# Spatial Planning: A Configuration Space Approach

TOMÁS LOZANO-PÉREZ

***Abstract*—This paper presents algorithms for computing constraints on the position of an object due to the presence of other objects. This problem arises in applications that require choosing how to arrange or how to move objects without collisions. The approach presented here is based on characterizing the position and orientation of an object as a single point in a configuration space, in which each coordinate represents a degree of freedom in the position or orientation of the object. The configurations forbidden to this object, due to the presence of other objects, can then be characterized as regions in the configuration space, called *configuration space obstacles.* The paper presents algorithms for computing these configuration space obstacles when the objects are polygons or polyhedra.**

***Index Terms*—Computational geometry, obstacle avoidance, robotics.**

## I. Introduction

INCREASINGLY, computer applications deal with models of two- and three-dimensional objects. Partly because of this, there has been rapid growth of interest in efficient algorithms for geometric problems. For example, research has focused on algorithms for 1) computing convex hulls [16], [32], 2) intersecting convex polygons and polyhedra [6], [27], [36], [38], 3) intersecting half-spaces [11], [33] 4) decomposing polygons [35], and 5) closest-point problems [37].[1] Another class of geometric problems involves placing an object among other objects or moving it without colliding with nearby objects. We call this class of problems: *spatial planning* problems. The following are representative applications where spatial planning plays an important role:

1) the layout of templates on a piece of stock [1]–[3], [13] so as to minimize the area of stock required:

2) machining a part using a numerically controlled machine tool [50], which requires plotting the path of one or more cutting surfaces so as to produce the desired part;

3) the layout of an IC chip [48] to minimize area, subject to geometric design constraints;

4) automatic assembly using an industrial robot [22], [23], [43], which requires grasping objects, moving them without collisions, and ultimately bringing them into contact.

One common spatial planning problem is to determine where an object $A$ can be placed, inside some specified region $R$, so that it does not collide with any of the objects $B_j$ already

Manuscript received August 15, 1980; revised June 29, 1981 and June 15, 1982. This work was supported in part by the Office of Naval Research under Contract N00014-81-K-0334 and in part by the Defense Advanced Research Projects Agency under Office of Naval Research Contracts N00014-80-C-0505, N00014-82-K-0494.

The author is with the Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.

[1] The references cited here are representative of the current literature; they are by no means a complete survey.

placed there. We call this the **Findspace** problem. Finding where to place another suitcase in the trunk of a car is an example of Findspace, where the new suitcase is $A$, the previous suitcases are the $B_j$, and the inside of the trunk is $R$. A related problem is to determine how to move $A$ from one location to another without causing collisions with the $B_j$. We call this the **Findpath** problem. For example, moving the suitcase mentioned above from its initial position outside the trunk to the desired position in the trunk, requires computing a path for the suitcase (and the mover's arms) that avoids the rest of the car. These two geometric problems, Findspace and Findpath, are the subject of this paper. Previous work on Findspace and Findpath is surveyed in Section VIII.

Findspace and Findpath can be defined more formally as follows.

*Definition:* Let $R$ be an object that completely contains $k_B$ other, possibly intersecting, objects $B_j$.

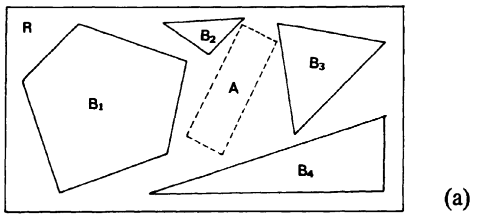
1) *Findspace*—Find a position for $A$, inside $R$, such that for all $B_j$, $A \cap B_j = \emptyset$. This is called a *safe position.*

2) *Findpath*—Find a path for $A$ from position $s$ to position $g$ such that $A$ is always in $R$ and all positions of $A$ on the path are safe. This is called a *safe path.*

Throughout this paper, the objects $R$ and $B_j$ are fixed convex polyhedra (or polygons). We take $A$ to be the set union of $k_A$ (possibly intersecting) convex polyhedra (or polygons) $A_i$. For example, $A$ may be a convex decomposition of a nonconvex polyhedron [35]. Fig. 1 illustrates the definitions of Findspace and Findpath for convex polygons.

The algorithm presented here for the Findspace and Findpath problems has two main steps: 1) building a data structure that captures the geometric constraints and 2) searching the data structure to find the solution. In this paper we focus on algorithms for constructing the appropriate data structure. In this sense, the approach is similar to many geometric search algorithms, for example, the Voronoi polygon approach to closest-point problems [37]. In the Findspace and Findpath algorithms described here, we build geometric objects, called *configuration space obstacles*, that represent all the positions of the object $A$ that cause collisions with the $B_j$. Given these objects, Findspace and Findpath correspond to the simpler problems of finding a single point (a position of $A$) or a path (a sequence of positions of $A$), outside of the configuration space obstacles. The advantage of this formulation is that the intersection of a point relative to a set of objects is easier to deal with than the intersection of objects among themselves.

Representing the positions of rigid objects requires specifying all their degrees of freedom, both translations and rotations. We will use the notion of configuration to unify our treatment of degrees of freedom. The *configuration* of a polyhedron is a set of independent parameters that characterize the position of every point[2] in the object. The configuration of



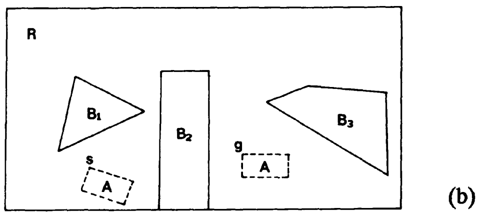


Fig. 1. $R$, $B_j$, and $A$ for Findspace and Findpath problems in two dimensions. (a) The Findspace problem is to find a configuration for $A$ where $A$ does not intersect any of the $B_j$. (b) The Findpath problem is to find a path for $A$ from $s$ to $g$ that avoids collisions with the $B_j$.

a polyhedron is defined relative to an initial configuration. In this initial configuration, by convention, a fixed vertex of the polyhedron coincides with the origin of the global coordinate frame. For a polyhedron $A$, this vertex is called the *reference vertex of* $A$, or $rv_A$.

The number of parameters required to specify the configuration of a $k$-dimensional polyhedron, $A$, relative to its initial configuration, is $d$, where $d = k + \binom{k}{2}$ [7, p. 10]; $k$ parameters are required to specify the position of $rv_A$ in $\mathcal{R}^k$ and $\binom{k}{2}$ are required to specify the orientation[3] of $A$. Thus, the configuration of $A$ can be regarded as a point $x \in \mathcal{R}^d$; this $d$-dimensional space of configurations of $A$ is denoted $Cspace_A$. $A$ in configuration $x$ is $(A)_x$; $A$ in its initial configuration is $(A)_0$. When an object's configuration is fixed, e.g., the $B_j$ mentioned earlier, we leave it unspecified.

If $A$ is a polygon in $\mathcal{R}^2$, the configuration of $A$ is specified by $(x, y, \theta)$, where $(x, y)$ is the position of $rv_A$ and $\theta$ is the rotation of $A$, about $rv_A$, relative to $(A)_0$. That is, for polygons in $\mathcal{R}^2$, $k = 2$, configurations are elements of $\mathcal{R}^3$, $d = 2 + 1$. If the orientation of $A$ is fixed, $(x, y)$ alone is sufficient to specify the polygons configuration; therefore, $Cspace_A$ is simply the $(x, y)$ plane. If $A$ is a polyhedron in $\mathcal{R}^3$, $k = 3$, the configurations of $A$ are elements of $\mathcal{R}^6$, $d = 3 + 3$. That is, three translations and three rotations are needed to specify the position and orientation of a rigid three-dimensional object [7].

Not all possible configurations in $Cspace_A$ represent legal configurations of $A$; in particular, configurations of $A$ where $A \cap B_j \neq \emptyset$ are illegal because they would cause collisions.

[2] In what follows, all geometric entities—points, lines, edges, planes, faces, and objects—will be treated as (infinite) sets of points. All of these entities will be in some $\mathcal{R}^n$, an $n$-dimensional real Euclidean space. $a$, $b$, $x$, and $y$ shall denote points of $\mathcal{R}^n$, as well as the corresponding vectors. $A$, $B$, and $C$ shall denote sets of points in $\mathcal{R}^n$, while $I$ and $K$ shall denote sets of integers. $\gamma$, $\theta$, and $\beta$, shall denote reals, while $i, j, k, l, m, n$ shall be used for integers. The coordinate representation of a point $c \in \mathcal{R}^n$, shall be $c = (\gamma_i) = (\gamma_1, \cdots, \gamma_n)$.

[3] The relative rotation of one coordinate system relative to another can be specified in terms of $\binom{k}{2}$ angles usually referred to as Euler angles [7]. These angles indicate the magnitude of three successive rotations about specified axes. Many conventions for the choice of axes exist, any of which is suitable for our purposes.

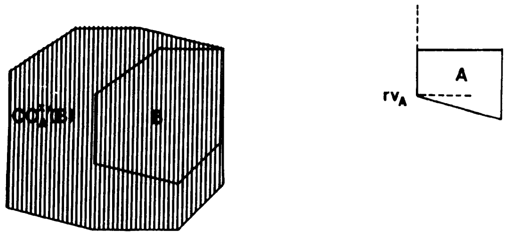


Fig. 2. The $Cspace_A$ obstacle due to $B$, for fixed orientation of $A$.

These illegal configurations are the result of a mapping of the $B_j$ into $Cspace_A$. This mapping exploits two fundamental properties of objects: 1) their *rigidity*, which allows their configurations to be characterized by a few parameters and 2) their *solidity*, which requires that a point not be inside more than one object.

*Definition:* The $Cspace_A$ obstacle due to $B$, denoted $CO_A(B)$, is defined as follows:

$$CO_A(B) \equiv \{x \in Cspace_A \,|\, (A)_x \cap B \neq \varnothing\}.$$

Thus, if $x \in CO_A(B)$ then $(A)_x$ intersects $B$, therefore $x$ is not safe. Conversely, any configuration $x \notin CO_A(B_j)$ (for all objects $B_j$) is safe. If $A$ is a convex polygon with fixed orientation, the presence of another convex polygon $B$ constrains the configuration of $A$, in this case simply the position of $rv_A$, to be outside of $CO_A(B)$, a larger convex polygon, shown as the shaded region in Fig. 2. The choice of a different vertex as $rv_A$ would result in translating $CO_A(B)$ relative to $B$ in the figure.

Just as $CO_A(B)$ defines those configurations for which $A$ intersects $B$, $CI_A(B)$ defines those configurations for which $A$ is completely inside $B$.

*Definition:* The $Cspace_A$ *interior* of $B$, denoted $CI_A(B)$, is defined as follows:

$$CI_A(B) \equiv \{x \in Cspace_A \,|\, (A)_x \subseteq B\}.$$

Clearly, $CI_A(B) \subseteq CO_A(B)$. Moreover, it is easy to see that for $A$ to be inside $B$, it must be outside of $B$'s complement. Therefore, letting $-X$ represent the complement of the set $X$, $CI_A(B) = -CO_A(-B)$.

A superscript to $CO_A(B)$ and $CI_A(B)$ will be used to indicate the coordinates of the configurations in the sets, e.g., $CO_A^{xy}(B)$ and $CO_A^{xy\theta}(B)$ denote sets of $(x, y)$ and $(x, y, \theta)$ values, respectively. When no superscript is used, as in $CO_A(B)$, we mean sets of configurations in the complete $Cspace_A$ for a polyhedron of $A$'s dimension, e.g., $\mathcal{R}^6$ for a three-dimensional polyhedron.

Using the definitions of *Cspace* obstacle and *Cspace* interior, Findspace and Findpath can be expressed as equivalent problems that involve placing one point, the configuration of $A$, relative to the $Cspace_A$ objects $CO_A(B_j)$ and $CI_A(R)$. In general, these problems are equivalent to finding either a single configuration of $A$ or a connected sequence of configurations of $A$ (a path), outside all of the $CO_A(B_j)$, but inside $CI_A(R)$

If $A$ and all of the $B_j$ are polygons and if the orientation of $A$ is fixed, then the $CO_A^{xy}(B_j)$ are also polygons. In that case, the shortest[4] safe paths for $A$ are piecewise linear paths connecting the start and the goal configurations via the vertices of the $CO_A^{xy}(B_j)$ polygons; see Fig. 3. Therefore, Findpath can be formulated as a graph search problem. The graph is formed by connecting all pairs of $CO_A^{xy}(B_j)$ vertices (and the start and goal) that can "see" each other, i.e., can be connected by a straight line that does not intersect any of the obstacles. The shortest path from the start to the goal in this *visibility graph* (**Vgraph**) is the shortest safe path for $A$ among the $B_j$ [24]. This algorithm solves two-dimensional Findpath problems when the orientation of $A$ is fixed, but the paths it finds are very susceptible to inaccuracies in the object model. These paths touch the $Cspace_A$ obstacles; therefore, if the model were exact, an object moving along this type of path would just touch the obstacles. Unfortunately, an inaccurate model or a slight error in the motion may result in a collision. Furthermore, the Vgraph algorithm does not find optimal paths among three-dimensional obstacles [24]. Alternative techniques for path-finding are treated in [23].

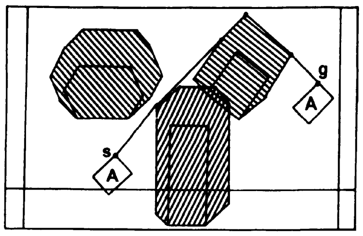


Fig. 3. The Findpath problem and its formulation using the $CO_A^{xy}(B_j)$. The shortest collision-free paths connect the origin and the destination via the vertices of the $CO_A^{xy}(B_j)$ polygons.

Here is a brief summary of the rest of the paper. Section II presents algorithms for computing $CO_A^{xy}(B)$. Section III characterizes $CO_A^{xy\theta}(B)$, the $Cspace_A$ obstacle for polygons that are allowed to rotate. Section IV describes an algorithm for computing $CO_A^{xyz}(B)$, the $Cspace_A$ obstacle for polyhedra with fixed orientation. Section V characterizes $CO_A(B)$, the $Cspace_A$ obstacle for polyhedra that are allowed to rotate. Section VI deals with slice projection, an approximation technique for higher dimensional $Cspace_A$ obstacles, for example, those obtained when a polyhedron is allowed to rotate. Section VII discusses the extensions to the Findspace and Findpath algorithms needed to plan the motions of industrial robots. Section VIII discusses related work in spatial planning.

## II. Computing $CO_A^{xy}(B)$

The crucial step in the *Cspace* approach to Findspace and Findpath is computing the $Cspace_A$ obstacles for the $B_j$. Thus far, we have only provided an implicit definition of $CO_A(B)$; we now provide, in Theorem 1, a characterization of $CO_A^{xy}(B)$ and $CI_A^{xy}(B)$ in terms of set sums that will lead us to an efficient algorithm for computing $Cspace_A$ obstacles.

*Set sum, set difference*, and *set negation* are defined on sets of points, eqivalently vectors, in $\mathcal{R}^n$ as follows:

[4] This assumes Euclidean distance as a metric. For the optimality conditions using a rectilinear (Manhattan) metric, see [19].

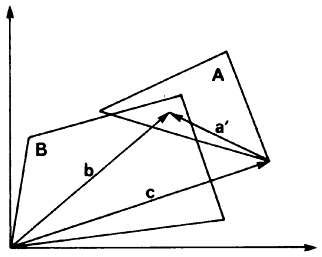


Fig. 4. Illustration of Theorem 1. Any location of $rv_A$, denoted $c$, for which $A$ and $B$ have a point in common (expressible as $b$ and $a'$), can be expressed as $c = b - a'$. Therefore, $CO^{xy}_A(B) = B \ominus (A)_0$.

$$A \oplus B = \{a + b \mid a \in A, b \in B\}$$
$$A \ominus B = \{a - b \mid a \in A, b \in B\}$$
$$\ominus A = \{-a \mid a \in A\}.$$

If a set $A$ consists of a single point $a$, then $a \oplus B = \{a\} \oplus B = A \oplus B$. Also, $A \ominus B = A \oplus (\ominus B)$. Note that, typically $A \oplus A \neq \{2a \mid a \in A\}$ and $A \ominus A \neq \emptyset$, although $A \oplus B = B \oplus A$.

We can characterize the $Cspace_A$ obstacle for objects with fixed orientation as a set difference of the objects' point sets:

*Theorem 1:* For $A$ and $B$, sets in $\mathcal{R}^2$, $CO^{xy}_A(B) = B \ominus (A)_0$.

*Proof:* If $c$ is an $(x, y)$ configuration of $A$, then $(A)_c = c \oplus (A)_0$. Therefore, if $a \in (A)_c$, then there exists $a' \in (A)_0$ such that $a = c + a'$; see Fig. 4. Thus, if $b \in B \cap (A)_c$, and therefore $b \in (A)_c$, then, for some $a' \in (A)_0$, $b = c + a'$ and therefore $c = b - a'$. Clearly, the converse is also true. ■

This theorem extends naturally to higher dimensions, e.g., $CO^{xyz}_A(B)$, as long as the orientation of $A$ is fixed.

If $A$ and $B$ are convex, then $A \oplus B$ and $A \ominus B$ are also convex [16, p. 9]; therefore $CO^{xy}_A(B)$ is convex. In fact, if $A$ and $B$ are convex polygons, then $CO^{xy}_A(B)$ is also a convex polygon. We can now use well-known properties of convex polygons and the set sum operation [5] to state an $O(n)$ algorithm for computing $CO^{xy}_A(B)$ when $A$ and $B$ are convex $n$-gons.

Let $\pi(A, u)$ denote the *supporting line* [5] of $A$ with outward normal $u$. All of $A$ is in one of the closed half-spaces bounded by $\pi(A, u)$ and $u$ points away from the interior of $A$. $\pi(A, u) \cap A$ is the set of boundary points of $A$ on the supporting line.

*Lemma 1* [5]: If $A$ and $B$ are convex sets and $u$ is an arbitrary unit normal, then

$$\pi(A \oplus B, u) \cap (A \oplus B) = (\pi(A, u) \cap A) \oplus (\pi(B, u) \cap B). \quad (1)$$

Fig. 5 illustrates this lemma.

*Lemma 2* [5]:

a) Let $s(a_1, a_2)$ be a line segment and $b$ a point, then $s(a_1, a_2) \oplus b = s(a_1 + b, a_2 + b)$ is a line segment parallel to $s(a_1, a_2)$ and of equal length. See Fig. 6(a).

b) Let $s(a_1, a_2)$ and $s(b_1, b_2)$ be parallel line segments such that $(a_2 - a_1) = k(b_2 - b_1)$ for $k > 0$. Then $s(a_1, a_2) \oplus s(b_1, b_2) = s(a_1 + b_1, a_2 + b_2)$ and the length of the sum is the sum of the lengths of the summands. See Fig. 6(b).

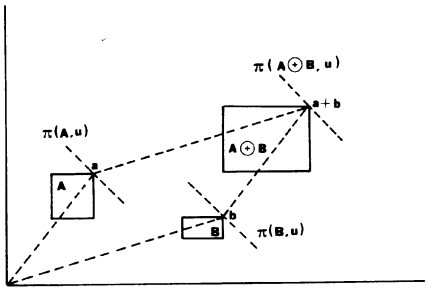


Fig. 5. Illustration for Lemma 1.

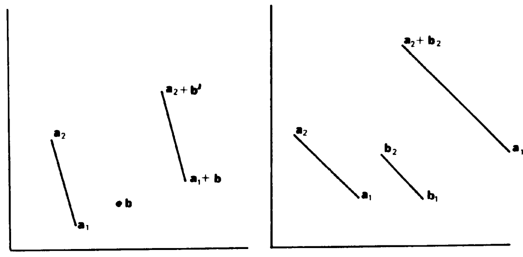


Fig. 6. Illustration for Lemma 2.

*Theorem 2:* For $A$ a convex $n$-gon and $B$ a convex $m$-gon, $CO^{xy}_A(B)$ can be computed in time $O(n + m)$.

*Proof:* For a polygon $P$, assume the $j$th edge, $s(v_j, v_{j+1})$, makes the angle $\theta_j$ with the $x$-axis. If $\theta(u)$ is the angle $u$ makes with the $x$ axis, then

$$\pi(P, u) \cap P = \begin{cases} v_j, & \text{if } \theta_{j-1} < \theta(u) < \theta_j \\ s(v_j, v_{j+1}), & \text{if } \theta(u) = \theta_j \\ v_{j+1}, & \text{if } \theta_j < \theta(u) < \theta_{j+1}. \end{cases}$$

We now apply Lemmas 1 and 2. Depending on the angle $\theta(u)$, each term on the right-hand side of (1) is either a line segment (edge) or a single point (vertex). It follows from Lemma 2 that the term on the left of (1) is one of:

a) a new vertex, when two vertices are combined;

b) a displaced edge, when an edge and a vertex are combined (Lemma 2a);

c) an edge, corresponding to a pair of displaced end-to-end edges, when two edges are combined (Lemma 2b).

As $u$ rotates counterclockwise, the boundary of $A \oplus B$ is formed by joining a succession of these elements. Note that, because of the convexity of $A$ and $B$, each edge is encountered exactly once [25, p. 13].

Polygons are stored as lists of vertices in the same order as they are encountered by the counterclockwise sweep of $u$. This is equivalent to a total order on the edges, based on the angle that the edge makes with the $x$ axis. These lists for $A$ and $B$ can be merged into a single total order on the angle in linear time, as they are traversed. At each step, we construct a new vertex (edges need not be represented explicitly) by the method indicated in the lemmas. The time for constructing the new vertices is bounded by a constant, since it involves at most two

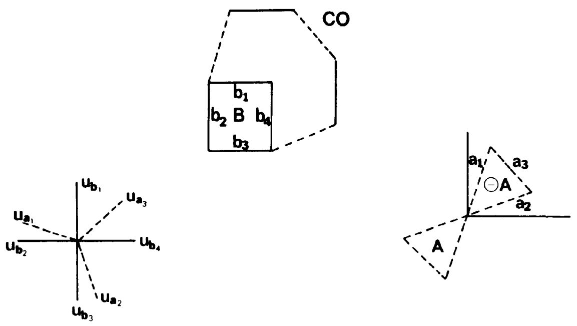


Fig. 7. The edges of $B \ominus (A)_0$, when $A$ and $B$ are convex polygons, are found by merging the edge lists of $B$ and $\ominus(A)_0$, ordered on the angle their normals make with the positive $x$ axis.

vector additions. Thus $A \oplus B$ can be computed in linear time during a scan of the vertices of $A$ and $B$; see Fig. 7. An implementation of this algorithm is included in Appendix I. Thus, $B \ominus (A)_0$ can be computed in linear time by first converting each vertex $a_i$ to $rv_A - a_i$; see Fig. 7. ■

This algorithm is similar to Shamos' diameter algorithm using antipodal pairs [36], but instead of dealing with two supporting lines on one polygon, it deals with two polygons and one supporting line on each. An algorithm, essentially identical to the one in Theorem 2, has been recently described in [39]. The proof Theorem 2, however, will be used in subsequent sections to derive characterizations and algorithms for other *Cspace* entities.

When $A$ or $B$ are nonconvex polygons, $CO_A^{xy}(B)$ can be computed by an extension of the algorithm above. The extension relies on decomposing the boundaries of the polygons into a sequence of polygonal arcs whose internal angles, i.e., the angle facing the inside of the polygon, are each less than $\pi$. The algorithm of Theorem 2 can then be applied to pairs of arcs; the result is a polygon whose boundary, in general, intersects itself. The algorithm requires, in the worst case, $O(n \times m)$ steps.

An alternative method of computing $CO_A^{xy}(B)$ for nonconvex $A$ and $B$ can be used when convex decompositions of $A$ and $B$ are available, e.g., the objects may have been designed by set operations on convex primitives. If $A$ is represented as the union of $k_A$ objects $A_i$, and $B$ is the union of $k_B$ objects $B_j$, then Theorem 3 follows directly from the definition of $CO_A(B)$.

*Theorem 3:* If $A = \bigcup_{i=1}^{k_A} A_i$ and $B = \bigcup_{j=1}^{k_B} B_j$:

$$CO_A(B) = \bigcup_{i=1}^{k_A} \bigcup_{j=1}^{k_B} CO_{A_i}(B_j).$$

This theorem simply says that the set of configurations that cause a collision between $A$ and $B$ are those for which any part of $A$ intersects any part of $B$.

## III. Characterizing $CO_A^{xy\theta}(B)$

We have so far restricted our attention to cases where $A$ remains in a fixed orientation. In these cases, all the geometric constraints for spatial planning are embodied in $CO_A^{xy}(B)$. However, $CO_A^{xy}(B)$ is only the cross section, for fixed $\theta$, of the three-dimensional full configuration space obstacle for polygons, $CO_A^{xy\theta}(B)$. In this section, we consider $CO_A^{xy\theta}(B)$, when

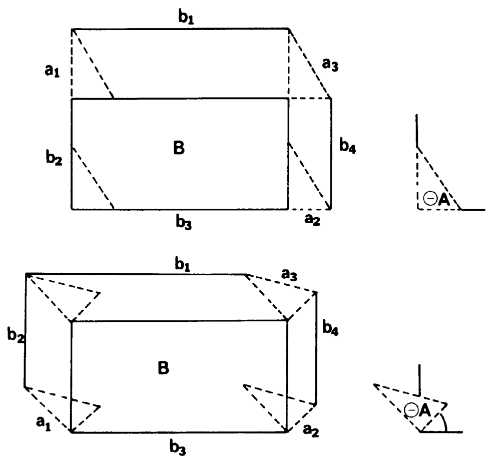


Fig. 8. When $A$ (and $\ominus(A)_0$) rotates by $\theta$, the $e_i^a$ rotate around $b_j$ and the $e_j^b$ are displaced. When an $e_i^a$ is aligned with an $e_j^b$ for some $\theta$, any additional rotation of $A$ will interchange the order in which they are encountered during the counterclockwise scan of Theorem 2. For example, in the top figure $a_1$ appears before $b_2$ on the boundary of $CO_A^{xy}(B)$ and is (nearly) aligned with $b_2$; in the bottom figure, after additional rotation of $A$, $a_1$ appears after $b_2$ on the boundary of $CO_A^{xy}(B)$. Therefore, the top figure illustrates $A$ in the orientation $\theta_{i,j}^*$.

$A$ and $B$ are convex, by examining changes in its cross section as $\theta$ changes.

For fixed $\theta$, we know from Theorem 2 that the edges of $CO_A^{xy}(B)$ are either displaced edges of $A$ or displaced edges of $B$. Therefore, for a small rotation of $A$, we expect that those edges of $CO_A^{xy}(B)$ corresponding to the edges of $A$ will rotate, but the edges corresponding to edges in $B$ will not. As $A$ rotates, however, the rate of displacement of these edges changes discontinuously when edges of $A$ and $B$ become parallel, as illustrated below.

Let $vert(B)$ denote the set of vertices of a polygon $B$, $b_j$ be the position vector of the $j$th member of $vert(B)$, and $a_i(\theta)$ be the position of the $i$th member of $vert(\ominus(A)_0)$, which depends on $\theta$. Assume that $A$ and $B$ have no parallel edges. For fixed $\theta$, the proof of Theorem 2 shows that each edge of $CO_A^{xy}(B)$ can be expressed as one of

$$e_i^a = b_j \oplus s(a_i(\theta), a_{i+1}(\theta)) \tag{2a}$$

$$e_j^b = a_i(\theta) \oplus s(b_j, b_{j+1}). \tag{2b}$$

The order in which the $a_i$ and $b_j$ are encountered in the counterclockwise scan described in Theorem 2 determines the $(i, j)$ pairings of vertices and edges. For example, in (2b), $a_i(\theta)$ is the vertex of $\ominus(A)_0$ that is on the supporting line of $\ominus(A)_0$ which is parallel to $s(b_j, b_{j+1})$, i.e., if $u_j$ is the normal to $s(b_j, b_{j+1})$, then $a_i = \pi(\ominus(A)_0, u_j) \cap \ominus(A)_0$.

Equation (2) shows that, for a given pairing of edges and vertices, the $e_i^a$ rotate around $b_j$, while the $e_j^b$ are simply displaced by the vector $a_i(\theta)$; see Fig. 8. The discontinuous changes occur at values of $\theta$, denoted $\theta_{i,j}^*$, where the $i$th edge of $A$ becomes parallel to the $j$th edge of $B$. For values of $\theta$ just greater than these $\theta_{i,j}^*$, some pair of edges has a different order in the scan of Theorem 2 from what they had when $\theta$ was just less than $\theta_{i,j}^*$; see Fig. 8. Therefore, at each $\theta_{i,j}^*$, the pairings between edges and vertices change. For $A$ a convex $n$-gon and

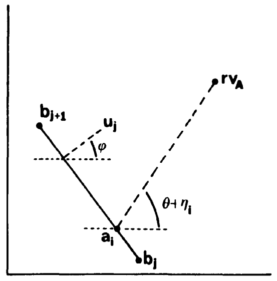


Fig. 9. Illustration of terms used in (3).

$B$ a convex $m$-gon, there are $O(n \times m)$ such $\theta^*_{ij}$ in $CO^{xy\theta}_A(B)$.

Between discontinuities, the lines defined by $e^b_j$ edges have a simple dependence on $\theta$. The edge $s(b_j, b_{j+1})$ is on a line whose vector equation[5] is: $\langle u_j, x\rangle = \langle u_j, b_j\rangle$ where $u_j$ is the constant unit normal to $s(b_j, b_{j+1})$. Let $a_i(\theta)$ make the angle $\theta + \eta_i$ with the $x$ axis, with $\eta_i$ constant, and $u_j$ make the angle $\phi_j$ with the $x$ axis. Then, if $\|a_i\|$ represents the vector magnitude of $a_i$, the equation for the line including $e^b_j$ is

$$\langle u_j, x\rangle = \langle u_j, a_i(\theta) + b_j\rangle$$
$$= \|a_i\| \cos(\theta + \eta_i - \phi_j) + \langle b_j, u_j\rangle. \quad (3)$$

The terms are illustrated in Fig. 9. This equation holds between discontinuities.

The equation for the $e^a_i$ edges is not as simple, because the orientation of the edge changes with $\theta$, i.e., the edge is a cross section of a curved surface in $(x, y, \theta)$ space. Let $v_i(\theta)$ be the normal vector to $s(a_i(\theta), a_{i+1}(\theta))$; then the vector equation of this curved surface is

$$\langle v_i(\theta), x\rangle = \langle v_i(\theta), a_i(\theta) + b_j\rangle. \quad (4)$$

This equation also holds only between discontinuities, i.e., for each pairing of vertices and edges.

Equations (3) and (4) define the shape of $CO^{xy\theta}_A(B)$. Since the resulting object is not polyhedral, however, it cannot be used for the Vgraph algorithm. Section VI discusses a technique for constructing lower dimension polyhedral approximations of $Cspace_A$ obstacles and an extended Vgraph algorithm to use them (see also [23]). These equations for $CO^{xy\theta}_A(B)$ are the basis for the Findpath algorithm described in [10].

## IV. Algorithm for $CO^{xyz}_A(B)$

Theorem 1 applies also to $CO^{xyz}_A(B)$, but the algorithm of Theorem 2 cannot be extended to polyhedra, since there is no similar total ordering of the faces of a polyhedron. However, Theorem 4 below follows easily from Theorem 1 and provides a way to compute $CO^{xyz}_A(B)$ for convex polyhedra $A$ and $B$. The method of Theorem 4 also applies to polygons, but is much less efficient than the linear algorithm of Theorem 2. Theorem 4 provides the basis for approximating $CO^{xyz}_A(B)$ when $A$ and $B$ are nonconvex, simply by replacing $A$ or $B$ by their convex hulls.

Let $conv(A)$ denote the *convex hull* of a polygon $A$, i.e., the smallest convex polygon enclosing $A$. We know that $conv(A)$, for a nonempty set $A \subseteq \mathcal{R}^d$, is $\{\sum_{i=1}^n \gamma_i x_i \mid x_i \in A, \gamma_i \geq 0, \sum_{i=1}^n \gamma_i = 1\}$, for some $n$ [16, p. 15]. This definition says that every point in the convex hull of $A$ can be written as a convex linear combination of points in $A$.

*Theorem 4:* For polyhedra $A$ and $B$,

$$conv(A \oplus B) = conv(A) \oplus conv(B)$$
$$= conv(vert(A) \oplus vert(B)).$$

*Proof:* First show that $conv(A \oplus B) = conv(A) \oplus conv(B)$.

($\supseteq$): By the definition of $\oplus$, if $x \in conv(A) \oplus conv(B)$, then there exist $a \in conv(A)$ and $b \in conv(B)$ such that $x = a + b$. The definition of convex hull states that any $a \in conv(A)$ can be expressed as a convex linear combination of points in $A$; likewise for any $b \in conv(B)$. Therefore, there exist $a_i \in A, b_i \in B, \sum_i \gamma_i = 1, \gamma_i \geq 0, \sum_j \beta_j = 1$, and $\beta_j \geq 0$ such that $a = \sum_i \gamma_i a_i$ and $b = \sum_j \beta_j b_j$ and thus

$$x = a + b = \left(\sum_i \gamma_i a_i\right) + b = \sum_i \gamma_i (a_i + b)$$
$$= \sum_i \gamma_i \left(a_i + \sum_j \beta_j b_j\right)$$
$$= \sum_i \gamma_i \left(\sum_j \beta_j a_i + \sum_j \beta_j, b_j\right)$$
$$= \sum_i \gamma_i \sum_j \beta_j (a_i + b_j) = \sum_i \sum_j \gamma_i \beta_j (a_i + b_j).$$

But, since $\sum_i \sum_j \gamma_i \beta_j = 1$ and $\gamma_i \beta_i \geq 0$, $x$ is a convex linear combination of points in $A \oplus B$ and therefore belongs to its convex hull. Therefore $conv(A) \oplus conv(B) \subseteq conv(A \oplus B)$.

($\subseteq$): If $x \in conv(A \oplus B)$, then there exist $\gamma_i \geq 0$ with $\sum_i \gamma_i = 1$, $a_i \in A$, and $b_i \in B$, such that

$$x = \sum_i \gamma_i (a_i + b_i) = \sum_i \gamma_i a_i + \sum_i \gamma_i b_i.$$

Therefore, $x \in conv(A) \oplus conv(B)$.

This establishes that $conv(A \oplus B) = conv(A) \oplus conv(B)$. Replacing $A$ by $vert(A)$ and $B$ by $vert(B)$, and using the fact that $conv(A) = conv(vert(A))$ [16], shows that $conv(vert(A) \oplus vert(B)) = conv(A) \oplus conv(B)$. ∎

*Corollary:* For convex polyhedra $A$ and $B$, $CO^{xyz}_A(B) = conv(vert(B) \ominus vert((A)_0))$.

*Proof:* $A \oplus B$ is convex, when $A$ and $B$ are both convex; thus, $A \oplus B = conv(A \oplus B)$. By Theorem 4, $A \oplus B = conv(vert(A) \oplus vert(B))$. Using Theorem 1 establishes the corollary. ∎

Several algorithms exist for finding the convex hull of a finite set of points on the plane, e.g., [15] and [32]. The latter [32] also describes an efficient algorithm for points in $\mathcal{R}^3$. These algorithms are known to run in worst case time $O(v \log v)$, where $v$ is the size of the input set. Therefore, Theorem 4 leads immediately to an algorithm for computing $CO^{xyz}_A(B)$ and an upper bound on the computational complexity of the problem or convex polyhedra.

*Theorem 5:* For convex polyhedra $A, B \subseteq \mathcal{R}^3$, each with

[5] The *scalar (dot) product* of vectors $a$ and $b$ will be denoted $\langle a, b\rangle$.

$O(n)$ vertices, $CO_A^{xyz}(B)$ can be computed in time $O(n^2 \log n)$.

*Proof:* The set $vert(B) \ominus vert((A)_0)$ is of size $O(n^2)$. Applying an $O(v \log v)$ convex hull algorithm to this set gives an $O(n^2 \log n)$ algorithm for computing $CO_A^{xyz}(B)$. ∎

The Vgraph algorithm discussed in Section I can be extended directly to deal with three-dimensional $Cspace_A$ obstacles, $CO_A^{xyz}(B_j)$. However, the paths found are not, in general, optimal paths [24]. Furthermore, with three-dimensional obstacles, the Vgraph algorithm is not even guaranteed to find a solution when one exists. This happens when the vertices of the $CO_A^{xyz}(B_j)$ are inaccessible, because they are outside of $CI_A^{xyz}(R)$. In that case, there may exist collision-free paths (via edges of the $CO_A^{xyz}(B_j)$), but the Vgraph algorithm will not find them. An alternative suboptimal, but complete, path searching strategy is described in [23]. A path searching algorithm based on mathematical optimization of a path along a fixed set of edges is described in [12].

## V. Characterizing $CO_A(B)$

The surfaces of $CO_A(B)$, when $A$ is three-dimensional and allowed to rotate, can be characterized in the same manner as the surfaces of $CO_A^{xy\theta}(B)$ were characterized in Section III. There are three types of surfaces that need to be considered, rather than two types as in two-dimensional objects. Let $f_i(\Theta)$ be the $i$th face of the convex polyhedron $A$, with $\Theta$ being the vector of three Euler angles indicating the orientation of $A$ relative to its initial orientation. Similarly, let $g_j$ be the $j$th face of the convex polyhedron $B$. As before, we let $a_i(\Theta)$ and $b_j$ denote vertices of $A$ and $B$, respectively. Each face of $CO_A(B)$ can be expressed as one of

$$f_i^a = b_j \oplus f_i(\Theta) \tag{5a}$$

$$f_j^b = a_i(\Theta) \oplus g_j \tag{5b}$$

$$f_{i,j}^{a \times b} = s(a_i(\Theta), a_{i+1}(\Theta)) \oplus s(b_j, b_{j+1}). \tag{5c}$$

The faces defined by (5a) and (5b) are parallel to the faces of $A$ and $B$, respectively. Each face defined by (5c) is a parallelogram, with edges parallel to the edges of $A$ and $B$ that give rise to the face. The vector equation for each type of surface follows the pattern of (3) and (4) above:

$$\langle N, x \rangle = \langle N, a_i(\Theta) + b_j \rangle \tag{6}$$

where $N$ is 1) the normal to $f_i(\Theta)$ for (5a) faces, 2) the normal to $g_j$ for (5b) faces, or 3) the cross product of the vectors along $s(a_i(\Theta), a_{i+1}(\Theta))$ and $s(b_j, b_{j+1})$ for (5c) faces. As above, this characterization only holds between discontinuities.

## VI. Approximating High-Dimension *Cspace* Obstacles

We have seen that when $A$ is a three-dimensional solid which is allowed to rotate, $CO_A(B)$ is a complicated curved object in a six-dimensional $Cspace_A$. An alternative to computing these objects directly is to use a sequence of low-dimensional projections of the high-dimensional $Cspace_A$ obstacles. For example, a three-dimensional $(x, y, \theta)$ $Cspace_A$ obstacle can be simply approximated by its projection on the $(x, y)$ plane, and any path of $A$ that avoided the projection would be safe for all orientations of $A$. On the other hand, there may be no paths that completely avoid the projection. A better approach is to divide the complete range of $\theta$ values into $k$ smaller ranges and, for each of these ranges, find the section of the $(x, y, \theta)$ obstacle in that range of $\theta$. These are called $\theta$-slices of the obstacle. The projection of these slices serves as an approximation of the obstacle. Paths that avoid individual slices are safe for orientations of $A$ in the $\theta$-range defining the slice.

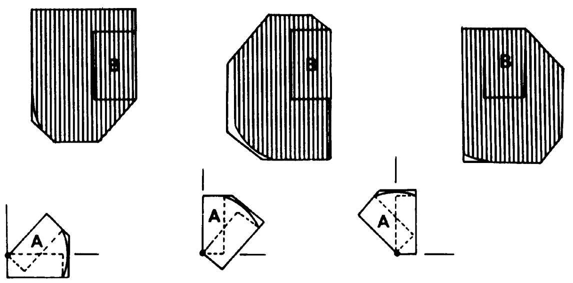


Fig. 10. Slice projections of $Cspace_A$ obstacles computed using the $(x, y)$-area swept out by $A$ over a range of $\theta$ values. Each of the shaded obstacles is the $(x, y)$-projection of a $\theta$-slice of $CO_A(B)$. The figure also shows a polygonal approximation to the slice projection and the polygonal approximation to the swept volume from which it derives.

The shaded areas in Fig. 10 are the $(x, y)$ projection of $\theta$-slices of $CO_A(B)$ when $A$ and $B$ are rectangles. These slices represent configurations where $A$ overlaps $B$ for some orientation of $A$ in the specified range of $\theta$. We will show that these slice projections are the $Cspace_A$ obstacles of the area swept out by $A$ over the range of orientations of the slice. The swept area under rotation of a polygon is not pologonal. To use the $CO_A^{xy}(B)$ algorithm developed earlier, we approximate the swept area as the union of polygons [23]. This polygonal approximations leads to a polygonal approximation for the projected slices, as shown in Fig. 10. Similar considerations apply to polyhedra.

The crucial properties of slice projection are: 1) a solution to a Findspace or Findpath problem in any of the slices is a solution to the original problem, although not all actual solutions can be found in the slices; and 2) the degree of approximation can be controlled by choosing the range of parameters of the slice, in particular the approximation need not be uniform across the range of parameters.

The Vgraph algorithm for Findpath, has been extended [24], by means of slice projection, to find paths when $A$ and all the $B_j$ are three-dimensional polyhedra that are allowed to rotate. Fig. 11 illustrates the basic idea of this algorithm. An alternative path-searching technique, also using slice projection is described in [23]. Because the slice projections are approximations to the $Cspace_A$ obstacles, neither of these algorithms is guaranteed to find solutions to Findpath problems. Paths found by Findpath algorithms that use slice projections are composed of sequences of translations interspersed with rotations, but where the rotations happen in quantized increments corresponding to the ranges of orientations that define the slices. Not all paths can be expressed in this fashion. For

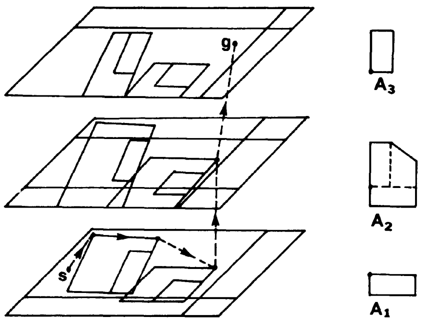


Fig. 11. An illustration of the Findpath algorithm using slice projection described by Lozano-Pérez and Wesley in [24]. A number of slice projections of the *Cspace* obstacles are constructed for different ranges of orientations of $A$. The problem of planning safe paths in the high-dimensional $C\,space_A$ is decomposed into 1) planning safe paths via $CO_A^{xyz}(B_j)$ vertices within each slice projection and 2) moving between slices, at configurations that are safe in both slices. $A_1$ represents $A$ in its initial configuration, $A_3$ represents $A$ in its final configuration and $A_2$ is a simple polyhedral approximation to the swept volume of $A$ between its initial and final orientation.

example, the classic problem of moving a rectangular sofa through a rectangular bend in a hallway that just fits the sofa requires continuous rotation during translation. However, a large class of useful problems can be solved using slice projection.

In the rest of the section we show how slice projections of $CO_A(B)$ may be computed using the $CO_A^{xy}(B)$ and $CO_A^{xyz}(B)$ algorithms of Section IV. The idea is simply that if a collision would occur for $A$ in some orientation, it would also occur for a swept volume of $A$ that includes $A$ in that orientation.

More formally, a $j$-slice of an object $C \in \mathcal{R}^n$ is defined to be $\{(\beta_1, \cdots, \beta_n) \in C \mid \gamma_j \le \beta_j \le \gamma_j'\}$, where $\gamma_j$ and $\gamma_j'$ are the lower and upper bounds of the slice, respectively. Then, if $I = \{1, \cdots, n\}$ and $K \subseteq I$, then a $K$-slice is the intersection of the $j$-slices for $j \in K$. Note that a $K$-slice of $C$ is an object of the same dimension as $C$. Slices can then be *projected* onto those coordinates in $I$ not in $K$, i.e., $I - K$, to obtain objects of lower dimension. A $j$-cross section is a $j$-slice whose lower and upper bounds are equal, e.g., $CO_A^{xy}(B)$, for some orientation of $A$, is the projection on the $(x, y)$ plane of a $\theta$-cross-section of $CO_A^{xy\theta}(B)$.

Slice projections are related to cross-section projections by the *swept volume* of an object. Intuitively, the swept volume of $A$ is all the space that $A$ covers when moving within a range of configurations. In particular, given two configurations for $A$, called $c$ and $c'$, then the union of $(A)_a$ for all $c \le a \le c'$ is the swept volume of $A$ over the configuration range $[c, c']$. Generally, $c$ and $c'$ differ only on some subset, $K$, of the configuration coordinates. For example, if $c$ and $c'$ are of the form $(\beta_1, \beta_2, \beta_3)$ and $K = \{3\}$, then the swept volume of $A$ over the range $[c, c']_K$ refers to the union of $A$ over a set of configurations differing only on $\beta_3$. The swept volume of $A$ over a configuration range is denoted $A[c, c']_K$.

If $A[c, c']_K$ overlaps some object $B$ then, for some configuration $a$ in that range, $(A)_a$ overlaps $B$. The converse is also true. $CO_{A[c,c']_K}(B)$ is the set of $I - K$ projections of those configurations of $A$ within $[c, c']_K$ for which $A$ overlaps $B$.

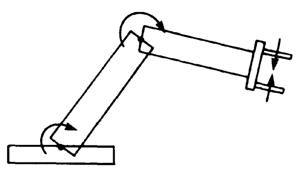


Fig. 12. Linked polyhedra can be used to model the gross geometry of industrial robot manipulators.

Equivalently, $CO_{A[c,c']_K}(B)$ is the $I - K$ projection of the $[c, c']_K$ slice of $CO_A(B)$. If the configurations of the swept volume are one of $(x, y)$, $(x, y, z)$, or $(x, y, \theta)$ then the algorithms of the previous sections can be used to compute $CO_{A[c,c']_K}(B)$ and thereby compute the required slice projections.[6]

A formal statement and proof of this result is included in Appendix II as Theorem 6. This theorem is of practical importance since it provides the mechanism underlying the Findspace and Findpath implementations described in [23] and [24].

## VII. Automatic Planning of Industrial Robot Motions

One application of the algorithms for Findspace and Findpath developed above is in the automatic planning of industrial robot motions [23], [43]. However, some extensions of the results for polyhedra are needed. In this section, we briefly discuss these extensions.

Industrial robots are open kinematic chains in which adjacent links are connected by prismatic or rotary joints, each with one degree of freedom [29]. We model them by *linked polyhedra*, kinematic chains with polyhedral links, each of which has either a translational or rotational degree of freedom relative to the previous joint in the chain; see Fig. 12. The relative position and orientation of adjacent links, $A_i$ and $A_{i+1}$, is determined by the $i$th *joint parameter* [29], [7]. The set of joint parameters of a linked polyhedron completely specifies the position and orientation of all the links. This type of model is clearly an approximation to the actual geometry; in particular, the shape of the joints is not represented and some values of the joint parameters may cause overlap of adjacent links.

The natural $Cspace_A$ for a linked polyhedron is that defined by the set of joint parameters. A point in this space determines the shape of the linked polyhedron and the configuration of each of its links. Unfortunately, the presence of rotary joints prevents the use of the $CO_A^{xyz}(B)$ algorithm of Section IV to plan the motions of linked polyhedra. However, there is an increasingly popular class of industrial robots, known as *Cartesian* robots, where the translational degrees of freedom of the robot are separate from the rotational. With this class of robots, we can use the $CO_A^{xyz}(B)$ algorithm and slice projection approach to plan collision-free paths and to plan how to grasp objects [23]. Actually doing this requires constructing the swept volume, over the rotational parameters, of the linked

[6] Of course, this requires computing a convex polyhedral approximation to the swept volume of $A$. Simple approximations are not difficult to compute [23], but this is an area where better algorithms are required. Nevertheless, the swept volume computation is a three-dimensional operation which can be defined and executed without recourse to six-dimensional constructs.

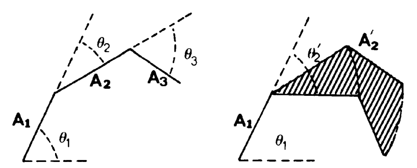


Fig. 13. Changes in the second joint angle from $\theta_2$ to $\theta_2'$ causes changes in the configurations of both link $A_2$ and link $A_3$.

polyhedron modeling the robot. The resulting swept volume can be viewed as a polyhedron with only translational degrees of freedom, for which the $CO_A^{xyz}(B)$ algorithm is applicable.

Using swept volumes of linked polyhedra for slice projection requires taking into consideration the interdependence of the joint parameters. Note that for a linked polyhedron, the position of link $j$ typically depends on the positions of links $k < j$, which are closer to the base than link $j$. Let $K = \{j\}$, $c = (\theta_i)$, $c' = (\theta_i')$, and $[c, c']_K$ define a range of configurations differing on the $j$th $Cspace_A$ parameter. Since joint $j$ varies over a range of values, links $l \geq j$ will move over a range of positions which depend on the values of $c$ and $c'$, as shown in Fig. 13. The union of each of the link volumes over its specified range of positions is the swept volume of the linked polyhedron. The swept volume of links $j$ through $n$ can be taken as defining a new $j$th link. The first $j - 1$ links and this new $j$th link define a new manipulator whose configuration can be described by the first $j - 1$ joint parameters. On the other hand, the shape of the new link $j$ depends not only on the $K$-parameters of $c$ and $c'$, i.e., $\theta_j$ and $\theta_j'$, but also on $\theta_l$ for $l > j$. This implicit dependence on parameters of $c$ and $c'$ that are not in $K$ is undesirable, since it means that the *shape* of the new $j$th link will vary as the swept volume is displaced, i.e., the $(I - K)$-parameters are changed. If we let $K = \{j, \cdots, n\}$, then the shape of the swept volume depends only on the $K$-parameters of $c$ and $c'$, while its configuration is determined by the $(I - K)$-parameters. A swept volume that satisfies this property is called *displaceable*. This property plays a crucial role in proving the fact, mentioned in Section VI, that slice projections of a $Cspace_A$ obstacle can be computed as the $Cspace_A$ obstacles of the swept volume of $A$ (see Theorem 6 in Appendix II).

In summary, with the extensions discussed in this section, the spatial planning algorithms developed for the case of rigid polyhedra can serve as the basis for planning the motions of industrial robots.

## VIII. Related Work in Spatial Planning

The definition of the Findspace problem used here is based on that in [49]. One previous approach to this problem is described in [30]; it is an application of the Warnock algorithm [47] for hidden line elimination. The idea is to recursively subdivide the workspace until an area "large enough" for the object is found. This approach has several drawbacks: 1) any nonoverlapping subdivision strategy will break up potentially useful areas and 2) the implementation of the predicate "large enough" is not specified (in general, the $CI_A(B)$ computation is required to implement this predicate). However, once the $Cspace_A$ obstacles have been computed, the Warnock search provides a good way of solving Findspace; since we only need space for a point, any free area is "large enough" [10].

The work by Udupa, reported in [45], [46], was the first to approach Findpath by explicitly using transformed obstacles and a space where the moving object is a point. Udupa used only rough approximations to the actual *Cspace* obstacles and had no direct method for representing constraints on more than three degrees of freedom. A survey of previous heuristic approaches to the Findpath problem for manipulators, for example, [20], [31], has been given in [46]. An early paper on Shakey [28] describes a technique for Findpath using a simple object transformation that defines safe points for a circular approximation to the mobile robot and uses a graph search formulation of the problem. More recent papers on navigation of mobile robots are also relevant to two-dimensional Findpath [14], [26] [44]. An early paper [18] reports on a program for planning the path of a two-dimensional sofa through a corridor. This program does a brute-force graph search through a quantized *Cspace*.

The *Cspace* approach to Findspace and Findpath described here is an extension of that reported in [24]. In that paper, an approximate algorithm for $CO_A^{xyz}(B)$ is described and the Vgraph algorithm for high-dimensional Findpath is first represented. An application of the Findpath and Findspace approach described in the current paper to automatic planning of manipulator motions is described in [23]. Alternative approaches to path searching in the presence of obstacles are described in [19], [12], [23]. The visibility computation needed in Vgraph is treated, in the context of hidden-line elimination, in [4], [51].

The basic idea of representing position constraints as geometric figures, e.g., $CO_A^{xy}(B)$, has been used (independently) in [1]–[3], where an algorithm to compute $CO_A^{xy}(B)$ for nonconvex polygons is used in a technique for two-dimensional layout. The template packing approach described in [13] uses a related computation based on a chain-code description of figure boundaries. Algorithms for packing of parallelopipeds, in the presence of forbidden volumes, using a construct equivalent to the $CO_A^{xyz}(B)$, but defined as "the hodograph of the close positioning function" are reported in [42]. The only use of this construct in the paper is for computing $CO_A^{xyz}(B)$ for aligned rectangular prisms.

An extension of the approach in [24] to the general Findpath problem is proposed in [34]. The proposal is based on the use of an exact representation of the high-dimensional *Cspace* obstacles. The basic approach is to define the general configuration constraints as a set of multinomials in the position parameters of $A$. However, the proposal still requires elaboration. It defines the configuration space constraints in terms of the relationships of vertices of one object to the faces of the other. This is adequate for polygons, but the equations in the paper only express the constraints necessary for vertices of $A$ to be outside of $B$, i.e., they are of the form of (3). They do not account for the positions of $A$ where vertices of $B$ are in contact with $A$ [see (4)]. The new equations will have terms of the form $x \cos\theta$ and $y \cos\theta$. Furthermore, the approach of defining the configuration constraints by examining the interaction of vertices and faces does not generalize to three-dimensional polyhedra. It is not enough to consider the interaction of vertices and faces; the interaction of edges and faces must also be taken into account (see Section V and [8]).

Two recent papers describe solutions for the Findpath problem with rotations in two [40] and three dimensions [41]. In [41], the *Cspace* surfaces are represented as algebraic manifolds in a 12-dimensional space; in this way the surfaces can be described as polynomials, allowing the use of some powerful mathematical machinery. The resulting algorithm has (large) polynomial time complexity, for fixed dimensionality of the *Cspace*.

A *Cspace* algorithm is described in [10] for solving Findpath, allowing rotations of the moving objects. The algorithm is based on recursively subdividing *Cspace* until a path of cells completely outside of the obstacles is found.

An alternative approach to two-dimensional Findpath with rotations is described in [9]. The algorithm is based on representing the empty space outside the objects $B_j$ explicitly as *generalized cones*. Motions of $A$ are restricted to be along the spines of the cones. The algorithm bounds the moving object by a convex polygon and characterizes the legal rotations of the bounding polygon along each spine.

## Appendix I

## Algorithm for $CO_A^{xy}(B)$

This Appendix shows an algorithm for computing $A \oplus B$, called SET-SUM(A, B, C), when $A$ and $B$ are convex polygons. Section II used this operation to compute $CO_A^{xy}(B)$.

Each polygon is described in terms of its vertices and the angles that the edges make with the positive $x$ axis. The edges and vertices are ordered in counterclockwise order, i.e., by increasing angle. The implementation assumes that a POLYGON record is available with the following components:

1) size—number of edges in the polygon.

2) vert [1:size + 1, 1:2]—an array of vectors representing the coordinates of a vertex. The $i$th edge, $i = 1, \cdots$, size, has the endpoints vert[i, k] and vert[i + 1, k], for $k \in \{1, 2\}$. Note that vert[size + 1, k] = vert[1, k].

3) angle [0:size]—the edge normal's angle (in the range $[0, 2\pi]$) with the $x$ axis, monotonically increasing. For convenience angle [0] = $2\pi$ − angle [size].

References to the components of a polygon, a, are written as one of a.size, a.vert, and a.angle.

The algorithm implements the angle scan in the proof of Theorem 2; in particular, the edges of the input polygons, a and b, are examined in order of angle. The algorithm determines the position of the vertices of c. It is clear that vertices can occur only at angles where there is either a vertex of a, or a vertex of b, or both. From Lemmas 1 and 2, it is easy to see that the position of the vertex of c is the sum of the positions of the corresponding vertices of a and b. The algorithm starts the scan at the angle determined by the first edge of b, the first loop in the program below serves to find the edges of a that straddle that angle. From there, the algorithm increments the edge index into a or b depending on which makes the smaller angle increment. In general, the algorithm requires incrementing the angle beyond $2\pi$ so as to consider all the edges of a before the edge found by the first loop of the program. Since the edges are stored with angles between 0 and $2\pi$, an offset variable is used to add $2\pi$ to the angle when the wraparound on polygon a is detected.

```
PROCEDURE setsum (a, b, c);
      POLYGON a, b, c;
      BEGIN INTEGER ea, eb, vc, i;
            REAL ang, offset;
            COMMENT Initialize an index into a, one into b, and one into c.
                  The value of offset will be either 0 or 2*pi, and it is used
                  to handle angle wraparound as described above;
            ea := 1;
            eb := 1;
            vc := 1;
            offset := 0;
            COMMENT Find adjacent edges in a whose angles straddle the angle
                  of the first edge of b;
            WHILE (a.angle[ea] <= b.angle[1] OR a.angle[ea - 1] >= b.angle[1])
                  DO ea := ea + 1;
            FOR i := 1 STEP 1 UNTIL 2 DO
                  c. vert[1, i] := a.vert[ea, i] + b.vert[1, i];
            COMMENT This loop implements the scan of Theorem 2 in the body of the
                  paper. The result of the loop is to fill the vertex array of c;
            WHILE (eb <= b.size) DO
                  BEGIN
                        vc := vc + 1;
                        ang := offset + a.angle[ea];
                        IF (ang <= b.angle[eb])
```

```
                THEN IF (ea >= a.size)
                    THEN BEGIN offset := 2*pi; ea := 1 END
                    ELSE ea := ea + 1;
                IF (ang >= b.angle[eb])
                THEN eb := eb + 1;
                FOR i := 1 STEP 1 UNTIL 2 DO
                    c.vert[vc, i] := a.vert[ea, i] + b.vert[eb, i]
            END;
        c.size := vc;
        FOR i := 1 STEP 1 UNTIL 2 DO
            c.vert[vc + 1, i] := c.vert[1, i];
END
```

## Appendix II

### Proof of Theorem 6

Assume that $Cspace_A \subseteq \mathcal{R}^d$, let $I = \{1, 2, \cdots, d\}$ and $K \subseteq I$. Let $I$, $K$ and $I - K$ denote sets of indexes for the coordinates of $a \in Cspace_A$. Define the following vectors, all in $Cspace_A$: $b = (\beta_i)$, $c = (\gamma_i)$ and $c' = (\gamma'_i)$ for $i \in I$. Then,

$$\Phi_K(c, c') \equiv \left\{ b \in \mathcal{R}^d \middle| \bigwedge_{k \in K} \gamma_k \le \beta_k \le \gamma'_k \right\}$$

$$\Phi_K(c) \equiv \Phi_K(c, c)$$

$$\Theta_K(c, c') \equiv \Phi_K(c, c') \cap \Phi_{I-K}(c, c)$$

These definitions are illustrated in Fig. 14.

The *projection operator*, denoted $P_K[\cdot]$: $\mathcal{R}^d \to \mathcal{R}^{|K|}$ is defined, for vectors and sets of vectors, by

$$P_K[b] = (\beta_k) \qquad k \in K$$

$$P_K[B] = \{P_K[b] | b \in B\}$$

Superscripts on vectors indicate projection, e.g., $b^K = P_K[b]$. In addition, the vector in $\mathcal{R}^{|I|}$ composed from one vector in $\mathcal{R}^{|K|}$ and one in $\mathcal{R}^{|I-K|}$ is denoted $(a^{I-K}\colon b^K)$, where $P_{I-K}[(a^{I-K}\colon b^K)] = a^{I-K}$ and $P_K[(a^{I-K}\colon b^K)] = b^K$.

In this notation, precise definitions for the notions of *cross section projection* and *slice projection* can be provided. The cross section projection of a $Cspace_A$ obstacle is written as follows:

$$P_{I-K}[CO_A(B) \cap \Phi_K(c)].$$

The slice projection, is similar to the cross section projection,

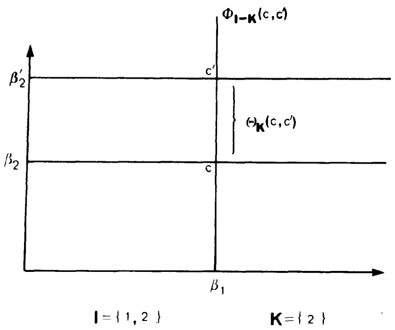


Fig. 14. Illustration of the definition of $\Phi_K(c,c')$ and $\Theta_K(c,c')$.

but carried out for all configurations between two cross sections:

$$P_{I-K}[CO_A(B) \cap \Phi_K(c, c')].$$

The $K$-parameters of the two configurations, $c$ and $c'$, define the bounds of the slice. Similarly, the swept volume can be defined in this notation.

*Definition:* The *swept volume* of $A$ over the configuration range $[c, c']_K$ is

$$(A[c, c']_K)_c \equiv \bigcup_{a \in \Theta_K(c,c')} (A)_a.$$

The requirement discussed in Section VII that the swept volume of $A$ be *displaceable* is embodied in the following condition:

$$\forall a\colon \bigcup_{x \in \Theta_K(c,c')} (A)_{(a^{I-K}:x^K)} = (A[c, c']_K)_{(a^{I-K}:c^K)} \qquad (7)$$

Note that the $I - K$ parameters may be changed, as in (7), but not those parameters in $K$. Therefore, $(A[c, c']_K)_a$ is defined only if $a \in \Phi_K(c)$.

Theorem 6: If (7) holds, then

$$P_{I-K}[CO_A(B) \cap \Phi_K(c, c')] = P_{I-K}[CO_{A[c,c']_K}(B) \cap \Phi_K(c)]$$

*Proof of Theorem:* Assume that the configuration $a$ is in the slice projection of $CO_A(B)$, that is,

$$a \in P_{I-K}[CO_A(B) \cap \Phi_K(c, c')].$$

This assumption and the definition of the projection operator allows us to deduce that some configuration in $Cspace_A$, whose $I - K$-projection is $a$, is in $CO_A(B)$:

$$\Leftrightarrow \exists x_1 \in \Phi_K(c, c')\colon ((a^{I-K}\colon x_1^K) \in CO_A(B)).$$

In fact, since we are only interested in the $K$-parameters of $x_1$ and $\Theta_K(c, c') \subseteq \Phi_K(c, c')$, we can assume without loss of generality that $x_1$ is in the smaller set, i.e.,

$$\Leftrightarrow \exists x_1 \in \Theta_K(c, c')\colon ((a^{I-K}\colon x_1^K) \in CO_A(B)).$$

Simply using the definition of $CO_A(B)$, it follows that

$$\Leftrightarrow (A)_{(a^{I-K}:x_1^K)} \cap B \ne \emptyset,$$

but if $A$ in this configuration intersects $B$, then any set including $A$ in that configuration will also intersect $B$. In particular,

$$\Leftrightarrow \bigcup_{x \in \Theta_K(c,c')} (A)_{(a^{I-K}:x^K)} \cap B \neq \emptyset.$$

We are assuming that swept volumes are displaceable, i.e., that (7) holds. Therefore, using (7), we get

$$\Leftrightarrow (A[c, c']_K)_{(a^{I-K}:c^K)} \cap B \neq \emptyset$$

Hence, by the definitions of $CO_A(B)$ and $\Phi_K(c)$,

$$\Leftrightarrow (a^{I-K}:c^K) \in CO_{A[c,c']}(B) \text{ and } (a^{I-K}:c^K) \in \Phi_K(c)$$

Applying the definition of the projection operator completes the proof:

$$\Leftrightarrow a \in P_{I-K}[CO_{A[c,c']_K}(B) \cap \Phi_K(c)]. \qquad \blacksquare$$

## Acknowledgment

The author would like to thank M. Brady, R. Brooks, J. Hollerbach, B. Horn, T. Johnson, M. Mason, P. Winston, B. Woodham, and the referees for their suggestions on the content and presentation of this paper. J. Hollerbach suggested the proof of Theorem 6 in the current version of the paper, which is much simpler than the original proof.

## References

[1] M. Adamowicz, "The optimum two-dimensional allocation of irregular, multiply-connected shapes with linear, logical and geometric constraints," Ph.D. dissertation, Dep. Elec. Eng., New York Univ., 1970.

[2] M. Adamowicz and A. Albano, "Nesting two-dimensional shapes in rectangular modules," *Computer Aided Design*, vol. 8, pp. 27-32, Jan. 1976.

[3] A. Albano and G. Sapuppo, "Optimal allocation of two-dimensional irregular shapes using heuristic search methods," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-10, pp. 242-248, May 1980.

[4] D. Avis and G. T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," Sch. Comput. Sci., McGill Univ., Rep. SOCS-80.2, Feb. 1980.

[5] R. V. Benson, *Euclidean Geometry and Convexity.* New York: McGraw-Hill, 1966.

[6] J. I. Bentley and T. Ottman, "Algorithms for reporting and counting geometric intersections," *IEEE Trans. Comput.*, vol. C-28, Sept. 1979.

[7] O. Bottema and B. Roth, *Theoretical Kinematics.* Amsterdam: North-Holland, 1979.

[8] J. W. Boyse, "Interference detection among solids and surfaces," *Commun. Ass. Comput. Mach.*, vol. ACM 22, pp. 3-9, Jan. 1979.

[9] R. A. Brooks, "Solving the Find-Path problem by representing free space as generalized cones," M.I.T. Artificial Intell. Lab., Rep. AIM-674, May 1982.

[10] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for Findpath with rotation," M.I.T. Artificial Intell. Lab., Rep. AIM-684, Dec. 1982.

[11] K. Q. Brown, "Fast intersection of half spaces," Dep. Comput. Sci., Carnegie-Mellon Univ., Rep. CS-78-129, June 1978.

[12] C. E. Campbell and J. Y. S. Luh, "A preliminary study on path planning of collision avoidance for mechanical manipulators," Sch. Elec. Eng., Purdue Univ., Rep. TR-EE 80-48, Dec. 1980.

[13] H. Freeman, "On the packing of arbitrary-shaped templates," in *Proc. 2nd USA-Japan Comput. Conf.*, 1975, pp. 102-107.

[14] G. Giralt, R. Sobek, and R. Chatila, "A multilevel planning and navigation system for a mobile robot," in *Proc. 6th Int. Joint Conf. Artificial Intell.*, Tokyo, Japan, Aug. 1979, pp. 335-338.

[15] R. L. Graham, "An efficent algorithm for determining the convex hull of a finite planar set," *Inform. Processing Lett.*, vol. 1, pp. 132-133, 1972.

[16] B. Grunbaum, *Convex Polytopes.* New York: Wiley-Interscience, 1967.

[17] L. J. Guibas and F. F. Yao, "On translating a set of rectangles," in *Proc. 12th Annu. ACM Symp. Theory of Computing*, Los Angeles, CA, Apr. 1980, pp. 154-160.

[18] W. E. Howden, "The sofa problem," *Comput. J.*, vol. 11, pp. 299-301, Nov. 1968.

[19] R. C. Larson and V. O. K. Li, "Finding minimum rectilinear distance paths in the presence of obstacles," *Networks*, vol. 11, pp. 285-304, 1981.

[20] R. A. Lewis, "Autonomous manipulation on a robot: Summary of manipulator software functions," Jet Propulsion Lab., California Inst. Technol., TM 33-679, Mar. 1974.

[21] L. Lieberman and M. A. Wesley, "AUTOPASS: An automatic programming system for computer controlled assembly," *IBM J. Res. Develop.*, vol. 21, July 1977.

[22] T. Lozano-Pérez, "The design of a mechanical assembly system," M.I.T. Artificial Intell. Lab., Rep. TR-397, Dec. 1976.

[23] ——, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-11, pp. 681-698, Oct. 1981.

[24] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. Ass. Comput. Mach.*, vol. ACM 22, pp. 560-570, Oct. 1979.

[25] L. Lyusternik, *Convex Figures and Polyhedra.* Dover, 1963, original copyright, Moscow, 1956.

[26] H. P. Moravec, "Visual mapping by a robot rover," in *Proc. 6th Int. Joint Conf. Artificial Intell.*, Tokyo, Japan, Aug. 1979.

[27] D. Mueller and F. Preparata, "Finding the intersection of two convex polyhedra," Coordinated Sci. Lab., Univ. Illinois, Urbana, Rep. R-793, Oct. 1977.

[28] N. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," in *Proc. 2nd Int. Joint Conf. Artificial Intell.*, 1969, pp. 509-520.

[29] R. P. Paul, "Manipulator Cartesian path control," *IEEE Trans. Syst., Man., Cybern.*, vol. SMC-9, pp. 702-711, Nov. 1979.

[30] G. Pfister, "On solving the FINDSPACE problem, or how to find where things aren't," M.I.T. Artificial Intell. Lab., Working Paper 113, Mar. 1973.

[31] D. L. Pieper, "The kinematics of manipulators under computer control," Stanford Artificial Intell. Lab., AIM-72, Oct. 1968.

[32] F. Preparata and S. Hong, "Convex hulls of finite sets of point in two and three dimensions," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 87-93, Feb. 1977.

[33] F. Preparata and D. Mueller, "Finding the intersection of a set of half spaces in time $O(n \log n)$," Coordinated Sci. Lab., Univ. Illinois, Urbana, Rep. R-803, Dec. 1977.

[34] J. Reif, "On the movers problem," in *Proc. 20th Annu. IEEE Symp. Foundation of Comput. Sci.*, 1979, pp. 421-427.

[35] B. Schachter, "Decomposition of polygons into convex sets," *IEEE Trans. Comput.*,vol. C-27, pp. 1078-1082, Nov. 1978.

[36] M. I. Shamos, "Geometric complexity," in *Proc. 7th Annu. ACM Symp. Theory of Computing*, 1975, pp. 224-233.

[37] M. I. Shamos and D. Hoey, "Closest-point problems," in *Proc. 16th Annu. IEEE Symp. Foundation of Comput. Sci.*, 1975, pp. 151-161.

[38] ——, "Geometric intersection problems," in *Proc. 17th Annu. IEEE Symp. Foundation of Comput. Sci.*, 1976, pp. 208-215.

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

[40] J. T. Schwartz and M. Sharir, "On the piano movers' problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," Dep. Comput. Sci., Courant Inst. Math. Sci., New York Univ., Rep. 39, 1981.

[41] ——, "On the piano movers' problem II. General techniques for computing topological properties of real manifolds," Dep. Comput. Sci., Courant Inst. Math. Sci., New York Univ., Rep. 41, 1982.

[42] Y. G. Stoyan and L. D. Ponomarenko, "A rational arrangement of geometric bodies in automated design problems," *Eng. Cybern.*, vol. 16, Jan. 1978.

[43] R. Taylor, "A synthesis of manipulator control programs from task-level specifications," Stanford Artificial Intell. Lab., Rep. AIM-282, July 1976.

[44] A. M. Thompson, "The navigation system of the JPL robot," in *Proc. 5th Int. Joint Conf. Artificial Intell.*, Massachusetts Inst. Technol., 1977.

[45] S. Udupa, "Collision detection and avoidance in computer controlled manipulators," in *Proc. 5th Int. Joint Conf. Artificial Intell.*, Massachusetts Inst. Technol., 1977.

[46] ——, "Collision detection and avoidance in computer controlled manipulators," Ph.D. dissertation, Dep. Elec. Eng., California Inst. Technol., 1977.

[47] J. E. Warnock, "A hidden line algorithm for halftone picture representation," Dep. Comput. Sci., Univ. Utah, Rep. TR4-5, May 1968.

[48] J. Williams, "STICKS—A new approach to LSI design," S.M. thesis, Dep. Elec. Eng., Massachusetts Inst. Technol., 1977.

[49] T. Winograd, *Understanding Natural Language.* New York: Aademic, 1972.

[50] T. C. Woo, "Progress in shape modelling," *IEEE Computer*, vol. 10, Dec. 1977.

[51] F. F. Yao, "On the priority approach to hidden-surface algorithms," in *Proc. 21st Symp. Foundation of Comput. Sci.*, 1980, pp. 301-307.