F Polyhedral Robots in Polyhedral Worlds

LINEAR REPRESENTATIONS are concise. In this appendix we consider the special case in which both the robot and all obstacles in the workspace are polygons (for two-dimensional worlds) or polyhedra (for three-dimensional worlds). Since polyhedra are three-dimensional solids whose faces are polygons, we begin by developing representations and computational methods for dealing with polygons. Although the restriction to polygonal obstacle may seem to be unrealistic, nearly all modern motion planning systems use polygonal models to represent obstacles (e.g., facet models that are common in computer graphics and so-called *polygon soup* models that are used in many CAD applications).

We begin the appendix by describing the representation of polygons in two dimensions. Following this, we describe an algorithm for determining whether two polygons intersect. This is the fundamental operation used by collision detection algorithms. We then describe an efficient algorithm that constructs a boundary representation for the configuration space obstacle region for the special case of $\mathcal{Q} = \mathbb{R}^2$ and discuss configuration space obstacles for the case of $\mathcal{Q} = SE(2)$.

F.1 Representing Polygons in Two Dimensions

A straight line in the plane divides the plane into three disjoint regions: the line itself, and the two regions that lie on either side of the line. To make this more precise, consider the line given by

$$h(x, y) = ax + by - c = 0. \qquad \text{(F.1)}$$

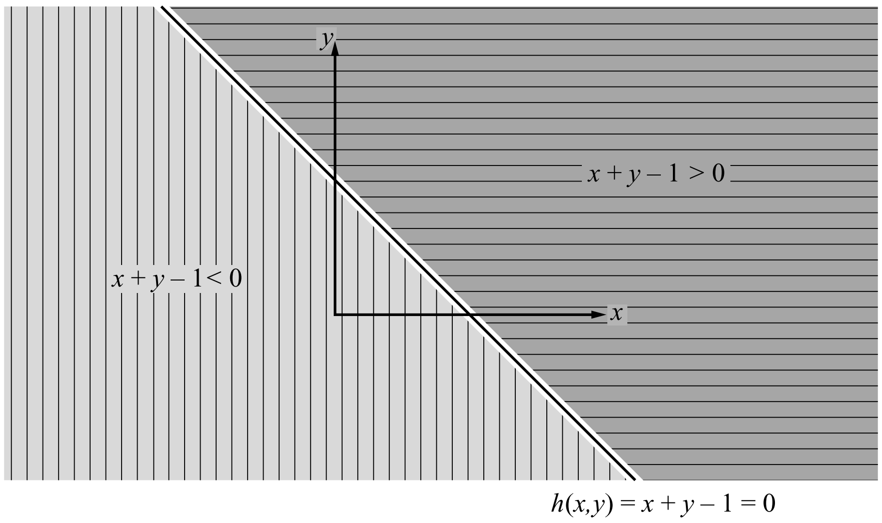

Figure F.1 Half-planes defined by $h(x, y) = x + y - 1$.

This equation implicitly defines a line to be the set of points whose projection onto the vector (a, b) is given by c. Thus, the vector (a, b) defines the normal to the line and c gives the signed perpendicular distance from the origin to the line. We can evaluate h for any point in the plane. Those points such that $h(x, y) \geq 0$ are said to lie in the *positive half plane,* represented by h^+. Points in h^+ are those points whose projection onto the normal is greater than the signed distance to the line. Those points such that $h(x, y) \leq 0$ are said to lie in the *negative half plane,* represented by h^-. The line itself is the intersection $h^- \cap h^+$. Figure F.1 shows an example for which the points (0, 5), (3, 5) lie in the negative half plane, while points (0, 0), (2, 2) lie in the positive half plane. Note that we can easily change the sense of the half planes by multiplying h by -1.

We can use half planes to construct polygons. In particular, we define a *convex polygonal region* in $\mathbb{R}^2$ to be the intersection of a finite number of half planes. For example, the three lines

$$h_1(x, y) = -x + y - 3$$
$$h_2(x, y) = -y$$
$$h_3(x, y) = x$$

can be used to construct a convex polygonal region by taking the intersection of the three half planes h_1^-, h_2^-, and h_3^-, as shown in figure F.2. For consistency, we will

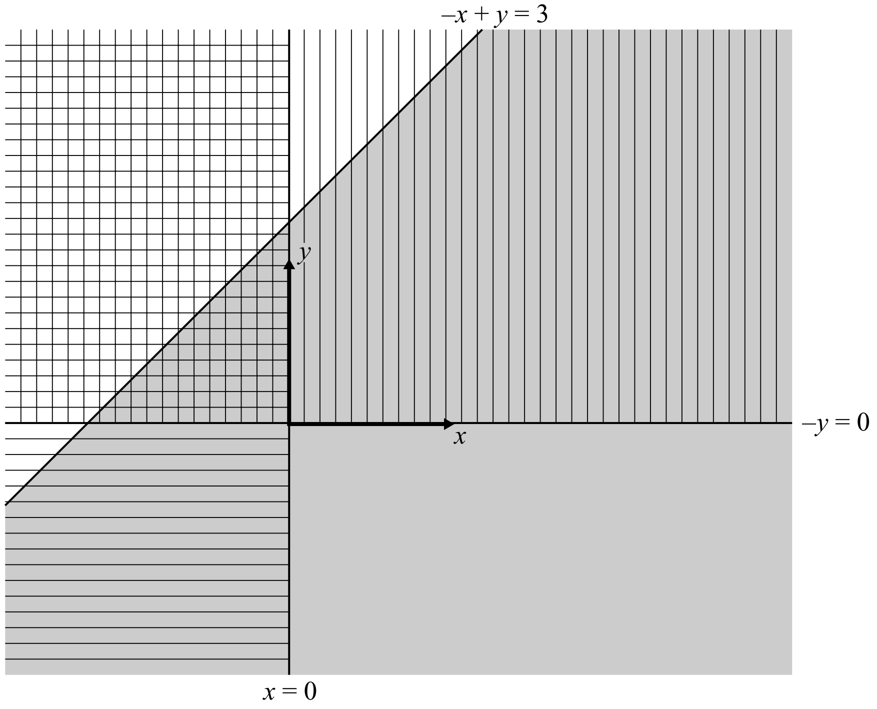

Figure F.2 A convex polygonal region constructed from the half planes h_1^-, h_2^-, and h_3^-.

always define convex polygonal regions as the intersection of negative half planes. If $h_i(x, y) \leq 0$ for each line that defines the convex polygonal region, then the point (x, y) lies inside the corresponding polygonal region. If $h_i(x, y) > 0$ for any line that defines the convex polygonal region, then the point lies outside the corresponding polygonal region. Note that a convex polygonal region need not be finite. For example, by our definition, the half space $x + y - 1 \leq 0$ is a valid convex polygonal region, even though it is unbounded (recall that a region is said to be convex if for all pairs of points in the region, the line segment connecting those points lies entirely within the region).

We define a *polygonal region* (possibly nonconvex) to be any subset of $\mathbb{R}^2$ obtained by taking the union of a finite number of convex polygonal regions. Polygonal regions need not be bounded or connected, and connected polygonal regions need not be simply connected (e.g., the union of two disjoint convex polygons is a polygonal region, but it is not connected). Finally, a *polygon* is any closed, simply connected polygonal region (alternatively, a polygonal region that is homeomorphic to a closed unit disk in the plane).

It is often convenient to represent a polygon by listing its vertices, e.g., in counterclockwise order (it is straightforward to determine the h_i given the set of vertices). This approach is used in sections F.2 and F.3, where we discuss how to construct the configuration space obstacle and then how to determine if a robot intersects it.

F.2 Intersection Tests for Polygons

In this section, we develop an algorithm for determining whether two polygons have a nonempty intersection. Such intersection tests are the essential primitive operations for collision detection algorithms used by most all modern path planners. Furthermore, for the specific case of $\mathcal{Q} = \mathbb{R}^2$ with polygonal obstacles, the intersection test that we develop here provides useful insight for developing an algorithm to explicitly construct the configuration space obstacle region, as described below in section F.3. We begin by considering the specific problem of testing for the intersection of a convex, polygonal robot with a specific convex, polygonal obstacle.

We will assume that the configuration of the robot is specified by $q = (x, y, \theta)$ and that the obstacle polygon is specified by a list of its vertices. It will also be convenient to explicitly represent the normal vectors for each edge of both the robot and the obstacle. We denote these normal vectors by n_i^R for the normal to edge i of the robot and $n_j^{\mathcal{W}}$ for the normal to edge j of obstacle $\mathcal{W}$. Note that the normals for the robot edges depend on the orientation (but not the x, y-coordinates) of the robot; we will often explicitly represent this dependence by the notation $n_i^R(\theta)$. We denote the vertices of the robot by r_i, and the edges by E_i^R. Similarly, we will denote the vertices of obstacle $\mathcal{W}$ by o_j and edges $E_j^{\mathcal{W}}$. Figure F.3 illustrates the notation.

Under these conditions, the problem of determining whether the robot intersects the obstacle is equivalent to determining whether the robot configuration lies within the configuration space obstacle region. The approach that we develop here identifies the defining half spaces for the configuration space obstacle region for a fixed robot orientation, θ. If the robot configuration is contained in *each* of these half spaces, then it lies in the configuration space obstacle polygon (since this polygon is merely the intersection of the half spaces), and the robot and obstacle intersect.

The problem of identifying these defining half spaces is equivalent to determining the boundary of the configuration space obstacle polygon. Recall that for a fixed value of θ, the boundary of this polygon corresponds to the set of configurations in which the robot and obstacle touch, but do not overlap. (If the robot and obstacle overlap, then we can move the robot to any configuration in a neighborhood and remain within the obstacle polygon.) For the kth obstacle, this condition can be expressed by

$$R(q) \cap \mathcal{W} \neq \emptyset \quad \text{and} \quad \text{int}\,(R(q)) \cap \text{int}\,(\mathcal{W}) = \emptyset. \tag{F.2}$$

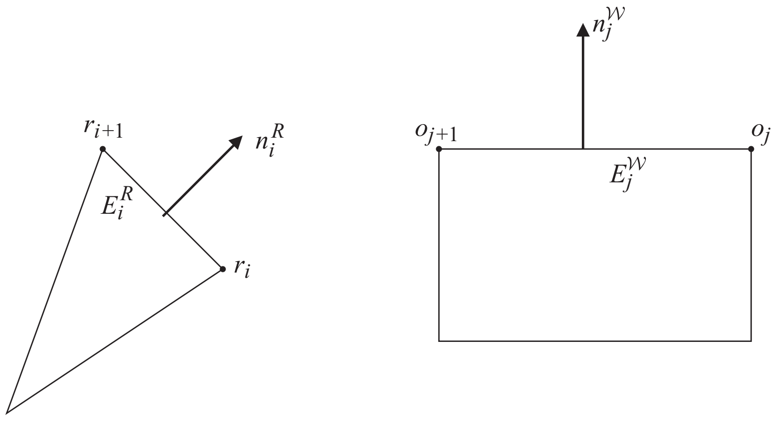

Figure F.3 Notation used to define vertices, normals and edges of the robot and obstacle polygon.

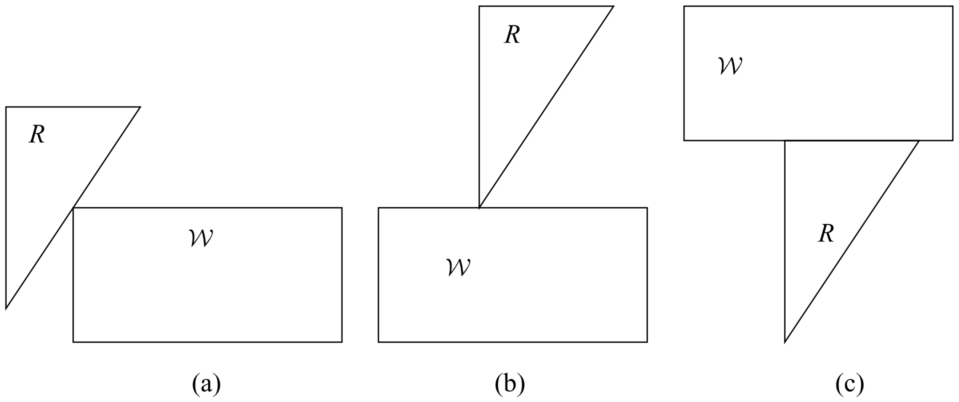

Figure F.4 (a) Type A contact, (b) Type B contact, (c) Both Type A and Type B contact.

For configurations that satisfy (F.2), there are only two possible kinds of contacts:

Type A Contact: an edge of R, say E_i^R, contains a vertex, o_j, of $\mathcal{W}$.

Type B Contact: an edge of $\mathcal{W}$, say $E_j^{\mathcal{W}}$, contains a vertex, r_i, of R.

Each possible type A or type B contact defines one half space that defines the configuration space obstacle polygon. Type A and B contacts are illustrated in figure F.4. Note that in figure F.4(c), both type A and B contacts occur simultaneously.

We begin with the case of type A contact. Type A contact between edge E_i^R and vertex o_j is possible only for certain orientations θ. In particular, such contact can occur only when θ satisfies

$$(o_{j-1} - o_j) \cdot n_i^R(\theta) \geq 0 \quad \text{and} \quad (o_{j+1} - o_j) \cdot n_i^R(\theta) \geq 0. \tag{F.3}$$

This condition is sometimes referred to as an *applicability condition*. Note that the normals for the edges of R are a function of configuration, but only of θ, and not of the x, y coordinates. The condition in (F.3) can also be expressed as the condition that a negated edge normal of the robot lies between the normals of an adjacent obstacle edge. This latter formulation of the condition is used below in section F.3. Note that (F.3) is satisfied with equality when an edge of the obstacle is coincident with an edge of the robot.

Each pair, E_i^R and o_j, that satisfies (F.3), defines a half space that contains the configuration space obstacle polygon. This half space is defined by

$$f_{ij}^R(x, y, \theta) = n_i^R(q) \cdot (o_j - r_i(x, y, \theta)) \leq 0. \tag{F.4}$$

This is illustrated in figure F.5.

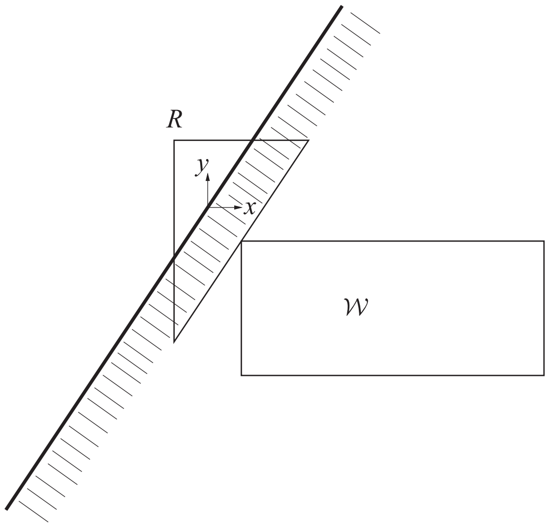

Figure F.5 The half space defined by this contact is below the thick black line that passes through the origin of the robot's coordinate frame.

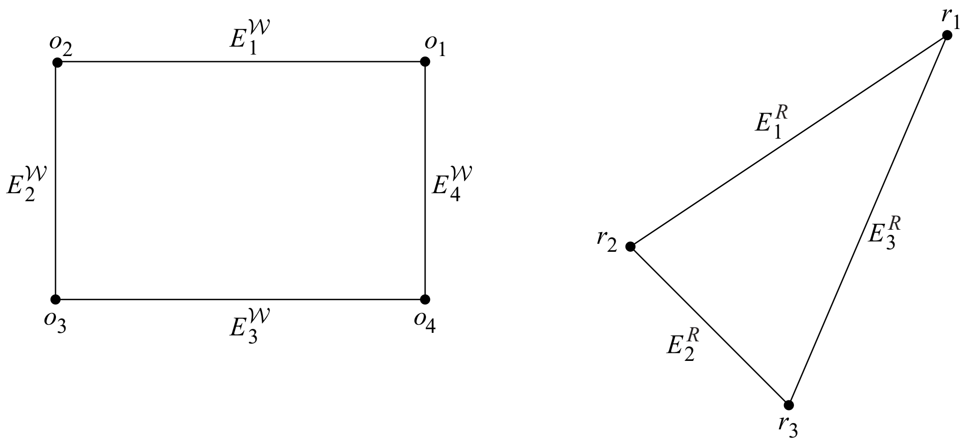

Figure F.6 The obstacle is shown on the left, and the robot on the right.

Type B contact is analogous to type A contact, but the roles of robot and obstacle are reversed. In particular, type B contact can occur between obstacle edge $E_j^{\mathcal{W}}$ and robot vertex r_i when

$$(r_{i-1}(\theta) - r_i(\theta)) \cdot n_j^{\mathcal{W}} \geq 0 \quad \text{AND} \quad (r_{i+1}(\theta) - r_i(\theta)) \cdot n_j^{\mathcal{W}} \geq 0. \tag{F.5}$$

The corresponding half space is defined by

$$f_{ij}^{\mathcal{W}}(x, y, \theta) = n_j^{\mathcal{W}} \cdot (r_i(x, y, \theta) - o_j) \leq 0. \tag{F.6}$$

Each type A or B contact defines one half space that contains the configuration space obstacle polygon. The configuration (x, y, θ) causes a collision only if it lies in *each* of these half spaces. Therefore, determining collision amounts to determining which i, j satisfy (F.3) or (F.5), and then verifying (F.4) or (F.6), respectively.

As an example, consider the robot and obstacle shown in figure F.6. Figure F.7(a) shows a case in which the robot and obstacle have a nonempty intersection. The following table shows the possible type A and B contacts (the first three entries of the table are the type A contacts), the definitions of the corresponding half spaces, and whether or not the half space constraints are satisfied. As can be seen, each applicable half space constraint is satisfied, and thus it is determined that the robot and obstacle are in collision.

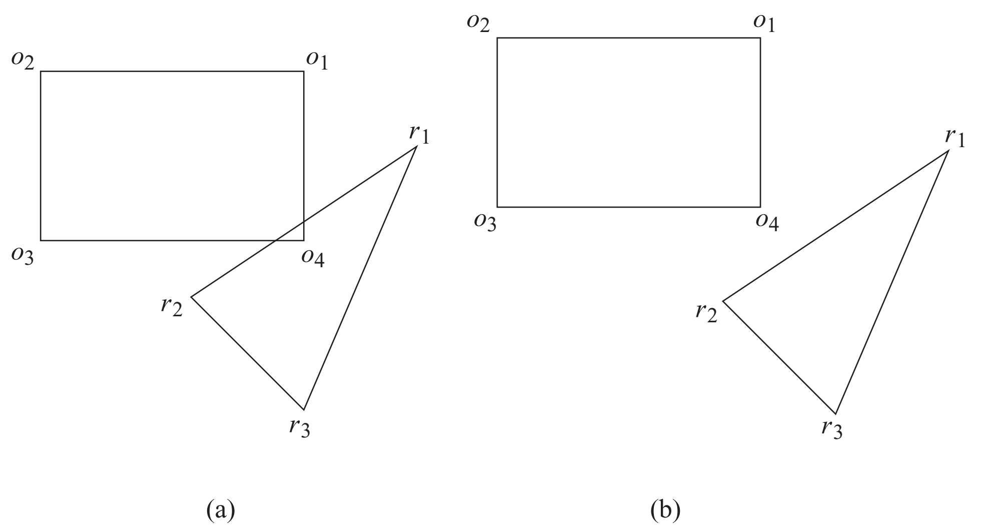

Figure F.7 The applicability conditions and half spaces for these two cases are shown in the table below.

Contact pair	**half space inequality**	**satisfied?**
E_1^R, o_4^k	$n_1^R(\theta) \cdot (o_4^k - r_1(x, y, \theta) \leq 0$	yes
E_2^R, o_1^k	$n_2^R(\theta) \cdot (o_1^k - r_2(x, y, \theta) \leq 0$	yes
E_3^R, o_3^k	$n_3^R(\theta) \cdot (o_3^k - r_3(x, y, \theta) \leq 0$	yes
$E_1^{\mathcal{W}_k}, r_3$	$n_1^{\mathcal{W}_k} \cdot (r_3(x, y, \theta) - o_1^k \leq 0$	yes
$E_2^{\mathcal{W}_k}, r_3$	$n_2^{\mathcal{W}_k} \cdot (r_3(x, y, \theta) - o_2^k \leq 0$	yes
$E_3^{\mathcal{W}_k}, r_1$	$n_3^{\mathcal{W}_k} \cdot (r_1(x, y, \theta) - o_3^k \leq 0$	yes
$E_4^{\mathcal{W}_k}, r_2$	$n_4^{\mathcal{W}_k} \cdot (r_2(x, y, \theta) - o_4^k \leq 0$	yes

Figure F.7(b) shows a case in which the robot and obstacle do not intersect. The following table shows the possible type A and B contacts, the definitions of the corresponding half spaces, and whether or not the half space constraints are satisfied. As can be seen, one of the applicable half space constraints is not satisfied, and thus it is determined that the robot and obstacle are not in collision.

Contact pair	**half space inequality**	**satisfied?**
E_1^R, o_4^k	$n_1^R(\theta) \cdot \big(o_4^k - r_1(x, y, \theta)\big) \le 0$	no
E_2^R, o_1^k	$n_2^R(\theta) \cdot \big(o_1^k - r_2(x, y, \theta)\big) \le 0$	yes
E_3^R, o_3^k	$n_3^R(\theta) \cdot \big(o_3^k - r_3(x, y, \theta)\big) \le 0$	yes
$E_1^{\mathcal{W}_k}, r_3$	$n_1^{\mathcal{W}_k} \cdot \big(r_3(x, y, \theta) - o_1^k\big) \le 0$	yes
$E_2^{\mathcal{W}_k}, r_3$	$n_2^{\mathcal{W}_k} \cdot \big(r_3(x, y, \theta) - o_2^k\big) \le 0$	yes
$E_3^{\mathcal{W}_k}, r_1$	$n_3^{\mathcal{W}_k} \cdot \big(r_1(x, y, \theta) - o_3^k\big) \le 0$	yes
$E_4^{\mathcal{W}_k}, r_2$	$n_4^{\mathcal{W}_k} \cdot \big(r_2(x, y, \theta) - o_4^k\big) \le 0$	yes

Suppose the robot and obstacles are not convex (note, the case of a nonconvex obstacle includes the case of multiple disconnected obstacle regions in the workspace). In this case, one can always partition the robot and obstacle into collections of convex polygons, $\{R_l\}$ and $\{\mathcal{W}_k\}$, respectively. To determine if the robot and obstacle are in collision, we merely check to see if any pair R_l and $\mathcal{W}_k$ are in collision, using the method described above.

F.3 Configuration Space Obstacles in $\mathcal{Q} = \mathbb{R}^2$: The Star Algorithm

It is sometimes convenient to explicitly represent the configuration space obstacle region in the special case of $\mathcal{Q} = \mathbb{R}^2$ (e.g., when using the visibility graph approach described in section 5.1). For a convex robot and obstacle, it is straightforward to derive a boundary representation for the configuration space obstacle region using the ideas developed in the preceding section.

As described above, for each satisfied applicability condition, (F.3) or (F.5), one half space is defined by (F.4) or (F.6), respectively. To construct the representation of the boundary of the configuration space obstacle region, we need only find the vertices that are defined by the intersections of the lines that define these half spaces. The algorithm that we develop here, sometimes called the *star algorithm,* is a particularly efficient way to do so.

The heart of the algorithm lies in the following observations. When the applicability condition

$$(o_{j-1} - o_j) \cdot n_i^R(\theta) \ge 0 \quad \text{and} \quad (o_{j+1} - o_j) \cdot n_i^R(\theta) \ge 0$$

is satisfied and there is a contact between E_i^R and vertex o_j, of $\mathcal{W}$, this contact will be maintained as the robot translates, maintaining contact with the vertex. At one extreme of this motion, the vertices o_j and r_i coincide, while at the other extreme, vertices o_j and r_{i+1} coincide. These extremes define two vertices of the configuration space obstacle region

$$o_j - r_i(0, 0, \theta), \quad \text{and} \quad o_j - r_{i+1}(0, 0, \theta).$$

Analogously, when the applicability condition

$$(r_{i-1}(\theta) - r_i(\theta)) \cdot n_j^{\mathcal{W}} \geq 0 \quad \text{and} \quad (r_{i+1}(\theta) - r_i(\theta)) \cdot n_j^{\mathcal{W}} \geq 0$$

is satisfied and there is a contact between obstacle edge $E_j^{\mathcal{W}}$ and robot vertex r_i, this contact will be maintained as the robot translates, maintaining contact with the edge. At one extreme of this motion, the vertices o_j and r_i coincide, while at the other extreme, vertices o_{j+1} and r_i coincide. These extremes define two vertices of the configuration space obstacle region

$$o_j - r_i(0, 0, \theta), \quad \text{and} \quad o_{j+1} - r_i(0, 0, \theta).$$

The enumeration of satisfied applicability conditions can be made particularly efficient by recalling that these conditions can be expressed in terms of the orientations of the robot and obstacle edge normals. We first negate the edge normals of the robot, then sort the merged list of obstacle and negated robot edge normals by orientation. We then scan this sorted list, and construct the appropriate vertices each time a negated robot edge normal lies between adjacent obstacle edge normals, or vice versa.

We note here that the algorithm described above is an implementation of the *Minkowski difference,* a useful operation in many computational geometry applications. The Minkowski difference between the robot and a convex obstacle is defined by

$$\mathcal{WO} \ominus R(q) = \{q \in \mathcal{Q} : q = c - r \quad \text{where } r \in R(q) \quad \text{and} \quad c \in \mathcal{WO}\} \tag{F.7}$$

where $\ominus$ is the *Minkowski* difference operator [124].

F.4 Configuration Space Obstacles in $\mathcal{Q} = SE(2)$

As we have seen in chapter 3, a polygon in the plane has three degrees of freedom, two for translation and one for rotation, and its configuration space is $\mathcal{Q} = SE(2)$. Consider a polygonal robot in a workspace that contains a single obstacle. For a fixed orientation, the configuration space of the polygon is reduced to $\mathbb{R}^2$. Thus, one way to visualize this configuration space is to "stack" a set of two-dimensional configuration spaces, where each slice in the stack corresponds to the (x, y) configurations of the robot at a fixed orientation θ and the vertical axis represents the orientation of the robot. An example is shown in figure F.8.

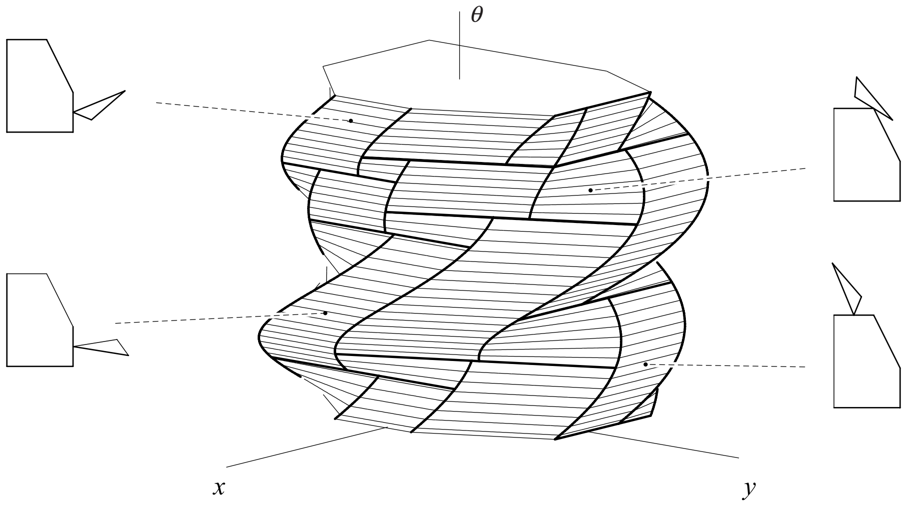

Figure F.8 The configuration space obstacle for a triangle-shaped robot in a workspace that contains a single, five-sided obstacle [69, 70].

F.5 Computing Distances between Polytopes in $\mathbb{R}^2$ and $\mathbb{R}^3$

In many applications, it is useful to know the minimum distance between two objects in addition to knowing whether or not they are in contact. We have seen in chapter 2 that knowledge of distance is essential for implementing the Bug family of algorithms. Moreover, minimum distance calculations are essential for collision detection, which is merely a special case of minimum distance calculations: if the minimum distance between two objects is zero, then they are in contact. In this section, we present an algorithm originally described by Gilbert, Johnson and Keerthi for computing the distance between convex polytopes, commonly referred to as the GJK distance computation algorithm [163].

We define the distance between polytopes A and B as

$$d(A, B) = \min_{a \in A, b \in B} \|a - b\|. \tag{F.8}$$

We reformulate (F.8) in terms of the Minkowski difference of two polytopes, i.e.,

$$A \ominus B = \{z \mid z = a - b, a \in A, b \in B\} = Z. \tag{F.9}$$

Using (F.9) we can rewrite (F.8) as

$$d(A, B) = \min_{a \in A, b \in B} \|a - b\| = \min_{z \in A \ominus B} \|z\|, \tag{F.10}$$

and we have reduced the problem of computing the distance between two polytopes to the problem of computing the minimum distance from one polytope to the origin.

In section F.3 we have seen an implementation of the Minkowski difference to construct the configuration space obstacle region. From this, it is easy to see that the Minkowski difference of two convex polytopes is itself a convex polytope. Since $Z = A \ominus B$ is a convex set, and since the norm, $\|z\|$, is a convex function, $z^* = \arg\min_{z \in Z} \|z\|$ is unique. Thus, there is a unique solution to (F.10). Note that the values of a and b that achieve this minimum are *not* necessarily unique.

Although finding the distance from Z to the origin may seem simpler than computing the distance between A and B, it should be noted that this is actually the case only if the necessary computations to determine $\min_{z \in Z} \|z\|$ are simpler than the computations required to compute $d(A, B)$ directly. This turns out to be the case for the GJK algorithm. Before we examine how this algorithm can be applied to the Minkowski difference of A and B, we first describe the algorithm for the case of computing the distance from Z to the origin, for Z any convex polytope.

Suppose Z is a polytope in $\mathbb{R}^n$ (i.e., $n = 2$ for polygons, and $n = 3$ for polyhedra). The GJK algorithm iteratively constructs a sequence of polytopes, each of which is the convex hull of some subset of the vertices of Z, such that at each iteration the distance from the origin to the new polytope decreases. Before describing the algorithm more formally, we define some useful terminology and notation.

The *convex hull* of a set of points in $\mathbb{R}^n$ is the smallest convex set in $\Re^n$ that contains those points. Efficient algorithms exist for computing the convex hull of general point sets, but for our purposes, we will not require such algorithms, since the GJK algorithm only deals with point sets of size three for polygons and size four for polyhedra. The convex hull of a set of three (noncollinear) points is the triangle defined by those points, and the convex hull of a set of four (noncoplanar) points is the tetrahedron defined by those points.

The GJK algorithm relies heavily on the notion of projection. In particular, for a convex set Z and a point x, the GJK algorithm computes the point $z \in Z$ with maximal projection onto x. The value of this projection operation is defined by

$$h_Z(x) = \max\{z \cdot x \mid z \in Z\}. \tag{F.11}$$

and the point z^* that achieves this maximum is defined by

$$s_Z(x) = z^* \quad s.t. \quad z^* \cdot x = h_Z(x). \tag{F.12}$$

The GJK algorithm for polygons is given as Algorithm 23 below. In the first step, the working vertex set V_0 is initialized to contain three arbitrarily selected vertices of the polygon, Z. At iteration k, the point x_k is determined as the point in the convex hull of the vertices in V_k that is nearest to the origin. Once x_k has been determined,

Algorithm 23 GJK Algorithm

Input: A polytope, $Z \subset \Re^2$.
Output: Minimal $\|z\|$, for $z \in Z \subset \Re^2$

```
1: $V_0 \leftarrow \{y_1, y_2, y_3\}$ with $y_i$ vertices of $Z$
2: $k \leftarrow 0$
3: Compute $x_k$, the point in the convex hull of $V_k$ that is nearest the origin, i.e., $x_k = \arg\min_{x \in hull(V_k)} \|x\|$.
4: Compute $h_Z(-x_k)$, and terminate if $\|x_k\| = h_Z(-x_k)$.
5: $z_k \leftarrow s_Z(-x_k)$, i.e., the projection of $z_k$ onto $x_k$ is nearer the origin than the projection onto $x_k$ of any other point in $Z$.
6: $x_k$ is contained in an edge of the convex hull of $V_k$. Let $V_{k+1}$ contain the two vertices that bound this edge and the point $z_k$.
7: $k \leftarrow k + 1$
8: Go to 3.
```

in step 5 a new vertex z_k is chosen as the vertex of the original polygon, Z, whose projection onto $-x_k$ is maximal. The point z_k then replaces a vertex in the current working vertex set to obtain a new working vertex set, V_{k+1}. The algorithm terminates (step 4) when x_k is itself the closest point in Z to the origin.

It is a fairly simple matter to extend the GJK algorithm (Algorithm 23) to the case in which $Z = A \ominus B$. Note that in the GJK algorithm, we never need an explicit representation of Z. We only need to compute two functions of Z: $h_Z(x)$ and $s_Z(x)$. Each of these can be computed without explicitly constructing Z. Let $Z = A \ominus B$. We can compute $h_{A \ominus B}(x)$ as follows,

$$\begin{aligned} h_{A\ominus B}(x) &= \max\{z \cdot x \mid z \in Z\} \\ &= \max\{z \cdot x \mid z \in A \ominus B\} \\ &= \max\{(a-b) \cdot x \mid a \in A, b \in B\} \\ &= \max\{a \cdot x - b \cdot x \mid a \in A, b \in B\} \\ &= \max\{a \cdot x \mid a \in A\} - \min\{b \cdot x \mid b \in B\} \\ &= \max\{a \cdot x \mid a \in A\} + \max\{b \cdot (-x) \mid b \in B\} \\ &= h_A(x) + h_B(-x). \end{aligned} \tag{F.13}$$

Now, suppose that a^* achieves the value $h_A(x)$ and b^* achieves the value $h_B(-x)$. Then $z^* = a^* - b^*$, and therefore we have

$$s_{A\ominus B}(x) = s_A(x) - s_B(-x). \tag{F.14}$$

Thus, we see that the GJK algorithm is easily extended to the case of the Minkowski difference of convex polygons. In steps 4 and 5, merely replace h_z and s_Z with the expressions (F.13) and (F.14). To extend the algorithm to convex polyhedra, merely replace step 1 of the algorithm by

1. $V_0 \leftarrow \{y_1, y_2, y_3, y_4\}$ with y_i vertices of Z

and step 6 of the algorithm by

6. x_k is contained in a face of the convex hull of V_k. Let V_{k+1} contain the three vertices that bound this face and the point z_k.