RSD-TR-26-86

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

by

E.G. Gilbert

D.W. Johnson

and

S.S. Keerthi[†]

The University of Michigan
Ann Arbor, MI 48109

October 1986

CENTER FOR RESEARCH ON INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109-1109

[†]This research was partly supported by the Center for Research in Integrated Manufacturing at the University of Michigan

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 make its application in a variety of robotics problems attractive. These are discussed and an example of collision detection is given.

TABLE OF CONTENTS

1. Introduction

In this paper we present an efficient algorithm for determining the Euclidean distance between two convex sets in three dimensional space. This problem is important in robotics and occurs also in other fields such as computer aided design and computer graphics. For convex polytopes and their spherical extensions, the algorithm terminates finitely. 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. Because the algorithm is so 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 of the algorithm have been to optimal path planning in the presence of obstacles [15], [17], [18].

Since there is an extensive literature concerning the polytope distance problem, we limit ourselves to 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 [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 [11], but the $O(M)$ result there seems to be in error; the actual time appears to be $O(M \log M)$. See [24] for another $O(M \log M)$ result. Because of their 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. Other schemes have also been described: [25] presents a program which uses a projection/combinatoric approach for polyhedra with facial representations, [5] and [7] are concerned with "directed" or "translational" distances (more about this later), [23] considers boxes [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 procedure 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 in continuum collision detection problems, they allow significant reductions in the total computation time. The algorithm has good numerical properties and bounds on the computational errors are available. An early version of the algorithm due to D.W. Johnson was used in the optimal path planning computations described in [18]. A detailed treatment of underlying algorithmic questions in a broader setting is given in [16].

The plan of the paper is as follows. In Section 2 we formulate distance measures for complex, not necessarily convex, objects and suggest how our algorithm may be applied to their computation. We also review what happens when the position and orientation of the objects is specified by a set of configuration variables. Section 3 shows how the support properties of the Minkowski set difference between the two sets can be computed efficiently. This leads to the basic problem of finding the distance between the origin and a single convex set. Section 4 describes the theoretical algorithm for solving this problem; Section 5 presents the efficient procedure for elementary polytopes; Section 6 introduces modifications to account for the effects of numerical errors. Many numerical experiments have been carried out; these are reported in Section 7. In Section 8 the algorithm is applied to a collision detection problem due to Canny [8]. A conclusion summarizes the key contributions and indicates some extensions.

2. Object Representations and Distance Measures

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:

$$d(K_A, K_B) = \min\left\{ |x - y| : x \in K_A, y \in K_B \right\}. \tag{2.1}$$

While computational considerations may suggest the use of other metrics in (2.1), 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 (2.1) exists and d is defined. However, it is only for simple objects such as spheres and line segments that formulas for d may be given. For some examples see [19].

If A and B are each composed of a collection of objects, the distance $d(K_A, K_B)$ may be computed in terms of the distances between the constituent objects. Specifically, suppose K_i, $i \in I = \{1, \cdots, N\}$, are compact sets in R^3, I_A and I_B are disjoint index sets in I, and

$$K_A = \bigcup_{i \in I_A} K_i, \quad K_B = \bigcup_{j \in I_B} K_j. \tag{2.2}$$

Then

$$d(K_A, K_B) = \min\left\{ d_{ij} : i \in I_A, j \in I_B \right\}, \tag{2.3}$$

where

$$d_{ij} = \min\left\{ |x-y| : x \in K_i, y \in K_j \right\} = d(K_i, K_j). \tag{2.4}$$

See the example in Figure 1. When K_A and K_B are not convex, they can often be represented by (2.2) where the K_i, $i \in I$, are convex. This allows our algorithm, which works on convex sets, to be applied to non-convex objects.

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

$$K^r = \left\{ x: |x-y| \leq r, y \in K \right\}, r \geq 0. \tag{2.5}$$

It is easy to verify that

$$d(K_A^{r_A}, K_B^{r_B}) = (d(K_A, K_B) - r_A - r_B)^+, \tag{2.6}$$

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

$$K_C = \bigcup_{i \in I_A} K_i^{r_i}, \quad K_D = \bigcup_{j \in I_B} K_j^{r_j} \tag{2.7}$$

implies

$$d(K_C, K_D) = \min\left\{(d_{ij} - r_i - r_j)^+ : i \in I_A, j \in I_B\right\}. \tag{2.8}$$

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 lead to a rich family of geometric shapes, convex polytopes and their spherical extensions, for which our algorithm is effective. Object A in Figure 1 is a simple example of how the family can be exploited. It is the union of two spheres (extensions of points) and a circular cylinder with end caps (an extension of a line segment). A somewhat more complex example is a solid rectangular plate of thickness $2r$ with round edges; it is modelled 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 (2.2) 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. To be more precise, K_i, $i \in I$, is obtained by translating and rotating a closed point set C_i:

$$K_i(q) = \left\{T_i(q)w + p_i(q): w \in C_i\right\}. \tag{2.9}$$

Here: $p_i(q) \in R^3$ is the translation, $T_i(q) \in R^{3\times3}$ 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.,

$$K_i^{r_i}(q) = \left\{ T_i(q)w + p_i(q): w \in C_i^{r_i} \right\}. \tag{2.10}$$

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 $d_{ij}(q)$ 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 $z^*_k \in K_k(\bar{q})$, $k=i,j$, are uniquely determined, $d_{ij}(q)$ does have a gradient at $\bar{q}$. In particular, $\nabla_q d_{ij}(\bar{q}) = \nabla_q \mid T_i(q)w^*_i + p_i(q) - T_j(q)w^*_j - p_j(q) \mid$, $q=\bar{q}$, where $w^*_k = T_k^T(\bar{q})(z^*_k - p_k(\bar{q}))$, $k=i,j$. Once z^*_i and z^*_j have been found (by the distance algorithm) the evaluation of this expression is relatively easy to carry out.

We conclude this section with a few remarks about a distance measure which gives further information when K_A and K_B intersect. For X, $Y \subset R^3$ let $X \pm Y = \{x \pm y: x \in X, y \in Y\}$ and let ϕ denote the empty set. The condition $K_A \bigcap (K_B + \{z\}) \neq \phi$ occurs if and only if $z \in K_A - K_B$. Thus $K_A - K_B$ is the set of translations z for object B which cause A and the translate of B to intersect. Thus,

$$d(K_A, K_B) = \min\left\{ \mid z \mid : z \in K_A - K_B \right\} \tag{2.11}$$

may be interpreted as the smallest translational distance between A and B which allows A and B to touch. Similarly,

$$\begin{aligned} d^-(K_A, K_B) &= inf\left\{ \mid z \mid : z \notin (K_A - K_B) \right\} \\ &= \min\left\{ \mid z \mid : z \in cl(K_A - K_B)' \right\}, \end{aligned} \tag{2.12}$$

where $cl(K_A - K_B)'$ denotes the closure of the complement of $K_A - K_B$, is the lower bound on the translational distances which allow A and B to be separated. Using different notations, the measure d^- has been proposed by Buckley [5] (for convex objects) and Cameron and Culley [7] as a measure of the depth of

intersection. Clearly, $d > 0$ implies $d^- = 0$ and $d^- > 0$ implies $d = 0$.

Unfortunately, d^- is much more difficult to deal with than d. In general, $cl(K_i - K_j)'$ is not convex, even when K_i and K_j are. Thus, the computation of d^-_{ij} is much more difficult than the computation of d_{ij}. When K_A and K_B are given by (2.2) it is easy to see that $d^-(K_A, K_B) \geq d^-_{ij}$, $i \in I_A$, $j \in I_B$. But, (2.3) does not apply to $d^-(K_A, K_B)$. For an example, replace K_A in Figure 1 by $K_6 = \{a\}$. Then $d^-_{65} = d^-_{64} = 0$, and $d(K_A, K_B) > 0$. This shows there is a further difficulty. Even if d^- can be computed for convex objects, it is not possible to compute d^- for the union of convex objects.

There is one case of interest where d^- can be obtained easily. Suppose the following assumptions hold: K_A and K_B are convex, $d(K_A, K_B) > 0$ and $d(K_A^{r_A}, K_B^{r_B}) = 0$. Then it is possible to prove $d^-(K_A^{r_A}, K_B^{r_B}) = r_A + r_B - d(K_A, K_B)$. Without the assumptions, it is only true that $d^-(K_A^{r_A}, K_B^{r_B}) \geq r_A + r_B - d(K_A, K_B)$.

3. 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 $|x|^2 = x \cdot x$. Throughout the section $X \subset R^m$ is compact and $Y \subset R^m$ is finite, i.e., $Y = \{y_1, \cdots, y_v\}$.

The affine and convex hulls of X are given by

$$\mathrm{aff}X = \left\{ \sum_{i=1}^{l} \lambda^i x_i : x_i \in X, \lambda^1 + \cdots + \lambda^l = 1 \right\}, \tag{3.1}$$

$$coX = \left\{ \sum_{i=1}^{l} \lambda^i x_i : x_i \in X, \lambda^i > 0, \lambda^1 + \cdots + \lambda^m = 1 \right\}. \tag{3.2}$$

It is easily confirmed that $\mathrm{aff}X$ is the translate of a linear space. For example, $\mathrm{aff}Y = \mathrm{Y} + \{y_1\}$ where Y is the linear span of $\{y_2 - y_1, \cdots, y_v - y_1\}$. Y is *affinely independent* if $dim\ \mathrm{aff}Y = dim\ \mathrm{Y} = v - 1$. If Y is not affinely independent, it is always possible to pick an affinely independent set $\bar{Y} \subset Y$ such that $\mathrm{aff}\bar{Y} = \mathrm{aff}Y$. The set coY 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 (3.2) l is restricted so that $l \leq dim X + 1$.

The nearest point in X to the origin, $\nu(X)$, is determined by

$$\nu(X) \in X, \ |\nu(X)| = \min\{ |x| : x \in X \} . \tag{3.3}$$

Suppose X is convex. Then the near point $\nu(X)$ is unique (otherwise it is easy to specify $x \in X$ with $|x| < |\nu(X)|$) and has the representation

$$\nu(X) = \sum_{i=1}^{l} \lambda^i x_i, \ x_i \in X, \lambda^i > 0, \lambda^1 + \cdots + \lambda^l = 1, \tag{3.4}$$

where $l \leq m + 1$ (use the Caratheodory theorem). If $\nu(X) \neq 0$, the even stronger result, $l \leq m$, holds. (Since $\nu(X)$ is a boundary point of X, $\nu(X) \in X \bigcap H$ where H is a support plane of X at $\nu(X)$ [26]. Because $dim H = m-1$, the Caratheodory theorem implies $l \leq m$.)

Often, the representation (3.4) is not unique. If the set $\{x_1, \cdots, x_l\}$ is not affinely independent, it is possible (see a proof of the Caratheodory theorem) to obtain a representation of the form (3.4) using a proper subset of $\{x_1, \cdots, x_l\}$. Thus, in (3.4) there is always a choice for l and $\{x_1, \cdots, x_l\}$ such that $\{x_1, \cdots, x_l\}$ is affinely independent.

The support function of X, $h_X: R^n \rightarrow R$, is defined by

$$h_X(\eta) = \max\{ x \cdot \eta : x \in X \}. \tag{3.5}$$

We use $s_X(\eta)$ to denote any solution of (3.5). Specifically, $s_X(\eta)$ satisfies

$$h_X(\eta) = s_X(\eta) \cdot \eta, \quad s_X(\eta) \in X. \tag{3.6}$$

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

$$\begin{aligned} h_{coY}(\eta) = h_Y(\eta) &= \max\{ y_i \cdot \eta : i=1, \cdots, v \} \\ s_{coY}(\eta) = s_Y(\eta) &= y_j, \ y_j \cdot \eta = h_Y(\eta) . \end{aligned} \tag{3.7}$$

Thus, (3.7) provides a simple procedure for evaluating h_X and s_X when X is the polytope coY.

It is apparent from Section 2 that the problem of finding $d_{ij} = d(K_i, K_j)$ is equivalent to the problem of finding $\nu(K)$ where K is the (Minkowski) set difference $K_i - K_j$. For convenience we set $i = 1$, $j = 2$. If K_1 and K_2 are the polytopes coZ_1 and coZ_2, where $Z_k = \{z_{kl} : l=1, \cdots, M_k\}$, then $K = K_1 - K_2$ is the polytope coZ, where $Z = \{z_{1i} - z_{2j} : i=1, \cdots, M_1, j=1, \cdots, M_2\}$. 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 work of others who have used the set difference.

Our algorithm for computing d_{12} is stated in terms of K and requires only the computation of h_K and s_K. These data are particularly simple to evaluate. In fact, it is directly verified that

$$h_K(\eta) = h_{K_1}(\eta) + h_{K_2}(-\eta), \; s_K(\eta) = s_{K_1}(\eta) - s_{K_2}(-\eta). \tag{3.8}$$

For the polytope case, (3.7)-(3.8) 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.

When the algorithm stops, it produces the following data : $l \leq m + 1$, $\lambda^i > 0$, $z_i \in K$, $i = 1, \cdots, l$,

$$\nu(K) = \sum_{i=1}^{l} \lambda^i z_i, \quad d_{12} = |\nu(K)| . \tag{3.9}$$

In the algorithm, the z_i are obtained either from initial data taken from K_1 and K_2 or, the evaluation of $s_K(\eta)$ for various η. Hence, $z_i = z_{1i} - z_{2i}$, $z_{1i} \in K_1$, $z_{2i} \in K_2$ and (3.9) yields

$$d_{12} = | z_1^* - z_2^* |, \quad z_1^* \in K_1 , \quad z_2^* \in K_2 , \tag{3.10}$$

where

$$z_1^* = \sum_{i=1}^{l} \lambda^i z_{1i} , \quad z_2^* = \sum_{i=1}^{l} \lambda^i z_{2i} . \tag{3.11}$$

With all these facts in mind we can put K_1 and K_2 aside and concentrate on the computation of $\nu(K)$ from h_K and s_K.

4. The Theoretical Algorithm

We now present the iterative procedure for determining the near point $\nu(K)$ of any compact convex set $K \subset R^m$. When K is a polytope, it is shown that this procedure terminates after a finite number of steps.

The basic idea is due to Barr and Gilbert [1], [2]: generate a sequence of polytopes contained in K such that their near points converge to $\nu(K)$. It is necessary to compute the near points of these polytopes, but this is a relatively easy calculation since these polytopes will have at most $m+1$ vertices.

To state the algorithm, we first establish criteria for descent and optimality and a bound on approximation error. These results appear elsewhere (e.g. in [14] and [29]), but proofs are given since they are brief and insightful.

Theorem 4.1: Consider a compact convex set $K \subset R^m$ and an arbitrary point $x \in K$. Then: (1) if $|x|^2 + h_K(-x) > 0$, there exists a point $z \in co\{x, s_K(-x)\} \subset K$ satisfying $|z| < |x|$; (2) $x = \nu(K)$ if and only if $|x|^2 + h_K(-x) = 0$; and (3) $|x - \nu(K)|^2 \leq |x|^2 + h_K(-x)$.

Proof: Result (1) is obvious if $|s_K(-x)| < |x|$ so assume $|s_K(-x)| \geq |x|$ and define $z = x + \lambda(s_K(-x) - x)$, $\lambda = (|x|^2 + h_K(-x))/|x - s_K(-x)|^2$. Note that $0 < \lambda \leq 1/2$ since $|x - s_K(-x)|^2 \geq 2(|x|^2 + h_K(-x))$. Hence, $z \in co\{x, s_K(-x)\}$ and $|z|^2 = |x|^2 - \lambda(|x|^2 + h_K(-x)) < |x|^2$. To show result (2), first let $|x|^2 + h_K(-x) = 0$. Since $|x|^2 = -h_K(-x) = \min\{z \cdot x: z \in K\}$, it is clear that $|x|^2 \leq |x|^2 + |z - x|^2 = |z|^2 + 2(|x|^2 - z \cdot x) \leq |z|^2$ for all $z \in K$. Therefore, $x = \nu(K)$. Now let $x = \nu(K)$ and assume $|x|^2 + h_K(-x) > 0$. Since result (1) implies $x \neq \nu(K)$, we must have $|x|^2 + h_K(-x) \leq 0$. However, $|x|^2 + h_K(-x) \geq 0$ since $\min\{z \cdot x: z \in K\} \leq |x|^2$. Therefore, $|x|^2 + h_K(-x) = 0$. Result (3) follows since result (2) implies $|\nu(K)|^2 \leq z \cdot \nu(K)$ for all $z \in K$ and consequently $|x - \nu(K)|^2 \leq |x|^2 - x \cdot \nu(K) \leq |x|^2 + h_K(-x)$.

Distance Algorithm. Given a compact convex set $K \subset R^m$ and $v \in \{1, \cdots, m+1\}$, perform the following steps:

(1) select $V_0 = \{y_1, \cdots, y_v\} \subset K$ and let $k = 0$;

(2) determine $\nu_k = \nu(coV_k)$;

(3) if $|\nu_k|^2 + h_K(-\nu_k) = 0$, set $\nu(K) = \nu_k$ and stop;

(4) let $V_{k+1} = \hat{V}_k \bigcup \{s_K(-\nu_k)\}$ where $\hat{V}_k \subset V_k$ has m elements or less and satisfies $\nu_k \in co\hat{V}_k$, increment k and proceed to step 2.

If the algorithm does not stop in step 3, then $\nu_k \neq 0$ and $|\nu_k|^2 + h_K(-\nu_k) > 0$. Hence, the existence of $\hat{V}_k$ in step 4 is guaranteed (see the comment following (3.4)), and V_{k+1} will always have $m+1$ or fewer elements. Furthermore, descent in the next iteration is guaranteed since result (1) of Theorem 4.1 implies $|\nu_{k+1}| = |\nu(coV_{k+1})| \leq |\nu(co\{\nu_k, s_K(-\nu_k)\})| < |\nu_k|$. The choice of V_0 is quite arbitrary. Ideally, $\nu(coV_0) \approx \nu(K)$. In the absence of any such insight, a variety of single point initializations, such as the one in Section 6, may be used.

For $K \subset R^m$ compact and convex, the algorithm generates a sequence $\{\nu_k\}$ which converges to $\nu(K)$. 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 $\nu(K)$ is obtained after a finite number of steps.

Theorem 4.2: Let K be the convex polytope coZ, where $Z \subset R^m$ is finite. If $s_K(\eta) \in Z$ for all $\eta \in R^m$, the Distance Algorithm generates $\nu(K)$ in a finite number of steps.

Proof: Clearly, $K = co(Z \bigcup V_0)$ and $Z \bigcup V_0 \subset R^m$ is finite. Assume $\nu(K)$ is not generated in N steps where N is the number of nonempty subsets of $Z \bigcup V_0$. In this case, $|\nu_k|^2 + h_K(-\nu_k) > 0$ and $|\nu_{k+1}| < |\nu_k|$ for all $0 \leq k < N$. Since near points are unique $V_k \neq V_l$ for any $0 \leq l < k \leq N$, and since $V_k \subset Z \bigcup V_0$ for all $0 \leq k \leq N$ every subset of $Z \bigcup V_0$ must have entered the algorithm. However, $\nu(K) \in co\overline{V}$ for some $\overline{V} \subset Z \bigcup V_0$ (see Section 3). Therefore, $\nu(K) = \nu(coV_k)$ for some $k < N$.

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 (3.7) - (3.8) and the associated discussion.

If $s_K(\eta) \in Z$ and $V_0 \subset Z$, it follows from the steps of the algorithm that $V_k \subset Z$ for all $k > 0$. Thus, when the algorithm terminates, $\nu(K)$ has a representation of the form (3.9) where the $z_i \in Z$. This observation is useful for initializing the algorithm in continuum problems. See Section 8.

5. The Distance Subalgorithm

Each iteration of the Distance Algorithm requires the determination of $\nu(coY)$, $Y = \{y_1, \cdots, y_v\} \subset R^m$, $1 \leq v \leq m+1$. In this section we describe a procedure originated by Johnson [17] for doing this. It is particularly efficient when m is small (say, $m \leq 4$) and yields a representation of the form

$$\nu(coY) = \sum_{i \in I_s} \lambda^i y_i \ , \ \sum_{i \in I} \lambda^i = 1 \ , \ \lambda^i > 0 \ , \ i \in I_s \subset \{1, \cdots, v\} \ ,$$
$$Y_s = \{y_i : i \in I_s\} \subset Y \text{ is affinely independent} \ , \tag{5.1}$$

where s indicates a particular member of the family of all nonempty subsets of Y. That such a representation exists follows from Section 3.

Since m is small, it is effective to take a combinatoric approach where the $\sigma = \sum_{r=1}^{v} [v!/(r!(v-r)!)]$ subsets of Y are successively tested until a representation of the form (5.1) is found. Geometrically, this test involves checking the open subsets of the polytope coY (e.g., a vertex, an open line segment or an open face) to see if they contain $\nu(coY)$. If $m=3$, there are at most $\sigma = 15$ such subsets to examine. To develop the approach, we first consider a simpler problem: the determination of $\nu(\mathrm{aff} Y_s)$, $Y_s \subset Y$.

It is straightforward to solve for $\nu(\mathrm{aff} Y_s)$. 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, $\nu(\mathrm{aff} Y_s) = \sum_{i=1}^{r} \lambda^i x_i$ 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 $f(\lambda^2, \cdots, \lambda^r) = | x_1 + \sum_{i=2}^{r} \lambda^i (x_i - x_1) |^2$. 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

$$A_s \lambda = b \ , \ A_s \in R^{r \times r} \ , \ b \in R^r \tag{5.2}$$

where

$$A_S(x_1, \cdots, x_r) = \begin{bmatrix} 1 & \cdots & 1 \\ (x_2 - x_1)\cdot x_1 & \cdots & (x_2 - x_1)\cdot x_r \\ \vdots & & \vdots \\ (x_r - x_1)\cdot x_1 & \cdots & (x_r - x_1)\cdot x_r \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{5.3}$$

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 of the elements of Y_S. If we define $\Delta(Y_S)$ as the determinant of A_S, then a first row expansion yields

$$\Delta(Y_S) = \sum_{i \in I_S} \Delta_i(Y_S). \tag{5.4}$$

If $\Delta(Y_S) > 0$ then the solution to the linear system (5.2) is unique, and expressing it by Cramer's rule yields

$$\nu(\mathrm{aff} Y_S) = \sum_{i \in I_S} \left[\Delta_i(Y_S)/\Delta(Y_S)\right] y_i \quad . \tag{5.5}$$

This representation holds when Y_S is affinely independent.

Theorem 5.1: $\Delta(Y_S) > 0$ if and only if Y_S is affinely independent.

Proof: If to each row $i > 1$ of A_S we add the product of the first row 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 $Q_S = \left[(x_2 - x_1)\cdots(x_r - x_1)\right] \in R^{m \times (r-1)}$. Thus, $\Delta(Y_S)$ is the grammian of the vectors $(x_2 - x_1), \cdots, (x_r - x_1)$ and $(x_2 - x_1), \cdots, (x_r - x_1)$ are linearly independent if and only if $\Delta(Y_S) > 0$. By the sentences below (3.2) the proof is complete.

We may efficiently calculate the near point to any affinely independent subset of Y using (5.4) - (5.5) and a recursive formula for the cofactors. This formula is developed by appending a row and column to A_S using $x_{r+1} = y_j$ ($j \in I_S'$, I_S' being the complement of I_S in $\{1, \cdots, v\}$) as additional data,

and expanding the cofactor $\Delta_j(Y_s \bigcup \{y_j\})$ of this larger matrix about the first r elements in the appended row. The result is given by

$$\Delta_j(Y_s \bigcup \{y_j\}) = \sum_{i \in I_s} \Delta_i(Y_s) y_i \cdot (y_k - y_j), \; k \in I_s, \; j \in I_s' ,$$
$$\Delta_i(\{y_i\}) = 1, \; i=1, \cdots, v . \tag{5.6}$$

This equation is valid for any $k \in I_s$ (i.e., Δ_j does not vary with k), so we set $k = \min\{i \in I_s\}$ to be definitive. When $v=4$, only 36 multiplies and 10 inner-product evaluations are required to evaluate all the cofactors of all the subsets of Y. Moreover, these data coupled with the evaluation of $\Delta(Y_s)$, $Y_s \subset Y$ are all that is needed to determine the subset of Y required for (5.1).

Theorem 5.2: Consider a finite set $Y = \{y_1, \cdots, y_v\} \subset R^m$ and a nonempty subset $Y_s \subset Y$. Then $\nu(coY)$ may be written in the form of (5.1) using Y_s if and only if (1) $\Delta(Y_s) > 0$, (2) $\Delta_i(Y_s) > 0$ for each $i \in I_s$ and (3) $\Delta_j(Y_s \bigcup \{y_j\}) \leq 0$ for each $j \in I_s'$. Furthermore,

$$\nu(coY) = \sum_{i \in I_s} \left[\Delta_i(Y_s)/\Delta(Y_s)\right] y_i \tag{5.7}$$

whenever Y_s satisfies these conditions.

Proof: It is geometrically obvious, and can be proved from (5.2) that $y = \nu(\mathrm{aff} Y_s)$ if and only if

$$y \cdot (y - y_k) = 0, \; k \in I_s . \tag{5.8}$$

Let $y = \nu(\mathrm{aff} Y_s)$, and suppose (1) and (2) are satisfied. Then from Theorem 5.1 and equations (5.4)-(5.5), y has the form of the right hand side of equation (5.1). In addition, (1), (3), and (5.5)-(5.6) imply $y \cdot (y_k - y_j) \leq 0$, $j \in I_s'$, $k \in I_s$. Using this and (5.8) we obtain $y \cdot (y - y_i) \leq 0$, $i \in \{1, \cdots, v\}$. Hence, for any $x \in coY$ we have $x = \sum_{i=1}^{v} \alpha^i y_i$, $\sum_{i=1}^{v} \alpha^i = 1$, $\alpha^i \geq 0$, $i \in \{1, \cdots, v\}$, $y \cdot (y - x) = \sum_{i=1}^{v} \alpha^i \, y \cdot (y - y_i) \leq 0$. Therefore, by result 2 of Theorem 4.1, $\nu(coY) = y$.

We now show the converse. Assume $y = \nu(coY)$ is given by (5.1). Theorem 4.1, 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 $\nu(\mathrm{aff} Y_s) = y$. However, Theorem 5.1 yields $\Delta(Y_s) > 0$. Therefore, the coefficients in (5.5) and (5.1) are unique, $\lambda^i = \Delta_i(Y_s)/\Delta(Y_s)$, $i \in I_s$ and, since $\lambda^i > 0$, $i \in I_s$ we have $\Delta_i(Y_s) > 0$, $i \in I_s$. Finally, subtracting (5.8) 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 $\Delta(Y_s) > 0$ and equations (5.5)-(5.6), we must have $\Delta_j(Y_s \bigcup \{y_j\}) \leq 0$, $j \in I_s'$.

Distance Subalgorithm. Given a finite set $Y = \{y_1, \cdots, y_v\} \subset R^m$, perform the following steps:

(1) select an ordering Y_s, $s=1, \cdots, \sigma$ of all subsets of Y and set $s = 1$;

(2) if $\Delta(Y_s) > 0$ and $\Delta_j(Y_s) > 0$, $j \in I_s$ and $\Delta_j(Y_s \bigcup \{y_j\}) \leq 0$, $j \in I_s'$, then calculate $\nu(coY)$ using (5.7) and stop;

(3) if $s < \sigma$, increment s and proceed to step (1);

(4) stop and indicate failure.

From Section 3 we know $\nu(coY)$ can be written in the form of (5.1). Thus, according to Theorem 5.2, there exists a $Y_s \subset Y$, $1 \leq s \leq \sigma$, satisfying the conditions in step (2). Since the algorithm can evaluate every $Y_s \subset Y$, it must terminate in step (2) with the correct value of $\nu(coY)$ regardless of the order selected in step (1).

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 $1 \leq s \leq \sigma$. We need to account for this possibility in the next section. Thus we have added step (4).

Suppose we obtain (5.1) with $\nu = m+1$ and $Y_s = Y$. Then coY is a simplex and $\nu(coY) \in$ interior coY. Hence, $\nu(coY) = 0$ and there is a sphere of maximum radius, centered on the origin, contained in coY. The radius of this sphere is $d^-(\{0\}, coY)$ and it is given by the distance to the m dimensional face of coY which is closest to the origin. Hence,

$$d^-(\{0\}, coY) = \min\left\{ |\nu(\mathrm{aff} Y_s)| : Y_s \subset Y \text{ has } m \text{ elements} \right\}. \qquad (5.9)$$

From $y = \nu(\mathrm{aff} Y_s)$, (5.5) and (5.8),

$$|\nu(\mathrm{aff} Y_s)| = (\Delta(Y_s)^{-1} \sum_{i \in I_s} \Delta_i(Y_s) y_i \cdot y_k)^{\frac{1}{2}}, \quad k \in I_s . \tag{5.10}$$

The data $\Delta(Y_s)$, $\Delta_i(Y_s)$, $y_i \cdot y_k$ are all needed in the determination of $\nu(coY)$ and require no additional computational effort. Thus (5.10) is evaluated with only m multiplies and one divide. If $m = 3$, this means (5.9) takes 12 multiplies and 4 divides.

When the distance algorithm of Section 4 stops with $\nu_k = 0$ and $V_k = coY$, where Y has $m+1$ points, it is clear that $coY \subset K$ and $d^-(\{0\}, coY)$ in (5.9) is a lower bound in $d^-(\{0\}, K)$. The lower bound may be important in applications (see, e.g., Section 8) and is computationally inexpensive.

6. 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 round-off errors. This is followed by some comments on the efficient implementation of the algorithm.

Errors do not accumulate in our algorithm since at every iteration k, $\nu_k = \nu(coY_k)$ results from the explicit evaluation of formulas which are only dependent on the set Y_k. This helps the ultimate accuracy of the results and simplifies the error analysis.

Inner product evaluations are one source of error. When $K = coZ_1 - coZ_2$, $Z_i = \{z_{ij} : j = 1, \cdots, M_i\}$, $i = 1,2$, we 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

$$\rho_c = \frac{1}{2}(\bar{z}_1 + \bar{z}_2), \ \bar{z}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} z_{ij}, \ i = 1,2 . \tag{6.1}$$

Since $co\bar{Z}_1 - co\bar{Z}_2 = K$, all our previous notations apply to the transformed problem.

Other sources of error include the evaluation of the sums in (5.4), (5.6)-(5.7) and step 3 of the Distance Algorithm. To account for these errors and those from the inner products, it is reasonable to replace the convergence criterion in step (3) by

$$| \nu_k |^2 + h_K(-\nu_k) \leq \epsilon D^2(K) \tag{6.2}$$

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

$$D(K) = \max\{ | z | : z \in K \} . \tag{6.3}$$

Since ϵD^2 is very small, result (3) of the Theorem 4.1 shows that the effect on the accuracy of the final result should be small. If $K = coZ_1 - coZ_2$ as in the last paragraph, and the origin of the system is translated as indicated, then the upper bound on $D(K)$, given by

$$D(K) \leq D(coZ_1 - \{\overline{z}_1\}) + D(coZ_2 - \{\overline{z}_2\}) + | \overline{z}_1 - \overline{z}_2 | , \tag{6.4}$$

may be appropriately used in (6.2). When Z_1 and Z_2 are dependent on q (see the paragraph containing (2.9)), the first two terms in (6.4) are independent of q and may be computed from the w_{ij} which specify C_i, i=1,2.

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

Given $Y = \{y_1, \cdots, y_v\}$, the Backup Procedure determines $\nu(coY)$ by evaluating $\nu(\mathrm{aff}Y_s)$ 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 (5.1). The Backup Procedure merely picks the best of the Y_s and sets $\nu(coY) = \nu(\mathrm{aff}Y_s)$:

$$\nu(coY) = \text{argument } \min\Big\{ \mid y \mid : y = \nu(\text{aff}Y_s),\ Y_s \subset Y, \\ \Delta(Y_s) > 0,\ \Delta_j(Y_s) > 0,\ j \in I_s \Big\} \tag{6.5}$$

where $\mid \nu(\text{aff}Y_s) \mid$ is calculated using (5.10). In most cases (6.5) 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 $v \in \{1, \cdots, m+1\}$, perform the following steps:

(1) select $V_0 = \{y_1, \cdots, y_v\} \subset K$ and let $k = 0$;

(2) set $Y = V_k$ and apply the Distance Subalgorithm; if it succeeds set $alg = DS$, otherwise use the Backup Procedure (6.5) and set $alg = BP$; set $\nu_k = \nu(coY)$ and $\hat{V}_k = Y_s$ where Y_s satisfies (5.1);

(3) if (6.2) holds, set $\nu(K) = \nu_k$ and stop;

(4) if (k=0 or $\mid \nu_k \mid < \mid \nu_{k-1} \mid$) and ($\hat{V}_k$ has m elements or less), then let $V_{k+1} = \hat{V}_k \bigcup \{s_K(-\nu_k)\}$, increment k and proceed to step (2);

(5) if $alg = BP$, set $\nu(K) = \nu_k$, indicate the error tolerance (6.2) is not satisfied and stop;

(6) if $alg = DS$, recalculate $\nu_k = \nu(coV_k)$ using the Backup Procedure (6.5), set $alg = BP$ and proceed to step (3).

It is easy to see that the algorithm always terminates, even if $\epsilon = 0$. If ϵ is small but reasonable (say 100 $\times$ 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 inconsistant with theory. The condition, $\mid \nu_k \mid \geq \mid \nu_{k-1} \mid$, $k \geq 1$, 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 $\nu_k = 0$. But $\nu_k = 0$ contradicts the failure of (6.2) 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 produ a near point of high accuracy. This is in part due to the structure of the num cal algorithm. Theoretically, both the Distance Algorithm and the Backup P cedure produce affinely independent sets $\hat{V}_k$, and $s_K(-\nu_k)$ should be affin independent of $\hat{V}_k$. Thus, the V_k, $k \geq 1$, should be affinely independent. E if V_0 is affinely dependent, or V_k, $k \geq 1$, is nearly so, the Distance Algorit usually functions well. We have confirmed this independently of the numeri algorithm by extensive experimentation with the Distance Algorithm.

When K is the set difference of two polytopes it is not obvious how the tial set V_0 should be chosen. We have tested a variety of schemes. In absence of additional information about K such as that described in Section the single point initialization $V_0 = \{s_K(-\bar{z}_1 + \bar{z}_2)\}$ has worked as well as a Here, $\bar{z}_1 - \bar{z}_2$ is the direction between centroids (see (6.1)) and serves as a ro estimate of $\nu(K)$. Note that the initialization is easy to compute using the p cedures outlined in Section 3.

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

Another aid to efficiency is the choice of ordering in step (1) of the Dista Subalgorithm. The sets most likely to produce the near point should be put the beginning of the list. Some of the subsets of $Y = V_k$ have already b tested in $Y = V_{k-1}$, and they are put at the end of the list (essentially, they eliminated). We have also found it effective to put one face of coV_k at the h of the list. It is $Y_1 \subset Y = V_k = \hat{V}_{k-1} \bigcup \{s_K(-\nu_{k-1})\}$ such that $V_k = Y_1 \bigcup$ and y maximizes $y \cdot s_K(-\nu_{k-1})$ over all $y \in \hat{V}_{k-1}$. Our experiences indicates t coY_1 contains $\nu(coV_k)$ about 80% of the time. The complete description of ordering procedure is too lengthy for inclusion here.

7. Numerical Experiments

The algorithm described in the previous section has been programmed as a Fortran subroutine and applied to a large number of examples in three space. Figure 2 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 in two parallel planes (M_i=20, 40, 50, 60, 100, 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. 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 $|z|^2 + h_K(-z)$ was excellent; typical values were in the order of 10^{-10}.

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:

$$EF = (t_M N_M + t_A N_A + t_D N_D + t_C N_C)/(t_M + t_A), \qquad (7.1)$$

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 Figure 2 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 Figure 2.

The results can be summarized as follows. 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 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% at most. The fact that the effort is proportional to $M_i + M_j$ is most encouraging. In combinatoric procedures it is proportional to $M_i M_j$.

8. 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 (3.11). From the comments in Sections 3 and 4, it is reasonable to assume the z_{1i} and the z_{2i} are points taken respectively from the finite sets $Z_1(q(s))$ and $Z_2(q(s))$ which generate $K_1 = coZ_1$ and $K_2 = coZ_2$. If T is large the geometry changes only slightly in one time increment and it is likely that the elements from $Z_1(q((t+1)/T))$ and $Z_2(q((t+1)/T))$ which have the same indices as those in (3.11) for $s = t/T$ can be used in (3.11) when $s = (t+1)/T$. Thus, the algorithm is started at $s = (t+1)/T$ with $V_0 = \{z_{1i} - z_{2i}, i=1, \cdots, l\}$ where $z_{1i} \in Z_1(q((t+1)/T))$ and $z_{2i} \in Z_2(q((t+1)/T))$ have the same indices as the elements in (3.11) 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 (3.11). 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 6.

Figure 3 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, $K_A = K_6 \bigcup K_7$, 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.

Figure 4 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 $d_{6i} \leq d_{7i}$, 1, $\cdots$,5, so that $d_i = d_{6i}$, i=1, $\cdots$,5. The computational times are shown in Table 1 for both the special initialization described above and the single point initialization of Section 6. The improvement due to the special initialization is significant, and as expected, gets better as T increases. 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 seconds on a Symbolics 3600 computer.

When $d_{68} = 0$ or $d_{65} = 0$, we have tabulated the lower bounds on d^-_{68} and d^-_{65} produced by the algorithm. The absolute value of the negative distances in Figure 4 correspond to these lower bounds. It is not known how closely they estimate d^-_{68} and d^-_{65}, but they do determine that significant collision penetrations have occurred.

We have run a simple test problem where $d^-_{ij}(q(s))$ can be obtained analytically. The computed lower bounds range from good to poor, and are best when $d^-_{ij}(q(s))$ is not too large. Fortunately, this is the situation of greatest interest.

Table 1. CPU times (Harris 800) in seconds for the example of Figure 3.

Number of Intervals in Grid (T)	Time with Single Point Initialization	Time with Special Initialization	Ratio of Times
10	.22	.13	1.7
100	2.00	.69	2.9
1000	19.81	6.32	3.1

9. 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 in Section 2 that it is difficult to compute the translational distance d^- for intersecting objects. Our algorithm provides, with essentially no additional cost, a lower bound on d^-. The use of this lower bound in applications such as those just mentioned remains to be explored.

Finally, a few comments should be made about sets which aren't 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.

10. References

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

[2] R.O. Barr, E.G. Gilbert, "Some efficient algorithms for a class of abstract optimization problems arising in optimal control," *IEEE Trans. Automatic*

Control, vol. AC-14, pp. 640-652, 1969.

[3] J.W. Boyse, "Interference detection among solids and surfaces," *Communications of the ACM,* vol. 22, pp. 3-9, 1979.

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

[5] C.E. Buckley, L.J. Leifer, "A proximity metric for continuum path planning," *Proc. 9th International Joint Conf. Art. Intelligence,* pp. 1096-1102, 1985.

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

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

[8] J. Canny, "Collision detection for moving polyhedra," MIT Artificial Intelligence Lab. Report No. 806, 1984.

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

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

[11] D.P. Dobkin, D.G. Kirkpatrick, "A Linear algorithm for determining the separation of convex polyhedra," *Journal 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," *Proc. IEEE International Conf. Robotics and Automation,* pp. 536-541, 1985.

[13] H. Edelsbrunner, "On computing the extreme distances between two convex polygons," Report No 96, Tech. Univ. of Graz, Graz, Austria, 1982.

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

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

[16] E.G. Gilbert, S.S. Keerthi, D.W. Johnson, "A family of algorithms for determining the distance between convex sets," Report in preparation, Center for Research in Integrated Manufacturing, Univ. of Michigan, 1986.

[17] D.W. Johnson, *The Optimization of Robot Motion in the Presence of Obstacles,* University of Michigan Ph.D. Dissertation in preparation.

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

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal Robotics Research,* vol. 5, pp. 90-98, 1986.

[20] D.T. Lee, F.P. Preparata, "Computational geometry - a survey," *IEEE Trans. Computing,* vol. C-33, pp. 1072-1101, 1984.

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

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

[23] W. Meyer, "Distances between boxes: applications to collision detection and clipping," *IEEE International Conf. Robotics and Automation,* pp. 597-602, 1986.

[24] M. Orlowski, "The computation of the distance between polyhedra in 3-space," *SIAM Conf. Geometric Modelling 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 Univ. Press, Princeton, NJ, 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, M. Sharir, "On the piano movers problem I, the special case of a rigid polygonal body moving admidst polygonal barriers," *Comm. Pure Appl. Math.,* vol. 36, pp. 345-398, 1983.

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

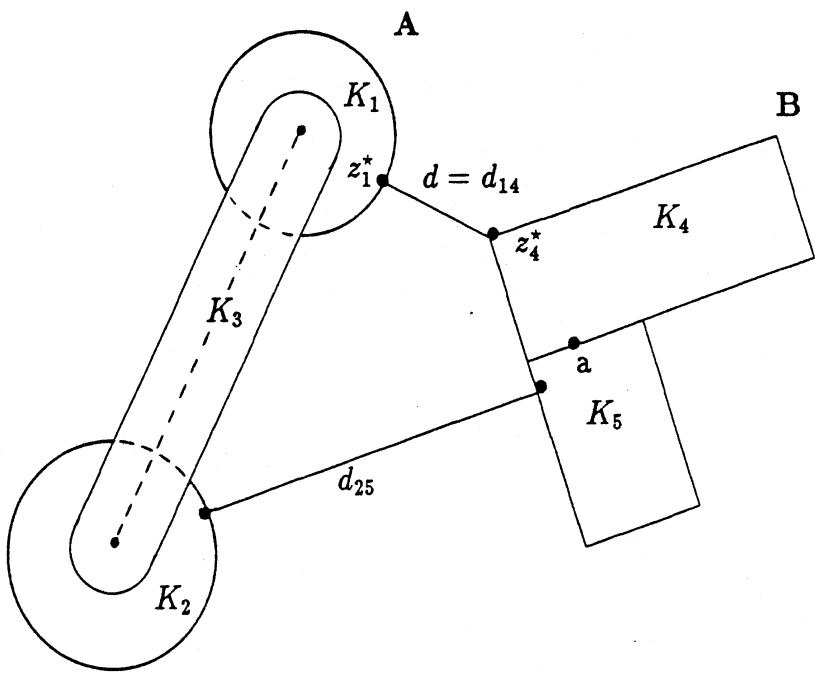

Figure 1 An example of object representation

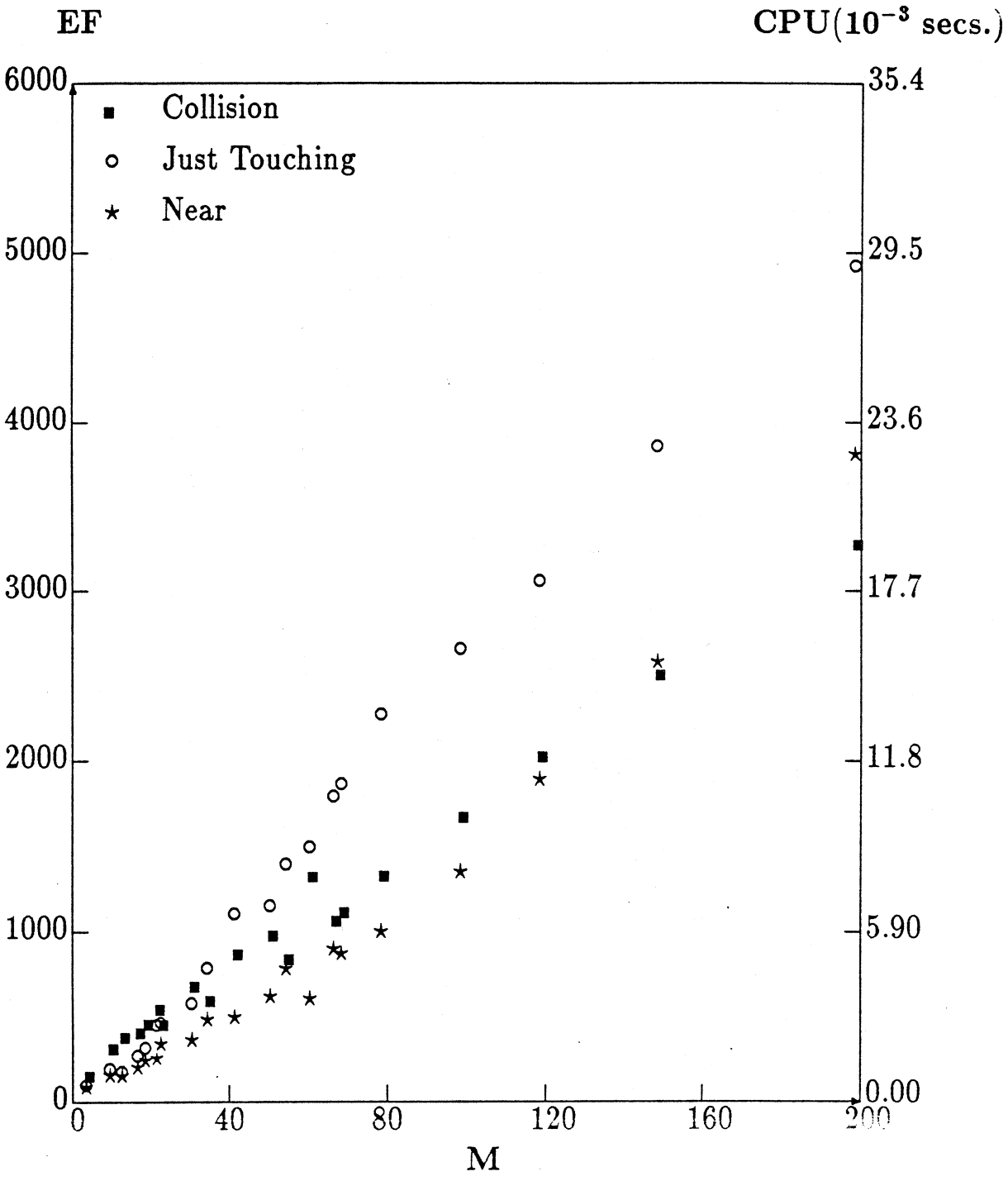

Figure 2 **Equivalent flops (EF) and CPU times vs. total number of vertices (M). Each data point is the average of 100 randomly generated examples.**

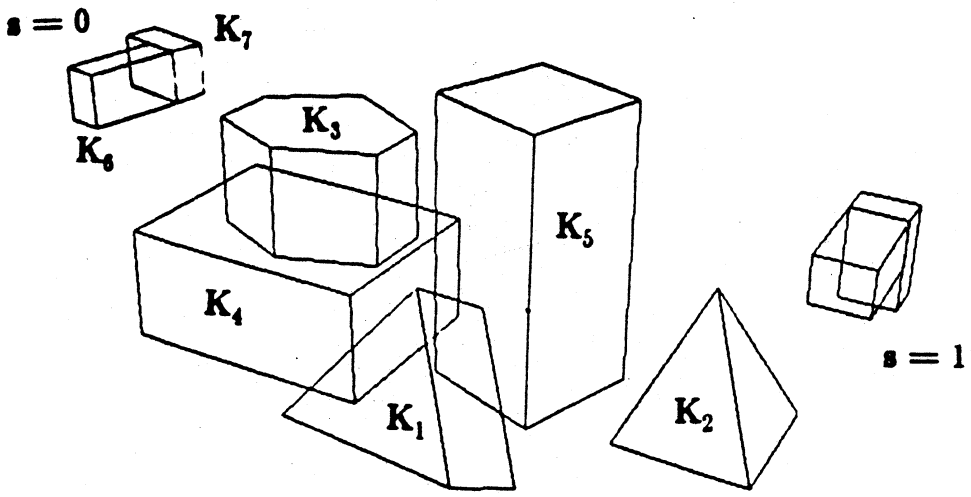

Figure 3. The example of collision detection.

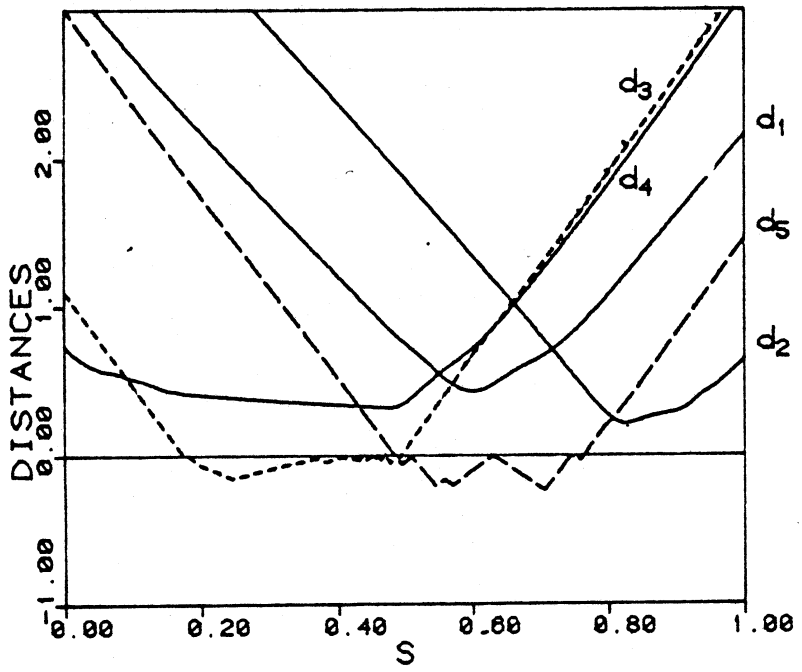

Figure 4. Results for the examples in Figure 3. The d_i are the distances between K_A and the K_i.