A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space

ELMER G. GILBERT, FELLOW, IEEE, DANIEL W. JOHNSON, AND S. SATHIYA KEERTHI

Abstract—**An efficient and reliable algorithm for computing the Euclidean distance between a pair of convex sets in R^m is described. Extensive numerical experience with a broad family of polytopes in R^3 shows that the computational cost is approximately linear in the total number of vertices specifying the two polytopes. The algorithm has special features which makes its application in a variety of robotics problems attractive. These are discussed and an example of collision detection is given.**

I. INTRODUCTION

IN ROBOTICS and other fields, such as computer-aided design and computer graphics, it is important to know if two objects, characterized by mathematical models in three-dimensional space, intersect or are in near proximity. The most natural measure of proximity is the Euclidean distance between the objects, i.e., the length of a shortest line segment joining the two objects. In this paper we present an approach for computing this distance. It applies to a complex family of shape models and is particularly convenient when the objects are subject to changes in position and orientation. Because the approach is highly efficient, we expect that it will become a useful tool in solving collision detection problems and path finding problems (see, e.g., [3], [6], [8], [10] and [4], [5], [12], [21], [28]). Our own applications have been to optimal path planning in the presence of obstacles [15], [17], [18].

The key element of the approach is an algorithm for computing the distance between convex sets in m-dimensional space. The algorithm is designed to be particularly efficient when $m = 3$ and the convex sets are polytopes defined by their vertices. While it is iterative, the algorithm terminates in a finite number of steps when the sets are polytopes. Numerical experience with such problems is most encouraging. For a wide variety of examples the computational times are nearly linear in the total number of vertices, $M = M_1 + M_2$, required to specify the two polytopes. Moreover, the coefficient of linear growth is quite small.

Manuscript received October 24, 1986; revised July 27, 1987. This research was partially supported by the Center for Research in Integrating Manufacturing at the University of Michigan.

E. G. Gilbert is with the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109.

D. W. Johnson was with the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109. He is now with Martin Marietta Aero and Naval Systems, Baltimore, MD 21220.

S. S. Keerthi was with the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109. He is now with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India.

IEEE Log Number 8718909.

There is an extensive literature concerning the polytope distance problem, so we will be content to give a brief review of some representative papers. The problem is in the field of computational geometry [20]. Consequently, many algorithms are specifically designed to achieve bounds on the form of the asymptotic computational time. For two-dimensional problems, Schwartz [27] gives an $O(\log^2 M)$ algorithm, and more recently, $O(\log M)$ algorithms have been exhibited [9], [13]. The three-dimensional problem has been considered in [24] and [11] with respective times of $O(M \log M)$ and $O(M)$. Because of their complexity and special emphasis on asymptotic performance, it is not clear that the algorithms in the preceding papers are efficient for practical problems where M is large, but not exceedingly large. Unfortunately, very few, if any, computational experiments have been done. Other schemes have also been described: Red [25] presents a program which uses a projection/combinatoric approach for polyhedra with facial representations, the authors of [5] and [7] are concerned with "negative" distances for intersecting objects (more about this later), Meyer [23] considers boxes, Lumelsky [22] considers line segments. It is also possible to convert the distance problem to a quadratic programming problem and apply any of the well-developed computer programs which are applicable.

Unlike the procedures of the previous paragraph, our algorithm has its origins in mathematical programming and treats directly the specification of the convex sets in terms of their support properties (for polytopes these properties are obtained easily from their vertices). The algorithm is in the same family as the algorithms described originally by Barr, Gilbert, and Wolfe [1], [2], [29] and may be viewed as a descent procedure which works on the distance between elementary polytopes contained in the convex sets. We have devised a special subalgorithm for evaluating the distance between the elementary polytopes. It contributes significantly to the overall efficiency of the algorithm. An important feature of the algorithm is its very general initialization features. When used to detect collision along a continuous path in the configuration space which describes the position and orientation of the objects, they allow significant reductions in the total computation time. The algorithm has good numerical properties and a thoroughly tested Fortran subroutine is available.[1] An early version of the algorithm due to Johnson was used in

[1] Contact E. G. Gilbert.

0882-4967/88/0400-0193$01.00

the optimal path planning computations described in [18]. This paper is similar to [16].

The plan of the paper is as follows. In Section II we consider the distance between a pair of objects taken from a general family of nonconvex objects and suggest how our algorithm may be applied to its computation. We also review what happens to the distance when the position and orientation of the objects is specified by a set of configuration variables. Section III reviews some results from convex analysis, and shows how the problem of computing the distance between convex sets can be reduced to the basic problem of finding the distance between the origin and a single convex set. Section IV describes the theoretical algorithm for solving the basic problem; Section V presents the distance subalgorithm for elementary polytopes; and Section VI introduces modifications in the overall algorithm to account for the effects of numerical errors. Many numerical experiments have been carried out; these are reported in Section VII. In Section VIII, the algorithm is applied to a collision detection problem due to Canny [8]. A conclusion summarizes the key contributions and indicates some extensions. An initial overview of the paper can be had by reading Sections III, IV, VII, and VIII.

II. Object Representations and Distance

Given two objects A and B in three space, it is convenient to represent them by compact sets: $K_A, K_B \subset R^3$. In particular, the points in K_A and K_B describe, respectively, the space occupied by the objects A and B. For $z = (z^1, z^2, z^3) \in R^3$, let $|z|$ denote the Euclidean length $\sqrt{(z^1)^2 + (z^2)^2 + (z^3)^2}$. The distance between the objects A and B is defined by the closest points in K_A and K_B:

$$d(K_A, K_B) = \min \{|x-y| : x \in K_A, y \in K_B\}. \quad (1)$$

While computational considerations may suggest the use of other metrics in (1), the Euclidean distance $|x - y|$ is the most natural. It conforms with the "physical" notion of distance and makes d invariant with respect to different choices for the origin and orientation of the coordinate system. Because K_A and K_B are compact, the minimum in (1) exists and d is defined.

It is only for simple object pairs such as a sphere and a line segment that formulas for d may be given [19]. Since our algorithm allows d to be computed for convex polytopes, a much wider class of object pairs is permitted. In fact, it is possible to treat conveniently a rich family of nonconvex shapes: objects which are the union of convex polytopes and their spherical extensions.

Suppose A and B are each the union of several objects. Then $d(K_A, K_B)$ may be computed in terms of the distances between the constituent objects. Specifically, let

$$K_A = \bigcup_{i \in I_A} K_i \qquad K_B = \bigcup_{j \in I_B} K_j \quad (2)$$

where the K_i, $i \in I = \{1, \cdots, N\}$ are compact sets in R^3, and I_A and I_B are disjoint index sets in I. Then

$$d(K_A, K_B) = \min \{d_{ij} : i \in I_A, j \in I_B\} \quad (3)$$

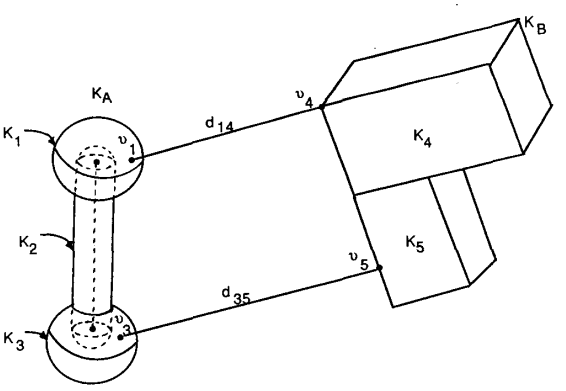


Fig. 1. An example of object representation. The near point pair v_1, v_4 for K_1 and K_4 is the near point pair for K_A and K_B; thus $d(K_A, K_B) = d_{14}$.

where

$$d_{ij} = \min \{|x-y| : x \in K_i, y \in K_j\} = d(K_i, K_j). \quad (4)$$

See the example in Fig. 1.

If the distance between objects A and B is known, so is the distance between their spherical extensions [18]. The r-spherical extension of $K \subset R^3$ is defined by

$$K^r = \{x : |x-y| \le r, y \in K\}, \qquad r \ge 0. \quad (5)$$

It is easy to verify that

$$d(K_A^{r_A}, K_B^{r_B}) = (d(K_A, K_B) - r_A - r_B)^+ \quad (6)$$

where $(\alpha)^+ = \alpha$, $\alpha > 0$, and $(\alpha)^+ = 0$, $\alpha \le 0$. More generally

$$K_A = \bigcup_{i \in I_A} K_i^{r_i} \qquad K_B = \bigcup_{j \in I_B} K_j^{r_j} \quad (7)$$

implies

$$d(K_A, K_B) = \min \{(d_{ij} - r_i - r_j)^+ : i \in I_A, j \in I_B\} \quad (8)$$

where the d_{ij} are given by (4). If the K_i are convex polytopes, (7) is the promised family of nonconvex shapes. Moreover, the polytope algorithm may be applied to the d_{ij}.

Spherical extensions are valuable for several reasons. They may be used to cover an object with a shell of safety: if $x \notin K^r$, it is clear that the distance between x and K exceeds r. More importantly, they may lead to economical representations of complex objects. Object A in Fig. 1 is a simple example. It is the union of two spheres (extensions of points) and a circular cylinder with spherical end caps (an extension of a line segment). Another example is a solid rectangular plate of thickness $2r$ with round edges; it is modeled by an r-spherical extension of a planar polytope with four vertices. Similarly, more general wire-frame objects can be given rounded representations.

Often the position and orientation of the objects K_i in (2) are specified by a configuration vector $q \in R^n$. For instance, if A and B are interacting manipulators whose links and payloads are the K_i, the components of q are the joint variables for the two manipulators. Our approach to object representation

handles such situations conveniently. To be more specific, K_i is obtained by translating and rotating a compact set C_i:

$$K_i(q)=\{T_i(q)w+p_i(q) : w \in C_i\}. \tag{9}$$

Here $p_i(q) \in R^3$ is the translation, $T_i(q) \in R^{3\times 3}$ is the (orthogonal) rotation matrix, and C_i describes K_i in its reference position. In practice, there are various ways of obtaining $p_i(q)$ and $T_i(q)$. For example, they may be extracted from the usual 4×4 homogeneous transformation matrix. If C_i is a polytope with vertices $w_{ij} \in R^3, j = 1, \cdots, M_i$, the corresponding vertices of $K_i(q)$ are given by $z_{ij} = T_i(q)w_{ij} + p_i(q)$, a simple computation. It follows from the orthogonality of $T_i(q)$ that the reference object for a spherical extension is independent of q; i.e.,

$$K_i^{r_i}(q)=\{T_i(q)w+p_i(q) : w \in C_i^{r_i}\}. \tag{10}$$

In [15] the dependence of d_{ij} on q has been examined in detail. Suppose the elements of $T_k(q)$ and $p_k(q)$, $k = i, j$ are continuously differentiable in q. Then it follows [15] that $d_{ij}(q)$ is Lipschitz continuous and has a gradient (Frechet derivative) almost everywhere. It is easy to give examples (K_i and K_j may be convex) where at a specific q, $d_{ij}(q)$ does not have a gradient. If $d_{ij}(\bar{q}) > 0$ and the nearest points $v_k \in K_k(\bar{q})$, $k = i, j$, are uniquely determined, $d_{ij}(q)$ does have a gradient at $\bar{q}$. In particular [15], we have $\nabla_q d_{ij}(\bar{q}) = \nabla_q |T_i(q)w_i^* + p_i(q) - T_j(q)w_j^* - p_j(q)|$, $q = \bar{q}$, where $w_k^* = T_k^T(\bar{q})(v_k - p_k(\bar{q}))$, $k = i, j$, and the super T denotes matrix transpose. Once v_i and v_j have been found (by the distance algorithm) the evaluation of this expression is relatively easy to carry out. We have used $\nabla_q d_{ij}(q)$ in an optimum path planning procedure [15], [17], [18], but its general value in anticipating collision is evident.

III. Preliminaries

In this section we introduce some notations and basic results which are required for the algorithm. Everything is stated in R^m, because the results are not restricted to $m = 3$. We use $x \cdot y$ for the inner product of $x, y \in R^m$ and $|x|^2 = x \cdot x$. For $X_1, X_2 \subset R^m$, $X_1 \pm X_2 = \{x_1 \pm x_2 : x_1 \in X_1, x_2 \in X_2\}$ denotes the Minkowski set sum or difference. Throughout the section $X \subset R^m$ is compact and $Y \subset R^m$ is a finite set of points $\{y_1, \cdots, y_v\}$.

The affine and convex hulls of X are given by

$$\text{aff } X=\left\{\sum_{i=1}^{l} \lambda^i x_i : x_i \in X, \lambda^1+\cdots+\lambda^l=1\right\} \tag{11}$$

$$\text{co } X=\left\{\sum_{i=1}^{l} \lambda^i x_i : x_i \in X, \lambda^i \geq 0, \lambda^1+\cdots+\lambda^l=1\right\}. \tag{12}$$

It is easily confirmed that aff X is the translate of a linear space. For example, aff $Y = Y + \{y_1\}$, where Y is the linear span of $\{y_2 - y_1, \cdots, y_v - y_1\}$. Y is *affinely independent* if dim aff Y = dim $Y = v - 1$. If Y is not affinely independent, it is always possible to pick an affinely independent set $\bar{Y} \subset Y$ such that aff $\bar{Y}$ = aff Y. The set co Y is a convex polytope whose vertices are contained in Y. Suppose X belongs to the translate of a linear space X. The Caratheodory theorem [26] states that there is no loss of generality if in (12) l is restricted so that $l \leq \dim X + 1$.

A nearest point in X to the origin $v(X)$ is determined by

$$v(X) \in X, \ |v(X)|=\min\{|x| : x \in X\}. \tag{13}$$

In general, there may be several points $v(X)$ satisfying (13). If X is convex, it is well known that $v(X)$ is uniquely determined [26].

By (12) it is obvious that the unique point $v(\text{co } X)$ has a representation of the form

$$v(\text{co } X)=\sum_{i=1}^{l} \lambda^i x_i, \ x_i \in X, \ \lambda^i>0, \ \lambda^1+\cdots+\lambda^l=1. \tag{14}$$

Often, the λ^i, the x^i, and l are not unique. However, it is always possible to obtain a representation with: a) $l \leq m + 1$ when $v(\text{co } X) = 0$ and $l \leq m$ when $v(\text{co } X) \neq 0$, b) $\{x_1, \cdots, x_l\}$ affinely independent.

The first part of a) is obvious from the Caratheodory theorem. In the second part of a), $v(\text{co } X)$ is in the boundary of co X. Thus $v(\text{co } X) \in H \cap \text{co } X$ where H is a support hyperplane of co X [26]. Since dim $H = m - 1$ the Caratheodory theorem implies $l \leq m$. If (14) holds and $\{x_1, \cdots, x_l\}$ is affinely dependent it is always possible to eliminate some of the x_i and obtain an affinely independent subset; the procedure for doing this is the same as the one used in the usual proof of the Caratheodory theorem.

The support function of X, $h_X: R^m \to R$, is defined by

$$h_X(\eta)=\max\{x \cdot \eta : x \in X\}. \tag{15}$$

We use $s_X(\eta)$ to denote any solution of (15). Specifically, $s_X(\eta)$ satisfies

$$h_X(\eta)=s_X(\eta) \cdot \eta \qquad s_X(\eta) \in X. \tag{16}$$

Since it is easy to prove that $h_X = h_{\text{co}X}$ and $s_X = s_{\text{co}X}$, $h_{\text{co}Y}(\eta)$ and $s_{\text{co}Y}(\eta)$ can be determined by a simple enumeration of inner products:

$$h_{\text{co}Y}(\eta)=h_Y(\eta)=\max\{y_i \cdot \eta : i=1, \cdots, v\}$$
$$s_{\text{co}Y}(\eta)=s_Y(\eta)=y_j, \qquad y_j \cdot \eta=h_Y(\eta). \tag{17}$$

Thus (17) provides a simple procedure for evaluating the support properties of the polytope co Y.

Fig. 2 displays some of the above notation and results. Note $v(\text{co } Y)$ has the representation (14) with $X = Y$, $l = 2$, $x_1 = y_5$, and $x_2 = y_6$.

Consider now the problem of finding d_{ij} in (4) when K_i and K_j are compact and convex. For convenience, assume hereafter that $i = 1$ and $j = 2$. Clearly

$$d_{12}=\min\{|z| : z \in K\}=|v(K)|, \qquad K=K_1-K_2. \tag{18}$$

This seems like a simpler problem than (4), but it is only superficially so. While the set difference $K = K_1 - K_2$ is convex, it is generally more complex than either K_1 or K_2.

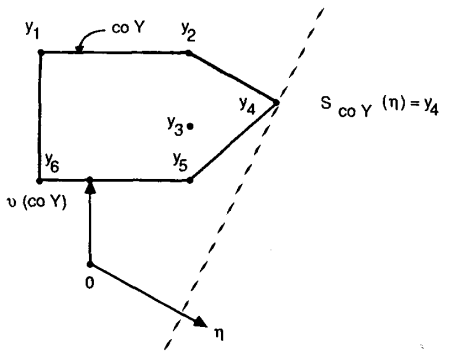


Fig. 2. An example in R^2 showing notation from Section III for $Y = \{y_1, \cdots, y_6\}$.

This is apparent when K_1 and K_2 are the polytopes

$$K_k = \text{co } Z_k, \qquad Z_k = \{z_{kl} : l = 1, \cdots, M_k\}. \tag{19}$$

Then it is easy to show that K is the polytope

$$K = \text{co } Z,$$

$$Z = \{z_{1i} - z_{2j} : i = 1, \cdots, M_1, j = 1, \cdots, M_2\}. \tag{20}$$

Since Z has M_1M_2 elements, K is much more complex than either K_1 or K_2. This complexity appears in [21] and in the work of others who have used the set difference.

Our algorithm determines d_{12} by solving the problem (18). Its steps require only the evaluation of h_K and s_K. Despite the complexity of K, these functions are easy to compute from K_1 and K_2:

$$h_K(\eta) = h_{K_1}(\eta) + h_{K_2}(-\eta), \; s_K(\eta) = s_{K_1}(\eta) - s_{K_2}(-\eta). \tag{21}$$

For the polytope case, (17) and (21) show that the computational effort associated with h_K and s_K is proportional to $M_1 + M_2$, not M_1M_2 as might first be expected.

In our subsequent description of the algorithm it simplifies matters to base everything on K and avoid specific reference to K_1 and K_2. Let us indicate how this can be done. When the algorithm stops, it produces the following data: $l \le m + 1$, $\lambda^i > 0$, $y_i \in K$, $i = 1, \cdots, l$,

$$\nu(K) = \sum_{i=1}^{l} \lambda^i y_i \qquad d_{12} = |\nu(K)|. \tag{22}$$

As input the algorithm uses initial points from K_1 and K_2 and evaluations of h_K and s_K by (21). Hence, $y_i = y_{1i} - y_{2i}$, $y_{1i} \in K_1$, $y_{2i} \in K_2$, and (22) yields

$$\nu(K) = \nu_1(K_1, K_2) - \nu_2(K_1, K_2) \tag{23}$$

where

$$\nu_1(K_1, K_2) = \sum_{i=1}^{l} \lambda^i y_{1i}$$

$$\nu_2(K_1, K_2) = \sum_{i=1}^{l} \lambda^i y_{2i} \tag{24}$$

are, respectively, near points in K_1 and K_2. If $\nu_1(K_1, K_2)$ and $\nu_2(K_1, K_2)$ are in "parallel sides" of K_1 and K_2, they may not be unique (even though $\nu(K)$ is unique). In case of nonuniqueness, the pair of near points provided by the algorithm has no special properties.

IV. The Theoretical Algorithm

We now present the algorithm for determining $\nu(K)$ when $K \subset R^m$ is compact and convex. If K is a polytope, it is shown that this algorithm terminates after a finite number of steps.

The basic idea is due to Barr and Gilbert [1], [2]: generate a sequence of polytopes co V_k contained in K such that their near points $\nu(\text{co } V_k)$ converge to $\nu(K)$. It is necessary to compute the $\nu(\text{co } V_k)$, but this takes little time because the polytopes have at most $m + 1$ vertices. See the next section.

To state the algorithm, we first introduce criteria for descent and optimality and establish a bound on approximation error.

Theorem 1: Let $K \subset R^m$ be compact and convex and define $g_K: R^m \to R$ by

$$g_K(x) = |x|^2 + h_K(-x). \tag{25}$$

Suppose $x \in K$. Then 1) if $g_K(x) > 0$ there is a point z in the line segment co $\{x, s_K(-x)\}$ satisfying $|z| < |x|$; 2) $x = \nu(K)$ if and only if $g_K(x) = 0$; 3) $|x - \nu(K)|^2 \le g_K(x)$.

For completeness, a proof of the theorem is given in Appendix I; its details have appeared in similar contexts before [14], [29].

Distance Algorithm: Given a compact convex set $K \subset R^m$ and initial points $y_1, \cdots, y_v \in K$, $1 \le v \le m + 1$, perform the following steps:

1) set $V_0 = \{y_1, \cdots, y_v\}$ and $k = 0$;
2) determine $\nu_k = \nu(\text{co } V_k)$;
3) if $g_K(\nu_k) = 0$, set $\nu(K) = \nu_k$ and stop;
4) set $V_{k+1} = \hat{V}_k \cup \{s_K(-\nu_k)\}$, where $\hat{V}_k \subset V_k$ has m elements or less and satisfies $\nu_k \in \text{co } \hat{V}_k$, increment k, and proceed to step 2).

Clearly, $V_k \subset K$ and $\nu_k \in K$ for $k \ge 0$. If the algorithm does not stop in step 3), then $g_K(\nu_k) > 0$ and Theorem 1 implies $\nu_k \ne 0$. Hence, the existence of $\hat{V}_k$ in step 4) is guaranteed; see result a) following (14). Furthermore, descent in the next iteration is guaranteed since result 1) of Theorem 1 implies

$$|\nu_{k+1}| = |\nu(\text{co } V_{k+1})| \le |\nu(\text{co } \{\nu_k, s_K(-\nu_k)\})| < |\nu_k|. \tag{26}$$

Fig. 3 illustrates the steps of the algorithm when K is a polytope co $\{z_1, \cdots, z_5\}$ in R^2. For $V_0 = \{z_1, z_2, z_3\}$ it can be seen that

$$\hat{V}_0 = \{z_2, z_3\}, \quad V_1 = \hat{V}_0 \cup \{z_4\}, \quad \hat{V}_1 = \{z_3, z_4\}$$

$$V_2 = \hat{V}_1 \cup \{z_5\}, \quad \nu_2 = \nu(K) \in \text{co } \{z_4, z_5\}. \tag{27}$$

When V_0 is the single point $\{z_2\}$ it may be verified that

$$\hat{V}_0 = \{z_2\}, \quad V_1 = \hat{V}_0 \cup \{z_5\}, \quad \hat{V}_1 = \{z_2, z_5\}$$

$$V_2 = \hat{V}_1 \cup \{z_4\}, \quad \nu_2 = \nu(K). \tag{28}$$

In general, the algorithm generates an infinite sequence

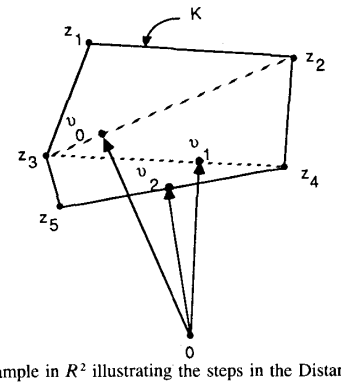


Fig. 3. An example in R^2 illustrating the steps in the Distance Algorithm.

$\{v_k\}$ which converges to $v(K)$. The proof follows from the same arguments used in the convergence proof for the method of Barr and Gilbert (see [1], [14]). The convergence proof is simpler when K is a polytope because $v(K)$ is obtained after a finite number of steps.

Theorem 2: Suppose $Z \subset R^m$ is a finite set of points and K is the polytope co Z. Assume $s_K(\eta) \in Z$ for all $\eta \in R^m$. Then the Distance Algorithm generates $v(K)$ in a finite number of steps.

Proof: Clearly, $K = \text{co}\,(Z \cup V_0)$ and $Z \cup V_0 \subset R^m$ is finite. Assume $v(K)$ is not generated in N steps where N is the number of nonempty subsets of $Z \cup V_0$. In this case, $g_K(v_k) > 0$ and $|v_{k+1}| < |v_k|$ for all $0 \le k < N$. Since near points are unique $V_k \ne V_l$ for any $0 \le l < k \le N$, and since $V_k \subset Z \cup V_0$ for all $0 \le k \le N$, every subset of $Z \cup V_0$ must have entered the algorithm. However, $v(K) \in \text{co}\,\bar{V}$ for some $\bar{V} \subset Z \cup V_0$ (see result a) following (14)). Therefore, $v(K) = v(\text{co}\,V_k)$ for some $k < N$, which is a contradiction.

The requirement that $s_K(\eta) \in Z$ is easy to obtain even when K is the set difference of two polytopes. This is clear from (17) and (21) and the associated discussion.

If $V_0 \subset Z$, it follows from the steps of the algorithm that $V_k \subset Z$ for all $k > 0$. Thus when the algorithm terminates, $v(K)$ has a representation of the form (22), where the $y_i \in Z$. This observation is useful for initializing the algorithm in continuum problems. See Sections VI and VIII for more details concerning this and the choice of the initial points.

V. The Distance Subalgorithm

In each iteration of the Distance Algorithm step 2) requires the determination of $v(\text{co}\ Y)$, $Y = \{y_1, \cdots, y_v\} \subset R^m$, where $Y = V_k$. In this section we describe a procedure originated by Johnson [17] for doing this. It is particularly efficient when v is small and yields a representation of the form

$$v\ (\text{co}\ Y) = \sum_{i \in I_s} \lambda^i y_i,$$

$$\sum_{i \in I_s} \lambda^i = 1,\ \lambda^i > 0,\ i \in I_s \subset \{1, \cdots, v\}$$

$$Y_s = \{y_i : i \in I_s\} \text{ is affinely independent} \qquad (29)$$

where s indicates a particular member of the family of all nonempty subsets of Y. The existence of such a representation is a consequence of result b) following (14). Note that Y_s becomes $\hat{V}_k$ in step 4) of the Distance Algorithm. Because it is affinely independent, $\hat{V}_k$ has a minimal number of elements. This simplifies V_{k+1} and makes the computations in the next iteration of the Distance Algorithm easier.

Since v is small, it is effective to take a combinatoric approach where all

$$\sigma = \sum_{k=1}^{v} [v!/k!(v-k)!]$$

subsets of Y are successively tested until a representation of the form (29) is found. Geometrically, this involves checking the open subsets of the polytope co Y to see if they contain $v(\text{co}\ Y)$. If $v = 4$, there are at most $\sigma = 15$ such subsets to examine. For example, if $v = 4$, $m = 3$, and Y is affinely independent, co Y is a three-dimensional simplex in R^3 and the $\sigma = 15$ open subsets are the 4 vertices, 6 open edges, 4 open faces, and interior of the simplex.

We begin by stating a theorem which characterizes the representation (29). Let I_s' be the complement of I_s in I and Y_s, $s = 1, \cdots, \sigma$, be an ordering of the subsets of Y. Define real numbers $\Delta_i(Y_s)$, $i \in I_s$, and $\Delta(Y_s)$ by

$$\Delta_i(\{y_i\}) = 1,\ i \in I$$

$$\Delta_j(Y_s \cup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s)(y_i \cdot y_k - y_i \cdot y_j), \quad k \in I_s,\ j \in I_s'$$

$$\Delta(Y_s) = \sum_{i \in I_s} \Delta_i(Y_s). \qquad (30)$$

It turns out that the choice of $k \in I_s$ does not affect the value of $\Delta_i(Y_s \cup \{y_j\}$. To be definitive, we have set $k = \min i$, $i \in I_s$ in our computations. Inspection shows that (30) can be viewed as a recursion which determines the $\Delta_i(Y_s)$ in order of increasing cardinality of Y_s. The total number of operations for all subsets of Y is modest: $v = 2$ requires 3 inner product evaluations, $v = 3$ requires 6 inner product evaluations and 12 multiplies, $v = 4$ requires 10 inner product evaluations and 36 multiplies.

Theorem 3: The representation (29) holds for Y_s if and only if: 1) $\Delta(Y_s) > 0$, 2) $\Delta_i(Y_s) > 0$ for each $i \in I_s$, 3) $\Delta_j(Y_s \cup \{y_j\}) \le 0$ for each $j \in I_s'$. Furthermore, the λ^i in (29) are given by

$$\lambda^i = \Delta_i(Y_s)/\Delta(Y_s). \qquad (31)$$

The theorem is proved in Appendix II. In brief: condition 1) implies Y_s is affinely independent; condition 2) implies $v(\text{co}\ Y_s)$ is in the relative interior of co Y_s; condition 3) implies $v(\text{co}\ Y_s) = v(\text{co}\ Y)$. Usually, Y_s is uniquely determined; multiple representations (29) can occur only if Y is affinely dependent.

Because a representation of the form (29) exists, conditions 1), 2), 3) must hold for some $s \in \{1, \cdots, \sigma\}$. Thus in principle, the following algorithm must terminate in step 2).

Distance Subalgorithm: Given a finite set $Y = \{y_1, \cdots, y_v\} \subset R^m$, and an ordering $Y_s, s = 1, \cdots, \sigma$, of all subsets of Y, perform the following steps:

1) set $s = 1$;
2) if $\Delta(Y_s) > 0$ and $\Delta_j(Y_s) > 0$, $j \in I_s$, and $\Delta_j(Y_s \cup \{y_j\}) \leq 0, j \in I_s'$, define $\nu(\text{co } Y)$ by (29) and (31) and stop;
3) if $s < \sigma$, increment s and proceed to step 1);
4) stop and indicate failure.

If there are numerical errors in the computation of the data in step 2), it may turn out on rare occasions that the conditions of step 2) are not satisfied for any $s \in \{1, \cdots, \sigma\}$. We need to account for this possibility in the next section. Thus we have added step 4).

Suppose we obtain (29) with $v = m + 1$ and $Y_s = Y$. Then co Y is a simplex and $\nu(\text{co } Y) \in$ interior co Y. Hence, $\nu(\text{co } Y) = 0$ and there is a sphere of maximum radius d^- centered on the origin, contained in co Y. The radius is of interest because it is a lower bound on the distance which co Y must be translated if the origin is to be exterior to co Y. Clearly, d^- is given by the distance to the m-dimensional face of co Y which is closest to the origin. Hence

$$d^- = \min \{|\nu(\text{aff } Y_s)| : Y_s \subset Y \text{ has } m \text{ elements}\}. \quad (32)$$

In Appendix II it is shown that

$$|\nu(\text{aff } Y_s)|^2 = \Delta(Y_s)^{-1} \sum_{i \in I_s} \Delta_i(Y_s) y_i \cdot y_k, \qquad k \in I_s. \quad (33)$$

The data $\Delta(Y_s)$, $\Delta_i(Y_s)$, $y_i \cdot y_k$ are all needed in the determination of $\nu(\text{co } Y)$ and require no additional computational effort. Thus (33) is evaluated with only m multiplies and one divide. If $m = 3$, this means (32) takes 12 multiplies, 4 divides, and 1 square root. The choice of $k \in I_s$ in (33) is arbitrary.

VI. The Numerical Algorithm

Having fully established the theoretical algorithm for computing the distance between compact convex sets, we now present modifications of the algorithm to make it totally reliable in the presence of roundoff errors. This is followed by some comments on the efficient implementation of the algorithm.

Errors do not accumulate in the Distance Algorithm since at every iteration k, $v_k = \nu(\text{co } V_k)$ results from the explicit evaluation of formulas which are only dependent on the set V_k. This helps the ultimate accuracy of the results and simplifies the error analysis.

Inner product evaluations are one source of error. When

$$K = \text{co } Z_1 - \text{co } Z_2, \qquad Z_i = \{z_{ij} : j = 1, \cdots, M_i\}, \quad i = 1, 2$$

we can reduce these errors by moving the origin of the system to a point located on the line segment joining the centroids of the sets Z_1 and Z_2. That is, we replace Z_1 and Z_2 by $\bar{Z}_i = \{z_{ij} - \rho_c : j = 1, \cdots, M_i\}$, $i = 1, 2$, where

$$\rho_c = \frac{1}{2}(\bar{z}_1 + \bar{z}_2) \qquad \bar{z}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} z_{ij}, \qquad i = 1, 2. \quad (34)$$

Since co $\bar{Z}_1$ − co $\bar{Z}_2 = K$, all our previous notations apply. The translation of the origin is only worthwhile when $d(K_1, K_2)$ is small, $|\rho_c|$ is large, and high accuracy is needed.

Other sources of error include the evaluation of (29)–(31) and $g_K(v_k)$. To account for these errors and those from the inner products, it is reasonable to replace the convergence criterion in step 3) of the Distance Algorithm by

$$g_K(v_k) \leq \epsilon (D(K))^2 \quad (35)$$

where $\epsilon > 0$ is related to floating-point accuracy and

$$D(K) = \max \{|z| : z \in K\}. \quad (36)$$

Since ϵD^2 is very small, result 3) of Theorem 1 shows that the effect on the accuracy of the final result should be small. If $K = \text{co } Z_1 - \text{co } Z_2$ as in the preceding paragraph, and the origin of the system is translated as indicated, then the upper bound on $D(K)$, given by

$$D(K) \leq D(\text{co } Z_1 - \{\bar{z}_1\}) + D(\text{co } Z_2 - \{\bar{z}_2\}) + |\bar{z}_1 - \bar{z}_2|, \quad (37)$$

may be appropriately used in (35). When Z_1 and Z_2 are dependent on q (see end of Section II), the first two terms in (37) are independent of q and may be computed from the w_{ij} which specify C_i, $i = 1, 2$.

On rare occasions, numerical errors may also cause the Distance Subalgorithm to fail, especially when $Y_s = V_k$ is affinely dependent or nearly so. For example, if $y_j, j \in I_s'$, is close to aff Y_s, $\Delta_j(Y_s \cup \{y_j\})$ is close to zero. If the numerical value of $\Delta_j(Y_s \cup \{y_j\})$ is positive when the actual value is negative, the exit through step 4) may occur. If the Distance Subalgorithm does fail, we resort to the following Backup Procedure which always runs to completion.

Given $Y = \{y_1, \cdots, y_v\}$, the Backup Procedure determines $\nu(\text{co } Y)$ by evaluating $\nu(\text{aff } Y_s)$ for all $Y_s \subset Y$ such that $\Delta(Y_s) > 0$, $\Delta_j(Y_s) > 0$, $j \in I_s$. Clearly, such Y_s are all candidates for the representation (29). The Backup Procedure merely picks the best of the Y_s and sets $\nu(\text{co } Y) = \nu(\text{aff } Y_s)$

$$\nu(\text{co } Y) = \arg\min \{|y| : y = \nu(\text{aff } Y_s), \quad s = 1, \cdots, \sigma, \quad \Delta(Y_s) > 0, \quad \Delta_j(Y_s) > 0, \quad j \in I_s\} \quad (38)$$

where $|\nu(\text{aff } Y_s)|$ is calculated using (33). In most cases (38) involves more effort than the Distance Subalgorithm, but it always succeeds since $\Delta(Y_s) > 0$, $\Delta_j(Y_s) > 0$, $j \in I_s$ when Y_s is a single element of Y.

The above comments lead to the following algorithm.

Numerical Algorithm: Given a compact convex set $K \subset R^m$ and initial points $y_1, \cdots, y_v \in K$, $1 \leq v \leq m + 1$, perform the following steps:

1) set $V_0 = \{y_1, \cdots, y_v\}$ and $k = 0$;
2) set $Y = V_k$ and apply the Distance Subalgorithm; if it succeeds set alg = *DS*, otherwise use the Backup Procedure (38) and set alg = *BP*; set $v_k = \nu(\text{co } Y)$ and $\hat{V}_k = Y_s$ where Y_s satisfies (29);
3) if (35) holds, output v_k, Y_s, and λ^i in (29), $g_K(v_k)$, and stop;

4) if ($k = 0$ or $|v_k| < |v_{k-1}|$) and ($\hat{V}_k$ has m elements or less), let $V_{k+1} = \hat{V}_k \cup \{s_K(-v_k)\}$, increment k, and proceed to step 2);
5) if alg = *BP*, indicate the error tolerance (35) is not satisfied and stop with the output data indicated in step 3);
6) recalculate $v_k = v(\text{co } V_k)$ using the Backup Procedure (38), set alg = *BP*, and proceed to step 3).

The algorithm stops with $v_k \cong v(K)$ and, within the roundoff errors involved in computing $g_K(v_k)$, $|v_k - v(K)| \leq (g_K(v_k))^{1/2}$. By relabeling the elements of Y_s and the λ^i, the expression for v_k has the form (22), which in turn may be broken down into (23).

It is easy to see that the algorithm always terminates, even if $\epsilon = 0$. If ϵ is small but reasonable (say 100 × machine error), the algorithm generally stops in step 3) and rarely passes through steps 5) and 6). Entrance to steps 5) and 6) implies the occurrence of a numerical result which is inconsistent with theory. The condition $|v_k| \geq |v_{k-1}|$, $k \geq 1$ contradicts the expected descent. Furthermore, by the design of the Distance Subalgorithm and the Backup Procedure, $\hat{V}_k$ has $m + 1$ elements only if $v_k = 0$. But $v_k = 0$ contradicts the failure of (35) which is necessary for entrance to step 5). The algorithm exits in step 5) only after both the Distance Subalgorithm and the Backup Procedure have been tried. Step 6) guarantees that the Backup Procedure is always tried before stopping in step 5).

In practice, the Distance Subalgorithm almost always succeeds and produces a near point of high accuracy. Theoretically, both the Distance Subalgorithm and the Backup Procedure produce affinely independent sets $\hat{V}_k$, and $s_K(-v_k)$ should be affinely independent of $\hat{V}_k$. Thus the V_k, $k \geq 1$ should be affinely independent. Even if V_0 is affinely dependent, or V_k, $k \geq 1$, is nearly so, the Distance Subalgorithm usually functions well. We have confirmed this independently of the Numerical Algorithm by extensive experimentation with the Distance Subalgorithm.

When K is the set difference of two polytopes it is not obvious how the initial set V_0 should be chosen. We have tested a variety of schemes. In the absence of additional information about K such as that described in Section VIII, the single point initialization $V_0 = \{s_K(-\bar{z}_1 + \bar{z}_2)\}$ has worked as well as any. Here, $\bar{z}_1 - \bar{z}_2$ is the direction between centroids (see (34)) and serves as a rough estimate of $v(K)$. Note that the initialization is easy to compute using the procedures outlined in Section III.

Attention to details in the implementation of the overall algorithm adds considerably to its efficiency. For example, the inner products of the elements in $\hat{V}_k$ appear in the Distance Subalgorithm (or Backup Procedure) for both $Y = V_k$ and $Y = V_{k+1}$ and can be saved for the $Y = V_{k+1}$ computation. Hence, if $\hat{V}_k$ has v elements, only $(v + 1)$ new inner products need to be calculated when $v(\text{co } V_{k+1})$ is determined.

Another aid to efficiency is the choice of ordering of the sets Y_s, $s \in \{1, \cdots, \sigma\}$, in the Distance Subalgorithm. The sets most likely to produce the near point should be put at the beginning of the list. Some of the subsets of $Y = V_k$ have already been tested in $Y = V_{k-1}$, and they are put at the end of the list (essentially, they are eliminated). We have found it especially effective to put one face of co V_k at the head of the list. It is determined by $Y_1 \subset Y = V_k = \hat{V}_{k-1} \cup \{s_K(-v_{k-1})\}$ such that $V_k = Y_1 \cup \{y\}$ and y maximizes $y \cdot s_K(-v_{k-1})$ over all $y \in \hat{V}_{k-1}$. Our experience indicates that co Y_1 contains $v(\text{co } V_k)$ about 80 percent of the time. The complete description of our ordering procedure is too lengthy for inclusion here.

It is worth noting that we have experimented with many variants of the basic distance algorithm, including those described in [1] and [29]. Our implementation is in part motivated by these experiments; for $m = 3$ it has proved to be significantly more efficient than the others which were tried.

VII. The Subroutine and Numerical Experiments

The algorithm described in the previous section has been programmed as a Fortran subroutine which is well commented and provides a number of options. The input data are the vertices of the two polytopes. The output data include: near points in each of the two polytopes, the vertices used in their representation and the corresponding values of the λ^i (recall (23) and (24)), the final value of $g_K(v_k)$, and a variety of error messages. Options include: the tolerance parameter ϵ, the shifting of the origin for improved accuracy, the outputting of internal algorithmic data, and internal or external initialization procedures. The subroutine has been applied to a large number of examples in three space. Fig. 4 summarizes the main results.

The examples were generated by selecting 20 pairs of polytopes from a family of 12 polytopes. The members of the family were centered on the origin and were of varying size (contained in spheres of radius 1 to 4). They included: a line segment ($M_1 = 2$), an equilateral triangle ($M_2 = 3$), a rectangular box ($M_3 = 8$), a truncated cone with hexagonal ends ($M_4 = 12$), truncated cylinders with octagonal and decagonal cross sections ($M_5 = 16$ and $M_6 = 20$), and a collection of irregular polytopes generated by placing an equal number of vertices randomly on circles in two parallel planes ($M_7 = 20$, $M_8 = 40$, $M_9 = 50$, $M_{10} = 60$, $M_{11} = 100$, $M_{12} = 100$). The twenty pairs selected were: $(i, j) = (i, 2)$, $(i, 4)$, $(i, 5)$, $(i, 10)$ with $i = 1, 3, 6, 8$ and $(7, 9)$, $(7, 12)$, $(11, 9)$, $(11, 12)$. For each of the 20 pairs three cases were considered: polytopes separated, just touching, or intersecting. In each of the cases there were 100 different examples, generated by random translations and rotations of the two polytopes. For the separated cases the expectation of the relative translation between the two polytopes was 10/3. The just touching and intersecting examples were generated by appropriate translations of the polytopes along the line joining the near points for the separated examples. The total number of examples was 6000.

The examples were run on a Harris 800 computer, which is somewhat faster than a VAX 780. The machine precision is 10^{-11} and the parameter ϵ was set equal to 10^{-9}. In every example the program ran to completion and did not require the use of steps 5), 6), or the Backup Procedure. The accuracy of the final results as measured by $g_K(v_k)$ was excellent; typical values were in the order of 10^{-10}.

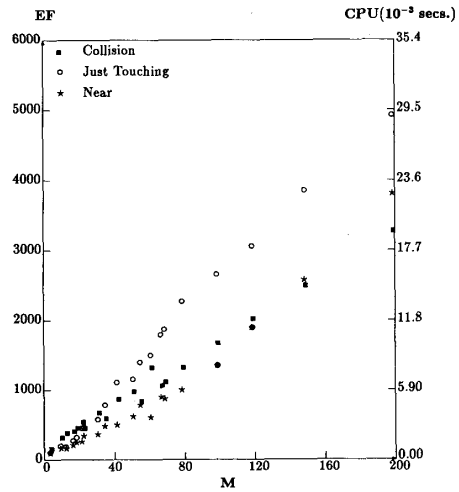


Fig. 4. Equivalent flops (EF) and CPU time versus the total number of vertices (M). Each point is the average of 100 randomly generated examples.

The actual number of operations (multiplies N_M, adds N_A, divides N_D, and comparisons N_C) were counted for each example. These were converted to equivalent flops, EF, by the following formula:

$$\mathrm{EF} = (t_M N_M + t_A N_A + t_D N_D + t_C N_C)/(t_M + t_A) \quad (39)$$

where the t's denote the times required for the operations. For the Harris the times in microseconds are: $t_M = 3.8$, $t_A = 2.1$, $t_D = 6.7$, $t_C = 1.7$. For a different machine, EF would be different because the relative times required for the operations would be different. However, the variation of EF from machine to machine should not be very great. The EF's plotted in Fig. 4 are the averages over the 100 examples in each case. The approximate times in seconds for the Harris computer can be obtained by multiplying EF by 6×10^{-6}. See the CPU scale in Fig. 4.

The results can be summarized as follows. Remarkably, the number of iterations k required for termination did not vary appreciably; for problems of all types and sizes it was generally in the range 3 to 6. For problems of moderate size, $M = M_i + M_j \leq 40$, the intersection cases are the most difficult. They require approximately 24 EF/M. For larger problems the just touching cases are most difficult, with EF/M ranging between 24 and 27. There is some evidence that EF/M grows slightly with M, but the increase is definitely less than log M. When the data for the three cases are averaged together, the performance is more uniform with EF/M ranging between 14 and 19 for all values of M.

Additional examples have been considered. When the algorithm is run on polytopes which are very near to each other, the computational times become close to those for the just touching cases; but on the average, never do they take more time than the just touching case. When the polytopes are widely separated the times drop significantly, with EF/$M \leq$ 7.

Pairs of line segments, $M = 4$, were tried using the same cases and numbers of runs described above. The results for EF were: separated, 36; just touching, 39; intersecting, 96. For line segments, the intersecting case (both segments contained in a common line) is truly pathological and should probably be discarded. It is interesting to compare our algorithm with the efficient algorithm developed by Lumelsky [22] for the special case of line segments. When his algorithm is arranged to produce the same results as ours, EF ranges between 38 and 40 (using the Harris time weights). Thus our algorithm appears to be competitive even though it is designed to handle the general polytope problem.

In general, one might expect the computational effort to be dependent on the shape of the objects and, for fixed M, M_i and M_j. In a variety of experiments which have been performed to test such behavior, some variation has been noted. But it is not very great, about 25 percent at most. The fact that the effort is proportional to $M_i + M_j$ is most encouraging. In combinatoric procedures it is proportional to $M_i M_j$.

There is no reason to expect that the algorithm is linear in M for all classes of polytope pairs. We have constructed an example in R^2 with $M_1 = 1$ and $M_2 \geq 1$ which shows the computations grow as $O(M_2^2) = O(M^2)$. This example is pathological in that the vertices of K_2 must be clustered ever closer to the near point in K_2 as M_2 increases. Even in this "bad" example the algorithm works well in the sense that $|\nu_k - \nu(K)|$ is exceedingly small after a modest number of iterations.

VIII. An Example of Collision Detection

In this section we consider an object which is continuously translated and rotated through a field of obstacles. Specifically, its position and orientation are given on a configuration space path defined by a continuous function $q(s)$. The initial position corresponds to $s = 0$ and the terminal position to $s = 1$. To locate approximately the points of collision on the path, the distances between the object and each of the obstacles is evaluated for $s = t/T$, where t and T are integers and $t = 0, \cdots, T$. If T is large, the collision points are located closely by the values of t where the distance just goes to zero.

The computational time can be decreased by using the general initialization feature of the distance algorithm. Suppose, for instance, $d_{12}(q(s))$ has been determined for $s = t/T$ and the corresponding near points are given by (24). From the comments in Sections III and IV, it is reasonable to assume the y_{1i} and the y_{2i} are points taken, respectively, from the finite sets $Z_1(q(s))$ and $Z_2(q(s))$ which generate $K_1 = \mathrm{co}\, Z_1$ and $K_2 = \mathrm{co}\, Z_2$. If T is large, the position and orientation of K_1 and K_2 change only slightly in one time increment and it is likely that the vertices in (24) for $s = t/T$ and $s = (t+1)/T$ are the same. Thus the algorithm is started at $s = (t+1)/T$ with $V_0 = \{y_{1i} - y_{2i}, i = 1, \cdots, l\}$ where $y_{1i} \in Z_1(q((t+1)/T))$ and $y_{2i} \in Z_2(q((t+1)/T))$ have the same indices as the elements in (24) from the previous stage. Of course, the λ^i change to account for the motion of the sets and the algorithm must determine these changes. But it does not have to spend time finding the points in (24). Even if new points must be found by the algorithm, the starting set V_0 is likely to be more effective than the single point initialization described in Section VI.

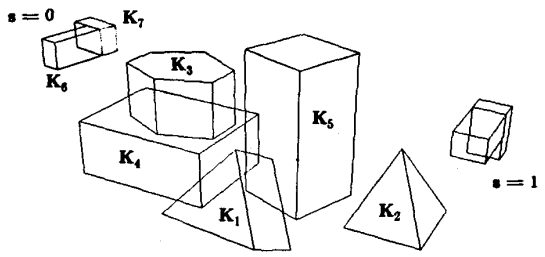


Fig. 5. The example of collision detection.

Fig. 5 shows a particular example. It was provided to us by John Canny who used it to demonstrate his quaternion technique [8] for computing collision times. The initial and terminal positions of the moving object, $K_A = K_6 \cup K_7$, together with the fixed objects, $K_1, \cdots, K_5$, are indicated. The configuration variables specifying the motion are the Cartesian position and the quaternion representation of rotation given in [8]. The configuration variables vary linearly in s from the initial position to the terminal position.

Fig. 6 shows the results of the computations. The distances between K_A and each of the five obstacles are denoted by $d_1, \cdots, d_5$. For all values of s it turns out that $d_{6i} \leq d_{7i}$ so that $d_i = d_{6i}$, $i = 1, \cdots, 5$. The computational times are shown in Table I for both the special initialization described above and the single point initialization of Section VI. The improvement due to the special initialization is significant, and as expected, gets better as T increases.

Inspection of Fig. 6 shows that object K_A collides with objects K_3 and K_5 during the course of the movement from $s = 0$ to $s = 1$. The collision points are determined by those values of s for which d_3 and d_5 just become zero. Our procedure determines these points within the resolution of the grid. Since Canny's algorithm is a root finding procedure on s, it locates the collision points precisely but does not determine the separation distances. His computational time is 11.6 s, on a Symbolics 3600 computer.

When object K_A intersects K_3 or K_5 it is of interest to know the degree of the penetration. Buckley and Leifer [5] and Cameron and Culley [7] have proposed a negative distance which measures the penetration. It can be shown that the negative distance between intersecting objects K_i and K_j is $d(K_i, K_j) = -\min \{|z|: z \in \text{boundary of } K\}$, where $K = K_i - K_j$. The computation of the negative distance is difficult [5]. However, when K_i and K_j are compact, convex sets, and the distance algorithm terminates with $\nu_k = 0 = \nu(\text{co } Y)$ where $Y = \hat{V}_k$ has $m + 1$ elements, it is clear from Section V that $d(K_i, K_j) \leq -d_{ij}^-$ where d_{ij}^- is given by d^- in (32). Also, when K_A and K_j intersect $d(K_A, K_j) \leq d(K_i, K_j)$, $i = 6, 7$. Thus when K_A and K_j intersect, the negative distance has the bound $d(K_A, K_j) \leq -d_j^-$, where $d_j^- = \max \{d_{6j}^-, d_{7j}^-\}$. Since $-d_3^-$ and $-d_5^-$ are easily computed, they are plotted in Fig. 6 as negative extensions of d_3 and d_5. It is not known how closely these bounds estimate the negative distances $d(K_A, K_3)$ and $d(K_A, K_5)$, but they do determine that significant collision penetrations have occurred.

We have run a simple test problem where the negative distance $d(K_i, K_j)$ can be obtained analytically as a function of

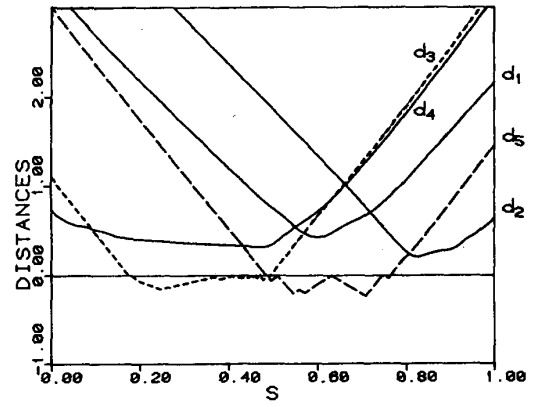


Fig. 6. Results for the example in Fig. 5. The d_i are the distances between K_A and the K_i.

TABLE I
CPU TIMES (Harris 800) IN SECONDS FOR THE EXAMPLE OF FIG. 5

Number of Intervals in Grid (T)	Time with Single Point Initialization	Time with Special Initialization	Ratio of Times
10	0.22	0.13	1.7
100	2.00	0.69	2.9
1000	19.81	6.32	3.1

q. The computed lower bounds range from good to poor, and are best when $|d(K_i, K_j)|$ is not too large. Fortunately, this is the situation of greatest interest.

IX. Conclusion

We have presented an algorithm for determining the Euclidean distance between compact sets in R^m. The emphasis has been on polytopes in R^3, since this is the single most important case in applications. Input data for the algorithm are in the form of finite sets of points whose convex hulls define the polytopes. This data format is particularly convenient in robotics applications where the position and orientation of the polytopes may be functions of configuration variables such as joint angles. Extensive numerical experience shows that the algorithm is efficient and reliable with a computational cost which is approximately linear in the total number of points specifying the polytopes.

The algorithm has some other special advantages. It provides the nearest points in the two polytopes. These are of direct interest and can also be used to compute the gradient of the distance with respect to the configuration variables. In continuum problems the algorithm may be initialized in a special way so that the computational time is significantly reduced. We have demonstrated this advantage in the collision detection problem, but it occurs in other applications too, such as the mapping of collision-free regions in configuration space, path finding, and path planning. It has been noted that it is difficult to compute the negative distances of [5], [7] for intersecting objects. Our algorithm provides, with essentially no additional cost, a bound on the negative distance.

Finally, a few comments should be made about sets which are not polytopes or spherical extensions of polytopes.

Suppose the algorithm is applied to the vertex sets of nonconvex polytopes. Then it is easy to see that it produces the distance between the convex hulls of the nonconvex polytopes. This distance is a conservative measure of collision and may be useful. When the distance between an infinite polyhedral cylinder and a polytope is computed, the computations are actually simplified: the vertex points are projected on a plane normal to the axis of the cylinder and the algorithm is applied in the plane (R^2). In the case of general convex sets, it is necessary to have a procedure for evaluating the support function of the sets. This is easy to arrange for ellipsoids and some other special objects. The convergence is not finite, but the algorithm can be made, through the choice of ϵ, to stop with a solution of specified accuracy. Prior experience with a similar algorithm [1] indicates that convergence rates for general sets should be good.

Appendix I

Proof of Theorem 1

Result 1) is obvious if $|s_K(-x)| < |x|$ so assume $|s_K(-x)| \geq |x|$ and define

$$z = x + \lambda(s_K(-x) - x)$$
$$\lambda = g_K(x)/|x - s_K(-x)|^2.$$

It is easy to see that $|x - s_K(-x)|^2 \geq 2g_K(x)$. Thus $0 < \lambda \leq 0.5$, $z \in \text{co}\,\{x, s_K(-x)\}$, and $|z|^2 = |x|^2 - \lambda g_K(x) < |x|^2$. To show result 2), first let $g_K(x) = 0$. Since

$$|x|^2 = -h_K(-x) = \min\{z \cdot x : z \in K\}$$

it is clear that

$$|x|^2 \leq |x|^2 + |z-x|^2 = |z|^2 + 2(|x|^2 - z \cdot x) \leq |z|^2, \quad \text{for all } z \in K.$$

Therefore, $x = \nu(K)$. Now let $x = \nu(K)$ and assume $g_K(x) > 0$. Since result 1) implies $x \neq \nu(K)$, we must have $g_K(x) \leq 0$. However, $x \in K$, (15) and (25) imply $g_K(x) \geq 0$. Therefore, $g_K(x) = 0$. Result 2) implies $|\nu(K)|^2 = -h_K(-\nu(K)) \leq z \cdot \nu(K)$ for all $z \in K$; consequently, $|x - \nu(K)|^2 \leq |x|^2 - x \cdot \nu(K) \leq |x|^2 + h_K(-x)$.

Appendix II

Proof of Theorem 3

First, we consider the determination of $\nu(\text{aff } Y_s)$ for $Y_s \subset Y$. If Y_s is a singleton the solution is trivial, so assume Y_s has $r > 1$ elements and let $x_1, \cdots, x_r$ represent an arbitrary ordering of these elements. In this case

$$\nu(\text{aff } Y_s) = \sum_{i=2}^{r} \lambda^i x_i$$

where

$$\lambda^1 = 1 - \sum_{i=2}^{r} \lambda^i$$

and the $\lambda^2, \cdots, \lambda^r \in R$ result from the unconstrained minimization of

$$f(\lambda^2, \cdots, \lambda^r) = \left| x_1 + \sum_{i=2}^{r} \lambda^i (x_i - x_1) \right|^2.$$

Since f is convex, the necessary and sufficient conditions for optimality are $\partial f(\lambda^2, \cdots, \lambda^r)/\partial \lambda^i = 0$, $i = 2, \cdots, r$. Consequently, $\lambda \in R^r$ solves the linear system

$$A_s \lambda = b \qquad A_s \in R^{r \times r} \qquad b \in R^r \tag{40}$$

where

$$A_s = \begin{bmatrix} 1 & \cdots & 1 \\ (x_2 - x_1) \cdot x_1 & \cdots & (x_2 - x_1) \cdot x_r \\ \vdots & & \vdots \\ (x_r - x_1) \cdot x_1 & \cdots & (x_r - x_1) \cdot x_r \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{41}$$

To determine λ, define $\Delta_i(Y_s)$, $i \in I_s$, as the cofactor of element $A_s^{1j}(x_1, \cdots, x_r)$ where j satisfies $x_j = y_i$. This is notationally correct since one may show, using elementary row and column operations on the matrix $A_s(x_1, \cdots, x_r)$, that these cofactors are invariant with respect to the selected order to the elements of Y_s. If we define $\Delta(Y_s)$ as the determinant of A_s, then a first row expansion yields the last equation in (30). If $\Delta(Y_s) > 0$ then the solution to the linear system (41) is unique, and expressing it by Cramer's rule yields

$$\nu(\text{aff } Y_s) = \sum_{i \in I_s} [\Delta_i(Y_s)/\Delta(Y_s)] y_i. \tag{42}$$

The existence of this representation is guaranteed by the following Lemma.

Lemma 1: $\Delta(Y_s) > 0$ if and only if Y_s is affinely independent.

Proof: If to each row $i > 1$ of A_s we add the product of row 1 times $(x_1 - x_i) \cdot x_1$, then it is clear that $\Delta(Y_s)$ is equal to the determinant of $Q_s^T Q_s$ where

$$Q_s = [(x_2 - x_1) \cdots (x_r - x_1)] \in R^{m \times (r-1)}.$$

Note that Y_s is affinely independent if and only if the columns of Q_s are linearly independent. Since $\Delta(Y_s)$ is the Gramian of the columns of Q_s the result follows immediately.

The recursive formulas (30) follow from the above cofactor definition of the $\Delta_i(Y_s)$. Append a row and column to A_s using $x_{r+1} = y_j$, $j \in I_s'$, as additional data. Then expanding the cofactor $\Delta_j(Y_s \cup \{y_j\})$ of this larger matrix about the first r elements in the appended row gives (30).

We now show that 1), 2), 3) imply the existence of (29) with the λ^i defined by (31). It is geometrically obvious, and can be proved from (41) that $y = \nu(\text{aff } Y_s)$ if and only if

$$y \cdot (y - y_k) = 0, \qquad \text{for all } k \in I_s. \tag{43}$$

Let $y = \nu(\text{aff } Y_s)$, and suppose 1) and 2) are satisfied. Then from Lemma 1 and (42), y is expressed in the form given by (29) and (31). In addition, 1), 3), (42), and (30) imply $y \cdot (y_k - y_j) \leq 0$, $j \in I_s'$, $k \in I_s$. Using this and (43) we obtain $y \cdot (y - y_i) \leq 0$, $i \in \{1, \cdots, v\}$. Hence, for any $x \in \text{co}\, Y$ we have

$$x = \sum_{i=1}^{v} \alpha^i y_i, \quad \sum_{i=1}^{v} \alpha^i = 1, \qquad \alpha^i \geq 0,\ i \in \{1, \cdots, v\}$$

$$y \cdot (y-x)=\sum_{i=1}^{v} \alpha^i y \cdot (y-y_i)\leq 0.$$

Therefore, $g_{\text{co}Y}(y) = 0$ and by result 2) of Theorem 1, $\nu(\text{co } Y) = y$.

We now show the converse. Assume $y = \nu(\text{co } Y)$ is given by (29). Theorem 1, result 2), implies $y\cdot(y - y_i) \leq 0$, $i \in \{1, \cdots, v\}$. Since $\lambda^i > 0$, $i \in I_s$, and

$$\sum_{i\in I_s} \lambda^i y \cdot (y-y_i)=0$$

it is clear $y\cdot(y - y_i) = 0$, $i \in I_s$, and $\nu(\text{aff } Y_s) = y$. However, Lemma 1 yields $\Delta(Y_s) > 0$. Because the coefficients in (42) and (29) are unique, (31) holds. Since $\lambda^i > 0$, $i \in I_s$, we have $\Delta_i(Y_s) > 0$, $i \in I_s$. Finally, subtracting (43) from $y\cdot(y - y_j) \leq 0, j \in \{1, \cdots, v\}$, results in $y\cdot(y_k - y_j) \leq 0, j \in I_s', k \in I_s$. Hence, using $\Delta(Y_s) > 0, y = \nu(\text{aff } Y_s)$, (42), and (30) we must have $\Delta_j(Y_s \cup \{y_j\}) \leq 0, j \in I_s'$.

Result (33) follows from (42) and (43) by setting $y = \nu(\text{aff } Y_s)$.

References

[1] R. O. Barr, "An efficient computational procedure for a generalized quadratic programming problem," *SIAM J. Contr.*, vol. 7, pp. 415–429, 1969.

[2] R. O. Barr and E. G. Gilbert, "Some efficient algorithms for a class of abstract optimization problems arising in optimal control," *IEEE Trans. Automat. Contr.*, vol. AC-14, pp. 640–652, 1969.

[3] J. W. Boyse, "Interference detection among solids and surfaces," *Commun. ACM*, vol. 22, pp. 3–9, 1979.

[4] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in configuration space for findpath with rotation," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 224–233, 1985.

[5] C. E. Buckley and L. J. Leifer, "A proximity metric for continuum path planning," in *Proc. 9th Int. Joint Conf. on Artificial Intelligence*, pp. 1096–1102, 1985.

[6] S. A. Cameron, "A study of the clash detection problem in robotics," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 488–493, 1985.

[7] S. A. Cameron and R. K. Culley, "Determining the minimum translational distance between two convex polyhedra," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 591–596, 1986.

[8] J. Canny, "Collision detection for moving polyhedra," MIT Artificial Intelligence Lab. Rep. 806, 1984.

[9] F. Chin and C. A. Wang, "Optimal algorithms for the intersection and minimum distance problems between planar polygons," *IEEE Trans. Comput.*, vol. C-32, pp. 1203–1207, 1983.

[10] R. K. Culley and K. G. Kempf, "A collision detection algorithm based on velocity and distance bounds," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1064–1069, 1986.

[11] D. P. Dobkin and D. G. Kirkpatrick, "A Linear algorithm for determining the separation of convex polyhedra," *J. Algorithms*, vol. 6, pp. 381–392, 1985.

[12] B. R. Donald, "On motion planning with six degrees of freedom: solving the intersection problems in configuration space," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 536–541, 1985.

[13] H. Edelsbrunner, "On computing the extreme distances between two convex polygons," *J. Algorithms,* vol. 6, pp. 515–542, 1985.

[14] E. G. Gilbert, "An iterative procedure for computing the minimum of a quadratic form on a convex set," *SIAM J. Contr.*, vol. 4, pp. 61–80, 1966.

[15] E. G. Gilbert and D. W. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE J. Robotics Automat.*, vol. RA-1, pp. 21–30, 1985.

[16] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three space," in *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 1883–1889, 1987.

[17] D. W. Johnson, "The optimization of robot motion in the presence of obstacles," Univ. of Michigan, Ph.D. dissertation, 1987.

[18] D. W. Johnson and E. G. Gilbert, "Minimum time robot path planning in the presence of obstacles," in *Proc. IEEE Conf. on Decision and Control*, pp. 1748–1753, 1985.

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Robotics Res.*, vol. 5, pp. 90–98, 1986.

[20] D. T. Lee and F. P. Preparata, "Computational geometry—A survey," *IEEE Trans. Comput.*, vol. C-33, pp. 1072–1101, 1984.

[21] T. Lozano-Perez, "Spacial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, 1983.

[22] V. J. Lumelsky, "On fast computation of distance between line segments," *Inform. Proc. Letters*, vol. 21, pp. 55–61, 1985.

[23] W. Meyer, "Distances between boxes: Applications to collision detection and clipping," in *IEEE Int. Conf. on Robotics and Automation*, pp. 597–602, 1986.

[24] M. Orlowski, "The computation of the distance between polyhedra in 3-space," presented at the SIAM Conf. on Geometric Modeling and Robotics, Albany, NY, July 1985.

[25] W. E. Red, "Minimum distances for robot task simulation," *Robotica*, vol. 1, pp. 231–238, 1983.

[26] R. T. Rockafellar, *Convex Analysis*. Princeton, NJ: Princeton Univ. Press, 1970.

[27] J. T. Schwartz, "Finding the minimum distance between two convex polygons," *Inform. Process. Lett.*, vol. 13, pp. 168–170, 1981.

[28] J. T. Schwartz and M. Sharir, "On the piano movers problem I, the special case of a rigid polygonal body moving amidst polygonal barriers," *Commun. Pure Appl. Math.*, vol. 36, pp. 345–398, 1983.

[29] P. Wolfe, "Finding the nearest point in a polytope," *Math. Programming*, vol. 11, pp. 128–149, 1976.

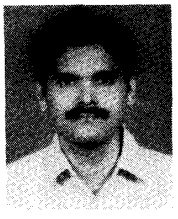
Elmer G. Gilbert (S'51–A'52–M'57–SM'78–F'79) received the B.S. and M.S. degrees in electrical engineering and the Ph.D. degree in instrumentation engineering, all from the University of Michigan, Ann Arbor.

Since 1957 he has been a faculty member in the Department of Aerospace Engineering at the University of Michigan, where he is currently a Professor. He is also a member of the Department of Electrical Engineering and Computer Science and a participant in the Center for Research on Integrated Manufacturing at the university. His current research interests include optimization, nonlinear control, and robotics. His special awards include the O. H. Schuck Award for best paper at the 1978 Joint Automatic Control Conference and a Control Systems Society best paper award in 1979. He has published numerous papers and holds eight patents.

Daniel W. Johnson was born in Detroit, MI, on April 23, 1955. He received the B.S. degrees in mathematics and physics from Lawrence Institute of Technology, Southfield, MI, in 1976. He received the M.S. degree in computer, information and control engineering and the Ph.D. degree in aerospace engineering from the University of Michigan, Ann Arbor, MI, in 1983 and 1987, respectively.

From 1976 to 1979, he was a Mathematician and later a Control Systems Engineer with Bendix Research Laboratories, Southfield, MI. From 1979 to 1984, he was a Member of the Technical Staff at Bendix Advanced Technology Center, Columbia, MD. He is currently a Senior Staff Engineer with Martin Marietta Aero and Naval Systems, Baltimore, MD. His professional interests include flexible structure dynamics and control, multivariable plant identification, optimization theory, and computer-controlled robotics.

Dr. Johnson is a member of the American Institute of Aeronautics and Astronautics.



S. Sathiya Keerthi was born in Aruppukottai, India, on March 21, 1959. He received the B.E. (Hons.) degree in mechanical engineering from University of Madras, India, in 1980, the M.S. degree in mechanical engineering from University of Missouri-Rolla, in 1982, and the Ph.D. degree in computer, information and control engineering, from the University of Michigan, Ann Arbor, in 1986.

He has recently joined the faculty of the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore, as Assistant Professor. His research interests are mainly in the areas of computational methods of optimal control, and optimization.