in X_j} \|\mathbf{x} - \bar{\mathbf{x}}_j\| = \|\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\| = \frac{1}{2} \max_{\mathbf{x} \in X} \|\mathbf{x} - \bar{\mathbf{x}}\|$. From (4) and (6), we further derive

$$\begin{aligned} \hat{h}_{S_j}(\mathbf{u}) &\triangleq \mathbf{u}^T \bar{\mathbf{s}}_j + L_h \max_{\mathbf{x} \in X_j} \|\mathbf{x} - \bar{\mathbf{x}}_j\| \\ &\leq \mathbf{u}^T \bar{\mathbf{s}} + L_h \|\bar{\mathbf{x}}_j - \bar{\mathbf{x}}\| + L_h \max_{\mathbf{x} \in X_j} \|\mathbf{x} - \bar{\mathbf{x}}_j\| \\ &= \mathbf{u}^T \bar{\mathbf{s}} + L_h \max_{\mathbf{x} \in X} \|\mathbf{x} - \bar{\mathbf{x}}\| \\ &\triangleq \hat{h}_S(\mathbf{u}). \end{aligned} \quad (7)$$

Therefore, the upper bound given by (6) is indeed monotonically decreasing during the dividing of X .

In fact, the definitions of support function (2) and its upper bound (6) can be extended to any domain. For a domain that is infinitely small and only a singleton, the upper bound is equal to the support function.

With the aforementioned upper bound, a B&B algorithm for $h_S(\mathbf{u})$, $\mathbf{s}_S(\mathbf{u})$, and \mathbf{x}^* is described in Algorithm 1. At every iteration of the algorithm, from the list \mathcal{L} of domains we select the one \hat{X} for which the upper bound is the maximum, denoted by \hat{h} , among all domains and divide \hat{X} into 2^m subdomains X_j . Then, we calculate the value of $\mathbf{u}^T \mathbf{s}$ at the center $\bar{\mathbf{x}}_j$ and the upper bound $\hat{h}_{S_j}(\mathbf{u})$ for each X_j and add X_j to the list \mathcal{L} if its upper bound is greater than the current best result h^* , which is the greatest value of $\mathbf{u}^T \mathbf{s}$ that we have so far. Since the upper bound is decreasing in the dividing of a domain as stated by (7), \hat{h} is monotonically decreasing as the algorithm iterates. On the other hand, h^* is monotonically increasing, and $h_S(\mathbf{u})$ is always bounded below by h^* but above by \hat{h} . Therefore, it is guaranteed that h^* converges to $h_S(\mathbf{u})$. Furthermore, we can derive that the difference of the final h^* terminated by the condition $\hat{h} - h^* > \epsilon$ from the true value of $h_S(\mathbf{u})$ is bounded by the termination tolerance ϵ , i.e.,

$$0 \leq h_S(\mathbf{u}) - h^* \leq \epsilon. \quad (8)$$

C. More Discussions

Here, we discuss several factors that may affect the computational efficiency of Algorithm 1.

Algorithm 1: Algorithm for $h_S(\mathbf{u})$ and $s_S(\mathbf{u})$.**Input:** \mathbf{u} and S given by f and X **Output:** $h_S(\mathbf{u})$ and $s_S(\mathbf{u})$ as well as the optimal solution

```

 $\mathbf{x}^*$ 
1:  $\mathbf{x}^* \leftarrow$  the middle point in domain  $X$ 
2:  $\mathbf{s}^* \leftarrow f(\mathbf{x}^*)$ 
3:  $h^* \leftarrow \mathbf{u}^T \mathbf{s}^*$ 
4:  $\hat{h} \leftarrow$  the upper bound of  $h_S(\mathbf{u})$  over  $X$ 
5:  $\hat{X} \leftarrow X$ 
6:  $\mathcal{L} \leftarrow \emptyset$ 
7: while  $\hat{h} - h^* > \epsilon$  do
8:   Divide  $\hat{X}$  into subdomains  $X_j$ 's
9:   for each  $X_j$  do
10:      $\bar{\mathbf{x}}_j \leftarrow$  the middle point in domain  $X_j$ 
11:      $\bar{\mathbf{s}}_j \leftarrow f(\bar{\mathbf{x}}_j)$ 
12:      $\bar{h}_j \leftarrow \mathbf{u}^T \bar{\mathbf{s}}_j$ 
13:     if  $\bar{h}_j > h^*$  then
14:        $h^* \leftarrow \bar{h}_j$ 
15:        $\mathbf{s}^* \leftarrow \bar{\mathbf{s}}_j$ 
16:        $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}_j$ 
17:     end if
18:      $\hat{h}_j \leftarrow$  the upper bound of  $h_S(\mathbf{u})$  over  $X_j$ 
19:     if  $\hat{h}_j > h^*$  then
20:       Add  $X_j$  to the list  $\mathcal{L}$ 
21:     end if
22:   end for
23:    $\hat{h} \leftarrow$  the maximum upper bound for domains in the
   list  $\mathcal{L}$ 
24:    $\hat{X} \leftarrow$  the domain in the list  $\mathcal{L}$  whose upper bound
   is maximal
25:   Remove  $\hat{X}$  from the list  $\mathcal{L}$ 
26: end while
27: return  $h^*$ ,  $\mathbf{s}^*$ , and  $\mathbf{x}^*$ 

```

First, the choice of the Lipschitz constant L_h in (6) affects the number of iterations required by Algorithm 1. One may simply take a sufficiently large L_h such that (4) holds for the entire domain X . However, this will lead to an overestimated upper bound as given by (6) and a slow convergence of the algorithm. Instead, if function f is differentiable over X , we can estimate L in (1) as follows and, then, take $L_h = L\|\mathbf{u}\|$ to be used in (4)

$$L = \max_{\mathbf{x} \in X} \left\| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \right\|_F \quad (9)$$

where $\partial f/\partial \mathbf{x}$ is the Jacobian matrix of f and $\|\cdot\|_F$ is the Frobenius norm. Another way to estimate L_h is to directly use the gradient of $\mathbf{u}^T \mathbf{s} = \mathbf{u}^T f(\mathbf{x})$ over X , i.e.,

$$L_h = \max_{\mathbf{x} \in X} \left\| \frac{\partial (\mathbf{u}^T f(\mathbf{x}))}{\partial \mathbf{x}} \right\|. \quad (10)$$

Usually (10) gives a smaller constant and is used in this article.

Second, we notice that Algorithm 1 could sometimes yield a number of domains with their upper bounds all close to each other and slightly greater than $h_S(\mathbf{u})$. The reason is that, for a domain containing or approaching the optimal solution \mathbf{x}^* , the value $\mathbf{u}^T \bar{\mathbf{s}}$ may be slightly less than $h_S(\mathbf{u})$ but its upper

Algorithm 2: Algorithm for $h_S(\mathbf{u})$ and $s_S(\mathbf{u})$.**Input:** \mathbf{u} and S given by f and X **Output:** $h_S(\mathbf{u})$ and $s_S(\mathbf{u})$ as well as the optimal solution

```

 $\mathbf{x}^*$ 
1:  $\mathbf{x}^* \leftarrow$  the middle point in domain  $X$ 
2:  $\mathbf{s}^* \leftarrow f(\mathbf{x}^*)$ 
3:  $h^* \leftarrow \mathbf{u}^T \mathbf{s}^*$ 
4:  $\hat{h} \leftarrow$  the upper bound of  $h_S(\mathbf{u})$  over  $X$ 
5:  $\hat{X} \leftarrow X$ 
6:  $\mathcal{L} \leftarrow \emptyset$ 
7:  $k \leftarrow 0$ 
8: while  $\hat{h} - h^* > \epsilon$  and  $k < K$  do
9:   Divide  $\hat{X}$  into subdomains  $X_j$ 's
10:  for each  $X_j$  do
11:     $\bar{\mathbf{x}}_j \leftarrow$  the middle point in domain  $X_j$ 
12:     $\bar{\mathbf{s}}_j \leftarrow f(\bar{\mathbf{x}}_j)$ 
13:     $\bar{h}_j \leftarrow \mathbf{u}^T \bar{\mathbf{s}}_j$ 
14:    if  $\bar{h}_j > h^*$  then
15:       $h^* \leftarrow \bar{h}_j$ 
16:       $\mathbf{s}^* \leftarrow \bar{\mathbf{s}}_j$ 
17:       $\mathbf{x}^* \leftarrow \bar{\mathbf{x}}_j$ 
18:    end if
19:     $\hat{h}_j \leftarrow$  the upper bound of  $h_S(\mathbf{u})$  over  $X_j$ 
20:    if  $\hat{h}_j > h^*$  then
21:      if  $|X_j| > \epsilon_X$  then
22:        Add  $X_j$  to the list  $\mathcal{L}$ 
23:      else
24:        Let  $\tilde{h}_j$ ,  $\tilde{\mathbf{s}}_j$ , and  $\tilde{\mathbf{x}}_j$  be the values of  $h_{S_j}(\mathbf{u})$ ,
         $s_{S_j}(\mathbf{u})$ , and  $\mathbf{x}_{S_j}^*$  computed by any
        optimization solver over domain  $X_j$ 
25:        if  $\tilde{h}_j > h^*$  then
26:           $h^* \leftarrow \tilde{h}_j$ 
27:           $\mathbf{s}^* \leftarrow \tilde{\mathbf{s}}_j$ 
28:           $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}_j$ 
29:        end if
30:      end if
31:    end for
32:     $\hat{h} \leftarrow$  the maximum upper bound for domains in the
    list  $\mathcal{L}$ 
33:     $\hat{X} \leftarrow$  the domain in the list  $\mathcal{L}$  whose upper bound
    is maximal
34:    Remove  $\hat{X}$  from the list  $\mathcal{L}$ 
35:     $k \leftarrow k + 1$ 
36:  end while
37: return  $h^*$ ,  $\mathbf{s}^*$ , and  $\mathbf{x}^*$ 

```

bound is greater than h^* due to the term $L_h \max_{\mathbf{x} \in X} \|\mathbf{x} - \bar{\mathbf{x}}\|$ in (6). Dividing such a domain could generate more suchlike subdomains and for a subdomain to be ruled out, the constant L_h and/or the size of the subdomain must be sufficiently small such that its upper bound is less than h^* . This may cause Algorithm 1 to iterate many times before the termination condition $\hat{h} - h^* > \epsilon$ can be reached. To avoid this situation, we propose a variant of Algorithm 1, as described in Algorithm 2, in which a local optimization solver is called to compute $h_{S_j}(\mathbf{u})$

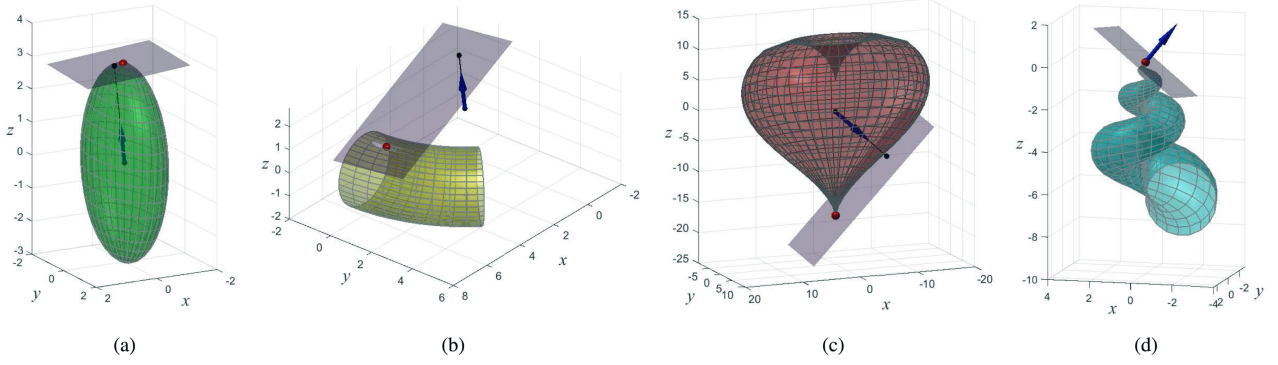


Fig. 2. Worst cases of computing the support function of (a) an ellipsoid, (b) one eighth of a torus, (c) a heart, and (d) a seashell. The blue arrow and the red dot represent a given direction \mathbf{u} and the corresponding support mapping $\mathbf{s}_S(\mathbf{u})$, respectively. The gray plane is the plane with normal \mathbf{u} passing through $\mathbf{s}_S(\mathbf{u})$ and a plane of support to the body S .

for any sufficiently small domain X_j (i.e., $|X_j| \leq \epsilon_X$) instead of adding it to the list \mathcal{L} , where $|X_j|$ denotes the size of X_j and ϵ_X is the tolerance on $|X_j|$. In addition, we set a maximum allowable iteration number K for Algorithm 2.

Here, Algorithm 2 has three user-specified parameters, namely the termination tolerance ϵ on the value $\hat{h} - h^*$, the tolerance ϵ_X on $|X_j|$, and the allowable iteration number K . The tolerance ϵ directly controls the accuracy of the computed result, as indicated by (8). It should be taken to be as small as needed in specific applications. For the number K , one can simply choose a relatively large value to avoid the early termination of the algorithm while $\hat{h} - h^*$ is still big. The tolerance ϵ_X also affects the accuracy and efficiency of the algorithm. In general, it should be set to a sufficiently small value such that the local optimization solver can confidently compute $h_{S_j}(\mathbf{u})$ over any such small domain. Nevertheless, it should also be noted that a too small ϵ_X will lead to more domains being added to the list \mathcal{L} , causing more domain divisions and iterations of the algorithm. In the following numerical tests, we try different values for ϵ_X and it turns out that 10^{-1} is small enough for attaining an accurate support function on four given bodies with an increasing level of complexity (from convexity to high nonconvexity).

D. Numerical Examples

We verify the performance of Algorithms 1 and 2 with numerical examples. The algorithms have been implemented in MATLAB and run on a desktop with an Intel Core i7-6700 3.40 GHz CPU and 16 GB RAM. The termination tolerance ϵ for $\hat{h} - h^*$ is set to 10^{-6} , the number of iterations is limited to $K = 10^5$, and the size $|X_j|$ of a domain is defined to be the maximum interval in X_j . In Algorithm 2, for a domain whose size is below a threshold ϵ_X , we call the “trust-region-reflective” algorithm with its default setting provided by the optimization toolbox of MATLAB to compute the support function over that domain.

Example 1: First, we use the algorithms to compute the support function of several 3-D bodies including an ellipsoid, one eighth of a torus, a heart, and a seashell, as shown in Fig. 2. Their parametric expressions are as follows.

1) *Ellipsoid:*

$$\mathbf{s} = \begin{bmatrix} \cos \theta \cos \phi \\ 2 \cos \theta \sin \phi \\ 3 \sin \theta \end{bmatrix}$$

where $\theta \in [-\pi/2, \pi/2]$ and $\phi \in [0, 2\pi]$.

2) *Torus:*

$$\mathbf{s} = \begin{bmatrix} (6 + 1.5 \cos \phi) \cos \theta \\ (6 + 1.5 \cos \phi) \sin \theta \\ 1.5 \sin \phi \end{bmatrix}$$

where $\theta \in [0, \pi/2]$ and $\phi \in [0, 2\pi]$.

3) *Heart:*

$$\mathbf{s} = \begin{bmatrix} 16 \sin^3 \theta \cos \phi \\ 8 \sin^3 \theta \sin \phi \\ 13 \cos \theta - 5 \cos 2\theta - 2 \cos 3\theta - \cos 4\theta \end{bmatrix}$$

where $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$.

4) *Seashell:*

$$\mathbf{s} = \begin{bmatrix} (1 - e^{\theta/6\pi}) \cos \theta (1 + \cos \phi) \\ -(1 - e^{\theta/6\pi}) \sin \theta (1 + \cos \phi) \\ 1 - e^{\theta/3\pi} - (1 - e^{\theta/6\pi}) \sin \phi \end{bmatrix}$$

where $\theta \in [0, 6\pi]$ and $\phi \in [0, 2\pi]$.

On the ellipsoid, the support function $h_S(\mathbf{u})$ and mapping $\mathbf{s}_S(\mathbf{u})$ can be calculated in a closed form as [2]

$$h_S(\mathbf{u}) = \sqrt{u_x^2 + 4u_y^2 + 9u_z^2},$$

$$\mathbf{s}_S(\mathbf{u}) = \begin{bmatrix} u_x & 4u_y & 9u_z \\ h_S(\mathbf{u}) & h_S(\mathbf{u}) & h_S(\mathbf{u}) \end{bmatrix}^T$$

where u_x , u_y , and u_z are the three components of \mathbf{u} . However, the computation of $h_S(\mathbf{u})$ and $\mathbf{s}_S(\mathbf{u})$ is much harder on the other bodies. In each case, we then let Algorithms 1 and 2 compute the support function along 10^3 random directions and

TABLE I
RESULTS OF ALGORITHM 1

Object	e_{\max}	e_{ave}^o	t_{ave}	t_{\max}	t_{\min}	N_{ave}	N_{\max}	N_{\min}
Ellipsoid	2.24×10^{-7}	0.04371950	0.009	0.054	0.007	349	1993	267
Torus	—	0.14482988	1.748	27.109	0.133	18141	10^5	4072
Heart	—	0.70123328	14.140	32.021	0.007	62457	10^5	215
Seashell	—	1.30161407	18.231	32.606	0.247	71530	10^5	6058

e —The value obtained by the closed-form solution minus the one obtained by the algorithm;

e^o —The value obtained by the algorithm minus the one obtained by one-time optimization;

t —CPU running time of the algorithm (unit: second);

N —Number of iteration of the algorithm;

The subscripts “max,” “min,” and “ave” represent the maximum, minimum, and average values of the corresponding quantities obtained in the tests, respectively.

TABLE II
RESULTS OF ALGORITHM 2

Object	ϵ_X	e_{\max}	e_{ave}^o	t_{ave}	t_{\max}	t_{\min}	N_{ave}	N_{\max}	N_{\min}	N_{ave}^o	N_{\max}^o	N_{\min}^o
Ellipsoid	10^{-1}	8.64×10^{-7}	0.04371952	0.033	0.100	0.019	75	102	56	14	44	9
	10^{-2}	8.19×10^{-7}	0.04371939	0.029	0.190	0.018	220	1028	160	14	96	8
	10^{-3}	6.06×10^{-7}	0.04371941	0.022	0.150	0.015	325	1801	249	14	96	8
Torus	10^{-1}	—	0.14483025	0.015	0.055	0.008	29	49	17	7	20	4
	10^{-2}	—	0.14483011	0.064	0.521	0.029	168	896	91	37	315	18
	10^{-3}	—	0.14483027	0.286	3.157	0.065	752	7340	274	193	2426	51
Heart	10^{-1}	—	0.70123298	0.081	0.334	0.007	81	199	22	32	132	3
	10^{-2}	—	0.70123264	0.469	1.810	0.009	640	2439	54	247	966	4
	10^{-3}	—	0.70123259	3.188	10.679	0.011	6254	22759	94	2296	7613	6
Seashell	10^{-1}	—	1.30161447	0.098	0.265	0.013	181	433	34	46	138	6
	10^{-2}	—	1.30161435	0.784	3.130	0.036	1208	4427	110	472	2278	22
	10^{-3}	—	1.30161435	5.795	35.081	0.080	12267	59474	361	4711	29851	1

N^o —Times of the optimization algorithm called in Algorithm 2. The unit for the CPU running time is second.

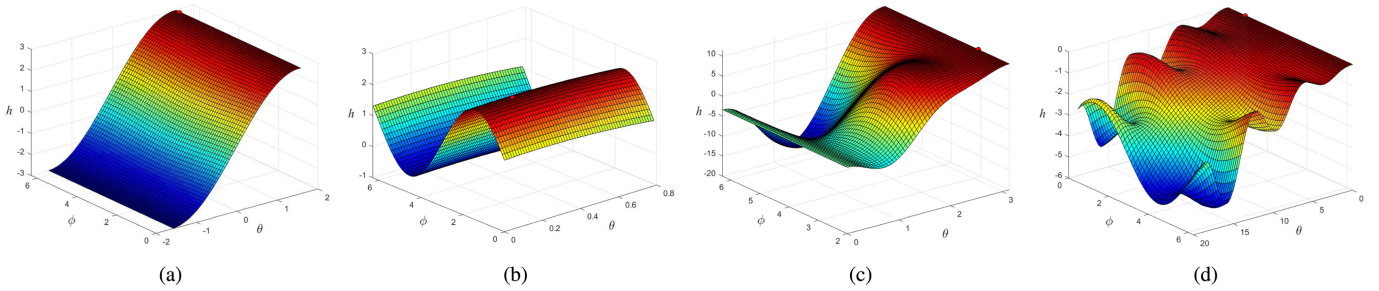


Fig. 3. Value of $u^T s$ on the (a) ellipsoid, (b) one eighth of the torus, (c) heart, and (d) seashell in the cases shown in Fig. 2. The red dot indicates the solution yielded by Algorithm 2, which gives the global maximum of $u^T s$ in these worst cases.

take the threshold ϵ_X of the domain’s size to be 10^{-1} , 10^{-2} , or 10^{-3} . Tables I and II exhibit the performance of the two algorithms, respectively. We first compare the results from the proposed algorithms on the ellipsoid with the ones obtained by the abovementioned closed-form expression, as displayed in the second column of Table I and third column of Table II, which shows that the proposed algorithms are accurate. For the other three objects, since no closed-form expression for the support function can be easily derived, we compare our results with the ones by using the MATLABs optimization algorithm to solve (2) over the initial domain with the domain’s centroid as the initial point, as shown in the next columns of the tables. It can be seen that the one-time optimization cannot guarantee the global maximum and the computed values are notably less than the results of our algorithms.

From Tables I and II, we further see that the two algorithms can achieve comparable levels of accuracy, but Algorithm 1 needs much longer computation time except on the ellipsoid. Fig. 2 shows several worst cases where both algorithms need a great number of iterations to terminate and Fig. 3 plots the value of $u^T s$ over the initial domains for θ and ϕ in these cases. It can be seen that the values of $u^T s$ are very close to each other or even the same somewhere and unfortunately some or even all of them are the global maximum of $u^T s$. In such a case, Algorithm 1 will generate a large number of tiny domains in order to reach the termination condition $\hat{h} - h^* \leq \epsilon$. The use of local optimization for sufficiently small domains in Algorithm 2 successfully prohibits unnecessary dividing of those domains and significantly reduces the number of iterations while keeping the result at the same level of accuracy. On the

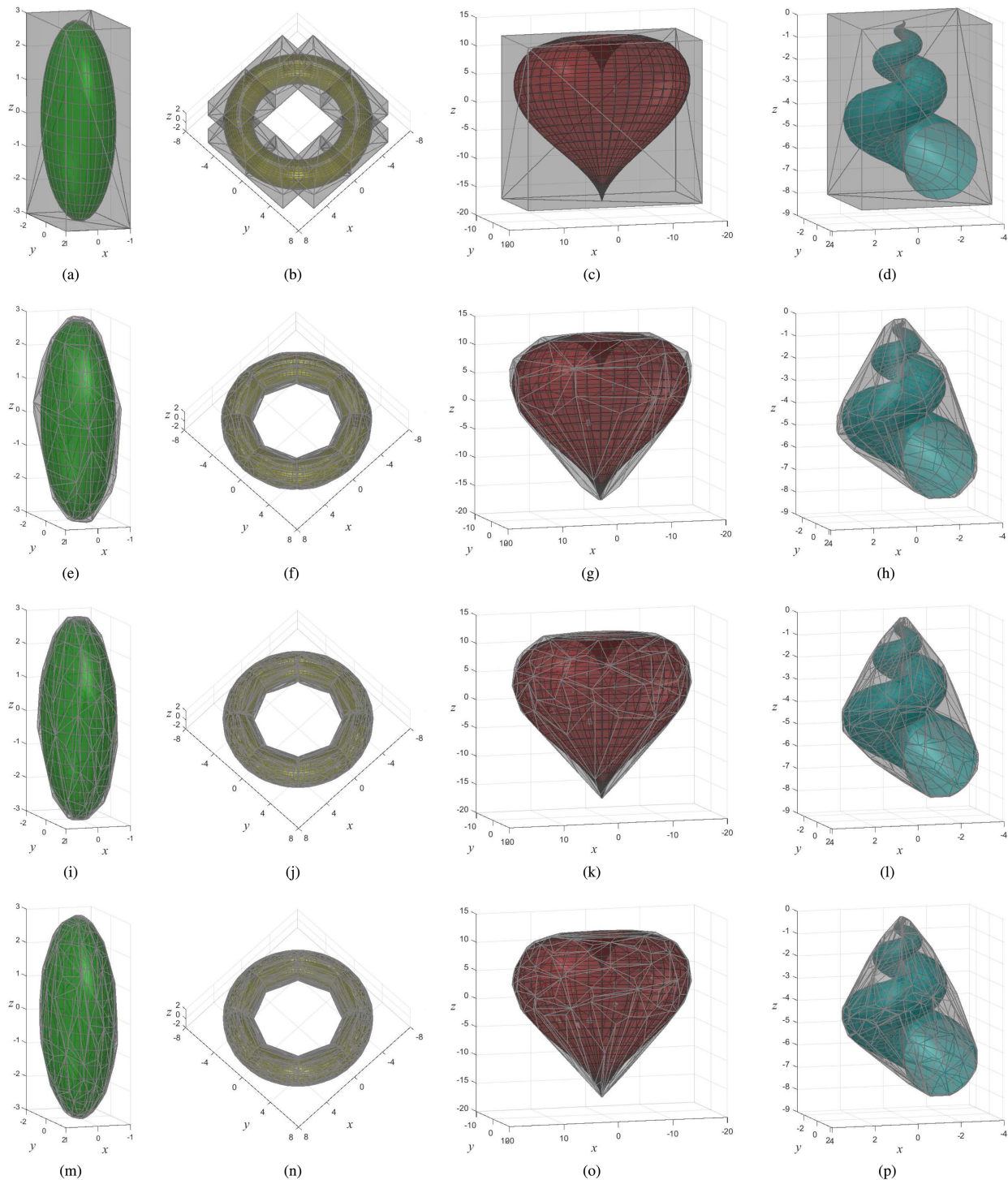


Fig. 4. Bounding polyhedra of the ellipsoid (first column), torus (second column), heart (third column), and seashell (fourth column) with axis-aligned planes (first row) and additional 30 (second row), 60 (third row), and 90 (fourth row) planes.

four objects, Algorithm 2 can achieve both high computational accuracy and efficiency by setting the threshold $\epsilon_X = 0.1$, as shown in Table II.

Example 2: Since the support function defines a plane that bounds and supports a body, we can compute a bounding polyhedron or even the convex hull of the body by computing the support function along a series of directions and forming a group

of planes supporting the body. Such an iterative algorithm has been described in the work [44]. Applying this algorithm with the proposed method for computing the support function, we can calculate bounding polyhedra for complex as well as simple geometric bodies, as shown in Fig. 4. As the algorithm iterates, more supporting planes are determined and the bounding polyhedron gets tighter and can eventually converge to the convex

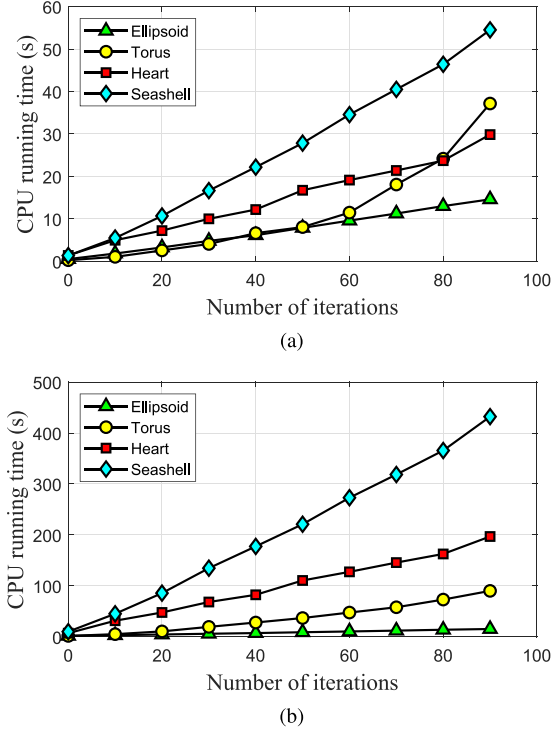


Fig. 5. CPU running time for computing bounding polyhedra. The threshold ϵ_X in Algorithm 2 is set to (a) 10^{-1} and (b) 10^{-2} , respectively.

hull of the body. Fig. 5 depicts the CPU running time of this computation in which Algorithm 2 is used to compute the support function.

III. MINIMUM DISTANCE COMPUTATION BETWEEN CONTINUOUS SURFACES

Distance is one of the most basic concepts in mathematics and has various applications in robotics. There are many definitions of distance and a common definition is the minimum Euclidean distance between points in two sets. Algorithms have been available for computing the minimum distance between compact convex sets [1], [2], [8]. For infinite point sets with nonconvex smooth boundaries, however, the computing of their minimum distance is difficult. In this section, based on the previous algorithm, we discuss a solution to this problem.

A. Mathematical Definitions

Let A and B be two infinite compact sets in \mathbb{R}^n , as depicted in Fig. 6. The Minkowski difference between them is the set defined as

$$A - B \triangleq \{\mathbf{a} - \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}. \quad (11)$$

It can be proved that $A \cap B = \emptyset$ (or $A \cap B \neq \emptyset$) is equivalent to $\mathbf{0} \notin A - B$ (or $\mathbf{0} \in A - B$). Throughout the following discussion, we assume that $A - B$ is of dimension n and has a nonempty interior in \mathbb{R}^n . The Euclidean distance between A

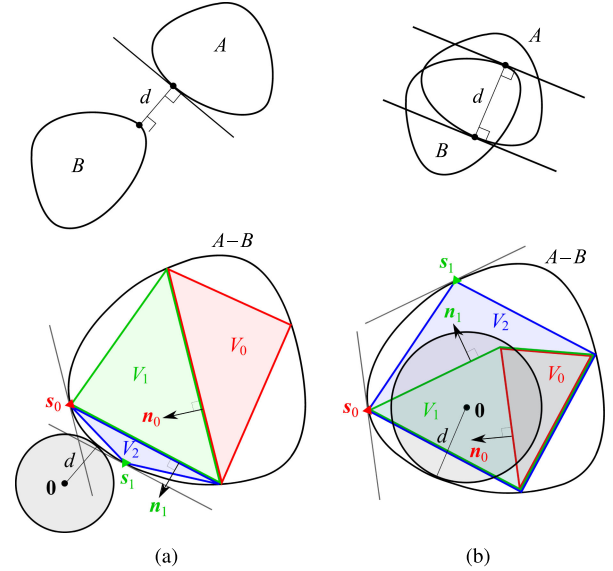


Fig. 6. Illustration of the minimum distance d between compact convex sets A and B . (a) A separated from B : The GJK algorithm iteratively generates a sequence of simplices in $A - B$ using the support mapping of $A - B$ and a face of the previous simplex such that the minimum distance \hat{d} from the origin to the simplex in the sequence converges to d [1], [2]. The iteration can be terminated once $\hat{d} + h_{A-B}(\mathbf{u}) < \epsilon_d$, where \mathbf{u} is the unit vector representing the direction of the minimum distance from the simplex to the origin and ϵ_d is the termination tolerance. (b) A intersecting with B : The algorithm iteratively expands a polytope in $A - B$ by adding the support mapping of $A - B$ as a new vertex to the polytope such that the minimum distance \hat{d} from the origin to the facets of the polytope converges to d [8], [12]. The iteration can be terminated once $h_{A-B}(\mathbf{u}) - \hat{d} < \epsilon_d$, where \mathbf{u} is the unit outward normal of the facet giving the minimum distance \hat{d} and ϵ_d is the termination tolerance. Please refer to relevant references for more details.

and B is defined as [46]

$$d(A, B) \triangleq \begin{cases} d^+(A, B) & \text{if } A \cap B = \emptyset \\ -d^-(A, B) & \text{if } A \cap B \neq \emptyset \end{cases} \quad (12)$$

where $d^+(A, B)$ and $d^-(A, B)$ are the Euclidean separation and penetration distances, respectively, which are defined as

$$d^+(A, B) \triangleq \min_{\mathbf{t} \in A - B} \|\mathbf{t}\| = \min_{\mathbf{t} \in \text{bd}(A - B)} \|\mathbf{t}\| \quad (13)$$

$$d^-(A, B) \triangleq \min_{\mathbf{t} \in A - B \setminus \text{int}(A - B)} \|\mathbf{t}\| = \min_{\mathbf{t} \in \text{bd}(A - B)} \|\mathbf{t}\| \quad (14)$$

where $\text{int}(\cdot)$ and $\text{bd}(\cdot)$ denote the interior and the boundary of a set, respectively. From (13), we see that $d^+(A, B)$ equals the minimum Euclidean distance from the origin to the points in $A - B$. Since $\mathbf{0} \notin A - B$ due to $A \cap B = \emptyset$ and we assume $\text{int}(A - B) \neq \emptyset$, $d^+(A, B)$ is obtained at a boundary point of $A - B$, as depicted in Fig. 6(a). As for $d^-(A, B)$, since $A - B \setminus \text{int}(A - B)$ is nothing but the boundary of $A - B$ when $\text{int}(A - B) \neq \emptyset$, $d^-(A, B)$ is the minimum Euclidean distance from the origin to the boundary of $A - B$, where the difference from $d^+(A, B)$ is $\mathbf{0} \in A - B$ because of $A \cap B \neq \emptyset$, as shown in Fig. 6(b).

B. Distance Algorithms

When both A and B are convex, $d^+(A, B)$ and $d^-(A, B)$ can be computed by the well-known GJK algorithm [1], [2] and the expanding polytope algorithm [8], [12], as depicted in Fig. 6(a) and (b), respectively. Starting with a simplex in $A - B$, the GJK algorithm calculates the support mapping of $A - B$ along the normal of the face containing the closest point in the simplex to the origin and uses the support mapping with the face to form a new simplex. By this iteration, the simplex progressively approaches the origin and the minimum distance between them converges to the minimum distance between the origin and $A - B$ or just $d^+(A, B)$ if $A \cap B = \emptyset$. To compute $d^-(A, B)$ when $A \cap B \neq \emptyset$, the expanding polytope algorithm iteratively grows a polytope containing the origin in $A - B$ by adding the support mapping of $A - B$ along the normal of the polytope's closest facet to the origin as a new vertex. As the algorithm iterates, the minimum distance from the origin to the facets of the polytope converges to the minimum distance from the origin to the boundary of $A - B$ or $d^-(A, B)$.

An essential operation in both algorithms is to calculate the support function h_{A-B} and mapping s_{A-B} of $A - B$, which can be simplified as

$$h_{A-B}(\mathbf{u}) = h_A(\mathbf{u}) + h_B(-\mathbf{u}) \quad (15a)$$

$$s_{A-B}(\mathbf{u}) = s_A(\mathbf{u}) - s_B(-\mathbf{u}). \quad (15b)$$

Thus, computing h_{A-B} and s_{A-B} has the same complexity as computing h_A , s_A and h_B , s_B separately, and there is no need to calculate the Minkowski difference $A - B$. When A and B are described by continuous parametric functions, their support functions can be computed by Algorithm 1 or 2.

Since the support function and mapping of the convex hull of a set are equal to those of the set itself [2], [45], if we apply the distance algorithms [1], [2], [8], [12] to the case where A or B is not convex, the yielded value, denoted by $\check{d}(A, B)$, is actually the minimum separation or penetration distance between the convex hull of A and the convex hull of B but it is worth noticing that none of the convex hulls is explicitly calculated. This can already be useful in some scenarios, such as collision detection between bodies as needed in robot motion planning and simulation. Nevertheless, we are still interested in the true minimum distance between A and B , for which a potential algorithm is discussed as follows.

In fact, $\check{d}(A, B)$ provides a lower bound of $d(A, B)$, since A or B is contained in its own convex hull and the minimum distance $\check{d}(A, B)$ between their convex hulls is less than or equal to their true minimum distance $d(A, B)$. Dividing A into smaller subsets A_j 's, we can deduce $\check{d}(A, B) \leq \check{d}(A_j, B)$ for $\forall j$ and $\check{d}(A, B) \leq \min_j \check{d}(A_j, B)$, which implies that the lower bound is increasing in the dividing of A . The same property can be derived for B as well. With this lower bound, we can derive a B&B algorithm as described in Algorithm 3. Assume that A and B are specified by parametric functions with all the parameters written as \mathbf{x} for which the total domain is denoted by X . At every iteration of the algorithm, X^* is the domain in the list \mathcal{L} whose lower bound is minimal and it is divided into subdomain X_j 's, by which we divide corresponding sets into subset A_j 's

Algorithm 3: Algorithm for the Minimum Distance.

Input: Sets A and B

Output: The minimum distance $d(A, B)$

```

1:  $d^* \leftarrow \check{d}(A, B)$ 
2:  $X^* \leftarrow$  domains of all parameters for  $A$  and  $B$ 
3:  $\mathcal{L} \leftarrow \emptyset$ 
4: while  $|X^*| > \epsilon_X$  do
5:   Divide  $X^*$  into subdomains  $X_j$ 's
6:   for each  $X_j$  do
7:      $\check{d}_j \leftarrow$  the lower bound  $\check{d}(A_j, B_j)$  over  $X_j$ 
8:     Add  $X_j$  to the list  $\mathcal{L}$ 
9:   end for
10:   $d^* \leftarrow$  the minimum lower bound for domains in
    the list  $\mathcal{L}$ 
11:   $X^* \leftarrow$  the domain in the list  $\mathcal{L}$  whose lower bound
    is minimal
12:  Remove  $X^*$  from the list  $\mathcal{L}$ 
13: end while
14: return  $X^*$  and  $d^*$ 

```

and B_j 's. Then, the lower bound $\check{d}(A_j, B_j)$ is calculated for each X_j added to \mathcal{L} . The iteration stops when the domain X^* is small enough. When $A \cap B = \emptyset$, there always exists a domain in the list \mathcal{L} such that the corresponding subsets of A and B contain the pair of points whose distance is the true minimum distance $d^+(A, B)$ between A and B . In this case, therefore, the minimum lower bound d^* in \mathcal{L} is always bounded above by $d^+(A, B)$ and guaranteed to converge to $d^+(A, B)$ even though A and B are not convex. In case that $A \cap B \neq \emptyset$, however, d^* is bounded above by and, thus, converges to zero rather than $d^-(A, B)$ as the algorithm iterates. It is worthwhile noting that this is still a useful result as it indicates that there are intersections between two sets and implies potential collisions to be noticed by motion planning algorithms.

C. Numerical Examples

Here, we report some numerical tests on minimum distance computation with the aid of Algorithm 2 or by Algorithm 3.

Example 3: We first use the GJK algorithm [1], [2] and the expanding polytope algorithm [8], [12] to compute the minimum distance between the convex hulls of two hearts, which are nonconvex bodies, as shown in Fig. 7(a). Originally, the two algorithms cannot be applied to this case because the computation of the support function of nonconvex bodies is unavailable. Now, we can do so with the aid of Algorithm 2. We first use the GJK algorithm [1], [2] to determine if two convex hulls intersect and then calculate their minimum separation distance when they do not. In case that they intersect, the algorithm [8] is called to compute the minimum penetration distance between the convex hulls. The support function of each heart is calculated by Algorithm 2 with $\epsilon_X = 10^{-1}$, which has turned out to be small enough for accurately calculating the support function, as revealed by the test results reported in Table II. We conduct 10^3 trials with randomized relative positions and orientations between two hearts and obtain the average computation times

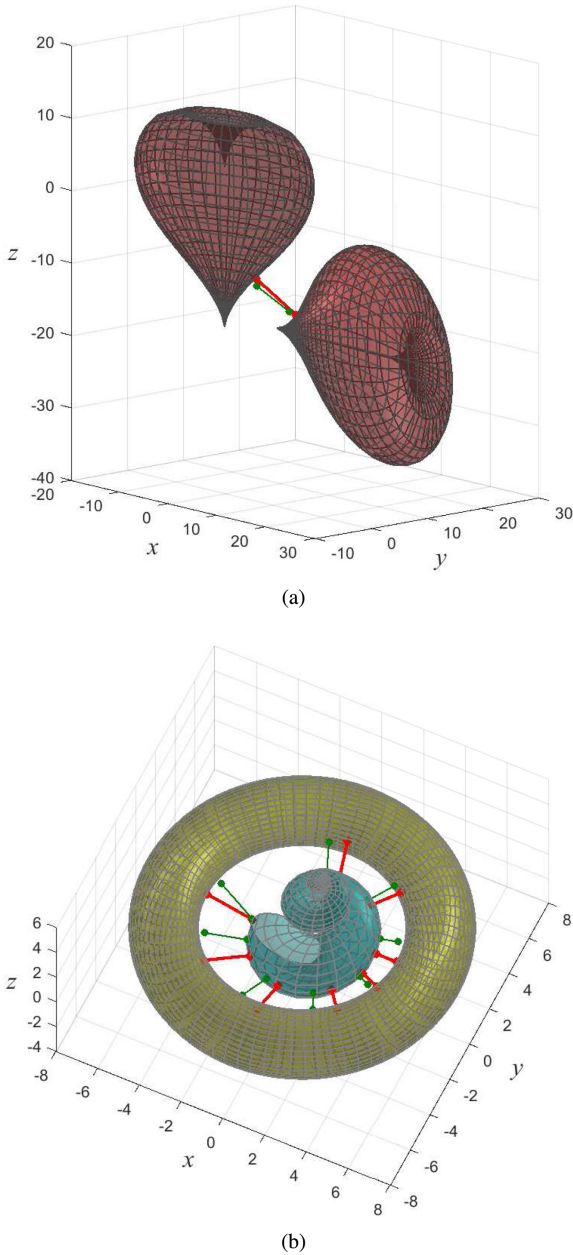


Fig. 7. Examples of minimum distance computation between (a) two hearts and (b) a torus and a seashell. The torus is divided into eight segments in the computation. The green and red lines represent the minimum distance between the convex hulls of and the true minimum distance between two bodies, respectively, where the true values are notably greater in the shown cases.

and numbers of iterations of the two distance algorithms with respect to different termination tolerances, as displayed in the left half of Table III. In the old case, where the given bodies are convex and their support functions can be calculated analytically, the computation times of the two distance algorithms range from several to tens of milliseconds on a modern PC [12]. Here, their computation times are two to three orders of magnitude longer while the numbers of iterations remain at almost the same level. The increase in the computation time is purely due to the computing of the support function with an iterative numerical algorithm rather than in an analytical way.

We also compute the true minimum separation distance between two hearts rather than their convex hulls with Algorithm 3 in the abovementioned random trials where the convex hulls of two hearts are determined to be separate. Here, we take $\epsilon_X = 10^{-1}$, 10^{-2} , and 10^{-3} , respectively, for Algorithm 3 and $\epsilon_d = 10^{-4}$ for the distance algorithms [1], [2], [8]. The results are collected in the left half of Table IV. From the second column of Table IV, we see that the true minimum distance between two hearts is often greater than the minimum distance between their convex hulls, which is in accordance with the fact that the hearts are concave and their convex hulls expand the actual bodies, leading to a reduced distance between them. Fig. 7(a) depicts one case where the true minimum distance is notably greater.

Example 4: We also conduct the same distance computation between a torus and a seashell, as depicted in Fig. 7(b). The torus is divided into eight identical segments as in Fig. 4. The minimum distance between the convex hulls of the seashell and each segment is computed by the distance algorithms [1], [2], [8] and the results are reported in the right half of Table III.

The right half of Table IV exhibits the results of Algorithm 3 to compute the true minimum separation distance between the seashell and each segment of the torus. These results reflect the same performance of the distance algorithms [1], [2], [8] with the aid of Algorithm 2 and of Algorithm 3 as in Example 3 for the minimum distance computation.

From the CPU running time and the number of iterations shown in Table IV, we notice that the current implementation of Algorithm 3 is not fast enough for real-time uses. To enhance its computational efficiency, we also tried the parallel implementation of the for-loop in Algorithm 3 to compute the lower bound $\tilde{d}(A_j, B_j)$ over each subdomain X_j using the “parfor” command provided by the parallel computing toolbox of MATLAB. The “parfor” command distributes the loop iterations onto a parallel pool of available local workers (four workers on the used desktop). By this straightforward parallelism, the computation time of Algorithm 3 is reduced by about three times, which is smaller than the ideal speed up of a factor of four on four workers due to the parallel overhead including the time required for data transfer. It is noted that we have the option to parallelize the for-loop in Algorithm 2, which actually runs as a subalgorithm in the for-loop in Algorithm 3. Then, parallelizing both for-loops causes nested parfor-loops, which is not allowed in MATLAB. Hence, we choose to parallelize the outer loop, namely the for-loop in Algorithm 3, which incurs a smaller parallel overhead. In the future, we will explore other parallel implementations, such as nested parallelism, to further improve the computational efficiency of the algorithms.

IV. OPTIMAL GRASP PLANNING ON CONTINUOUS SURFACES

A grasp on an object can be modeled as a set of contacts on the object’s surface and the goal of optimal grasp planning is to compute the contact locations providing the best performance quality. Owing to the nonlinearity of a grasp quality measure and a general object’s surface, which can be piecewise parameterized as nonlinear functions, optimal grasp planning

TABLE III
RESULTS OF DISTANCE COMPUTATION BY EXISTING ALGORITHMS

ϵ_d	Heart & Heart						Torus & Seashell					
	t^+	N^+	t^0	N^0	t^-	N^-	t^+	N^+	t^0	N^0	t^-	N^-
10^{-2}	2.198	10.41	0.808	3.53	2.954	11.09	0.901	4.37	0.561	2.64	2.146	9.60
10^{-3}	2.759	13.13	0.813	3.53	3.724	15.06	1.118	5.72	0.560	2.64	2.726	12.95
10^{-4}	3.282	15.76	0.812	3.53	4.435	18.99	1.328	7.05	0.560	2.64	3.279	16.17
10^{-5}	3.797	18.39	0.811	3.53	5.112	22.69	1.548	8.43	0.560	2.64	3.804	19.23

ϵ_d —Termination tolerance for the distance algorithms [1], [2], [8] as explained in the caption of Fig. 6;
 t , N —Average CPU running time (unit: second) and number of iterations of a distance algorithm;
 Superscript “+0” refers to the GJK algorithm [1], [2] in the separation/penetration case;
 Superscript “-” refers to the algorithm [8] for the penetration distance.

TABLE IV
RESULTS OF TRUE DISTANCE COMPUTATION BY ALGORITHM 3

ϵ_X	Heart & Heart							Torus & Seashell						
	Δ_{ave}	t_{ave}	t_{max}	t_{min}	N_{ave}	N_{max}	N_{min}	Δ_{ave}	t_{ave}	t_{max}	t_{min}	N_{ave}	N_{max}	N_{min}
10^{-1}	0.1726	47.85	261.32	13.32	41.53	2121	22	0.0309	18.00	197.95	3.47	48.36	1610	23
10^{-2}	0.1731	61.31	1213.6	14.98	218.81	17046	38	0.0310	22.30	945.99	4.14	168.92	23839	38
10^{-3}	0.1731	86.13	4546.0	15.26	812.23	10^6	50	0.0310	44.07	2664.2	4.73	1137.4	10^6	51

Δ —The minimum separation distance between two bodies minus the minimum distance between their convex hulls;
 t —CPU running time of Algorithm 3 (unit: second);
 N —Number of iterations of Algorithm 3 (10^6 is the maximum number of iterations allowed).

is a highly-nonlinear optimization problem, for which computing the globally optimal solution is extremely difficult. In this section, we attempt to solve this problem with another B&B algorithm, in which the proposed B&B algorithm for the support function is used in calculating an upper bound of grasps’ quality over any domain of the object’s surface.

A. Preliminary Knowledge

Consider using m contacts to form a grasp on a 3-D object. Let $\mathbf{p}_i \in \mathbb{R}^3$ for $i = 1, 2, \dots, m$ be the position of contact i with respect to the object coordinate frame attached at the center of mass of the object. Assume that the object’s surface is continuous and the three components of \mathbf{p}_i are all specified by continuous functions of the same surface parameter $\mathbf{x}_i \in \mathbb{R}^2$. Let X_i be the domain of \mathbf{x}_i . By differentiating \mathbf{p}_i with respect to \mathbf{x}_i , we can derive two orthogonal unit tangent vectors $\mathbf{o}_i \in \mathbb{R}^3$ and $\mathbf{t}_i \in \mathbb{R}^3$ as well as the inward unit normal $\mathbf{n}_i \in \mathbb{R}^3$ at \mathbf{p}_i such that $\mathbf{n}_i = \mathbf{o}_i \times \mathbf{t}_i$. Then, \mathbf{o}_i , \mathbf{t}_i , and \mathbf{n}_i establish a local right-handed coordinate frame at \mathbf{p}_i . Moreover, we can deduce that \mathbf{p}_i , \mathbf{n}_i , \mathbf{o}_i , and \mathbf{t}_i have Lipschitz continuity, i.e.,

$$\|\mathbf{p}_i(\mathbf{x}_i) - \mathbf{p}_i(\mathbf{x}'_i)\| \leq L_{pi} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (16a)$$

$$\|\mathbf{n}_i(\mathbf{x}_i) - \mathbf{n}_i(\mathbf{x}'_i)\| \leq L_{ni} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (16b)$$

$$\|\mathbf{o}_i(\mathbf{x}_i) - \mathbf{o}_i(\mathbf{x}'_i)\| \leq L_{oi} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (16c)$$

$$\|\mathbf{t}_i(\mathbf{x}_i) - \mathbf{t}_i(\mathbf{x}'_i)\| \leq L_{ti} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (16d)$$

where \mathbf{x}'_i is a value of the surface parameter other than \mathbf{x}_i and L_{pi} , L_{ni} , L_{oi} , and L_{ti} are nonnegative real constants, which can be estimated from the parametric expressions of \mathbf{p}_i , \mathbf{n}_i , \mathbf{o}_i , and \mathbf{t}_i , respectively, similarly to (9).

The contact force $\mathbf{f}_i \in \mathbb{R}^3$ at \mathbf{p}_i can be expressed in the local coordinate frame as $\mathbf{f}_i = [f_{i1} \ f_{i2} \ f_{i3}]^T$, where f_{i1} , f_{i2} , and f_{i3} are the components of \mathbf{f}_i along \mathbf{n}_i , \mathbf{o}_i , and \mathbf{t}_i , respectively. It can be converted to a wrench \mathbf{w}_i with respect to the object coordinate

frame by

$$\mathbf{w}_i = \mathbf{G}_i \mathbf{f}_i$$

where \mathbf{G}_i is a linear mapping that can be written as

$$\mathbf{G}_i = \begin{bmatrix} \mathbf{n}_i & \mathbf{o}_i & \mathbf{t}_i \\ \mathbf{p}_i \times \mathbf{n}_i & \mathbf{p}_i \times \mathbf{o}_i & \mathbf{p}_i \times \mathbf{t}_i \end{bmatrix} \in \mathbb{R}^{6 \times 3}. \quad (17)$$

To avoid slippage at contact, \mathbf{f}_i must belong to the following convex cone, known as the friction cone [47]

$$F_i \triangleq \left\{ \mathbf{f}_i \in \mathbb{R}^3 \mid f_{i1} \geq 0, \sqrt{f_{i2}^2 + f_{i3}^2} \leq \mu_i f_{i1} \right\} \quad (18)$$

where μ_i is the Coulomb friction coefficient. The primitive contact force set U_i consists of contact forces with unit normal component on the boundary of F_i [14]

$$U_i \triangleq \left\{ \mathbf{f}_i \in \mathbb{R}^3 \mid f_{i1} = 1, \sqrt{f_{i2}^2 + f_{i3}^2} = \mu_i \right\}. \quad (19)$$

The image W_i of U_i through \mathbf{G}_i as given below is called the primitive contact wrench set

$$W_i = \mathbf{G}_i(U_i). \quad (20)$$

The grasp wrench set is defined as the convex hull of the union of W_i , $i = 1, 2, \dots, m$ for all contacts [19], [48], i.e.,

$$W \triangleq \text{CH} \left(\bigcup_{i=1}^m W_i \right) \quad (21)$$

where $\text{CH}(\cdot)$ denotes the convex hull of a set. The set W consists of all the wrenches that can be applied to the object by all m contacts with unit sum of normal contact forces. A well-known grasp quality measure is defined in terms of the minimum distance (12) from the origin to W [19], [21], i.e.,

$$\sigma \triangleq -d(W, \mathbf{0}) = \min_{\mathbf{u}^T \mathbf{u} = 1} h_W(\mathbf{u}) \quad (22)$$

where $h_W(\mathbf{u})$ is the support function of W along a vector $\mathbf{u} \in \mathbb{R}^6$. The value σ is positive and nonpositive for a grasp having and not having force closure, respectively, and a greater σ implies a better grasp. Physically, σ reflects the overall ability of a grasp to apply wrenches to the object in all directions.

From the abovementioned arguments, σ depends on the parameters \mathbf{x}_i 's specifying the contact locations. Then, the goal of optimal grasp planning is to compute $\mathbf{x}_i \in X_i, i = 1, 2, \dots, m$ such that σ is maximal, which can be written as

$$\begin{cases} \text{maximize } \sigma \\ \text{subject to } \mathbf{x}_i \in X_i, i = 1, 2, \dots, m \end{cases}. \quad (23)$$

Because the object's surface, the friction cone (18), and the distance function in (12) are all nonlinear, σ is a nonlinear function of \mathbf{x}_i 's and (23) is a nonlinear optimization problem with many local maxima. It is extremely hard to compute the globally optimal solution to such a problem. To the best of authors' knowledge, there is no effective algorithm to solve this problem without any approximation until now.

B. Planning Algorithm

For problem (23), we propose a B&B algorithm, which is aimed at yielding a solution with guaranteed or even global optimality. To do this, we first introduce an upper bound of σ over the domain $X = X_1 \otimes X_2 \otimes \dots \otimes X_m$ of the problem, where \otimes represents the Cartesian product of sets. Referring to the primitive contact wrench set W_i defined by (20) for a single contact point, we can define a wrench set \hat{W}_i for the piece of the object's surface corresponding to the domain X_i as the union of W_i for all points in the piece

$$\hat{W}_i \triangleq \bigcup_{\mathbf{x}_i \in X_i} W_i. \quad (24)$$

Then, similarly to the grasp wrench set W and quality value σ , we can define a set \hat{W} as (21) with \hat{W}_i replacing W_i and a scalar value $\hat{\sigma}$ as (22) with \hat{W} replacing W . From (24), the grasp wrench set W for any grasp in the domain X is contained in \hat{W} , by which we obtain $h_W(\mathbf{u}) \leq h_{\hat{W}}(\mathbf{u})$ for $\forall \mathbf{u}$ and $\sigma \leq \hat{\sigma}$ from (22). Hence, $\hat{\sigma}$ is an upper bound of σ over X . Furthermore, for the same reason, we can derive $\hat{W}' \subset \hat{W}$ and $\hat{\sigma}' \leq \hat{\sigma}$ for any subdomain $X' \subset X$, where \hat{W}' and $\hat{\sigma}'$ are the wrench set and the upper bound for X' , respectively. This implies that the upper bound is monotonically decreasing during the dividing of a domain as needed in a B&B algorithm.

The wrench set \hat{W} and/or the scalar value $\hat{\sigma}$ have been proposed before for discrete surface point sets [42], special surface elements (points, line segments, and convex facets) [49], and the entire object's surface [22]. However, the computing of $\hat{\sigma}$ is limited to the special cases and there is no algorithm to compute $\hat{\sigma}$ for a general continuous surface. Prior to the discussion on how to compute the upper bound $\hat{\sigma}$, we recall how the grasp quality measure σ is computed. As defined in terms of the minimum distance in (22), σ can be calculated by the distance algorithms [2], [8], as depicted in Fig. 6, and the support function $h_W(\mathbf{u})$ and mapping $\mathbf{s}_W(\mathbf{u})$ of W need to be calculated at every iteration. Luckily, $h_W(\mathbf{u})$ and $\mathbf{s}_W(\mathbf{u})$ for

any \mathbf{u} can be calculated in a closed form as [14]

$$h_W(\mathbf{u}) = h_{W_{i^*}}(\mathbf{u}) \quad (25a)$$

$$\mathbf{s}_W(\mathbf{u}) = \mathbf{s}_{W_{i^*}}(\mathbf{u}) \quad (25b)$$

where $i^* = \arg \max_{i=1,2,\dots,m} h_{W_i}(\mathbf{u})$. Furthermore, $h_{W_i}(\mathbf{u})$ and $\mathbf{s}_{W_i}(\mathbf{u})$ can be calculated by

$$h_{W_i}(\mathbf{u}) = d_{i1} + \mu_i \sqrt{d_{i2}^2 + d_{i3}^2} \quad (26a)$$

$$\mathbf{s}_{W_i}(\mathbf{u}) = \mathbf{G}_i \begin{bmatrix} 1 & \frac{\mu_i d_{i2}}{\sqrt{d_{i2}^2 + d_{i3}^2}} & \frac{\mu_i d_{i3}}{\sqrt{d_{i2}^2 + d_{i3}^2}} \end{bmatrix}^T \quad (26b)$$

where d_{i1} , d_{i2} , and d_{i3} are the components of \mathbf{d}_i given by

$$\mathbf{d}_i = \mathbf{u}^T \mathbf{G}_i. \quad (27)$$

In case that $\sqrt{d_{i2}^2 + d_{i3}^2}$ in (26) is zero, which rarely happens though, $\mathbf{s}_{W_i}(\mathbf{u})$ can be taken to be the first column of \mathbf{G}_i , i.e., $\mathbf{s}_{W_i}(\mathbf{u}) = [\mathbf{n}_i^T \mathbf{p}_i^T \times \mathbf{n}_i^T]^T$, such that $h_{W_i}(\mathbf{u}) = d_{i1}$.

Also defined as the minimum distance, the upper bound $\hat{\sigma}$ can be calculated by the same distance algorithms [2], [8] as used to compute σ . However, we shall compute $h_{\hat{W}}(\mathbf{u})$ in this case, which is more difficult than the computing of $h_W(\mathbf{u})$, since \hat{W} is defined over m domains X_i 's rather than m contact points. A closed-form expression of $h_{\hat{W}}(\mathbf{u})$ might be derivable for some special surfaces, such as a planar surface. In general, for a curved continuous surface we can only compute $h_{\hat{W}}(\mathbf{u})$ numerically. To do this, similarly to (25), we first have

$$h_{\hat{W}}(\mathbf{u}) = h_{\hat{W}_{i^*}}(\mathbf{u}) \quad (28a)$$

$$\mathbf{s}_{\hat{W}}(\mathbf{u}) = \mathbf{s}_{\hat{W}_{i^*}}(\mathbf{u}) \quad (28b)$$

where $i^* = \arg \max_{i=1,2,\dots,m} h_{\hat{W}_i}(\mathbf{u})$. From (24), we further derive

$$h_{\hat{W}_i}(\mathbf{u}) = \max_{\mathbf{x}_i \in X_i} h_{W_i}(\mathbf{u}) \quad (29a)$$

$$\mathbf{s}_{\hat{W}_i}(\mathbf{u}) = h_{W_i^*}(\mathbf{u}) \quad (29b)$$

where W_i^* is the primitive wrench contact set W_i at \mathbf{x}_i^* for which $h_{W_i}(\mathbf{u})$ is maximal over X_i . Comparing (29a) with (2), we see that $h_{\hat{W}_i}(\mathbf{u})$ can be calculated in the same way as $h_S(\mathbf{u})$ if we can derive a monotonically decreasing upper bound for $h_{W_i}(\mathbf{u})$ as for $\mathbf{u}^T f(\mathbf{x})$. From (17), (26a), and (27) as well as the Lipschitz continuity of \mathbf{p}_i , \mathbf{n}_i , \mathbf{o}_i , and \mathbf{t}_i described by (16), it is not so difficult to deduce that $h_{W_i}(\mathbf{u})$ has Lipschitz continuity as well, which can be written as

$$|h_{W_i}(\mathbf{u}) - h_{W_i'}(\mathbf{u})| \leq L_{hi} \|\mathbf{x}_i - \mathbf{x}_i'\| \quad (30)$$

where W_i' is the primitive contact wrench set at \mathbf{x}_i' and \mathbf{x}_i' is an arbitrary point in X_i . Equation (30) is similar to (4). Then, following (5) and (6), we attain an upper bound of $h_{W_i}(\mathbf{u})$ as

$$\hat{h}_{W_i}(\mathbf{u}) \triangleq h_{\bar{W}_i}(\mathbf{u}) + L_{hi} \max_{\mathbf{x}_i \in X_i} \|\mathbf{x}_i - \bar{\mathbf{x}}_i\| \quad (31)$$

where $\bar{\mathbf{x}}_i$ is the center of domain X_i and \bar{W}_i is the primitive contact wrench set at $\bar{\mathbf{x}}_i$. Furthermore, similarly to (7), we can prove that the upper bound given by (31) is monotonically

Algorithm 4: Algorithm for the Optimal Grasp.

Input: Object surface functions with domains X_i ,
 $i = 1, 2, \dots, m$

Output: Solution \mathbf{x}^* giving an optimal grasp on the object

```

1:  $\hat{X} \leftarrow X$ 
2:  $\hat{\sigma} \leftarrow$  the upper bound of  $\sigma$  over  $X$ 
3:  $\mathbf{x}^* \leftarrow \emptyset$ 
4:  $\sigma^* \leftarrow 0$ 
5:  $\mathcal{L} \leftarrow \emptyset$ 
6: while  $\hat{X} \neq \emptyset$  and  $\hat{\sigma} - \sigma^* > \epsilon$  do
7:   Use any heuristic to find a relatively better grasp in
   domain  $\hat{X}$  with corresponding surface parameters
    $\tilde{\mathbf{x}}$  and quality value  $\tilde{\sigma}$ 
8:   if  $\tilde{\sigma} > \sigma^*$  then
9:      $\mathbf{x}^* \leftarrow \tilde{\mathbf{x}}$ 
10:     $\sigma^* \leftarrow \tilde{\sigma}$ 
11:   end if
12:   if  $|\hat{X}| > \epsilon_X$  then
13:     Divide  $\hat{X}$  into subdomains  $X_j$ 's
14:     for each  $X_j$  do
15:        $\hat{\sigma}_j \leftarrow$  the upper bound of  $\sigma$  over  $X_j$ 
16:       if  $\hat{\sigma}_j > \sigma^*$  then
17:         Add  $X_j$  to the list  $\mathcal{L}$ 
18:       end if
19:     end for
20:   end if
21:    $\hat{\sigma} \leftarrow$  the maximum upper bound for domains in
   the list  $\mathcal{L}$ 
22:    $\hat{X} \leftarrow$  the domain in the list  $\mathcal{L}$  whose upper bound
   is maximal
23:   Remove  $\hat{X}$  from the list  $\mathcal{L}$ 
24: end while
25: return  $\mathbf{x}^*$  and  $\sigma^*$ 

```

decreasing in dividing X_i . With this upper bound, therefore, $h_{\hat{W}_i}(\mathbf{u})$ can be calculated similarly by Algorithm 1 or 2 and consequently $\hat{\sigma}$ by the distance algorithms [2], [8].

With the upper bound of the grasp quality over a domain, the B&B algorithm for solving (23) is described in Algorithm 4. At every iteration of the algorithm, a local search or sampling in the selected domain \hat{X} is performed first to quickly obtain a grasp with relatively higher quality in \hat{X} and possibly update the current best grasp. If \hat{X} is small enough, it is believed that the attained grasp is close enough to the optimal grasp in \hat{X} and \hat{X} will not be further divided; otherwise, \hat{X} is divided into subdomains, which will be added to the list \mathcal{L} if their upper bounds are greater than the quality value σ^* of the current best grasp. Finally, \hat{X} is updated with the domain in \mathcal{L} whose upper bound is maximal. In case that \mathcal{L} is empty at this moment, we can simply set \hat{X} to empty to stop the algorithm.

Algorithm 4 is aimed at solving (23) globally, for which the condition is that the local optimizer can compute the globally optimal grasp in any domain \hat{X} with its size $|\hat{X}| \leq \epsilon_X$. To ensure this, the tolerance ϵ_X on $|\hat{X}|$ tends to be small, which will facilitate the computing of the globally optimal grasp in \hat{X} but on the other hand cause more domains to be added to

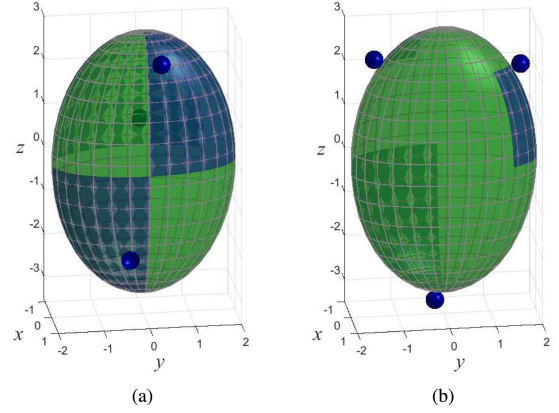


Fig. 8. Grasps with three contacts on the ellipsoid yielded by Algorithm 4 at the (a) 3rd iteration with $\sigma = 0.0730$ and (b) 139th iteration with $\sigma = 0.1632$.

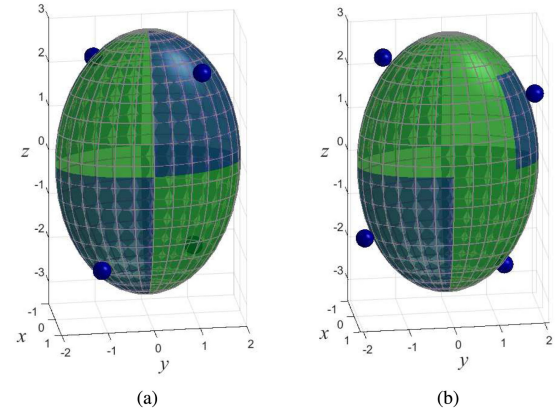


Fig. 9. Grasps with four contacts on the ellipsoid yielded by Algorithm 4 at the (a) 0th iteration with $\sigma = 0.2017$ and (b) 2nd iteration with $\sigma = 0.2385$. The shadow areas indicate the domains in which the contact locations are obtained by the local search using the interior-point algorithm.

the list \mathcal{L} and increase the required number of iterations by Algorithm 4. In practice, we may set other stopping criteria, such as the maximum allowable iteration number or running time, for the algorithm to compute near-optimal grasps within an acceptable time and give consideration to both its result's optimality and computational efficiency.

C. Numerical Examples

We again implemented Algorithm 4 in MATLAB on the desktop with an Intel Core i7-6700 3.40 GHz CPU and 16 GB RAM and tested it on several objects. To seek a good grasp in the selected domain \hat{X} at every iteration of Algorithm 4, we call the interior-point algorithm provided by the optimization toolbox of MATLAB with its default setting and the middle point of \hat{X} as the initial point to solve the optimization problem same as (23) over \hat{X} . The tolerance ϵ_X on $|\hat{X}|$ is taken to be 10^{-1} and $|\hat{X}|$ is defined to be the maximum interval in \hat{X} . In addition, we set the maximum running time allowed for Algorithm 4 to 48 h.

Example 5: We first use the algorithm to compute optimal grasps with three and four contacts on the ellipsoid, as shown in Figs. 8 and 9, respectively. Since the ellipsoid is a symmetric object, there will be many equivalent subdomains if we set the

TABLE V
RESULTS OF ALGORITHM 4 ON THE ELLIPSOID

i	Initial domains for (θ_i, ϕ_i)	t	N	Contacts (θ_i, ϕ_i)	σ	t	N	Contacts	σ
1	$[0, \pi/2] \times [0, \pi/2]$	1249.96	3	(0.9226, 0.2208)	0.0730	129198	139	(0.7291, 1.5672)	0.1632
2	$[0, \pi/2] \times [0, 3\pi/2]$			(0.1835, 3.1035)				(0.8365, 4.7098)	
3	$[-\pi/2, 0] \times [0, 2\pi]$			(-0.6731, 6.0299)				(-1.4892, 4.6953)	
1	$[0, \pi/2] \times [0, \pi/2]$	366.320	0	(0.7854, 0.7854)	0.2017	111718	2	(0.5194, 1.4132)	0.2385
2	$[0, \pi/2] \times [\pi, 3\pi/2]$			(0.7854, 3.9270)				(0.8712, 4.4825)	
3	$[-\pi/2, 0] \times [\pi/2, \pi]$			(-0.7854, 2.3562)				(-0.9334, 1.8419)	
4	$[-\pi/2, 0] \times [3\pi/2, 2\pi]$			(-0.7854, 5.4978)				(-0.5374, 4.9057)	

t , N —CPU running time (unit: second) and number of iterations of the algorithm.

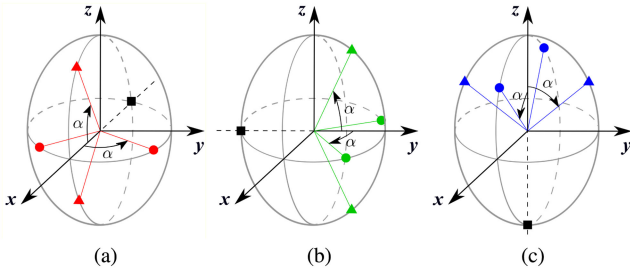


Fig. 10. Intuitive search for the globally optimal grasp with three contacts on the ellipsoid. One contact (marked by the black square) is outermost on the (a) $-x$, (b) $-y$, (c) $-z$ axis and the other two are symmetrically located on the boundary of the (a) xy or xz , (b) yx or yz , (c) zx or zy cross section.

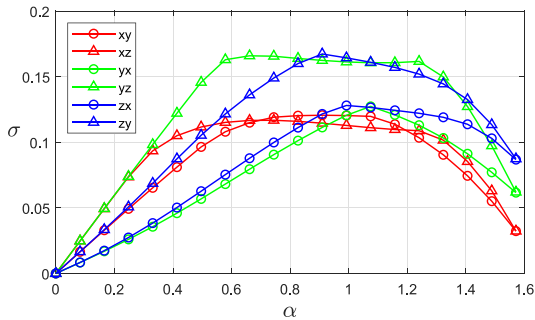


Fig. 11. Quality value σ of the grasp with respect to the variable α as depicted in Fig. 10.

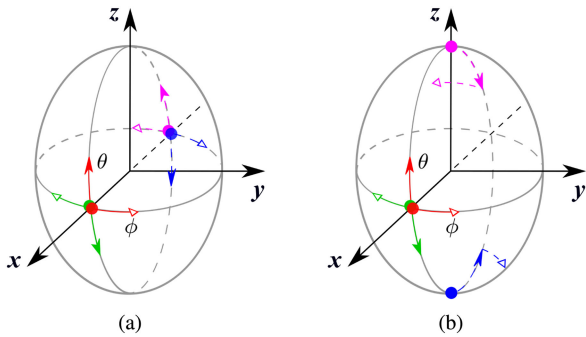


Fig. 12. Intuitive search for the globally optimal grasp with four contacts on the ellipsoid. One contact (in red color) moves on one eighth of the ellipsoid in the first octant and the other three (in different colors) move accordingly. The pair of solid and hollow arrows in the same color as a contact indicates two moving directions of the contact. The movements of contacts indicated by the solid or hollow arrows are described by the same parameter θ or ϕ , and the points shown in the figure are the initial contact locations.

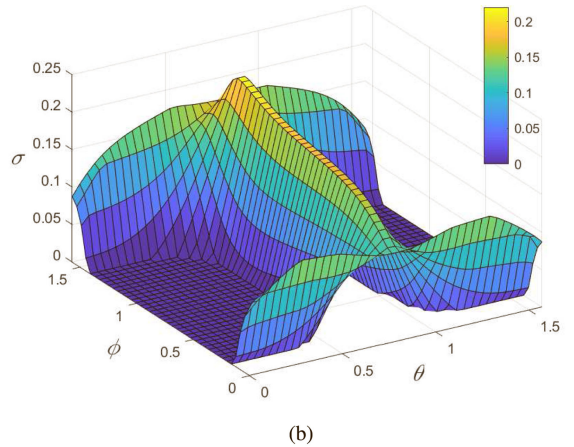
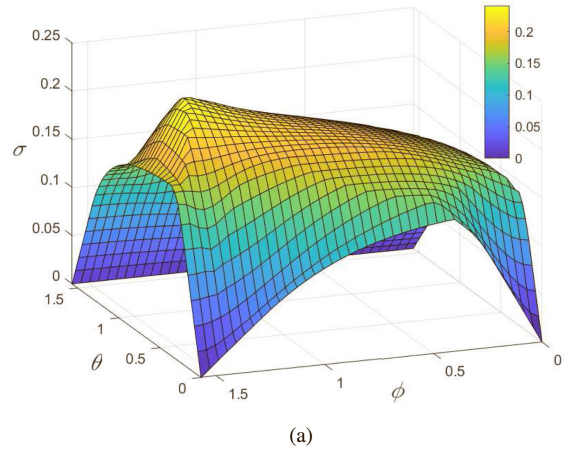


Fig. 13. Quality value σ of the grasp with respect to the variables θ and ϕ as depicted in Fig. 12. The values σ for nonforce-closure grasps are forced to be zero in these plots.

initial domain for each contact to be $\theta \in [-\pi/2, \pi/2]$ and $\phi \in [0, 2\pi]$. To avoid this, we set the initial domain as listed in the first column of Table V. In the case of three contacts, after 3 iterations of Algorithm 4 in about 20 min, we obtain the first force-closure grasp with $\sigma = 0.0730$, as shown in Fig. 8(a). After 139 iterations in nearly 36 h, we obtain a much better grasp with $\sigma = 0.1632$, as shown in Fig. 8(b), which is the best grasp that we obtain within the time limit. Intuitively, the globally optimal grasp should be attained at one of the cases where one contact is located at the outermost point on the ellipsoid along the $-x$ (resp. $-y$ and $-z$) direction and the other two contacts are

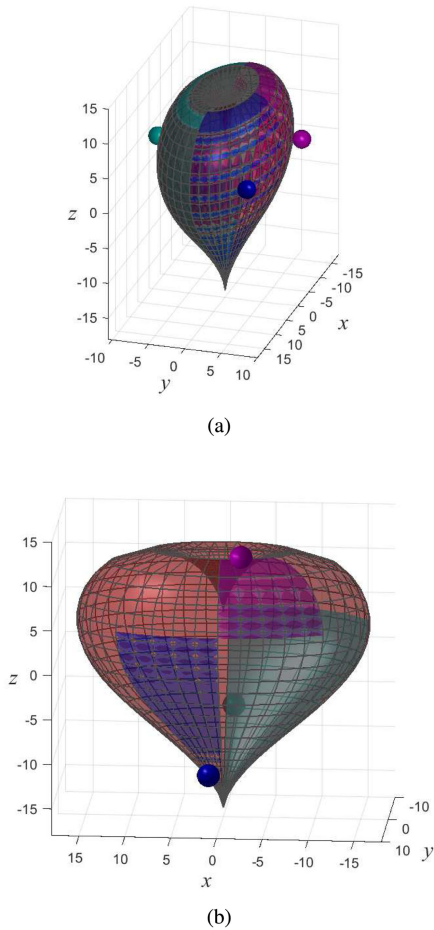


Fig. 14. Grasps with three contacts on the heart yielded by Algorithm 4 at the (a) 0th iteration with $\sigma = 0.1439$ and (b) 84th iteration with $\sigma = 0.1817$.

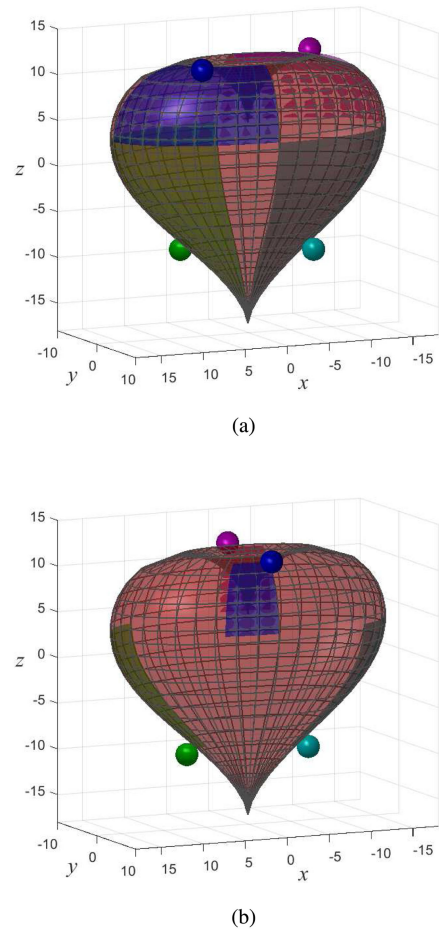


Fig. 15. Grasps with four contacts on the heart yielded by Algorithm 4 at the (a) 0th iteration with $\sigma = 0.2640$ and (b) 37th iteration with $\sigma = 0.3963$.

symmetrically distributed on the boundary of the cross section by the xy or xz (resp. yx or yz and zx or zy) plane, as depicted in Fig. 10. The quality values of all grasps in the six cases are plotted in Fig. 11, in which the maximum $\sigma = 0.1673$ is obtained at $\alpha = 0.9094$ in the zy case depicted in Fig. 10(c) and the corresponding grasp is very close to the one shown in Fig. 8(b).

The results in the case of four contacts are also displayed in Table V. The first force-closure grasp and the final grasp obtained within the time limit are shown in Fig. 9(a) and (b), respectively. Again, to verify the global optimality of the final grasp, we let four contacts move on the ellipsoid in two ways as indicated in Fig. 12(a) and (b), respectively, where we think the globally optimal grasp should exist. Fig. 13 describes the quality values of those grasps in the two cases. In Fig. 13(a), the maximum value of σ is 0.2399 and obtained at $\theta = 0.7250$ and $\phi = 1.3690$, while in Fig. 13(b) the maximum value of σ is 0.2202 and obtained at $\theta = 0.8055$ and $\phi = 1.3690$. Both give a grasp similar to the one shown in Fig. 9(b).

Example 6: We now apply Algorithm 4 to more complex objects, the heart and the seashell, as depicted in Figs. 14–17. Tables VI and VII show the results and key performance of the algorithm. On the heart, which is a symmetric object, it is relatively easier to attain a force-closure grasp with good quality

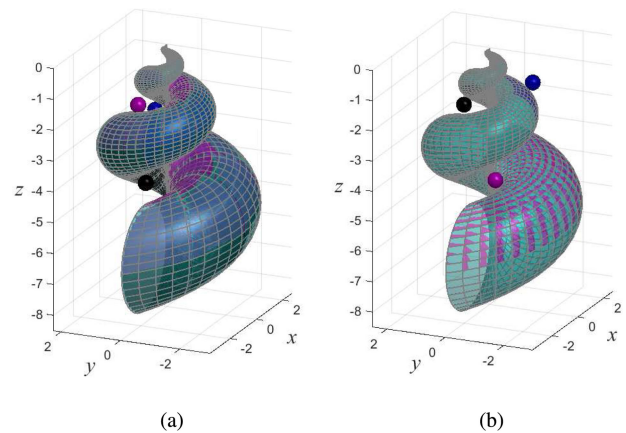


Fig. 16. Grasps with three contacts on the seashell yielded by Algorithm 4 at the (a) 1st iteration with $\sigma = 0.0245$ and (b) 66th iteration with $\sigma = 0.0670$.

as such a grasp has been found in the initial domain in both three- and four-contact cases, as indicated by the fourth column of Table VI. The final grasp accords with our intuition for being the optimal grasp, as shown especially in Fig. 15(b). By contrast, forming a good grasp on the irregular seashell is much more

TABLE VI
RESULTS OF ALGORITHM 4 ON THE HEART

i	Initial domains for (θ_i, ϕ_i)	t	N	Contacts (θ_i, ϕ_i)	σ	t	N	Contacts	σ
1	$[\pi/16, \pi] \times [0, \pi/2]$	1154.87	0	(1.6092, 0.7726)	0.1439	76025.6	84	(2.5430, 1.1372)	0.1817
2	$[\pi/16, \pi] \times [0, \pi]$			(1.6055, 1.9442)				(1.0408, 1.7943)	
3	$[\pi/16, \pi] \times [\pi, 2\pi]$			(1.5690, 4.7185)				(2.1414, 4.6416)	
1	$[\pi/16, \pi/2] \times [0, \pi/2]$	8510.53	0	(1.1104, 0.8611)	0.2640	141604	37	(1.1261, 1.5433)	0.3963
2	$[\pi/16, \pi/2] \times [\pi, 3\pi/2]$			(1.1351, 3.8467)				(1.1253, 4.6509)	
3	$[\pi/2, \pi] \times [\pi/2, \pi]$			(2.2548, 2.4754)				(2.3174, 3.1181)	
4	$[\pi/2, \pi] \times [3\pi/2, 2\pi]$			(2.2540, 5.6752)				(2.3133, 6.2594)	

t , N —CPU running time (unit: second) and number of iterations of the algorithm.

TABLE VII
RESULTS OF ALGORITHM 4 ON THE SEASHELL

i	Initial domains for (θ_i, ϕ_i)	t	N	Contacts (θ_i, ϕ_i)	σ	t	N	Contacts	σ
1	$[2\pi, 6\pi] \times [0, 2\pi]$	3909.32	1	(8.6009, 4.7045)	0.0245	83218.0	66	(10.137, 0.6318)	0.0670
2	$[2\pi, 6\pi] \times [0, \pi]$			(1.4286, 1.6078)				(17.833, 2.2038)	
3	$[2\pi, 6\pi] \times [\pi, 2\pi]$			(1.3508, 4.4185)				(7.5657, 5.1650)	
1	$[2\pi, 6\pi] \times [0, 2\pi]$	7262.01	8	(7.2083, 2.0026)	0.1149	47198.4	12	(9.0618, 2.4150)	0.1641
2	$[2\pi, 6\pi] \times [0, 2\pi]$			(7.6075, 4.8071)				(6.6175, 5.5518)	
3	$[2\pi, 6\pi] \times [0, \pi]$			(16.076, 1.3927)				(17.749, 1.6041)	
4	$[2\pi, 6\pi] \times [\pi, 2\pi]$			(16.122, 4.2519)				(15.493, 5.5964)	

t , N —CPU running time (unit: second) and number of iterations of the algorithm.

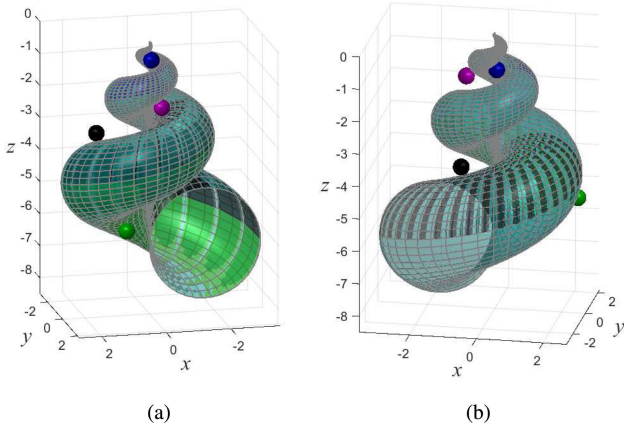


Fig. 17. Grasps with four contacts on the seashell yielded by Algorithm 4 at the (a) 8th iteration with $\sigma = 0.1149$ and (b) 12th iteration with $\sigma = 0.1641$.

V. CONCLUSION

In view of the necessity of computing the support function in many existing algorithms and its current limitation to simple convex sets, in this article we first presented a B&B algorithm to calculate the support function of complex sets that were described by functions with Lipschitz continuity and can have continuous nonconvex boundaries. Based on the Lipschitz continuity, we derived an upper bound on the support function of such a set and prove that the upper bound was decreasing and converges to the exact value of the support function through a B&B procedure. With this new general computational method for the support function, we, then, can 1) compute the bounding polytope of a complex set, 2) extend the existing algorithms to compute the minimum distance between the convex hulls of and derive a new algorithm to compute the true minimum separation distance between two complex sets, and finally 3) develop an algorithm for globally optimal grasps on objects with continuous surfaces.

difficult. The algorithm needs some iterations to produce the first force-closure grasp and the quality value σ of the final grasp is notably smaller.

From the abovementioned examples, we see that Algorithm 4 tends to yield the optimal grasp but its major limitation lies in the need of significant amount of computation time. A similar parallel implementation to Algorithm 3 as discussed in Section III-C can speed up Algorithm 4 by two to three times. Being intended for the optimal grasp planning in a highly nonlinear form, however, the algorithm takes hours to give a good result so far. The termination condition $\hat{X} \neq \emptyset$ or $\hat{\sigma} - \sigma^* > \epsilon$ was not reached in the reported examples and the algorithm was terminated because its running time exceeded the given limit. This leaves the risk that the computed grasp is not guaranteed to be the globally optimal grasp.

In terms of future work, first, we would like to explore possibilities to enhance the computational efficiency of the proposed algorithms. We may consider more appropriate upper/lower bounds and branching strategies such that a B&B procedure can more quickly narrow down the candidate domain and determine the optimal solution. Moreover, we can take advantage of parallel computing to speed up some operations in the algorithms, such as the computing of upper/lower bounds for subdomains. Second, we can extend the algorithms to other types of continuous functions, such as Hölder continuous functions, which possess a more general form of continuity than Lipschitz continuity. Third, the computation of the true minimum penetration distance between overlapping nonconvex sets remains as an open problem, which deserves further exploration. In addition, we will explore other situations that need the computing of

support function and can benefit from this new computational technique.

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 J. Robot. Autom.*, vol. 4, no. 2, pp. 193–203, Apr. 1988.
- [2] E. G. Gilbert and C. P. Foo, "Computing the distance between general convex objects in three-dimensional space," *IEEE Trans. Robot. Autom.*, vol. 6, no. 1, pp. 53–61, Feb. 1990.
- [3] S. A. Cameron, "Enhancing GJK: Computing minimum and penetration distances between convex polyhedra," in *Proc. IEEE Int. Conf. Robot. Autom.*, Albuquerque, NM, USA, Apr. 1997, pp. 3112–3117.
- [4] S. A. Cameron, "A comparison of two fast algorithms for computing the distance between convex polyhedra," *IEEE Trans. Robot. Autom.*, vol. 13, no. 6, pp. 915–920, Dec. 1997.
- [5] C. J. Ong and E. G. Gilbert, "The Gilbert-Johnson-Keerthi distance algorithm: A fast version for incremental motions," in *Proc. IEEE Int. Conf. Robot. Autom.*, Albuquerque, NM, USA, Apr. 1997, pp. 1183–1189.
- [6] C. J. Ong and E. G. Gilbert, "Fast versions of the Gilbert-Johnson-Keerthi distance algorithm: Additional results and comparisons," *IEEE Trans. Robot. Autom.*, vol. 17, no. 4, pp. 531–539, Aug. 2001.
- [7] K. Sridharan, H. E. Stephanou, K. C. Craig, and S. S. Keerthi, "Distance measures on intersecting objects and their applications," *Inf. Process. Lett.*, vol. 51, no. 4, pp. 181–188, 1994.
- [8] Y. Zheng, "An efficient algorithm for a grasp quality measure," *IEEE Trans. Robot.*, vol. 29, no. 2, pp. 579–585, Apr. 2013.
- [9] X.-Y. Zhu, H. Ding, and Y. L. Xiong, "Pseudo minimum translational distance between convex polyhedra (i): Definition and properties," *Sci. China, Ser. E*, vol. 44, no. 2, pp. 216–224, 2001.
- [10] X.-Y. Zhu and J. Wang, "Synthesis of force-closure grasps on 3-D objects based on the Q distance," *IEEE Trans. Robot. Autom.*, vol. 19, no. 4, pp. 669–679, Aug. 2003.
- [11] X.-Y. Zhu, H. Ding, and S. K. Tso, "A pseudodistance function and its applications," *IEEE Trans. Robot. Autom.*, vol. 20, no. 2, pp. 344–352, Apr. 2004.
- [12] Y. Zheng and K. Yamane, "Generalized distance between compact convex sets: Algorithms and applications," *IEEE Trans. Robot.*, vol. 31, no. 4, pp. 988–1003, Aug. 2015.
- [13] Y. Zheng and K. Yamane, "Ray-shooting algorithms for robotics," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 4, pp. 862–874, Oct. 2013.
- [14] 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, Dec. 2009.
- [15] Y. Zheng and C.-M. Chew, "Fast equilibrium test and force distribution for multi-contact robotic systems," *ASME J. Mech. Robot.*, vol. 2, no. 2, 2010, Art. no. 021001.
- [16] Y. Zheng, M. C. Lin, D. Manocha, A. H. Adiwahono, and C.-M. Chew, "A walking pattern generator for biped robots on uneven terrains," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Taipei, Taiwan, Oct. 2010, pp. 4483–4488.
- [17] Y. Zheng and K. Yamane, "Human motion tracking control with strict contact force constraints for floating-base humanoid robots," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Atlanta, GA, USA, 2013, pp. 34–41.
- [18] Y. Zheng and K. Yamane, "Adapting human motions to humanoid robots through time warping based on a general motion feasibility index," in *Proc. IEEE Int. Conf. Robot. Autom.*, Seattle, WA, USA, 2015, pp. 6281–6288.
- [19] C. Ferrari and J. F. Canny, "Planning optimal grasps," in *Proc. IEEE Int. Conf. Robot. Autom.*, Nice, France, May 1992, pp. 2290–2295.
- [20] M. Teichmann, "A grasp metric invariant under rigid motions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1996, pp. 2143–2148.
- [21] Y. Zheng and W.-H. Qian, "Improving grasp quality evaluation," *Robot. Auton. Syst.*, vol. 57, no. 6/7, pp. 665–673, 2009.
- [22] N. S. Pollard, "Parallel methods for synthesizing whole-hand grasps from generalized prototypes," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1994.
- [23] C. Borst, M. Fischer, and G. Hirzinger, "Grasp planning: How to choose a suitable task wrench space," in *Proc. IEEE Int. Conf. Robot. Autom.*, New Orleans, LA, USA, Apr. 2004, pp. 319–325.
- [24] R. Haschke, J. J. Steil, I. Steuwer, and H. Ritter, "Task-oriented quality measures for dextrous grasping," in *Proc. IEEE Int. Conf. Comp. Intell. Robot. Autom.*, Espoo, Finland, Jun. 2005, pp. 689–694.
- [25] M. Strandberg and B. Wahlberg, "A method for grasp evaluation based on disturbance force rejection," *IEEE Trans. Robot.*, vol. 22, no. 3, pp. 461–469, Jun. 2006.
- [26] Y. Lin and Y. Sun, "Grasp planning to maximize task coverage," *Int. J. Robot. Res.*, vol. 34, no. 9, pp. 1195–1210, 2015.
- [27] Y. Zheng and K. Yamane, "Evaluation of grasp force efficiency considering hand configuration and using novel generalized penetration distance algorithm," in *Proc. IEEE Int. Conf. Robot. Autom.*, Karlsruhe, Germany, 2013, pp. 1580–1587.
- [28] K. Hang, F. T. Pokorny, and D. Kragic, "Friction coefficients and grasp synthesis," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Tokyo, Japan, Nov. 2013, pp. 3520–3526.
- [29] A. T. Miller and P. K. Allen, "Examples of 3D grasp quality computation," in *Proc. IEEE Int. Conf. Robot. Autom.*, Detroit, MI, USA, 1999, pp. 1240–1246.
- [30] C. Borst, M. Fischer, and G. Hirzinger, "A fast and robust grasp planner for arbitrary 3D objects," in *Proc. IEEE Int. Conf. Mechatron. Autom.*, Detroit, MI, USA, May 1999, pp. 1890–1896.
- [31] Y. Zheng, "Computing the best grasp in a discrete point set with wrench-oriented grasp quality measures," *Auton. Robots*, vol. 43, no. 4, pp. 1041–1062, 2019.
- [32] A. T. Miller and P. K. Allen, "GraspIt! a versatile simulator for robotic grasping," *IEEE Robot. Autom. Mag.*, vol. 11, no. 4, pp. 110–122, 2004.
- [33] G. F. Liu, J. J. Xu, and Z. X. Li, "On quality functions for grasp synthesis, fixture planning, and coordinated manipulation," *IEEE Trans. Autom. Sci. Eng.*, vol. 1, no. 2, pp. 146–162, Oct. 2004.
- [34] X.-Y. Zhu and H. Ding, "Computation of force-closure grasps: An iterative algorithm," *IEEE Trans. Robot.*, vol. 22, no. 1, pp. 172–179, Feb. 2006.
- [35] Z. X. Xue, J. M. Zoellner, and R. Dillmann, "Automatic optimal grasp planning based on found contact points," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatron.*, Xi'an, China, Jul. 2008, pp. 1053–1058.
- [36] T. Watanabe and T. Yoshikawa, "Grasping optimization using a required external force set," *IEEE Trans. Autom. Sci. Eng.*, vol. 4, no. 1, pp. 52–66, Jan. 2007.
- [37] M. A. Roa and R. Suárez, "Computation of independent contact regions for grasping 3-D objects," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 839–850, Aug. 2009.
- [38] H. K. Dai, A. Majumdar, and R. Tedrake, "Synthesis and optimization of force closure grasps via sequential semidefinite programming," in *Proc. Int. Symp. Robot. Res.*, 2015, pp. 285–305.
- [39] A. Sintov, R. J. Menassa, and A. Shapiro, "A gripper design algorithm for grasping a set of parts in manufacturing lines," *Mechanism Mach. Theory*, vol. 105, pp. 1–30, 2016.
- [40] Y. Zheng, "Computing the globally optimal frictionless fixture in a discrete point set," *IEEE Trans. Robot.*, vol. 32, no. 4, pp. 1026–1032, Aug. 2016.
- [41] K. Hang *et al.*, "Hierarchical fingertip space: A unified framework for grasp planning and in-hand grasp adaptation," *IEEE Trans. Robot.*, vol. 32, no. 4, pp. 960–972, Aug. 2016.
- [42] K. Hang, J. A. Stork, N. S. Pollard, and D. Kragic, "A framework for optimal grasp contact planning," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 704–711, Apr. 2017.
- [43] J. A. Hausteine, K. Hang, and D. Kragic, "Integrating motion and hierarchical fingertip grasp planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, Singapore, May/June 2017, pp. 3439–3446.
- [44] Y. Zheng, "Computing bounding polytopes of a compact set and related problems in n -dimensional space," *Comput.-Aided Des.*, vol. 109, pp. 22–32, 2019.
- [45] S. R. Lay, *Convex Sets and their Applications*. New York, NY, USA: Wiley, 1982.
- [46] S. A. Cameron and R. K. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Proc. IEEE Int. Conf. Robot. Autom.*, San Francisco, CA, USA, Apr. 1986, pp. 591–596.
- [47] R. M. Murray, Z. X. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL, USA: CRC Press, 1994.
- [48] B. Mishra, J. T. Schwarz, and M. Sharir, "On the existence and synthesis of multifingered positive grips," *Algorithmica*, vol. 2, no. 4, pp. 541–558, 1987.
- [49] Y. Zheng and W.-H. Qian, "New advances in automatic selection of eligible surface elements for grasping and fixturing," *Robotica*, vol. 28, no. 3, pp. 341–348, 2010.



Yu Zheng (Senior Member, IEEE) received the B.E. degrees in both mechanical engineering and computer science and the Ph.D. degree in mechatronics from Shanghai Jiao Tong University, Shanghai, China, in 2001 and 2007, respectively, and the M.S. and Ph.D. degrees in computer science from the University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, in 2011 and 2014, respectively.

Between 2007 and 2009, he was a Postdoctoral Research Fellow with the Department of Mechanical Engineering, National University of Singapore. He worked as a Lab Associate, a Research Associate, and finally a Postdoctoral Researcher with Disney Research Pittsburgh between 2010 and 2014. From 2014 to 2018, he was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Michigan-Dearborn. He joined Tencent Robotics X in September 2018 and currently is a Principal Research Scientist. His research interests include multicontact/multibody robotic systems, robotic grasping and manipulation, legged robots, and various algorithms for robotics.

Dr. Zheng is an Associate Editor for the IEEE ROBOTICS AND AUTOMATION LETTERS.



Kaiyu Hang (Member, IEEE) received the B.S. degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2010, and the M.Sc. degree in communication systems and the Ph.D. degree in computer science, specialized in robotics and computer vision, from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2012 and 2016, respectively.

He is a Postdoctoral Associate with the GRAB Lab, Yale University, CT, USA. His research interests include representations and optimization for robotic manipulation, motion planning, adaptive grasping and in-hand manipulation, underactuated robotic hands, dual-arm manipulation, and mobile manipulation.