$$\phi_2(q_0, q_1, q_2, q_3) = \left(\frac{q_0}{|q_2|}, \frac{q_1}{|q_2|}, \frac{q_3}{|q_2|}\right)$$
$$\phi_3(q_0, q_1, q_2, q_3) = \left(\frac{q_0}{|q_3|}, \frac{q_1}{|q_3|}, \frac{q_2}{|q_3|}\right)$$

As we have seen above, $R_i \in SO(3)$ represents a rotation, and the composition of successive rotations, say $R_1$ and $R_2$, is represented by the rotation matrix $R = R_1 R_2$. Likewise, multiplication of quaternions corresponds to the composition of successive rotations. In particular, if $Q_1$ and $Q_2$ are two quaternions representing a rotation by $\theta_1$ about axis $n_1$ and a rotation by $\theta_2$ about axis $n_2$, respectively, then the result of performing these two rotations in succession is represented by the quaternion $Q = Q_1 Q_2$. Using (E.22) through (E.25) it is straightforward to determine the quaternion product. In particular, for two quaternions, $X$ and $Y$, we compute their product, $Z = XY$, as

$$\begin{aligned} z_0 + iz_1 + jz_2 + kz_3 &= (x_0 + ix_1 + jx_2 + kx_3)(y_0 + iy_1 + yx_2 + yx_3) \\ &= x_0y_0 - x_1y_1 - x_2y_2 - x_3y_3 \\ &\quad + i(x_0y_1 + x_1y_0 + x_2y_3 - x_3y_2) \\ &\quad + j(x_0y_2 + x_2y_0 + x_3y_1 - x_1y_3) \\ &\quad + k(x_0y_3 + x_3y_0 + x_1y_2 - x_2y_1). \end{aligned}$$

By equating the real parts on both sides of the final equality, and by equating the coefficients of $i$, $j$, and $k$ on both sides of the final equality, we obtain

$$\begin{aligned} z_0 &= x_0y_0 - x_1y_1 - x_2y_2 - x_3y_3 \\ z_1 &= x_0y_1 + x_1y_0 + x_2y_3 - x_3y_2 \\ z_2 &= x_0y_2 + x_2y_0 + x_3y_1 - x_1y_3 \\ z_3 &= x_0y_3 + x_3y_0 + x_1y_2 - x_2y_1. \end{aligned}$$

The quaternion $Q = (q_0, q_1, q_2, q_3)$ can be thought of as having the scalar component $q_0$ and the vector component $q = [q_1, q_2, q_3]^T$. Therefore, one often represents a quaternion by a pair, $Q = (q_0, q)$. Using this notation, $q_0$ represents the real part of $Q$, and $q$ represents the imaginary part of $Q$. Using this notation, the quaternion product $Z = XY$ can be represented more compactly as

$$\begin{aligned} z_0 &= x_0y_0 - x^T y \\ z &= x_0 y + y_0 x + x \times y, \end{aligned}$$

in which $\times$ denotes the vector cross product operator.

For complex numbers, the conjugate of $a + ib$ is defined by $a - ib$. Similarly, for quaternions we denote by $Q^*$ the conjugate of the quaternion $Q$, and define

$$Q^* = (q_0, -q_1, -q_2, -q_3). \tag{E.31}$$

With regard to rotation, if the quaternion $Q$ represents a rotation by $\theta$ about the axis $n$, then its conjugate $Q^*$ represents a rotation by $\theta$ about the axis $-n$. It is easy to see that

$$QQ^* = \left(q_0^2 + ||q||^2, 0, 0, 0\right) \tag{E.32}$$

and that

$$||QQ^*|| = \left|\left|\left(q_0^2 + q_1^2 + q_2^2 + q_3^2, 0, 0, 0\right)\right|\right| = \sum q_i^2 = ||Q||^2. \tag{E.33}$$

A quaternion, $Q$, with its conjugate, $Q^*$, can be used to perform coordinate transformations. Let the point $p$ be rigidly attached to a coordinate frame $\mathcal{F}$, with local coordinates $(x, y, z)$. If $Q$ specifies the orientation of $\mathcal{F}$ with respect to the base frame, and $T$ is the vector from the world frame to the origin of $\mathcal{F}$, then the coordinates of $p$ with respect to the world frame are given by

$$Q(0, x, y, z)Q^* + T, \tag{E.34}$$

in which $(0, x, y, z)$ is a quaternion with zero as its real component. Quaternions can also be used to transform vectors. For example, if $n = (n_x, n_y, n_z)$ is the normal vector to the face of a polyhedron, then if the polyhedron is rotated by $Q$, the new direction of the normal is given by

$$Q(0, n_x, n_y, n_z)Q^*. \tag{E.35}$$

# 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

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

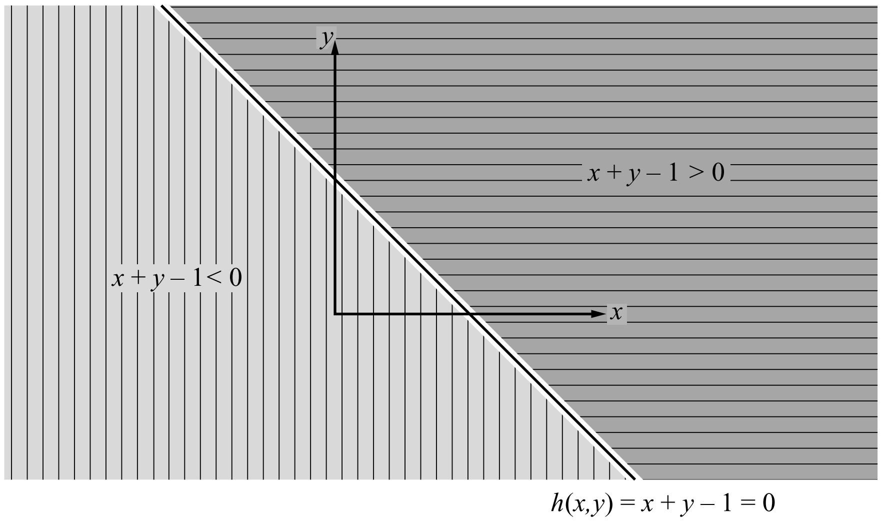


**Figure F.1** Half-planes defined by $h(x, y) = x + y - 1$.

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

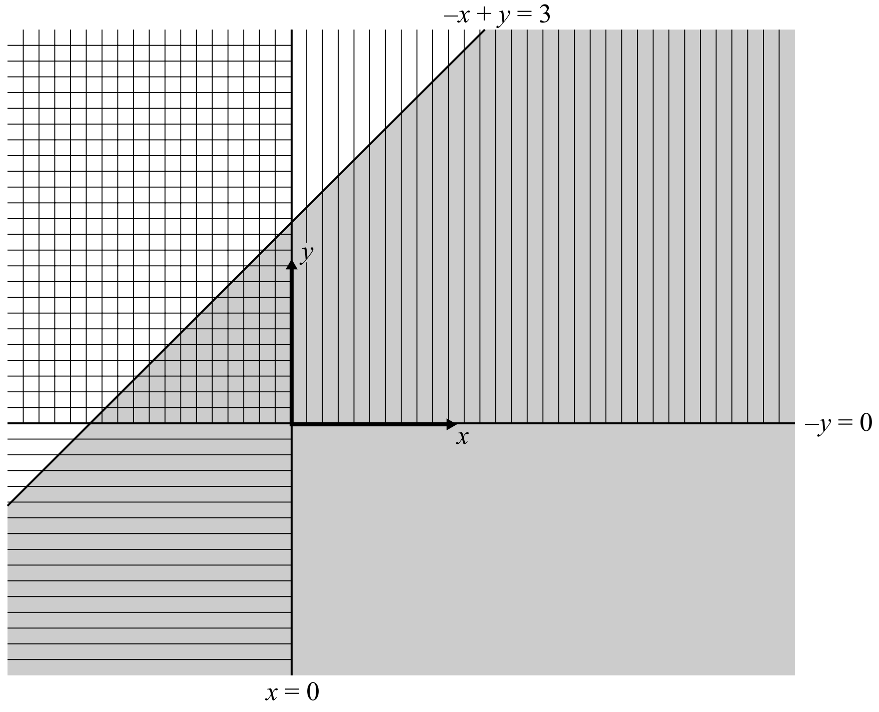


**Figure F.2** A convex polygonal region constructed from the half planes $h_1^-$, $h_2^-$, and $h_3^-$.

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, $\theta$. 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 $\theta$, 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 $k$th obstacle, this condition can be expressed by

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

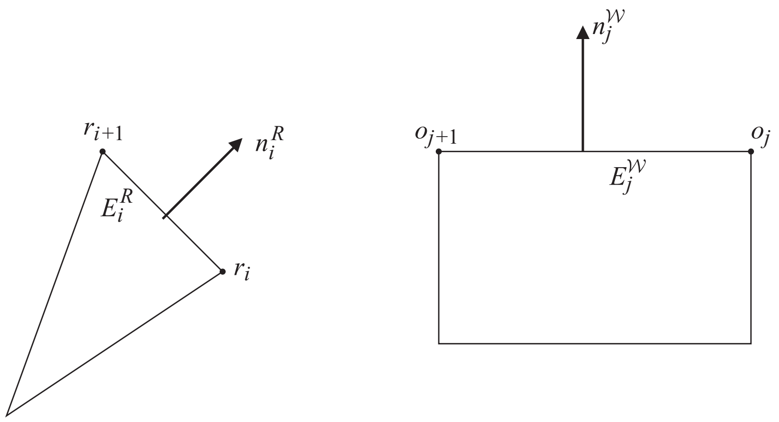


**Figure F.3** Notation used to define vertices, normals and edges of the robot and obstacle polygon.

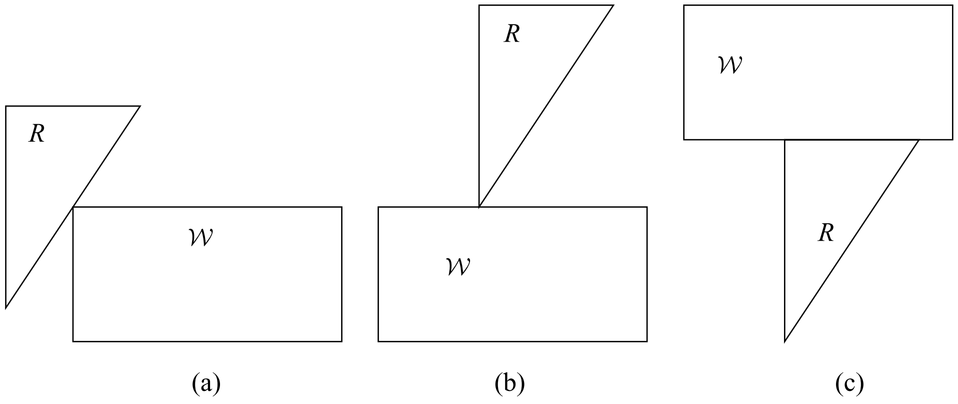


**Figure F.4** (a) Type A contact, (b) Type B contact, (c) Both Type A and Type B contact.

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 $\theta$. In particular, such contact can occur only when $\theta$ 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 $\theta$, 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.

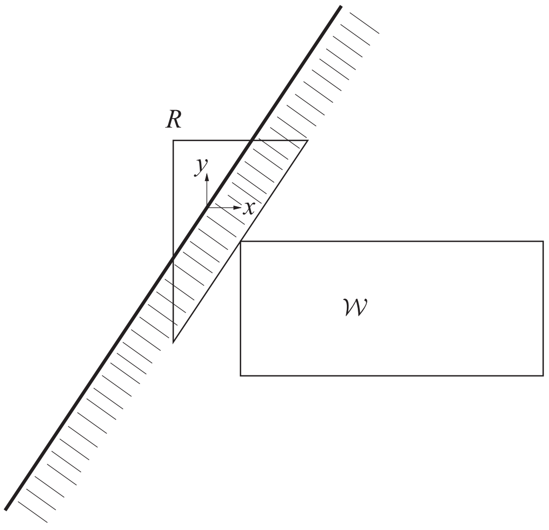


**Figure F.5** The half space defined by this contact is below the thick black line that passes through the origin of the robot's coordinate frame.

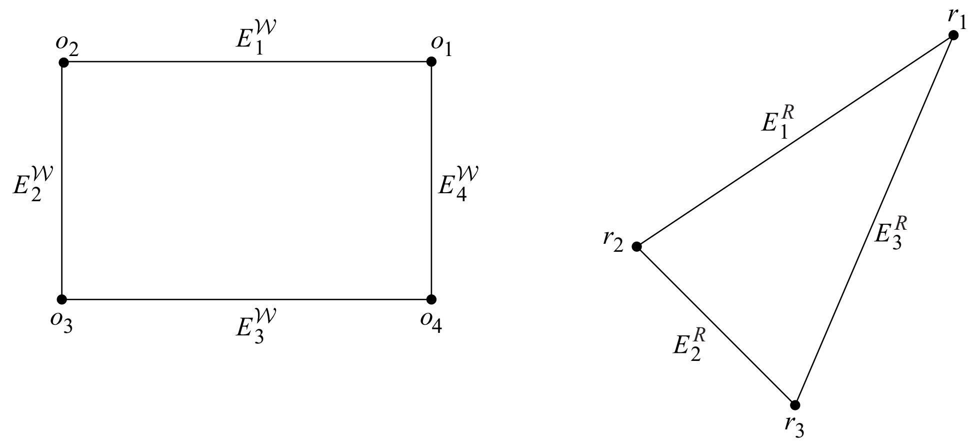


**Figure F.6** The obstacle is shown on the left, and the robot on the right.

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 $(x, y, \theta)$ 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.

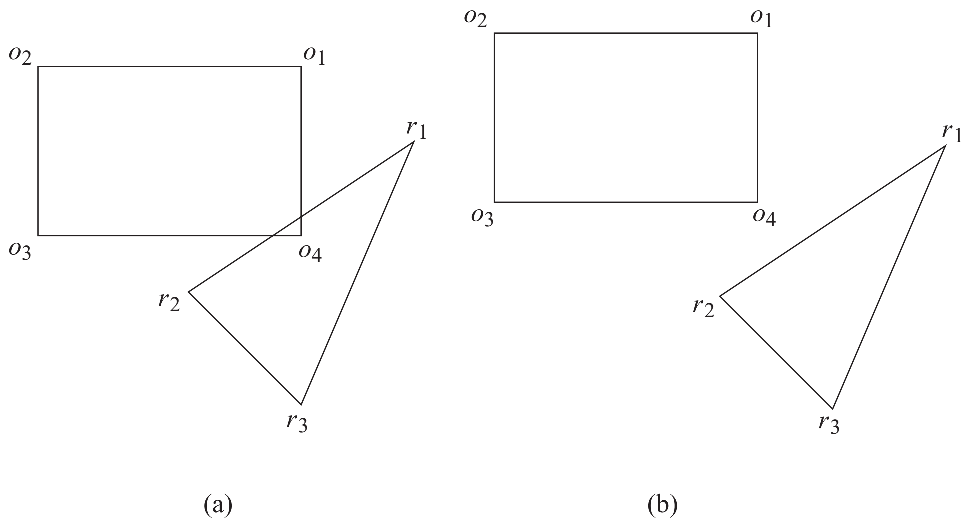


**Figure F.7** The applicability conditions and half spaces for these two cases are shown in the table below.

| **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?** |
|---|---|---|
| $E_1^R, o_4^k$ | $n_1^R(\theta) \cdot \left(o_4^k - r_1(x, y, \theta)\right) \leq 0$ | no |
| $E_2^R, o_1^k$ | $n_2^R(\theta) \cdot \left(o_1^k - r_2(x, y, \theta)\right) \leq 0$ | yes |
| $E_3^R, o_3^k$ | $n_3^R(\theta) \cdot \left(o_3^k - r_3(x, y, \theta)\right) \leq 0$ | yes |
| $E_1^{\mathcal{W}_k}, r_3$ | $n_1^{\mathcal{W}_k} \cdot \left(r_3(x, y, \theta) - o_1^k\right) \leq 0$ | yes |
| $E_2^{\mathcal{W}_k}, r_3$ | $n_2^{\mathcal{W}_k} \cdot \left(r_3(x, y, \theta) - o_2^k\right) \leq 0$ | yes |
| $E_3^{\mathcal{W}_k}, r_1$ | $n_3^{\mathcal{W}_k} \cdot \left(r_1(x, y, \theta) - o_3^k\right) \leq 0$ | yes |
| $E_4^{\mathcal{W}_k}, r_2$ | $n_4^{\mathcal{W}_k} \cdot \left(r_2(x, y, \theta) - o_4^k\right) \leq 0$ | 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

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

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 $\theta$ and the vertical axis represents the orientation of the robot. An example is shown in figure F.8.

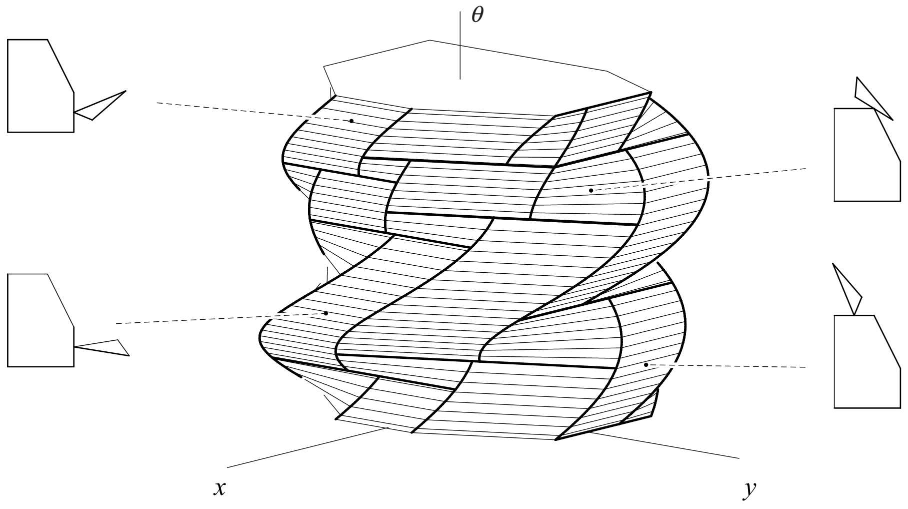


**Figure F.8** The configuration space obstacle for a triangle-shaped robot in a workspace that contains a single, five-sided obstacle [69, 70].

## 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,

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

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

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

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 $z_k$.

# G *Analysis of Algorithms and Complexity Classes*

## G.1 Running Time

Yet another way to study an algorithm is to compute the running time of the algorithm purely as a function of the length of the input. *Worst-case analysis* considers the longest running time of all inputs of a particular length. *Average-time analysis* considers the average of all the running times of inputs of a particular length. The worst-case analysis is typically referred to as the *running time* of an algorithm.

Finding an expression for the exact running time is often difficult, but in most cases close estimations are possible. *Asymptotic analysis* provides the means of analyzing the running time of the algorithms for large inputs. As the lengths of the inputs become large, the high-order terms dominate the value of the expression and the low-order terms have little or no effect. Hence, a close approximation can be obtained only by considering the highest-order term in an expression. For example, the highest-order term of the function $f(n) = 2n^5 + 100n^3 + 27n + 2003$ is $2n^5$. As $n$ becomes large, disregarding the coefficient 2, the function $f(n)$ behaves like $n^5$ and it is said that $f(n)$ is asymptotically at most $n^5$. The following are some common definitions that are useful in analyzing the asymptotic behavior of functions and hence algorithms.

DEFINITION G.1.1 *Let $f$ and $g$ be two functions $f, g : \mathbb{N} \to \mathbb{R}^+$.*

- *The function $g$ is an* asymptotically upper bound *for $f$, denoted $f(n) \in O(g(n))$ and read $f$ is big-O of $g$, if there* exists *a constant $c > 0$ and $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$,*

  $$f(n) \leq cg(n).$$

- *The function $g$ is an* asymptotically strict upper bound *for $f$, denoted $f(n) \in o(g(n))$ and read $f$ is small-o of $g$, if for* every *constant $c > 0$ and $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$,*

  $$f(n) < cg(n).$$

- *The function $g$ is an* asymptotically lower bound *for $f$, denoted $f(n) \in \Omega(g(n))$ and read $f$ is big-Omega of $g$, if there* exists *a constant $c > 0$ and $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$,*

  $$cf(n) \geq g(n).$$

- *The function $g$ is an* asymptotically strict lower bound *for $f$, denoted $f(n) \in \omega(g(n))$ and read $f$ is small-omega of $g$, if for* every *constant $c > 0$ and $n_0 \in \mathbb{N}$ such that $\forall n \geq n_0$,*

  $$f(n) > cg(n).$$

- *The function $g$ is* asymptotically equal *to $f$, denoted $f(n) \in \Theta(g(n))$ and read $f$ is theta of $g$, if*

  $$f(n) \in O(g(n)) \quad \textit{and} \quad f(n) \in \Omega(g(n)).$$

Considering only the highest terms and disregarding constant factors, the big-O notation says that the function $f$ is no more than the function $g$. The big-O notation is thought of as suppressing a constant factor. For example, $f(n) = 5n^4 + 7n^3 - 4n^2 \in O\left(n^4\right)$, $f(n) = n^{\log n} + n^{100} \in O\left(n^{\log n}\right)$, etc. When $f$ is strictly less than $g$, the small-o notation is used. The small o-notation indicates that the function $g$ grows much faster than the function $f$. For example, $f(n) = \log^{100} n \in o\left(n^{1/100}\right)$, $f(n) = n^{40} \in o(2^n)$, etc. The notations $\Omega$ and $\omega$ express the opposite of $O$ and $o$ notations, respectively. Thus, the big-Omega notation indicates that $f$ grows no slower than $g$, and the small-omega notation indicates that $f$ grows faster than $g$. When the functions $f$ and $g$ grow at the same rate, the $\Theta$ notation is used. For example, $f(n) = 3n^5 + n^4 \in \Theta\left(n^5\right)$.

When describing the running time of different algorithms, certain terms come up frequently. The running times of common algorithms such as matrix multiplication, sorting, shortest path, etc., are $O(n^c)$, where $c$ is some positive constant. In such cases, it is said that the running time is polynomial in the length of the input $n$. Other algorithms, such as satisfiability of Boolean expressions, Hamiltonian paths, decomposition of integers into prime factors, etc., are $O(2^{n^c})$, where $c$ is some positive constant. Such algorithms are said to be running in exponential time. The following table summarizes some of the common characterizations of the running time of algorithms ($c$ is some positive constant).

| | **Running time** |
|---|---|
| constant | $O(1)$ |
| logarithmic | $O(\log n)$ |
| polylogarithmic | $O(\log^c n)$ |
| linear | $O(n)$ |
| polynomial | $O(n^c)$ |
| quasipolynomial | $O\left(n^{\log^c n}\right)$ |
| exponential | $O\left(2^{n^c}\right)$ |
| doubly expnonential | $O\left(2^{2^{n^c}}\right)$ |

## G.2 Complexity Theory

The goal of *complexity theory* is to characterize the amount of resources needed for the computation of specific problems. Common resources include sequential time, sequential space, number of gates in Boolean circuits, parallel time in a multiprocessor machine, etc. The exact complexity of a problem is determined by the amount of resources that is both sufficient and necessary for its solution. Sufficiency implies an upper bound on the amount of resources needed to solve the problem for every instance of the input. Necessity implies a lower bound, i.e., for some instance of the input, at least a certain amount of resources is required to solve the problem.

The amount of resources that is needed to solve a problem allows for an elegant classification of problems according to their computational complexity. Researchers have developed the notion of *complexity classes*, where a complexity class is defined by specifying (a) the type of computation model M, (b) the resource R which is measured in this model, and (c) an upper bound U on this resource. A complexity class, then, consists of all problems requiring at most an amount U of resource R for their solution in the model M. Thus, the *complexity of a problem* is determined by finding to which complexity classes it belongs (by providing upper bounds on the resource) and to which complexity classes it does not belong (by providing lower bounds). To define complexity classes more precisely, we will need to make use of definitions of alphabets, strings, and languages.

## Input Representation

The amount of the resource used in a complexity class is expressed in terms of the length of the input. It is not clear, however, how to define the length of the input since it can be of different types and values, i.e., integers, names, graphs, matrices, etc. It is convenient to have a unique and clear definition of the length of the input. To this end, researchers have proposed the encoding of inputs as strings over a set of symbols and have defined the length of the input as the number of symbols of the encoding string.

DEFINITION G.2.1 *An alphabet, usually denoted by $\Sigma$, is any finite set of symbols.*

DEFINITION G.2.2 *A string $s$ over an alphabet $\Sigma$ is a sequence of symbols from $\Sigma$. The length of a string $s$, denoted $|s|$, is equal to the number of its symbols. The set of all strings over the alphabet $\Sigma$ is denoted by $\Sigma^*$.*

The encoding of an input $a$ is denoted by $\text{enc}(a)$. To illustrate, let $\Sigma = \{0, 1\}$. Then, integers can be encoded in standard binary form, e.g., the encodings of 5 and 35 are 101 and 100001 of lengths 3 and 6, respectively. The encoding of a graph $G = (V, E)$, where $V = \{v_1, \ldots, v_n\} \subset \mathbb{N}$ and $E = \{(v'_1, v''_1), \ldots, (v'_m, v''_m)\} \subseteq V \times V$, can be obtained by concatenating the encodings of its vertex set and its edge set. The vertex set and the edge set can be encoded by concatenating the encodings of the vertices and of the edges, respectively. Special markers can be used to indicate the ending of a vertex and edge encoding. Thus, $\text{enc}(G) = \text{enc}(v_1) \circ \text{enc}(*) \circ \cdots \circ \text{enc}(v_n) \circ \text{enc}(*) \circ \text{enc}(+) \circ \text{enc}(v'_1) \circ \text{enc}(*) \circ \text{enc}(v''_1) \circ \text{enc}(*) \circ \cdots \circ \text{enc}(v'_m) \circ \text{enc}(*) \circ \text{enc}(v''_m) \circ \text{enc}(*)$, where $\text{enc}(*)$ and $\text{enc}(+)$ are the encodings of special markers used to separate vertices and indicate the start of the edge encodings, respectively, and $\circ$ denotes concatenation.

## Problem Abstraction

A problem can be thought of as mapping an input instance to a solution. In many cases, we are interested in problems whose solution is either "yes" or "no." Such problems are known as *decision problems*. For example, the graph-coloring problem asks whether it is possible to color the vertices of a graph $G = (V, E)$ using $k$ different colors such that no two vertices connected by an edge have the same color. In many other cases, we are interested in finding the best solution according to some criteria. Such problems are known as *optimization problems*. To continue our example, we may be interested in determining the minimum number of colors needed to color a graph. Generally, an optimization problem can be cast as a decision problem by imposing an upper bound. In our example, we can determine the minimum number of colors needed to color a graph $G = (V, E)$ by invoking the corresponding decision problem with $k = 1, \ldots, |V|$ until the answer to the decision problem is "yes."

In the rest of the section, we restrict our attention to decision problems since their definition is more amendable to complexity analysis and since other problems can be cast as decision problems.

### Languages

Languages provide a convenient framework for expressing decision problems.

DEFINITION G.2.3 *A language $L$ over an alphabet $\Sigma$ is a set of strings over the alphabet $\Sigma$, i.e., $L \subseteq \Sigma^*$.*

The language defined by a decision problem includes all the input instances whose solution is "yes." For example, the graph-coloring problem defines the language whose elements are all the encodings of graphs that can be colored using $k$ colors.

### Acceptance of Languages

An algorithm $A$ accepts a string $s \in \Sigma^*$ if the output of the algorithm $A(s)$ is "yes." The string $s$ is rejected by the algorithm if its output $A(s)$ is "no." The language $L$ accepted by an algorithm $A$ is the set of strings accepted by the algorithm, i.e.,

$L = \{s : s \in \Sigma^* \text{ and } A(s) = \text{"yes"}\}$.

Note that even if $L$ is the language accepted by the algorithm $A$, given some input string $s \notin L$, the algorithm will not necessarily reject $s$. It may never be able to determine that $s \notin L$ and thus loop forever. Language $L$ is *decided* by an algorithm $A$ if for every string $s \in \Sigma^*$, $A$ accepts $s$ if $s \in L$ and $A$ rejects $s$ if $s \notin L$. If $L$ is decided by $A$, it guarantees that on any input string the algorithm will terminate.

DEFINITION G.2.4 *Let $t : \mathbb{N} \to \mathbb{N}$ be a function. An algorithm $A$* decides a language *$L$ over some alphabet $\Sigma$ in time $O(t(n))$ if for every string $s$ of length $n$ over $\Sigma$, the algorithm $A$ in $O(t(n))$ steps accepts $s$ if $s \in L$ or rejects $s$ if $s \notin L$. Language $L$ is decided in time $O(t(n))$.*

### Complexity Classes

We are now ready to define some of the most important complexity classes. We start with the definition of the polynomial-time complexity class.

DEFINITION G.2.5 *A language $L$ is in* P *if there exists a polynomial-time algorithm $A$ that decides $L$.*

The complexity class P encompasses a wide variety of problems such as sorting, shortest path, Fourier transform, etc. Roughly speaking, P corresponds to all the problems that admit an efficient algorithm. Generally, we think of problems that are solvable by polynomial time algorithms as being tractable, or easy, and problems that require superpolynomial time as being intractable, or hard.

Indeed, for many problems there are no polynomial-time algorithms. For example, deciding whether or not a graph $G = (V, E)$ can be colored with three colors is not known to be in P. These problems can be solved by brute-force algorithms in exponential time.

DEFINITION G.2.6 *A language $L$ is in* EXPTIME *if there exists an exponential-time algorithm $A$ that decides $L$.*

Interestingly enough, many of these hard problems have a feature that is called polynomial-time verifiability. That is, although currently it is not possible to solve these problems in polynomial time, if a candidate solution to the problem, called a certificate, is given, the correctness of the solution can be verified in polynomial time. For example, a certificate for the graph-coloring problem with three colors would be a mapping that for each vertex indicates its color. The correctness can be verified in polynomial time by examining all the edges and checking for each edge that the colors of its two vertices are different. This observation is captured by the following definition.

DEFINITION G.2.7 *A language $L$ is in* NP *if there exists a polynomial-time verifier algorithm $A$ and a constant $c$ such that for every string $s$ there exists a certificate $y$ of length $O(|s|^c)$ such that $A(s, y) =$ "yes" if $s \in L$ and $A(s, y) =$ "no" if $s \notin L$.*

It is clear that $\mathrm{P} \subseteq \mathrm{NP}$ since any language that can be decided in polynomial time can also be decided without the need of a certificate. The most fundamental question in complexity theory is whether $\mathrm{P} \subset \mathrm{NP}$ or $\mathrm{P} = \mathrm{NP}$. After many years of extensive research the question remains unanswered. An important step was made in the 70s when Cook and Levin related the complexity of certain NP problems to the complexity of all NP problems. They were able to prove that if a polynomial-time algorithm existed for one of these problems, then a polynomial-time algorithm could be constructed for any NP problem. These special problems form an important complexity class known as NP-complete.

DEFINITION G.2.8 *A language $L_1$ is* polynomial time reducible *to a language $L_2$, denoted $L_1 \leq_p L_2$, if there exists a polynomial time computable function $f : \Sigma^* \to \Sigma^*$, such that for every $s \in \Sigma^*$,*

$$s \in L_1 \iff f(s) \in L_2.$$

*$f$ is called the reduction function and the algorithm $F$ that computes $f$ is called the reduction algorithm.*

If a language $L_1$ is reducible to a language $L_2$ via some polynomial-time computable function $f$, and if $L_2$ has a polynomial-time algorithm $A_2$, then we can construct a polynomial-time algorithm $A_1$ for $L_1$. Given some input string $s$, algorithm $A_1$ invokes $F$ to compute $f(s)$ and then invokes $A_2$ on $f(s)$ and gives the same answer as $A_2$. Thus, via reductions, the solution of one problem can be used to solve other problems.

DEFINITION G.2.9 *A language $L$ is in* NP-complete *if*

*1.* $L \in$ NP, *and*

*2. if* $L' \in$ NP, *then* $L' \leq_p L$.

*If $L$ satisfies the second condition, but not necessarily the first condition, then $L$ is* NP-hard.

It is clear now that if an NP-complete problem has a polynomial-time algorithm, then via reductions it is possible to construct a polynomial-time algorithm for any problem in NP. This would imply that P $=$ NP.

In addition to time, another common resource of interest is space. Using the same framework, complexity classes can be defined based on the amount of space the algorithms use to solve problems.

DEFINITION G.2.10 *Let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a function. An algorithm $A$ decides a language $L$ over some alphabet $\Sigma$ in space $O(t(n))$ if for every string $s$ of length $n$ over $\Sigma$, the algorithm $A$ using at most $O(t(n))$ space accepts $s$ if $s \in L$ or rejects $s$ if $s \notin L$. The language $L$ is decided in space $O(t(n))$.*

DEFINITION G.2.11 *A language $L$ is in* PSPACE *if there exists a polynomial-space algorithm $A$ that decides $L$.*

DEFINITION G.2.12 *A language $L$ is in* PSPACE-complete *if*

*1.* $L \in$ PSPACE, *and*

*2. if* $L' \in$ PSPACE, *then* $L' \leq_p L$.

*If $L$ satisfies the second condition, but not necessarily the first condition, then $L$ is* PSPACE-hard.

It can be easily shown that the relationship between the different complexity classes that have been defined in this section is as follows:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}.$$

## G.3 Completeness

When describing robotics algorithms in this book, several notions of "completeness" are used.

DEFINITION G.3.1 *An algorithm A is* complete *if in a finite amount of time, A always finds a solution if a solution exists or otherwise A determines that a solution does not exist.*

DEFINITION G.3.2 *An algorithm A is* resolution complete *if in a finite amount of time and for some small resolution step* $\epsilon > 0$*, A always finds a solution if a solution exists or otherwise A determines that a solution does not exist.*

DEFINITION G.3.3 *An algorithm A is* probabilistically complete *if the probability of finding a solution, if a solution exists, converges to* 1*, when the running time approaches infinity.*

Complete algorithms include many common algorithms such as $A^*$, shortest-path, scheduling problems, etc. Resolution complete algorithms have to approximate a continuous measure by discretizing it at small steps. Ray tracing from graphics algorithms and sampling-based planning algorithms that use a grid representation of the configuration space are examples of resolution complete algorithms. Probabilistic completeness guarantees that given enough time, a solution will be found (if a solution exists). If a solution does not exist, the algorithm may not be able to necessarily detect this fact and thus runs forever. In practice, probabilistic algorithms terminate and declare failure if an upper bound on the amount of time the algorithm could use has passed and a solution has not been found. The basic Probabilistic RoadMap planner (`PRM`) is an example of a probabilistically complete algorithm. Such an algorithm trades completeness for efficiency; in many cases, for the same problem, a probabilistically complete algorithm will find a solution faster (if a solution exists) than a complete algorithm.

# H Graph Representation and Basic Search

## H.1 Graphs

A graph is a collection of *nodes* and *edges,* i.e., $G = (V, E)$. See figure H.1. Sometimes, another term for a node is *vertex,* and this chapter uses the two terms interchangeably. We use $G$ for graph, $V$ for vertex (or node), and $E$ for edge. Typically in motion planning, a node represents a salient location, and an edge connects two nodes that correspond to locations that have an important relationship. This relationship could be that the nodes are mutually accessible from each other, two nodes are within line of sight of each other, two pixels are next to each other in a grid, etc. This relationship does not have to be mutual: if the robot can traverse from nodes $V_1$ to $V_2$, but not from $V_2$ to $V_1$, we say that the edge $E_{12}$ connecting $V_1$ and $V_2$ is directed. Such a collection of nodes and edges is called a *directed graph*. If the robot can travel from $V_1$ to $V_2$ and vice versa, then we connect $V_1$ and $V_2$ with two directed edges $E_{12}$ and $E_{21}$. If for each vertex $V_i$ that is connected to $V_j$, both $E_{ij}$ and $E_{ji}$ exist, then instead of connecting $V_i$ and $V_j$ with two directed edges, we connect them with a single undirected edge. Such a graph is called an *undirected graph.* Sometimes, edges are annotated with a non-negative numerical value reflective of the costs of traversing this edge. Such values are called *weights.*

A *path* or *walk* in a graph is a sequence of nodes $\{V_i\}$ such that for adjacent nodes $V_i$ and $V_{i+1}$, $E_{i\ i+1}$ exists (and thus connects $V_i$ and $V_{i+1}$). A graph is *connected* if for all nodes $V_i$ and $V_j$ in the graph, there exists a path connecting $V_i$ and $V_j$. A *cycle* is a

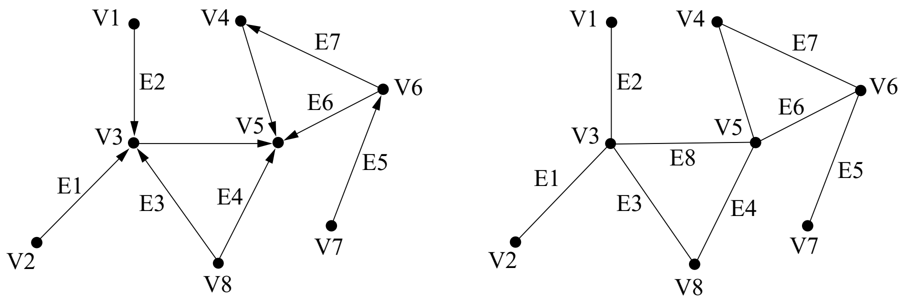


**Figure H.1** A graph is a collection of nodes and edges. Edges are either directed (left) or undirected (right).

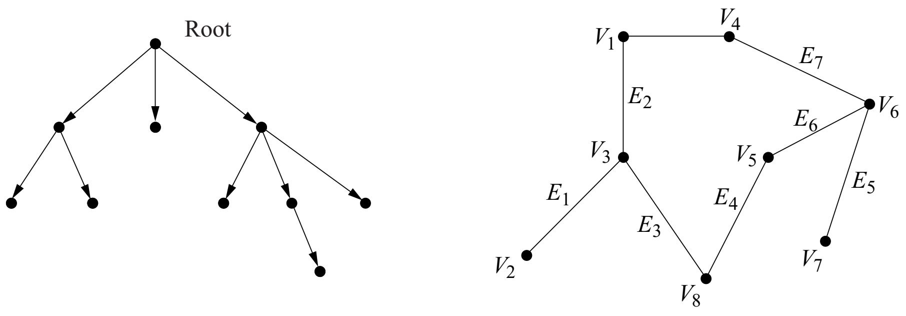


**Figure H.2** A tree is a type of directed acyclic graph with a special node called the *root*. A cycle in a graph is a path through the graph that starts and ends at the same node.

path of $n$ vertices such that first and last nodes are the same, i.e., $V_1 = V_n$ (figure H.2). Note that the "direction" of the cycle is ambiguous for undirected graphs, which in many situations is sufficient. For example, a graph embedded in the plane can have an undirected cycle which could be both clockwise and counterclockwise, whereas a directed cycle can have one orientation.

A *tree* is a connected directed graph without any cycles (figure H.2). The tree has a special node called the *root*, which is the only node that possesses no incoming arc. Using a parent-child analogy, a parent node has nodes below it called children; the root is a parent node but cannot be a child node. A node with no children is called a *leaf*. The removal of any nonleaf node breaks the connectivity of the tree.

Typically, one searches a tree for a node with some desired properties such as the goal location for the robot. A *depth-first search* starts at the root, chooses a child, then that node's child, and so on until finding either the desired node or a leaf. If the search encounters a leaf, the search then backs up a level and then searches through an unvisited child until finding the desired node or a leaf, repeating this process until the desired node is found or all nodes are visited in the graph (figure H.3).

Breadth-first search is the opposite; the search starts at the root and then visits all of the children of the root first. Next, the search then visits all of the grandchildren, and so forth. The belief here is that the target node is near the root, so this search would require less time (figure H.3).

A grid induces a graph where each node corresponds to a pixel and an edge connects nodes of pixels that neighbor each other. Four-point connectivity will only have edges to the north, south, east, and west, whereas eight-point connectivity will have edges to all pixels surrounding the current pixel. See figure H.4.

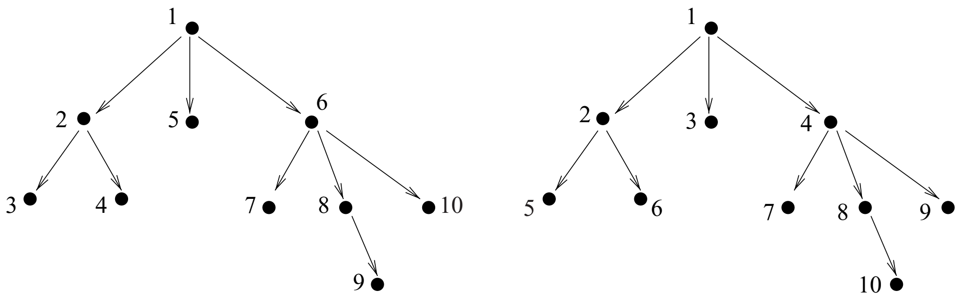


**Figure H.3** Depth-first search vs. breadth-first search. The numbers on each node reflect the order in which nodes are expanded in the search.

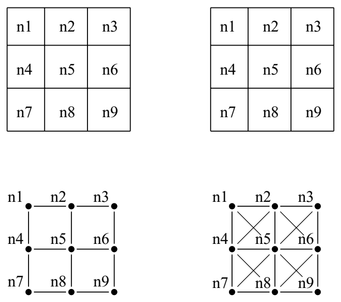


**Figure H.4** Four-point connectivity assumes only four neighbors, whereas eight-point connectivity has eight.

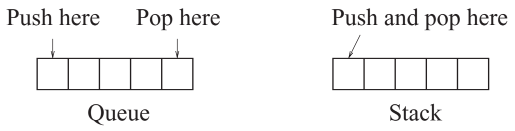


**Figure H.5** Queue vs. stack.

As can be seen, the graph that represents the grid is not a tree. However, the breadth-first and depth-first search techniques still apply. Let the *link length* be the number of edges in a path of a graph. Sometimes, this is referred to as edge depth. Link length differs from path length in that the weights of the edges are ignored; only the number of edges count. For a general graph, breadth-first search considers each of the nodes that are the same link length from the start node before going onto child nodes. In contrast, depth-first search considers a child first and then continues through the children successively considering nodes of increasing link length away from the start node until it reaches a childless or already visited node (i.e., a cycle). In other words, termination of one iteration of the depth-first search occurs when a node has no unvisited children.

The wave-front planner (chapter 4, section 4.5) is an implementation of a breadth-first search. Breadth-first search, in general, is implemented with a list where the children of the current node are placed into the list in a first-in, first-out (FIFO) manner. This construction is commonly called a *queue* and forces all nodes of the same linklength from the start to be visited first (figure H.5). The breadth-first search starts with placing the start node in the queue. This node is then *expanded* by it being popped off (i.e., removed from the front) the queue and all of its children being placed onto it. This procedure is then repeated until the goal node is found or until there are no more nodes to expand, at which time the queue is empty. Here, we expand all nodes of the same level (i.e., link length from the start) first before expanding more deeply into the graph.

Figure H.6 displays the resulting path of breadth-first search. Note that all paths produced by breadth-first search in a grid with eight-point connectivity are optimal with respect to the "eight-point connectivity metric." Figure H.7 displays the link lengths for all shortest paths between each pixel and the start pixel in the free space in Figure H.6. A path can then be determined using this information via a gradient descent of link length from the goal pixel to the start through the graph as similarly done with the wavefront algorithm.

Depth-first search contrasts breadth-first search in that nodes are placed in a list in a last-in, first-out (LIFO) manner. This construction is commonly called a *stack* and forces nodes that are of greater and greater link length from the start node to be

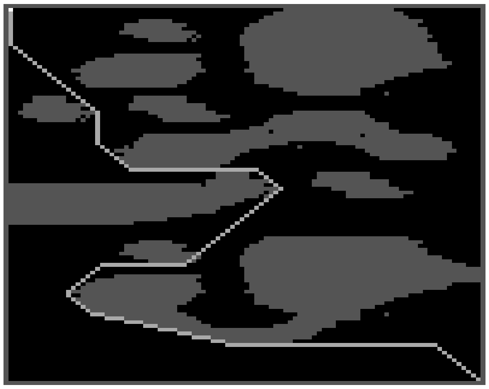


**Figure H.6** White pixels denote the path that was determined with breadth-first search.

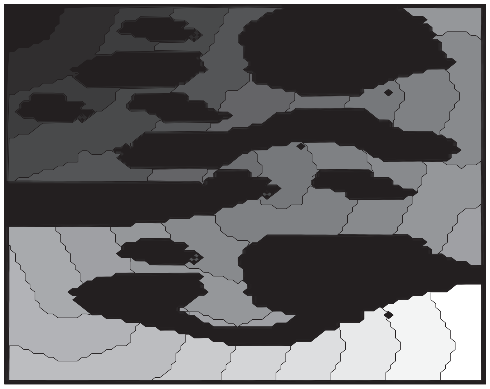


**Figure H.7** A plot of linklength values from the start (upper left corner) node where colored pixels correspond to link length (where the lighter the pixel the greater the linklength in the graph) and black pixels correspond to obstacles.

visited first. Now the expansion procedure sounds the same but is a little bit different; here, we pop the stack and push all of its children onto the stack, except popping and pushing occur on the same side of the list (figure H.5). Again, this procedure is repeated until the goal node is found or there are no more nodes to expand. Here, we expand nodes in a path as deep as possible before going onto a different path.

Figure H.8 displays the resulting path of depth-first search. In this example, depth-first search did not return an optimal path but it afforded a more efficient search

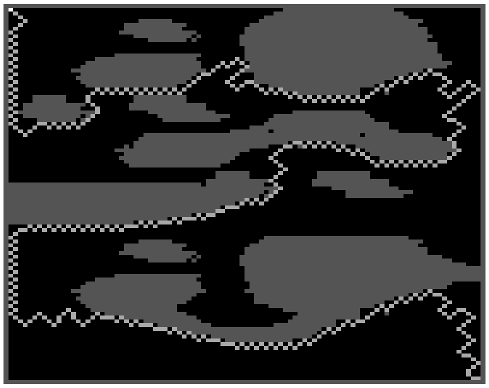


**Figure H.8** White pixels denote the path that was determined with depth-first search.

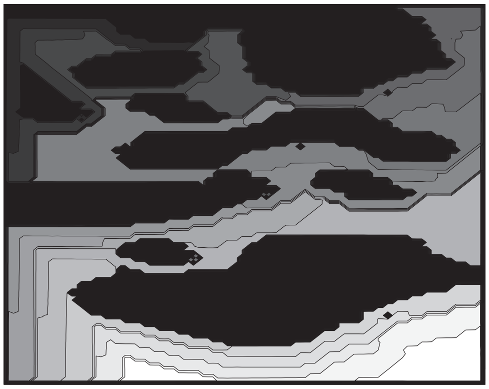


**Figure H.9** A plot of linklength values from the start (upper left corner) node where colored pixels correspond to link lengths of paths defined by the depth-first search. The lighter the pixel the greater the linklengths in the graph; black pixels correspond to obstacles.

in that the goal was found more quickly than breadth-first search. Figure H.9 is similar to figure H.7, except the link lengths here do *not* correspond to the shortest path to the start; instead, the link lengths correspond to the paths derived by the depth-first search. Again, we can use a depth-first search algorithm to fill up such a map and then determine a path via gradient descent from the goal pixel to the start.

Another common search is called a *greedy search* which expands nodes that are closest to the goal. Here, the data structure is called a *priority queue* in that nodes are placed into a sorted list based on a priority value. This priority value is a heuristic that measures distance to the goal node.

## H.2 $A^*$ Algorithm

Breadth-first search produces the shortest path to the start node in terms of link lengths. Since the wave-front planner is a breadth-first search, a four-point connectivity wave-front algorithm produces the shortest path with respect to the Manhattan distance function. This is because it implicitly has an underlying graph where each node corresponds to a pixel and neighboring pixels have an edge length of one. However, shortest-path length is not the only metric we may want to optimize. We can tune our graph search to find optimal paths with respect to other metrics such as energy, time, traversability, safety, etc., as well as combinations of them.

When speaking of graph search, there is another opportunity for optimization: minimize the number of nodes that have to be visited to locate the goal node subject to our path-optimality criteria. To distinguish between these forms of optimality, let us reserve the term *optimality* to measure the path and *efficiency* to measure the search, i.e., the number of nodes visited to determine the path. There is no reason to expect depth-first and breadth-first search to be efficient, even though breadth-first search can produce an optimal path.

Depth-first and breadth-first search in a sense are uninformed, in that the search just moves through the graph without any preference for or influence on where the goal node is located. For example, if the coordinates of the goal node are known, then a graph search can use this information to help decide which nodes in the graph to visit (i.e., expand) to locate the goal node.

Alas, although we may have some information about the goal node, the best we can do is define a *heuristic* which hypothesizes an expected, but not necessarily actual, cost to the goal node. For example, a graph search may choose as its next node to explore one that has the shortest Euclidean distance to the goal because such a node has highest possibility, based on local information, of getting closest to the goal. However, there is no guarantee that this node will lead to the (globally) shortest path in the graph to the goal. This is just a good guess. However, these good guesses are based on the best information available to the search.

The $A^*$ algorithm searches a graph efficiently, with respect to a chosen heuristic. If the heuristic is "good," then the search is efficient; if the heuristic is "bad," although a path will be found, its search will take more time than probably required and possibly return a suboptimal path. $A^*$ will produce an optimal path if its heuristic is *optimistic*.

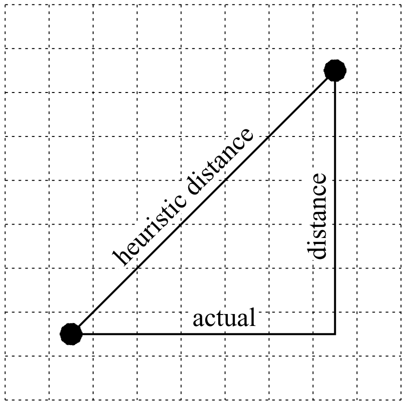


**Figure H.10** The heuristic between two nodes is the Euclidean distance, which is less than the actual path length in the grid, making this heuristic optimistic.

An optimistic, or admissible, heuristic always returns a value less than or equal to the cost of the shortest path from the current node to the goal node within the graph. For example, if a graph represented a grid, an optimistic heuristic could be the Euclidean distance to the goal because the $L^2$ distance is always less than or equal to the $L^1$ distance in the plane (figure H.10).

First, we will explain the $A^*$ search via example and then formally introduce the algorithm. See figure H.11 for a sample graph. The $A^*$ search has a *priority queue* which contains a list of nodes sorted by priority, which is determined by the sum of the distance traveled in the graph thus far from the start node, and the heuristic.

The first node to be put into the priority queue is naturally the start node. Next, we *expand* the start node by popping the start node and putting all adjacent nodes to the start node into the priority queue sorted by their corresponding priorities. Since node B has the greatest priority, it is expanded next, i.e., it is popped from the queue and its neighbors are added (figure H.12). Note that only unvisited nodes are added to the priority queue, i.e., do not re-add the start node.

Now, we expand node H because it has the highest priority. It is popped off of the queue and all of its neighbors are added. However, H has no neighbors, so nothing is added to the queue. Since no new nodes are added, no more action or expansion will be associated with node H (figure H.12). Next, we pop off the node with greatest priority, i.e., node A, and expand it, adding all of its adjacent neighbors to the priority queue (figure H.12).

Next, node E is expanded which gives us a path to the goal of cost 5. Note that this cost is the real cost, i.e., the sum of the edge costs to the goal. At this point, there are

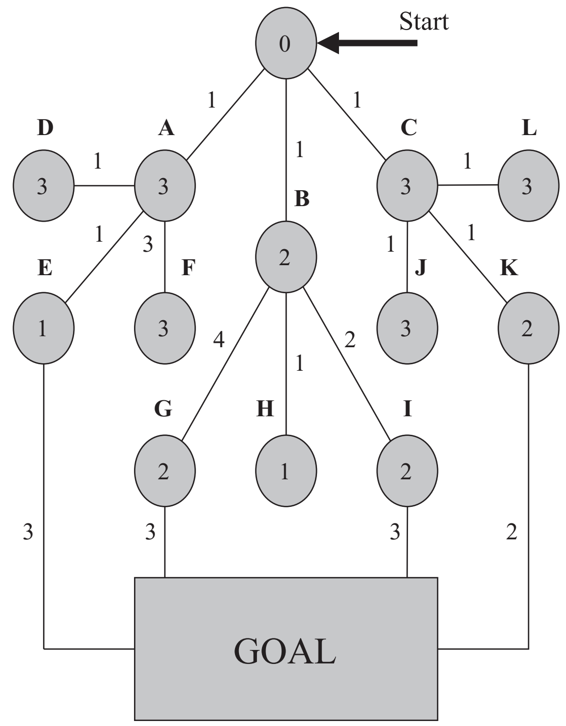


**Figure H.11** Sample graph where each node is labeled by a letter and has an associated heuristic value which is contained inside the node icon. Edge costs are represented by numbers adjacent to the edges and the start and goal nodes are labeled. We label the start node with a zero to emphasize that it has the highest priority at first.

nodes in the priority queue which have a priority value greater than the cost to the goal. Since these priority values are lower bounds on path cost to the goal, all paths through these nodes will have a higher cost than the cost of the path already found. Therefore, these nodes can be discarded (figure H.12).

The explicit path through the graph is represented by a series of *back pointers.* A back pointer represents the immediate history of the expansion process. So, the back pointers from nodes A, B, and C all point to the start. Likewise, the back pointers to D, E, and F point to A. Finally, the back pointer of goal points to E. Therefore, the path defined with the back pointers is start, A, E, and goal. The arrows in figure H.12 point in the reverse direction of the back pointers.

Even though a path to the goal has been determined, $A^*$ is not finished because there could be a better path. $A^*$ knows this is possible because the priority queue

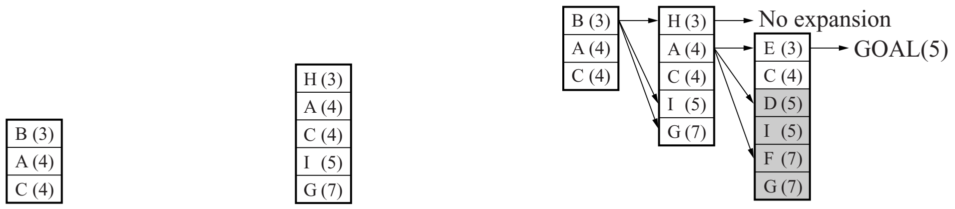


**Figure H.12** (Left) Priority queue after the start is expanded. (Middle) Priority queue after the second node, B, is expanded. (Right) Three iterations of the priority queue are displayed. Each arrow points from the expanded node to the nodes that were added in each step. Since node H had no unvisited adjacent cells, its arrow points to nothing. The middle queue corresponds to two actions. Node E points to the goal which provides the first candidate path to the goal. Note that nodes D, I, F, and G are shaded out because they were discarded.

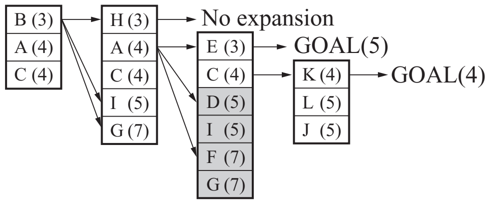


**Figure H.13** Four displayed iterations of the priority queue with arrows representing the history of individual expansions. Here, the path to the goal is start, C, K, goal.

still contains nodes whose value is smaller than that of the goal state. The priority queue at this point just contains node C and is then expanded adding nodes J, K, and L to the priority queue. We can immediately remove J and L because their priority values are greater than or equal the cost of the shortest path found thus far. Node K is then expanded finding the goal with a path cost shorter than the previously found path through node E. This path becomes the current best path. Since at this point the priority queue does not possess any elements whose value is smaller than that of the goal node, this path results in the best path (figure H.13).

## H.2.1 Basic Notation and Assumptions

Now, we can more formally define the $A^*$ algorithm. The input for $A^*$ is the graph itself. These nodes can naturally be embedded into the robot's free space and thus can have coordinates. Edges correspond to adjacent nodes and have values corresponding to the cost required to traverse between the adjacent nodes. The output of the $A^*$

algorithm is a back-pointer path, which is a sequence of nodes starting from the goal and going back to the start.

We will use two additional data structures, an open set $O$ and a closed set $C$. The open set $O$ is the priority queue and the closed set $C$ contains all processed nodes. Other notation includes

- Star($n$) represents the set of nodes which are adjacent to $n$.
- $c(n_1, n_2)$ is the length of edge connecting $n_1$ and $n_2$.
- $g(n)$ is the total length of a backpointer path from $n$ to $q_{start}$.
- $h(n)$ is the heuristic cost function, which returns the estimated cost of shortest path from $n$ to $q_{goal}$.
- $f(n) = g(n) + h(n)$ is the estimated cost of shortest path from $q_{start}$ to $q_{goal}$ via $n$.

The algorithm can be found in algorithm 24.

### H.2.2 Discussion: Completeness, Efficiency, and Optimality

Here is an informal proof of completeness for $A^*$. $A^*$ generates a search tree, which by definition, has no cycles. Furthermore, there are a finite number of acyclic paths in the tree, assuming a bounded world. Since $A^*$ uses a tree, it only considers acyclic paths. Since the number of acyclic paths is finite, the most work that can be done,

**Algorithm 24** $A^*$ Algorithm

**Input:** A graph

**Output:** A path between start and goal nodes

```
1: repeat
2:   Pick $n_{best}$ from $O$ such that $f(n_{best}) \leq f(n), \forall n \in O$.
3:   Remove $n_{best}$ from $O$ and add to $C$.
4:   If $n_{best} = q_{goal}$, EXIT.
5:   Expand $n_{best}$: for all $x \in$ Star($n_{best}$) that are not in $C$.
6:   if $x \notin O$ then
7:     add $x$ to $O$.
8:   else if $g(n_{best}) + c(n_{best}, x) < g(x)$ then
9:     update $x$'s backpointer to point to $n_{best}$
10:  end if
11: until $O$ is empty
```

searching all acyclic paths, is also finite. Therefore $A^*$ will always terminate, ensuring completeness.

This is not to say $A^*$ will always search all acyclic paths since it can terminate as soon as it explores all paths with greater cost than the minimum goal cost found. Thanks to the priority queue, $A^*$ explores paths likely to reach the goal quickly first. By doing so, it is efficient. If $A^*$ does search every acyclic path and does not find the goal, the algorithm still terminates and simply returns that a path does not exist. Of course, this also makes sense if every possible path is searched.

Now, there is no guarantee that the first path to the goal found is the cheapest/best path. So, in quest for optimality (once again, with respect to the defined metric), all branches must be explored to the extent that a branch's terminating node cost (sum of edge costs) is greater than the lowest goal cost. Effectively, all paths with overall cost lower than the goal must be explored to guarantee that an even shorter one does not exist. Therefore, $A^*$ is also optimal (with respect to the chosen metric).

### H.2.3 Greedy-Search and Dijkstra's Algorithm

There are variations or special cases of $A^*$. When $f(n) = h(n)$, then the search becomes a *greedy* search because the search is only considering what it "believes" is the best path to the goal from the current node. When $f(n) = g(n)$, the planner is not using any heuristic information but rather growing a path that is shortest from the start until it encounters the goal. This is a classic search called *Dijkstra's algorithm.* Figure H.14 contains a graph which demonstrates Dijkstra's Algorithm. In this example, we also show backpointers being updated (which can also occur with $A^*$). The following lists the open and closed sets for the Dijkstra search in each step.

1. $O = \{S\}$
2. $O = \{2, 4, 1, 5\}$; $C = \{S\}$ (1, 2, 4, 5 all point back to $S$)
3. $O = \{4, 1, 5\}$; $C = \{S, 2\}$ (there are no adjacent nodes not in C)

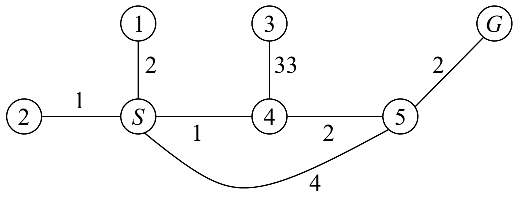


**Figure H.14** Dijkstra graph search example.

4. O = {1,5,3}; C = {S, 2, 4} (1, 2, 4 point to *S*; *5 points to 4*)

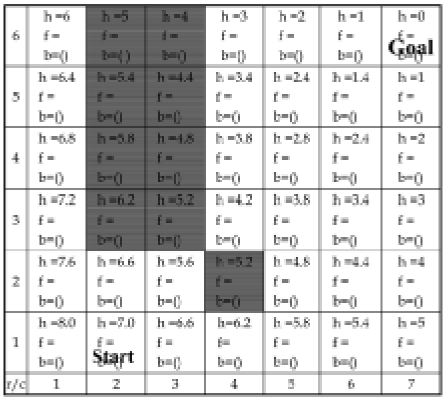
5. O = {5,3}; C = {S, 2, 4, 1}

6. O = {3, G}; C = {S, 2, 4, 1} (goal points to 5 which points to 4 which points to *S*)

## H.2.4 Example of $A^*$ on a Grid

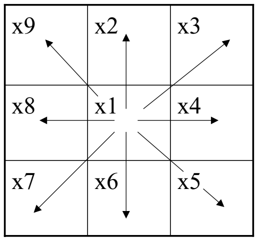
Figure H.15 contains an example of a grid world with a start and a goal identified accordingly. We will assume that the free space uses eight-point connectivity, and thus cell (3, 2) is adjacent to cell (4, 3), i.e., the robot can travel from (3, 2) to (4, 3). Each of the cells also has its heuristic distance to the goal where we use a modified metric which is not the Manhattan or the Euclidean distance. Instead, between free space pixels, a vertical or horizontal step has length 1 and a diagonal has length 1.4 (our approximation of $\sqrt{2}$). The cost of traveling from a free space pixel to an obstacle pixel is made to be arbitrarily high; we chose 10000. So one pixel step from a free space to an obstacle pixel along a vertical or horizontal direction costs 10000 and one pixel step along a diagonal direction costs 10000.4. Here, we are assuming that our graph connects *all* cells in the grid, not just the free space, and the prohibitively high cost of moving into an obstacle will prevent the robot from collision (figure H.16).

Note that this metric, in the free space, does not induce a true Euclidean metric because two cells sideways and one cell up is 2.4, not $\sqrt{5}$. However, this metric is quite representative of path length within the grid. This heuristic is optimistic because the actual cost to current cell to the goal will always be greater than or equal



| 6 | h =6<br>f =<br>b=() | h =5<br>f =<br>b=() | h =4<br>f =<br>b=() | h =3<br>f =<br>b=() | h =2<br>f =<br>b=() | h =1<br>f =<br>b=() | h =0<br>f =<br>Goal |
|---|---|---|---|---|---|---|---|
| 5 | h =6.4<br>f =<br>b=() | h =5.4<br>f =<br>b=() | h =4.4<br>f =<br>b=() | h =3.4<br>f =<br>b=() | h =2.4<br>f =<br>b=() | h =1.4<br>f =<br>b=() | h =1<br>f =<br>b=() |
| 4 | h =6.8<br>f =<br>b=() | h =5.8<br>f =<br>b=() | h =4.8<br>f =<br>b=() | h =3.8<br>f =<br>b=() | h =2.8<br>f =<br>b=() | h =2.4<br>f =<br>b=() | h =2<br>f =<br>b=() |
| 3 | h =7.2<br>f =<br>b=() | h =6.2<br>f =<br>b=() | h =5.2<br>f =<br>b=() | h =4.2<br>f =<br>b=() | h =3.8<br>f =<br>b=() | h =3.4<br>f =<br>b=() | h =3<br>f =<br>b=() |
| 2 | h =7.6<br>f =<br>b=() | h =6.6<br>f =<br>b=() | h =5.6<br>f =<br>b=() | h =5.2<br>f =<br>b=() | h =4.8<br>f =<br>b=() | h =4.4<br>f =<br>b=() | h =4<br>f =<br>b=() |
| 1 | h =8.0<br>f =<br>b=() | h =7.0<br>f =<br>Start | h =6.6<br>f =<br>b=() | h=6.2<br>f=<br>b=() | h =5.8<br>f =<br>b=() | h =5.4<br>f =<br>b=() | h =5<br>f =<br>b=() |
| r/c | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

**Figure H.15** Heuristic values are set, but backpointers and priorities have not.



c(x1, x2) = 1
c(x1, x9) = 1.4

c(x1, x8) = 10000, if x8 is in obstacle, x1 is a free cell

c(x1,x9) = 10000.4, if x9 is in obstacle, x1 is a free cell

**Figure H.16** Eight-point connectivity and possible cost values.

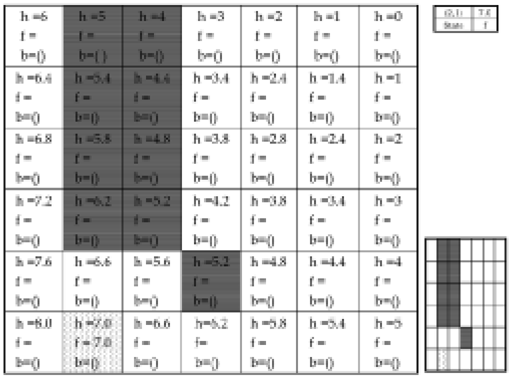


**Figure H.17** Start node is put on priority queue, displayed in upper right.

to the heuristic. Thus far, in figure H.15 the back pointers and priorities have not been set.

The start pixel is put on the priority queue with a priority equal to its heuristic. See figure H.17. Next, the start node is expanded and the priority values for each of the start's neighbors are determined. They are all put on the priority queue sorted in ascending order by priority. See figure H.18(left). Cell (3, 2) is expanded next, as depicted in figure H.18(right). Here, cells (4, 1), (4, 2), (4, 3), (3, 3), and (2, 3) are added onto the priority queue because our graph representation of the grid includes both free space and obstacle pixels. However, cells (4, 2), (3, 3), and (2, 3) correspond to obstacles and thus have a high cost. If a path exists in the free space or the longest path in the free space has a traversal cost less than our arbitrarily high number chosen for obstacles (figure H.16), then these pixels will never be expanded. Therefore, in the figures below, we did not display them on the priority queue.

Eventually, the goal cell is reached (figure H.19 (left)). Since the priority value of the goal is less than the priorities of all other cells in the priority queue, the resulting

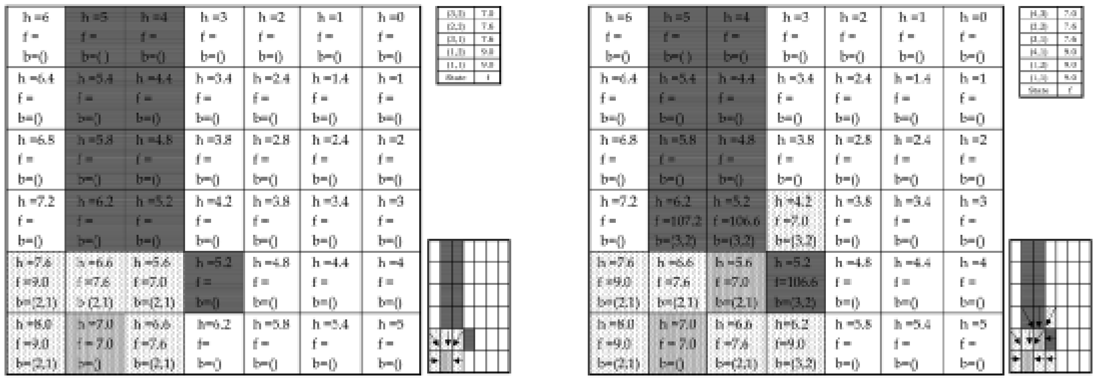
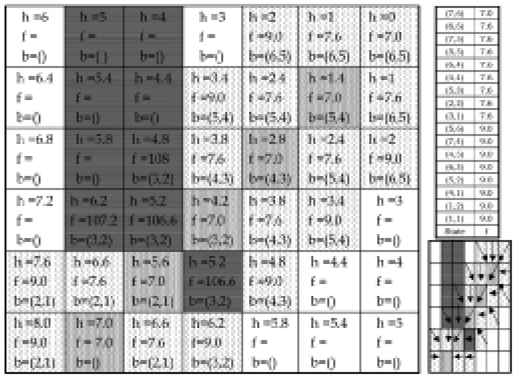


**Figure H.18** (Left) The start node is expanded, the priority queue is updated, and the back-pointers are set, which are represented by the right bottom icon. $b = (i, j)$ points to cell $(i, j)$. (Right) Cell (3, 2) was expanded. Note that pixels (3, 3), (2, 3), and (4, 2) are not displayed in the priority queue because they correspond to obstacles.



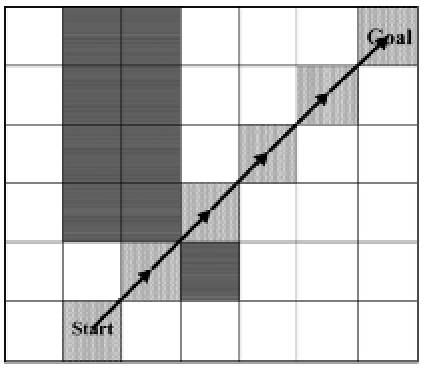


**Figure H.19** (Left) The goal state is expanded. (Right) Resulting path.

path is optimal and $A^*$ terminates. $A^*$ traces the backpointers to find the optimal path from start to goal (figure H.19 (right)).

## H.2.5 Nonoptimistic Example

Figure H.20 contains an example of a graph whose heuristic values are nonoptimistic and thus force $A^*$ to produce a nonoptimal path. $A^*$ puts node $S$ on the priority queue and then expands it. Next, $A^*$ expands node A because its priority value is 7. The goal

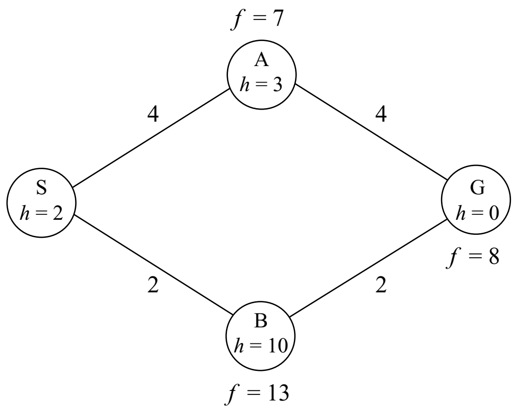


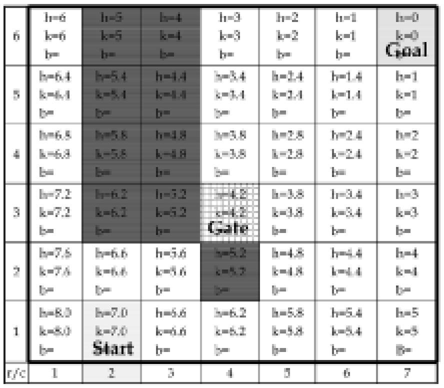
**Figure H.20** A nonoptimistic heuristic leads to a nonoptimal path with $A^*$.

node is then reached with priority value 8, which is still less than node B's priority value of 13. At this point, node B will be eliminated from the priority queue because its value is greater than the goal's priority value. However, the optimal path passes through B, not A. Here, the heuristic is not optimistic because from B to G, $h = 10$ when the actual edge length was 2.

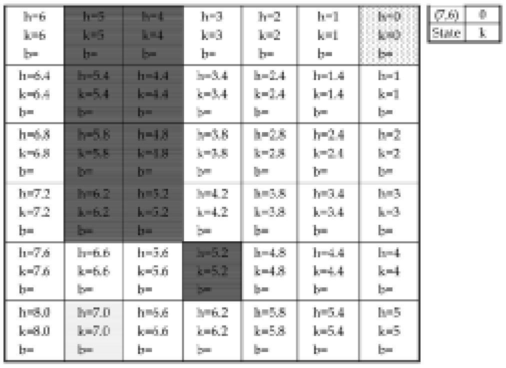
## H.3 $D^*$ Algorithm

So far we have only considered *static* environments where only the robot experiences motion. However, we can see that many worlds have moving obstacles, which could be other robots themselves. We term such environments *dynamic*. There are three types of dynamic obstacles: ones that move significantly slower than the robot, those that move at the same speed, and finally obstacles that move much faster than the robot. The superfast obstacle case is easy to ignore because the obstacles will be moving so fast that there probably is no need to plan for them because they will either move too fast for the planner to have time to account for them or they will be in and out of the robot's path so quickly that it does not require any consideration. In this section, we consider dynamic environments where the world changes at a speed much slower than the robot. An example can be a door opening and closing.

Consider the grid environment in figure H.21(left) which is identical to the one in figure H.15, except pixel (4, 3) is a gate which can either be a free-space pixel or an obstacle pixel. Let's assume it starts as a free-space pixel. We can run the $A^*$ or



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | h=6 k=6 b= | h=5 k=5 b= | h=4 k=4 b= | h=3 k=3 b= | h=2 k=2 b= | h=1 k=1 b= | h=0 k=0 Goal |
| 5 | h=6.4 k=6.4 b= | h=5.4 k=5.4 b= | h=4.4 k=4.4 b= | h=3.4 k=3.4 b= | h=2.4 k=2.4 b= | h=1.4 k=1.4 b= | h=1 k=1 b= |
| 4 | h=6.8 k=6.8 b= | h=5.8 k=5.8 b= | h=4.8 k=4.8 b= | h=3.8 k=3.8 b= | h=2.8 k=2.8 b= | h=2.4 k=2.4 b= | h=2 k=2 b= |
| 3 | h=7.2 k=7.2 b= | h=6.2 k=6.2 b= | h=5.2 k=5.2 b= | h=4.2 k=4.2 Gate | h=3.8 k=3.8 b= | h=3.4 k=3.4 b= | h=3 k=3 b= |
| 2 | h=7.6 k=7.6 b= | h=6.6 k=6.6 b= | h=5.6 k=5.6 b= | h=5.2 k=5.2 b= | h=4.8 k=4.8 b= | h=4.4 k=4.4 b= | h=4 k=4 b= |
| 1 | h=8.0 k=8.0 b= | h=7.0 k=7.0 Start | h=6.6 k=6.6 b= | h=6.2 k=6.2 b= | h=5.8 k=5.8 b= | h=5.4 k=5.4 b= | h=5 k=5 b= |
| r/c | 1 | 2 | 3 | 4 | 5 | 6 | 7 |



| | | | | | | |
|---|---|---|---|---|---|---|
| h=6 k=6 b= | h=5 k=5 b= | h=4 k=4 b= | h=3 k=3 b= | h=2 k=2 b= | h=1 k=1 b= | h=0 k=0 b= |
| h=6.4 k=6.4 b= | h=5.4 k=5.4 b= | h=4.4 k=4.4 b= | h=3.4 k=3.4 b= | h=2.4 k=2.4 b= | h=1.4 k=1.4 b= | h=1 k=1 b= |
| h=6.8 k=6.8 b= | h=5.8 k=5.8 b= | h=4.8 k=4.8 b= | h=3.8 k=3.8 b= | h=2.8 k=2.8 b= | h=2.4 k=2.4 b= | h=2 k=2 b= |
| h=7.2 k=7.2 b= | h=6.2 k=6.2 b= | h=5.2 k=5.2 b= | h=4.2 k=4.2 b= | h=3.8 k=3.8 b= | h=3.4 k=3.4 b= | h=3 k=3 b= |
| h=7.6 k=7.6 b= | h=6.6 k=6.6 b= | h=5.6 k=5.6 b= | h=5.2 k=5.2 b= | h=4.8 k=4.8 b= | h=4.4 k=4.4 b= | h=4 k=4 b= |
| h=8.0 k=8.0 b= | h=7.0 k=7.0 b= | h=6.6 k=6.6 b= | h=6.2 k=6.2 b= | h=5.8 k=5.8 b= | h=5.4 k=5.4 b= | h=5 k=5 b= |

| | |
|---|---|
| (7,6) | 0 |
| State | k |

**Figure H.21** (Left) A pixel world similar to figure H.15, except it has a gate, heuristic, and minimum heuristic values. (Right) Goal node is expanded.

Dijkstra's algorithm to determine a path from start to goal, and then follow that path until an unexpected change occurs, which in figure H.21(left) happens at (4, 3). When the robot encounters pixel (4, 3) and determines that it changed from a free-space to an obstacle pixel, it can simply reinvoke the $A^*$ algorithm to determine a new path. This, however, can become quite inefficient if many pixels are changing from obstacle to free space and back. The $D^*$ algorithm was devised to "locally repair" the graph allowing for an efficient updated searching in dynamic environments, hence the term $D^*$ [397].

$D^*$ initially determines a path starting with the goal and working back to the start using a slightly modified Dijkstra's search. The modification involves updating a heuristic and a minimum heuristic function. Each cell in figure H.21(left) contains a heuristic cost ($h$) which for $D^*$ is an *estimate* of path length from the particular cell to the goal, not necessarily the shortest path length to the goal as it was for $A^*$. In this example, the $h$ values do not respect the presence of obstacles when reflecting distance to the goal node; in other words, computation of $h$ assumes that the robot can pass through obstacles. For example, cell (1, 6) has an $h$ value of 6. These $h$ values will be updated during the initial Dijkstra search to reflect the existence of obstacles. The minimum heuristic values ($k$) are the *estimate* of the *shortest* path length to the goal. Both the $h$ and the $k$ values will vary as the $D^*$ search runs, but they are equal upon initialization, and were derived from the metric described in figure H.16.

Initially, the goal node is placed on the queue with $h = 0$ and then is expanded (figure H.21, right), adding (6, 6), (6, 5), and (7, 5) onto the queue. Next, pixel (6, 6) is expanded adding cells (5, 6), (5, 5) onto the queue. Note that the $k$ values are used

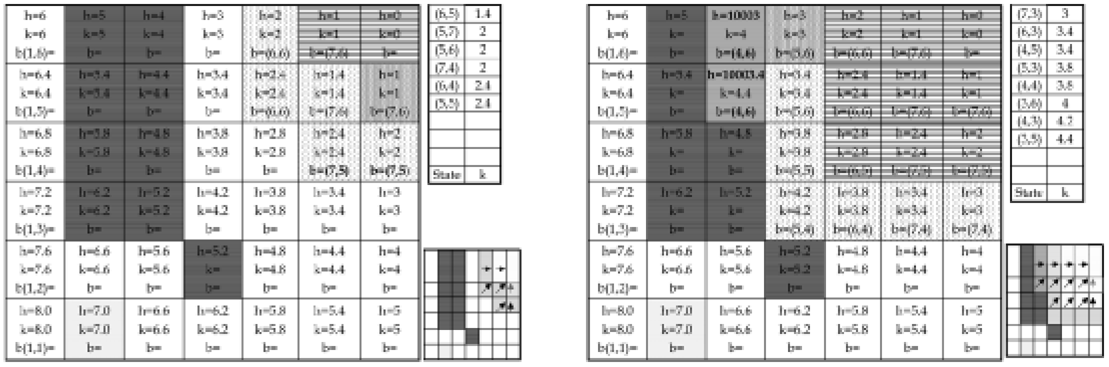


**Figure H.22** (Left) First (6, 6) and then (7, 5) is expanded. (Right) The $h$ values in obstacle cells that are put on priority queue are updated.

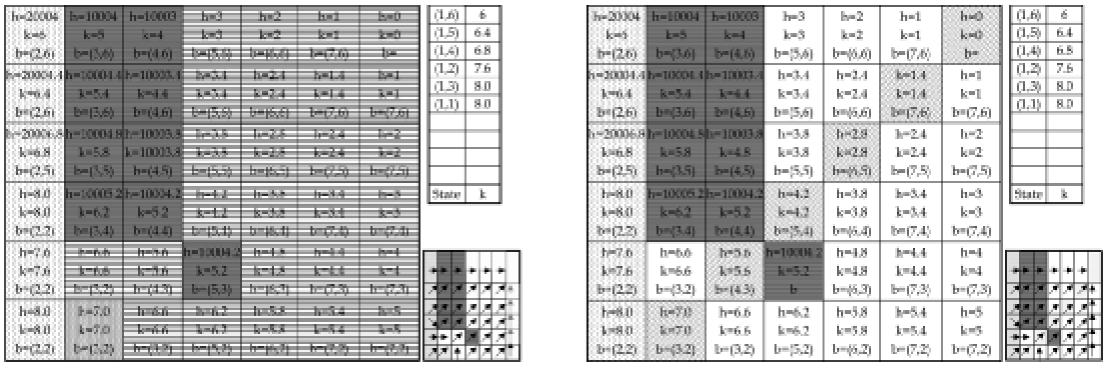


**Figure H.23** (Left) Termination of Dijkstra's search phase: start cell is expanded. (Right) Tracing backpointers yields the optimal path.

to determine the priority for the Dijkstra's search (and later on for the $D^*$ search) and that they are equal to the $h$ values for the initial Dijkstra's search.

Next, pixel (7, 5) is expanded adding cells (6, 4) and (7, 4) onto the queue (figure H.22, left). More pixels are expanded until we arrive at pixel (4, 6) (figure H.22, right). When (4, 6) is expanded, pixels (3, 6) and (3, 5), which are obstacle pixels, are placed onto the priority queue. Unlike our $A^*$ example, we display these obstacle pixels in the priority queue in figure H.22(right). Note that the $h$ values of the expanded obstacle pixels are all updated to prohibitively high values which reflects the fact that they lie on obstacles.

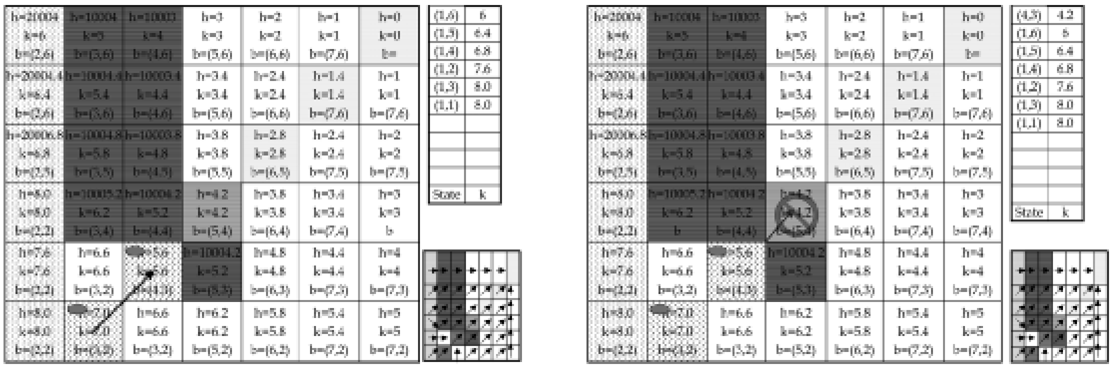


**Figure H.24** (Left) The robot physically starts tracing the optimal path. (Right) The robot cannot trace the assumed optimal path: gate (4, 3) is closed.

The Dijkstra's search is complete when the start node (2, 1) is expanded (figure H.23, left). The optimal path from start to goal (assuming that the gate pixel (4, 3) is open) is found by traversing the backpointers starting from the start node to the goal node (figure H.23(right)). The optimal path is (2, 1) $\longrightarrow$ (3, 2) $\longrightarrow$ (4, 3) $\longrightarrow$ (5, 4) $\longrightarrow$ (6, 5) $\longrightarrow$ (7, 6). Note that pixels (1, 1), (1, 2), (1, 3), (1, 4), and (1, 6) are still on the priority queue.

The robot then starts tracing the optimal path from the start pixel to the goal pixel. In figure H.24(left), the robot moves from pixel (2, 1) to (3, 2). When the robot tries to move from pixel (3, 2) to (4, 3), it finds that the gate pixel (4, 3) is closed (figure H.24, left). In the initial search for an optimal path, we had assumed that the gate pixel was open, and hence the current path may not be feasible. At this stage, instead of replanning for an optimal path from the current pixel (3, 2) to goal pixel using $A^*$, $D^*$ tries to make local changes to the optimal path.

$D^*$ puts the pixel (4, 3) on the priority queue because it corresponds to a discrepancy between the map and the actual environment. Note that this pixel must have the lowest minimum heuristic, i.e., $k$ value, because all other pixels on the priority queue have a $k$ value greater than or equal to the start and all pixels along the previously determined optimal path have a $k$ value less than the start. The idea here is to put the changed pixel onto the priority queue and then expand it again, thereby propagating the possible changes in the heuristic, i.e., the $h$ values, to pixels for which an optimal path to the goal passes through the changed pixel.

In the current example, pixel (4, 3) is expanded, i.e., it is popped off the priority queue and pixels whose optimal paths pass through (4, 3) are placed onto the priority

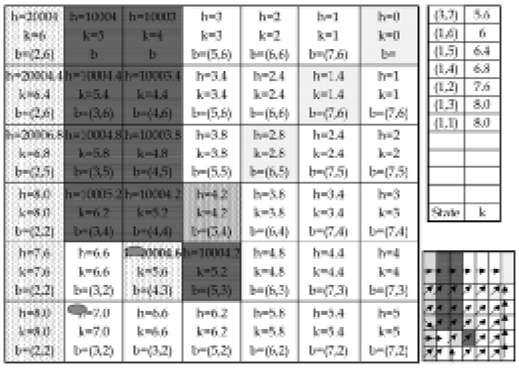
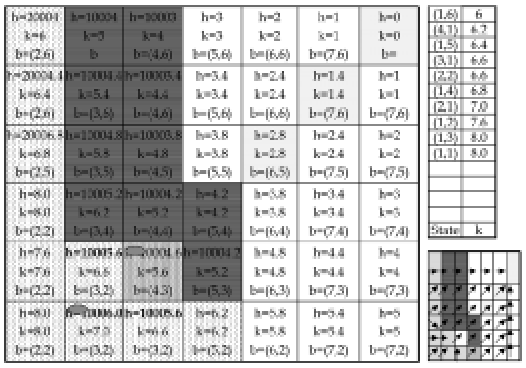



**Figure H.25** (Left) The gate pixel (4, 3) is put on priority queue and expanded. The assumed optimal path from (3, 2) to goal passed through (4, 3), so the $h$ value is increased to a high value to reflect that the assumed optimal path may not in fact be optimal. (Right) Pixel (3, 2) is expanded; the $h$ values of (2, 2),(2, 1) and (3, 1) are updated because the assumed optimal path from these cells passed through the expanded cell. (4, 1)) remains unaffected.

queue with updated heuristic values ($h$ values). The new $h$ values are the $h$ values of the changed pixel plus the path cost from the changed pixel to the given pixel. This path cost is a high number which we set to 10000.4 if it passes diagonally through an obstacle pixel. Therefore, pixel (3, 2) has an $h$ value equal to 10004.6 (figure H.25, left).

Next, pixel (3, 2) is expanded because its $k$ value is the smallest. However, its $k$ value is less than its $h$ value and we term such pixels as having a *raised state.* When a pixel is in a raised state, its back pointer may no longer point to an optimal path. Now, pixels (2, 2), (2, 1), (3, 1), and (4, 1) are on the priority queue. The $h$ values of cells (2, 2), (2, 1), and (3, 1) are updated to high values to reflect that the estimated optimal path from these cells to the goal passed through the gate cell, and may not be optimal anymore. However, the optimal path for pixel (4, 1) did not pass through the gate, and hence its $h$ value stays the same (figure H.25, right).

Pixel (1, 6) is expanded next, but it does not affect the $h$ values of its neighbors. Next pixel (4, 1) is expanded and pixels (3, 2), (3, 1), (5, 1), and (5, 2) are put onto the priority queue (figure H.26(left). Now the $h$ values of (5, 1) and (5, 2) remain unaffected, however, for cell (3, 2), the goal can now possibly be reached in $h(4, 1) + 1.4 = 6.2 + 1.4 = 7.6$ because cell (4, 1) is in a *lowered state,* i.e., it is a cell whose $h$ values did not have to be updated because of the gate. Therefore, cell (3, 2) receives an $h$ value of 7.6. The backpointer of (3, 2) is now set pointing toward (4, 1). The initially determined optimal path from (4, 1) to the goal did not pass through the

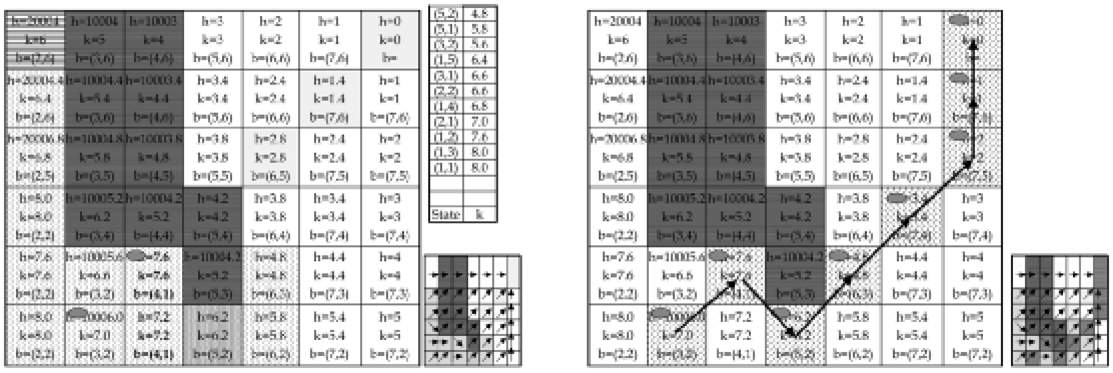


**Figure H.26** (Left) (4, 1) is expanded; (5, 1) and (5, 2) remain unaffected, the $h$ values of (3, 2) and (3, 1) are lowered to reflect the lowered heuristic value because of detour, while the minimum-heuristic values of these two cells are increased, and the backpointers of these cells are set pointing to (4, 1). (Right) The robot traces the new locally modified optimal path.

gate pixel, and hence it indeed is optimal even after the change of state of the gate pixel. Then, the path obtained by concatenating the $(3, 2) \rightarrow (4, 1)$ transition and the optimal path from (4, 1) will be optimal for (3, 2). Thus, the estimate for the best path from (3, 2) toward the goal, i.e., $k(3, 2)$, is now 7.6, and the process terminates. The robot then physically traces the new optimal path from (3, 2) to reach the goal (figure H.26(right)).

See algorithms 25–31 for a description of the $D^*$ algorithm. This algorithm uses the following notation.

- $X$ represents a state.
- $O$ is the priority queue.
- $L$ is the list of all states.
- $G$ is the goal state.
- $S$ is the start state.
- $t(X)$ is value of state with regards to the priority queue.
  - $t(X) = NEW$, if $X$ has never been in $O$,
  - $t(X) = OPEN$, if $X$ is currently in $O$, and
  - $t(X) = CLOSED$, if $X$ was in $O$ but currently is not.

**Algorithm 25** $D^*$ Algorithm

**Input:** List of all states $L$

**Output:** The goal state, if it is reachable, and the list of states $L$ are updated so that the backpointer list describes a path from the start to the goal. If the goal state is not reachable, return NULL.

```
1:  for each $X \in L$ do
2:     t(X) = NEW
3:  end for
4:  $h(G) = 0$
5:  $O = \{G\}$
6:  $X_c = S$
    {The following loop is Dijkstra's search for an initial path}
7:  repeat
8:     $k_{min} = PROCESS - STATE(O, L)$
9:  until ($k_{min} = -1$) or ($t(X_c) = CLOSED$)
10: $P = GET - BACKPOINTER - LIST(L, X_c, G)$ (algorithm 26)
11: if P = NULL then
12:    Return (NULL)
13: end if
14: repeat
15:    for each neighbor $Y \in L$ of $X_c$ do
16:       if $r(X_c, Y) \neq c(X_c, Y)$ then
17:          $MODIFY - COST(O, X_c, Y, r(X_c, Y))$
18:          repeat
19:             $k_{min} = PROCESS - STATE(O, L)$
20:          until ($k_{min} \geq h(X_c)$) or ($k_{min} = -1$)
21:          $P = GET - BACKPOINTER - LIST(L, X_c, G)$
22:          if P = NULL then
23:             Return (NULL)
24:          end if
25:       end if
26:    end for
27:    $X_c$ = the second element of $P$ {Move to the next state in $P$}.
28:    $P = GET - BACKPOINTER - LIST(L, X_c, G)$
29: until $X_c = G$
30: Return ($X_c$)
```

**Algorithm 26** $GET - BACKPOINTER - LIST(L, S, G)$

**Input:** A list of states $L$ and two states (start and goal)

**Output:** A list of states from start to goal as described by the backpointers in the list of states $L$

```
1: if path exists then
2:     Return (The list of states)
3: else
4:     Return ($NULL$)
5: end if
```

**Algorithm 27** $INSERT(O, X, h_{new})$

**Input:** Open list, a state, and an $h$-value

**Output:** Open list is modified

```
1: if $t(X) = NEW$ then
2:     $k(X) = h_{new}$
3: else if $t(X) = OPEN$ then
4:     $k(X) = \min(k(X), h_{new})$
5: else if $t(X) = CLOSED$ then
6:     $k(X) = \min(h(X), h_{new})$
7: end if
8: $h(X) = h_{new}$
9: $t(X) = OPEN$
10: Sort $O$ based on increasing $k$ values
```

- $c(X, Y)$ is the estimated path length between adjacent states $X$ and $Y$.
- $h(X)$ is the estimated cost of a path from $X$ to Goal (heuristic).
- $k(X)$ is the estimated cost of a shortest path from $X$ to Goal (minimum-heuristic $= \min h(X)$ before $X$ is put on $O$, values $h(X)$ takes after $X$ is put on $O$).
- $b(X) = Y$ implies that $Y$ is a parent state of $X$, i.e. the path is like $X \longrightarrow Y \longrightarrow G$.
- $r(X, Y)$ is the measured distance adjacent states $X$ and $Y$.

**Algorithm 28** $MODIFY-COST(O, X, Y, cval)$

**Input:** The open list, two states and a value

**Output:** A $k$-value and the open list gets updated

```
1: $c(X, Y) = cval$
2: if $t(X) = CLOSED$ then
3:     $INSERT(O, X, h(X))$
4: end if
5: Return $GET-KMIN(O)$ (algorithm 30)
```

**Algorithm 29** $MIN-STATE(O)$

**Input:** The open list $O$

**Output:** The state with minimum $k$ value in the list related values

```
1: if $O = \emptyset$ then
2:     Return $(-1)$
3: else
4:     Return $(\text{argmin}_{Y \in O} k(Y))$
5: end if
```

**Algorithm 30** $GET-KMIN(O)$

**Input:** The open list $O$

**Output:** Lowest $k$-value of all states in the open list

```
1: if $O = \emptyset$ then
2:     Return $(-1)$
3: else
4:     Return $(\min_{Y \in O} k(Y))$
5: end if
```

**Algorithm 31** PROCESS-STATE

**Input:** List of all states $L$ and the list of all states that are open $O$

**Output:** A $k_{min}$, an updated list of all states, and an updated open list

```
1: $X = MIN-STATE(O)$ (algorithm 29)
2: if $X = NULL$ then
3:     Return $(-1)$
4: end if
5: $k_{old} = GET-KMIN(O)$ (algorithm 30)
6: $DELETE(X)$
7: if $k_{old} < h(X)$ then
8:     for each neighbor $Y \in L$ of $X$ do
9:         if $h(Y) \leq k_{old}$ and $h(X) > h(Y) + c(Y, X)$ then
10:            $b(X) = Y$
11:            $h(X) = h(Y) + c(Y, X)$;
12:        end if
13:    end for
14: else if $k_{old} = h(X)$ then
15:    for each neighbor $Y \in L$ of $X$ do
16:        if $(t(Y) = NEW)$ or $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ or $(b(Y) \neq$
           $X$ and $h(Y) > h(X) + c(X, Y))$ then
17:            $b(Y) = X$
18:            $INSERT(O, Y, h(X) + c(X, Y))$ (algorithm 27)
19:        end if
20:    end for
21: else
22:    for each neighbor $Y \in L$ of $X$ do
23:        if $(t(Y) = NEW)$ or $(b(Y) = X$ and $h(Y) \neq h(X) + c(X, Y))$ then
24:            $b(Y) = X$
25:            $INSERT(O, Y, h(X) + c(X, Y))$
26:        else if $b(Y) \neq X$ and $h(Y) > h(X) + c(X, Y)$ then
27:            $INSERT(O, X, h(X))$
28:        else if $(b(Y) \neq X$ and $h(X) > h(Y) + c(X, Y))$ and $(t(Y) = CLOSED)$ and
           $(h(Y) > k_{old})$ then
29:            $INSERT(O, Y, h(Y))$
30:        end if
31:    end for
32: end if
33: Return $GET-KMIN(O)$ (algorithm 30)
```

## H.4 Optimal Plans

There exists a huge number of search algorithms in the literature, with the ones discussed here being just the most basic ones. All of the techniques discussed here result in a path. A path is sufficient if the robot is able to follow it. Sometimes, randomness may push the robot off its path. One possibility is to replan, as we did in $D^*$ (albeit for different reasons: above the environment changed and thereby mandated replanning). Another is to determine the best action for *all* nodes in the graph, not just the ones along the shortest path. A mapping from nodes to actions is called a *universal plan,* or *policy*. Techniques for finding optimal policies are known as universal planners and can be computationally more involved than the shortest path techniques surveyed here. One simple way to attain a universal plan to a goal is to run Dijkstra's algorithm backward (as in $D^*$): After completion, we know for each node in the graph the length of an optimal path to the goal, along with the appropriate action. Generalizations of this approach are commonly used in stochastic domains, where the outcome of actions is modeled by a probability distribution over nodes in the graph.

# I Statistics Primer

ELEGANT AND powerful techniques are at the fingertips of statisticians. Although difficult at first, speaking their language can be quite powerful. Probability theory provides a set of tools that can be used to quantify uncertain events. In the context of robotics, probability theory allows us make decisions in the presence of uncertainty caused by phenomena such as noisy sensor data or interaction with unpredictable humans. This section introduces a few fundamental concepts including probability, random variables, distributions, and Gaussian random vectors.

When we talk about probability, we generally talk in terms of *experiments* and *outcomes*. When an experiment is conducted, a single outcome from the set of possible outcomes for that experiment results. For example, an experiment could be flipping a coin and the set of possible outcomes is {heads, tails}. If the experiment were to take a measurement in degrees Kelvin, then the set of possible outcomes would be the interval $[0, \infty)$. An *event* is defined to be a subset of the possible outcomes.

Let $\mathcal{S}$ denote the set of all possible outcomes for a given experiment, and let $E$ be an event, i.e., $E \subset \mathcal{S}$. The *probability* of the event $E$ occurring when the experiment is conducted is denoted $\text{Pr}(E)$. Pr maps $\mathcal{S}$ to the interval $[0, 1]$. In the example of flipping a fair coin, $\text{Pr}(\text{heads}) = 0.5$, $\text{Pr}(\text{tails}) = 0.5$, and $\text{Pr}(\text{heads} \cup \text{tails}) = 1$. In general, the probability must obey certain properties:

1. $0 \leq \text{Pr}(E) \leq 1$ for all $E \subset \mathcal{S}$.

2. $\text{Pr}(\mathcal{S}) = 1$.

3. $\sum_i \Pr(E_i) = Pr(E_1 \cup E_2 \cup \ldots)$ for any countable disjoint collection of sets $E_1, E_2, \ldots$. This property is known as *sigma additivity*. In particular, we have $\sum_{i=1}^{n} \Pr(E_i) = Pr(E_1 \cup E_2 \cup \ldots \cup E_n)$.

4. $\Pr(\emptyset) = 0$.

5. $\Pr(E^c) = 1 - \Pr(E)$, where $E^c$ denotes the complement of $E$ in $\mathcal{S}$.

6. $\Pr(E_1 \cup E_2) = \Pr(E_1) + \Pr(E_2) - \Pr(E_1 \cap E_2)$.

Technically, the first three axioms imply the last three.

Events may or may not depend upon each other. If the occurance of $E_1$ has no effect on $E_2$, then $E_1$ and $E_2$ are *independent;* otherwise they are dependent. We say $E_1$ and $E_2$ are independent if $\Pr(E_1 \wedge E_2) = \Pr(E_1) \cdot \Pr(E_2)$. One way to express the dependence of two events is through *conditional probability*. For events $E_1$ and $E_2$, $\Pr(E_1 \mid E_2)$ is the *conditional probability* that $E_1$ occurs given that $E_2$ occurs. If $E_1$ and $E_2$ are independent and $\Pr(E_2) > 0$, then $\Pr(E_1 \mid E_2) = \Pr(\mathrm{E}_1)$. For dependent events, Bayes' rule expresses the relationship between the conditional probabilities for two events, again assuming $\Pr(E_2) > 0$:

$$\Pr(E_1 \mid E_2) = \frac{\Pr(E_2 \mid E_1)\Pr(E_1)}{\Pr(E_2)}.$$

Bayes' rule is a useful formula; it is the foundation of the estimation methods presented in chapter 9.

## I.1 Distributions and Densities

Within robotics, a somewhat simplified but nevertheless sufficient model of a *random variable* is a mapping from the set of events to the real line, usually denoted $X : \mathcal{S} \to \mathbb{R}$. A simple example of a random variable is to consider a single coin flip and define $X = 0$ when the outcome is heads and $X = 1$ when the outcome is tails. As another example, consider flipping a fair coin ten times. A random variable can be the number of heads that appeared or the number of times heads appeared sequentially, etc. Random variables are useful because they represent events as real numbers. With real numbers, we can perform calculations and analysis that are difficult or impossible to perform on the abstract events.

A *distribution* is an abstract concept that corresponds to all probability statements that can be made about a random variable. Before we discuss the various distributions used to describe random variables, we first distinguish between contiunous and discrete random variables. A random variable is said to be *discrete* if its range (the values that it maps to) is a set of discrete points. We call a random variable *continuous* if its range forms a continuum on the real line and, using a term that is defined further below, it possesses a probability density function.

Discrete random variables are commonly described using one of two types of distributions. The first is the *cumulutive distribution function* (CDF) which is denoted $F_X(a) = \Pr(X \leq a)$. The second is the *probability mass function* (PMF), which is defined to be $f_X(a) = \Pr(X = a)$.

Continuous random variables are described with two analogous distributions. The first is the cumulative distribution function (figure I.1), which is defined for continuous random variables exactly the same way that it is defined for discrete random variables. The second is the *probability density function* (PDF) (figure I.2), which is denoted $f_X$ and is defined such that

$$\Pr(a \leq X \leq b) = \int_{x=a}^{b} f_X(x)\, dx.$$

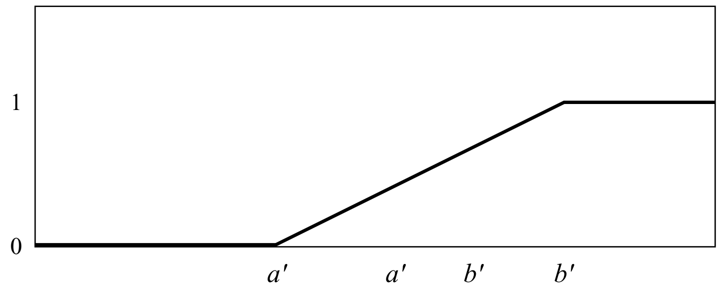


**Figure I.1** Cumulative uniform distribution.

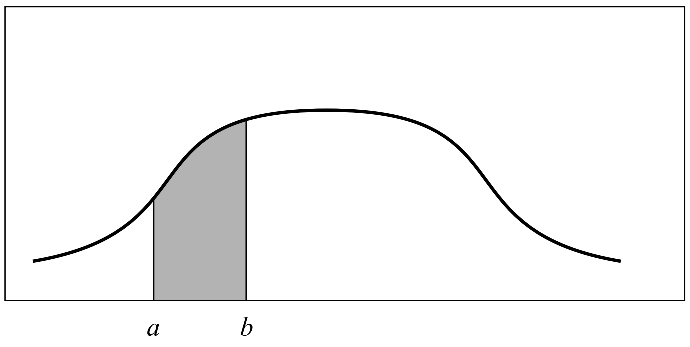


**Figure I.2** Probability density function.

Note that for a continuous random variable, $Pr((X = a)) = \int_{x=a}^{a} f_X(x)\, dx = 0$. This can be disconcerting to the newcomer. Another way to view this is: consider the odds of landing exactly on $a$. Since the point $a$ is a set of measure zero, it should have zero probability of occurring.

Some distributions are so common that they have their own name. For example, the uniform distribution is a family of continuous distributions over an interval. It can either be described by the CDF

$$U(x; a, b) = \begin{matrix} 0 & x < a \\ \frac{x-a}{b-a} & a \le x \le b \\ 1 & x \ge b \end{matrix}$$

or by the corresponding PDF

$$u(x; a, b) = \begin{matrix} 0 & x < a \\ \frac{1}{b-a} & a \le x \le b \\ 0 & x \ge b. \end{matrix}$$

One can calculate all sorts of probabilties using either the CDF or the PDF. Consider for $a', b' \in [a, b]$ with $a' < b'$. Using the CDF we can compute $Pr(a' \le x \le b') = U(b'; a, b) - U(a'; a, b)$. Alternatively we can use the PDF to compute $Pr(a' \le x \le b') = \int_{x=a'}^{b'} u(x; a, b)\, dx$.

## I.2 Expected Values and Covariances

We previously defined a random variable to be a function that maps the event space to the real line. Similarly, we define a *random vector* to be a mapping from the event space to the space of real-valued vectors of some dimension. In other words, a random vector $X$ is a map $X : \mathcal{S} \rightarrow \mathbb{R}^n$. Note that a random variable is just a special case of a random vector where $n = 1$.

The *expected value* (or *mean*) for a discrete random vector is defined to be

$$E(X) = \sum_i x_i f_X(x_i),$$

where $x_i$ is the $i$th value that random variable $X$ can take and $f_X$ is the PMF associated with $X$. Note that $E(X)$ is a vector in $\mathbb{R}^n$, where $n$ is the dimension of $X$. It is tempting to think that the expected value is the outcome most likely to occur, but this is not generally the case. The expected value of a single fair die roll is 3.5 which, of course, cannot occur.

The expected value (or mean) of a continuous random vector is defined to be

$$E(X) = \int_{x \in \mathbb{R}^n} x f_X(x)\, dx,$$

where $f_X$ is the PDF associated with $X$. As in the case of discrete random vectors, $E(X)$ is a vector in $\mathbb{R}^n$. We also denote $E(X)$ with $\bar{X}$. Expectation is a linear operator, which means that $E(aX + bY) = aE(X) + bE(Y)$.

The *variance* of a (scalar) random variable $x$ is $E((X - \bar{X})^2)$. For a scalar random variable the variance is denoted $\sigma^2$. For a random vector we can consider the variance of each element $X_i$ of $X$ individually. The variance of $X_i$ is denoted $\sigma_i^2$.

Now we want to consider the effect of one variable on another. This is termed *covariance* between two random variables $X_i$ and $X_j$. Let $\sigma_{ij} = E((X_i - \bar{X}_i)(X_j - \bar{X}_j))$. By this definition $\sigma_{ii}$ is the same as $\sigma_i^2$, the variance of $X_i$. For $i \neq j$, if $\sigma_{ij} = 0$, then $X_i$ and $X_j$ are independent of each other. The *covariance matrix* of a random vector $X$ is defined to be

$$P_X = E((X - \bar{X})(X - \bar{X})^T).$$

The $n \times n$ matrix $P_X$ contains the variances and covariances within the random vector $X$. Specifically, the element in the $i$th row, $j$th column of $P_X$ will be identical to the $\sigma_{ij}$ defined above.

## I.3 Multivariate Gaussian Distributions

A random vector $X$ is said to have a multivariate Gaussian distribution if it is described by the PDF

$$f_X(x) = \frac{1}{\sqrt{(2\pi)^n |P_X|}} e^{-\frac{1}{2}(x-\bar{X})^T P_X^{-1} (x-\bar{X})}, \tag{I.1}$$

where $\bar{X} \in \mathbb{R}^n$ is the mean vector and $P_X \in \mathbb{R}^{n \times n}$ is the covariance matrix. It can be verified by direct substitution that $\bar{X}$ and $P_X$ are in fact the mean and covariance matrix of $X$ as defined in the section above.

# J Linear Systems and Control

THIS APPENDIX gives a brief review of the theory of linear time invariant (LTI) dynamical systems. Many dynamical systems that appear in science and engineering can be approximated by LTI systems, and linear systems theory provides important tools to control and observe them. We focus on the so-called *state space* formulation of LTI systems because that is the formulation used in the Kalman filter (see chapter 8). In this appendix we present some of the more fundamental concepts of LTI state-space systems, including stability, feedback control, and observability.

## J.1 State Space Representation

Consider as an example the mass-spring-damper system depicted in figure J.1, where $z(t)$ denotes the position of the mass $m$ at time $t$. If we assume that the spring is linear, then the force applied by the spring is given as $F_s = -kz(t)$. Likewise, if we assume that the damper is linear, then the force applied by the damper is proportional to the velocity of the mass, yielding $F_d = -\gamma \frac{dz}{dt}(t) \overset{\triangle}{=} \gamma \dot{z}(t)$. For now we assume the externally applied force $F_{\text{ext}} = 0$. Summing these forces and applying Newton's law (force $=$ mass $\times$ acceleration) yields

$$m\ddot{z}(t) = -\gamma \dot{z}(t) - kz(t). \tag{J.1}$$

This second-order ordinary differential equation (ODE) provides a mathematical description of how the position and velocity of mass change with time. Accordingly, we call equation (J.1) a *model* of the mass-spring-damper system. If the position $z$ and

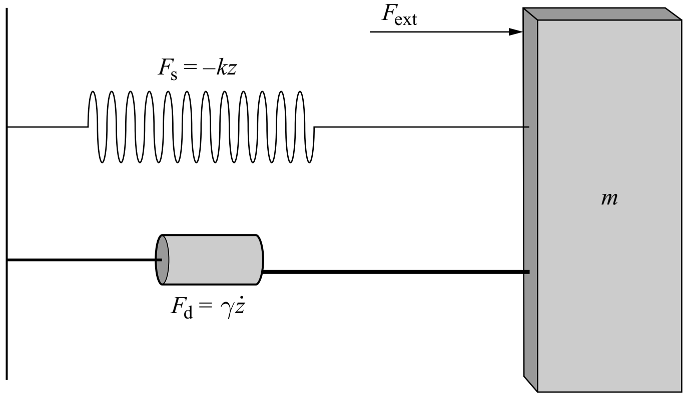


**Figure J.1** Mass–spring–damper system.

velocity $\dot{z}$ are known at some instant of time $t_0$, then the solution to equation (J.1) subject to initial conditions $z(t_0)$ and $\dot{z}(t_0)$ will match the trajectory of the physical system.

Now define the vector

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ \dot{z}(t) \end{bmatrix}.$$

Equation (J.1) can be rewritten in terms of $x$ as follows:

$$\dot{x}(t) = \begin{bmatrix} \dot{z}(t) \\ \ddot{z}(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -\frac{1}{m}\left(\gamma\dot{z}(t) + kz(t)\right) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -\frac{1}{m}\left(\gamma x_2(t) + kx_1(t)\right) \end{bmatrix},$$

which can finally be summarized as

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\gamma}{m} \end{bmatrix} x(t). \tag{J.2}$$

Thus we have taken a second-order scalar ODE and rewritten it as a first-order vector ODE. We call this first-order vector ODE the *state-space representation* of the mass-spring-damper system, and the state vector $x(t)$ is a member of the *state space*. Since the right hand side of equation (J.2) can be written as a constant matrix multiplied by the state vector, this system is both linear and time invariant.

Generally, an LTI state-space system can be written as the vector ODE,

$$\dot{x}(t) = Ax(t); \qquad x(t_0) = x_0, \tag{J.3}$$

where $x(t) \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$. This ODE is sometimes called a *vector field* because it assigns a vector $Ax$ to each point $x$ in the state space. This ODE has a unique solution, and the solution can be written in closed form,

$$x(t) = e^{A(t-t_0)}x_0, \tag{J.4}$$

where the matrix exponential is defined by the Peano-Baker series

$$\begin{aligned} e^{A(t-t_0)} &= \sum_{i=0}^{\infty} \frac{A^i\,(t-t_0)^i}{i!} \\ &= I_{n\times n} + A\,(t-t_0) + \frac{A^2(t-t_0)^2}{2!} + \cdots . \end{aligned}$$

## J.2 Stability

Assuming that the matrix $A$ has full rank, then the point $x = 0$ is the only point in the state space that satisfies the equilibrium condition $\dot{x} = 0$. The point $x = 0$ is called an equilibrium point. Note that the state will not move from an equilibrium point. If the initial condition is $x(t_0) = 0$, then $x(t)$ will remain at 0 for all time. In this section we discuss the stability of the origin of an LTI system.

We begin by defining a few notions of stability. An equilibrium point $x_e$ (in the case of LTI systems $x_e = 0$) is said to be *stable* if for every $\epsilon > 0$ there exists a $\delta > 0$ such that whenever the initial condition satisfies $\|x_e - x(t_0)\| < \delta$ the solution $x(t)$ satisfies $\|x_e - x(t)\| < \epsilon$ for all time $t > 0$. In other words, stable means that if the initial condition starts close enough to the equilibrium, then the solution will never drift very far away. $x_e$ is said to be *asymptotically stable* if it is stable and $\|x_e - x(t)\| \to 0$ as $t \to \infty$. Likewise, $x_e$ is said to be *unstable* if it is neither stable nor asymptotically stable.

It is worth noting that for LTI systems, the stability properties are global. If they hold on any open subset of the state space, then they hold everywhere. Stability can be characterized in terms of the eigenvalues of the matrix $A$, as stated in the following theorem:

THEOREM J.2.1 (LTI stability) *Consider the LTI system stated in equation (J.3), and let $\lambda_i$, $i \in \{1, 2, \ldots, n\}$ denote the eigenvalues of $A$. Let* $\mathrm{re}(\lambda_i)$ *denote the real part of $\lambda_i$ Then the following holds:*

1. $x_e = 0$ *is stable if and only if* $\mathrm{re}(\lambda_i) \le 0$ *for all $i$.*

2. $x_e = 0$ *is asymptotically stable if and only if* $\mathrm{re}(\lambda_i) < 0$ *for all $i$.*

3. $x_e = 0$ *is unstable if and only if* $\mathrm{re}(\lambda_i) > 0$ *for some $i$.*

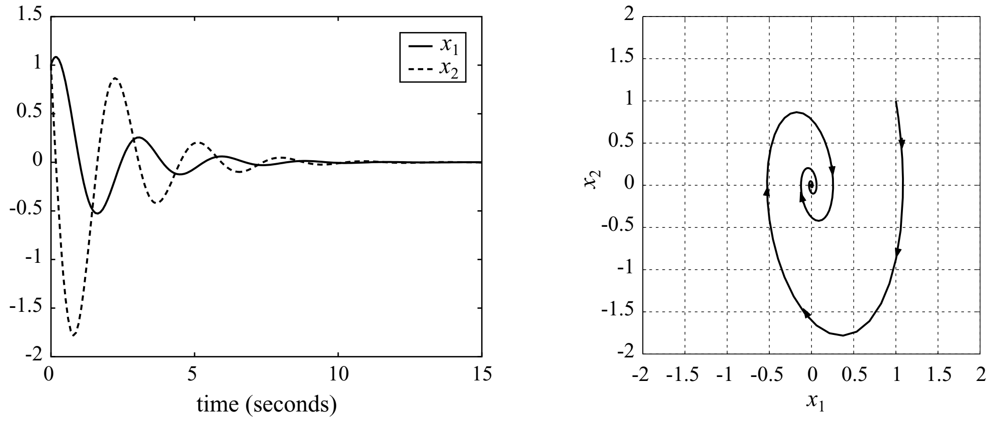


**Figure J.2** Asymptotical stability. (Left) The states $x_1$ and $x_2$ ($z$ and $\dot{z}$, respectively) plotted as time evolves. (Right) Phase plane plot of $x_2$ vs. $x_1$.

Consider the mass-spring-damper example. The eigenvalues of $A$ are

$$\frac{-\gamma \pm \sqrt{\gamma^2 - 4km}}{2m}.$$

When the damping term is positive, the real parts of the eigenvalues are negative and the system is asymptotically stable. Figure J.2 shows two different representations of the trajectory of the mass-spring-damper system with $m = 1$, $k = 5$, and $\gamma = 1$. The figure on the left shows the values of $x_1$ and $x_2$ plotted as functions of time. As expected for an asymptotically stable system, both converge to zero. The figure on the right shows the trajectory in state space by plotting $x_2$ vs. $x_1$. This is sometimes referred to as a “phase plane” plot. The direction in which the trajectory flows is depicted by arrows. Here the trajectory starts at the initial condition and spirals into the origin. When the damping is zero, the system solution is a bounded oscillation and hence is stable but not asymptotically stable. Figure J.3 plots the time and phase plane representations of the stable trajectory that results when $m = 1$, $k = 5$, and $\gamma = 0$. Note that in the phase plane the periodic oscillation becomes a closed loop. When the damping is negative the damping term actually adds energy to the system, creating an oscillation that grows without bound. Time and phase plane plots for the case where $m = 1$, $k = 5$, and $\gamma = -0.4$ are shown in figure J.4.

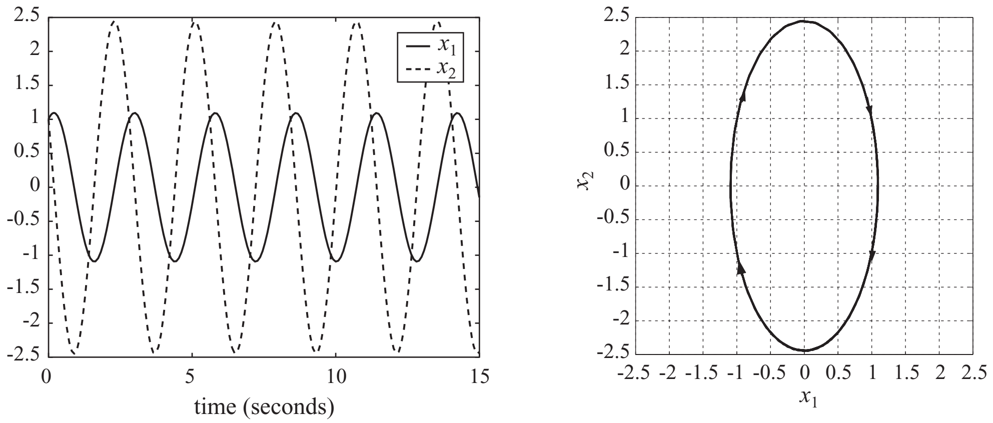


**Figure J.3** Stability. (Left) The states $x_1$ and $x_2$ ($z$ and $\dot{z}$, respectively) plotted as time evolves. (Right) Phase plane plot of $x_2$ vs. $x_1$.

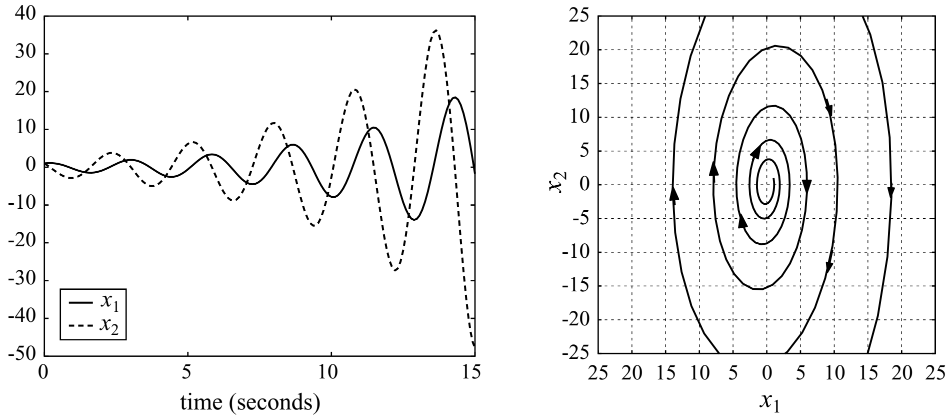


**Figure J.4** Instability. (Left) The states $x_1$ and $x_2$ ($z$ and $\dot{z}$, respectively) plotted as time evolves. (Right) Phase plane plot of $x_2$ vs. $x_1$.

## J.3 LTI Control Systems

Often one has the ability to affect the behavior of a dynamical system by applying some sort of external input. For example, in the mass-spring-damper system discussed earlier we can influence the trajectory of the system by applying a time-varying external force $F(t)$ to the mass. This results in the LTI control system

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\gamma}{m} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t). \tag{J.5}$$

More generically, we write an LTI control system as

$$\dot{x}(t) = Ax(t) + Bu(t); \qquad x(t_0) = x_0, \tag{J.6}$$

where the state vector $x(t) \in \mathbb{R}^n$ and the external input vector $u(t) \in \mathbb{R}^m$. The matrix $B \in \mathbb{R}^{n \times m}$. The matrix $A$ describes the system dynamics of the unforced system, i.e., $A$ describes how the state would evolve if the input were zero. $B$ describes how the inputs affect the evolution of the state.

The system described in equation (J.3) is said to be *controllable* if for any initial condition $x(t_0)$, there exists a continuous control input $u(t)$ that drives the solution $x(t)$ to the origin, $x = 0$. Note that the origin is an equilibrium point for the unforced system. This definition of controllability is equivalent to the definition of controllability for nonlinear systems presented in chapter 8, section 12.3 where the goal state is restricted to $x_{\text{goal}} = 0$.

THEOREM J.3.1 (LTI Controllability Test) *The LTI control system in equation (J.6) is controllable if and only if the matrix*

$$W_c = [B \; AB \; A^2B \; \cdots \; A^{n-1}B]$$

*has rank $n$.*

Because controllability is determined solely by the matrices $A$ and $B$, we can say that the pair $(A, B)$ is controllable if the system in equation (J.6) is controllable.

One common control objective is to make the origin of a naturally unstable system stable using state feedback. Consider the control input given by the state-dependent control law

$$u(t) = -Kx(t)$$

for some matrix $K \in \mathbb{R}^{m \times n}$. Substituting this into equation (J.6) yields

$$\dot{x}(t) = (A - BK)x(t).$$

As a result, we can examine the stability of this new system in terms of the eigenvalues of the matrix $A - BK$. One of the fundamental properties of real-valued matrices is that their eigenvalues must occur in complex conjugate pairs. If $a + bi$ is an eigenvalue of a matrix, then $a - bi$ must also be an eigenvalue of that matrix. Hence we define a collection of complex numbers $\Lambda = \{\lambda_i \mid i \in \{1, 2, \ldots, n\}\}$ to be *allowable* if for each $\lambda_i$ that has a nonzero imaginary part there is a corresponding conjugate $\lambda_j$. Now we are prepared to state an important result of linear control theory:

THEOREM J.3.2 (Eigenvalue Placement) *Consider the system of equation (J.6) and assume the pair $(A, B)$ is controllable and that the matrix $B$ has full column rank. Let $\Lambda = \{\lambda_i \mid i \in \{1, 2, \ldots, n\}\}$ be any allowable collection of complex numbers. Then there exists a constant matrix $K \in \mathbb{R}^{m\times n}$ such that the set of eigenvalues of $(A - BK)$ is equal to $\Lambda$.*

Under the assumptions of this theorem, we can place the eigenvalues of the matrix $A - BK$ in any allowable configuration using linear feedback. The task of stabilizing an LTI system is then simply a matter of finding a $K$ so that the corresponding eigenvalues have negative real parts. There are a number of algorithms to perform direct eigenvalue assignment (also sometimes called pole placement). Some of these are implemented in the MATLAB control systems toolbox. Similarly, the famous linear quadratic regulator (LQR) (see e.g., [396]) places the eigenvalues of $A - BK$ to optimize a user-defined cost function.

Consider as an example the mass-spring-damper system with negative damping. As was pointed out earlier, this system is unstable; solutions for initial conditions arbitrarily close to the origin will grow without bound. To use state feedback to stabilize this system, consider the matrix

$$A - BK = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\gamma}{m} \end{bmatrix} - \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} [k_1 \quad k_2] = \begin{bmatrix} 0 & 1 \\ -\frac{k+k_1}{m} & \frac{\gamma+k_2}{m} \end{bmatrix}.$$

The eigenvalues of $A - BK$ are

$$\frac{(-\gamma - k_2) \pm \sqrt{(-\gamma - k_2)^2 - 4(k - k_1)m}}{2m},$$

so we can ensure that the real part of both eigenvalues is negative by choosing $k_2$ such that $-\gamma - k_2 < 0$. This is equivalent to adding sufficient positive viscous damping to overcome the energy added by the negative damping term $\gamma$.

## J.4 Observing LTI Systems

Often it is not possible to directly measure the entire state of an LTI system. Rather, the state must be observed through the use of sensors that provide some lower-dimensional measurement of the current state. If it were possible to measure only velocity in the mass-spring-damper example, then equations of motion together with the output equation for the system would be

$$\begin{aligned} \dot{x}(t) &= \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{\gamma}{m} \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F(t), \\ y(t) &= [0 \quad 1]\, x(t), \end{aligned} \tag{J.7}$$

where $y(t)$ represents the output signal coming from the sensor. We write a general LTI system with output equation as

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t); \qquad x(t_0) = x_0, \\ y(t) &= Cx(t), \end{aligned} \tag{J.8}$$

where the state vector $x(t) \in \mathbb{R}^n$, the control vector $u(t) \in \mathbb{R}^m$, and the output vector $y(t) \in \mathbb{R}^p$. The constant matrix $C \in \mathbb{R}^{p \times n}$. Note that the matrix $C$ may not be invertible (it is usually not even square!), so the state at any instant $x(t)$ cannot be directly observed from the measurement at that instant $y(t)$. We must instead reconstruct the state by measuring the output over some interval of time and using knowledge of the system dynamics. A device that performs such a reconstruction is called an *observer*.

We say that the system of equation (J.8) is *observable* if it is possible to determine the initial state $x(t_0)$ by observing the known signals $y(t)$ and $u(t)$ over some period of time.

THEOREM J.4.1 (LTI Observability Test) *The LTI control system in equation (J.8) is observable if and only if the matrix*

$$W_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

*has rank n.*

As in the case of controllability, we say that the pair $(A, C)$ is observable if the system in equation (J.8) is observable. Note that the pair $(A, C)$ is observable if and only if

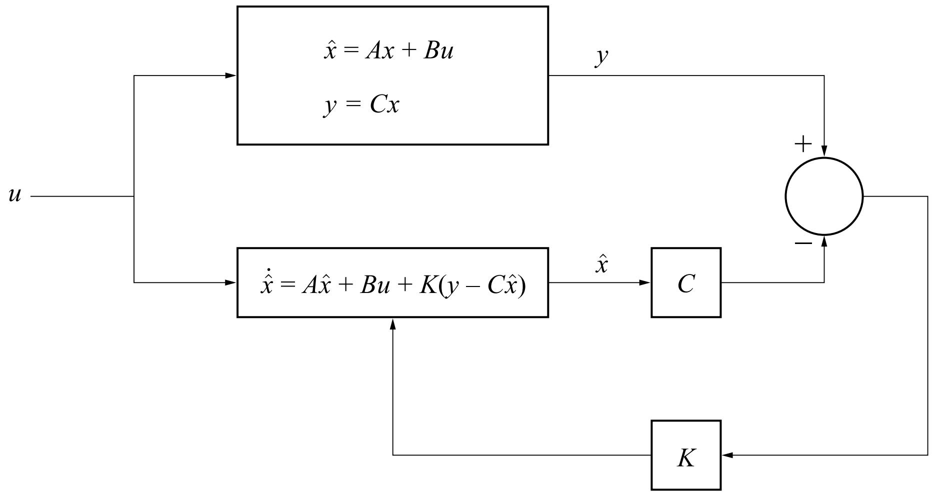


**Figure J.5** Block diagram for a linear observer.

the pair $(A^T, C^T)$ is controllable. If the pair $(A, B)$ is controllable and the pair $(A, C)$ is observable, then the system [and the triple $(A, B, C)$] is said to be *minimal*.

Now consider an observer defined by the ODE

$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t)). \tag{J.9}$$

Note that this ODE requires that we know the matrices $A$, $B$, and $C$ as well as the input $u(t)$ and output $y(t)$. The vector $\hat{x}(t)$ is called the *state estimate* produced by this observer. As shown in the block diagram in figure J.5, this observer is essentially a copy of the original dynamic system with a correcting term that is a linear function of the difference between the measured output $y(t)$ and the estimated output $C\hat{x}(t)$. The task is then to try to choose $K$ so that the correcting term forces the state estimate to converge to the actual value.

If we define the error signal $e(t) = x(t) - \hat{x}(t)$, we can examine how the error evolves with time:

$$\begin{aligned}\dot{e}(t) &= \dot{x}(t) - \dot{\hat{x}}(t)\\ &= Ax(t) + Bu(t) - (A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t)))\\ &= A(x(t) - \hat{x}(t)) - K(Cx(t) - C\hat{x}(t))\\ &= (A - KC)e(t)\end{aligned}$$

If $e(t) \to 0$, then $\hat{x}(t) \to x(t)$. So the state estimate $\hat{x}(t)$ that results from the observer presented in equation (J.9) converges to the actual state $x(t)$ if $K$ is chosen so that the unforced LTI system $\dot{e}(t) = (A - KC)e(t)$ is asymptotically stable.

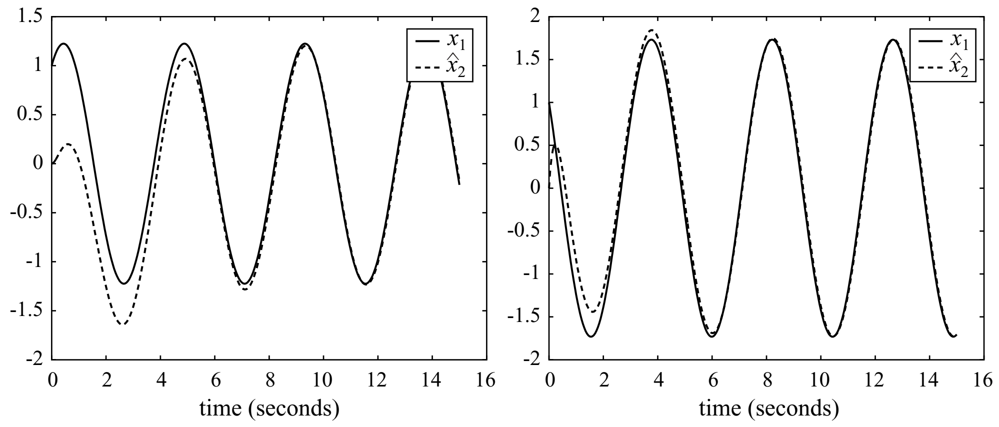


**Figure J.6** Solid lines represent the actual state and the dashed line represents the state estimate determined by the observer. The left figure depicts $x_1$ and the right $x_2$.

Recall that the eigenvalues of any matrix are equal to the eigenvalues of its transpose, so the eigenvalues of $A - KC$ are identical to the eigenvalues of $A^T - C^T K^T$. According to theorem J.3.2, we can place the eigenvalues of $A^T - C^T K^T$ in any allowable configuration provided that the pair $(A^T, C^T)$ is controllable and the matrix $C^T$ has full column rank. This is equivalent to saying that the eigenvalues of $A - KC$ can be placed in any allowable configuration provided that the pair $(A, C)$ is observable and $C$ has full row rank. Under these conditions, it is possible to chose a $K$ so that the observer estimate $\hat{x}(t)$ converges to $x(t)$.

Consider the mass-spring-damper system of equation (J.7). The matrix

$$A - KC = \begin{bmatrix} 0 & 1 - k_1 \\ -\frac{k}{m} & -\frac{\gamma}{m} - k_2 \end{bmatrix}.$$

The eigenvalues of this matrix are

$$\frac{-(\gamma + mk_2) \pm \sqrt{(-\gamma - mk_2)^2 - 4m(k - k_1)}}{2m},$$

so choosing $k_2$ such that $-\gamma - mk_2 < 0$ will guarantee that the observer given in equation (J.9) converges, meaning that after some initial transient, estimate $\hat{x}(t)$ will provide a good approximation of the state. For the case where $m = 1$, $k = 2$, and $\gamma = 0$, the choice of $K = [0\ 2]^T$ will provide a convergent observer. Figure J.6 shows how the estimates $\hat{x}_1(t)$ and $\hat{x}_2(t)$ converge to $x_1(t)$ and $x_2(t)$, respectively.

## J.5 Discrete Time Systems

The previous sections dealt with an LTI system whose trajectories were continuous in time. In practice, a continuous dynamical system is usually sampled at regular time intervals. The sampled or *discrete time* signal is then fed into a computer as a sequence of numbers. The computer can then use this sequence to calculate a desired control input or to estimate the state. In this section we present an overview of the theory of discrete time LTI systems and their relationship to their continuous time cousins.

Consider the continuous time signal $x(t)$. We define a sequence of vectors using the formula $x_s(k) = x(t_0 + kT)$. The sequence $x_s(k)$ is the *discrete time sampling* of the continuous signal $x(t)$. In the future, we will abuse notation and drop the $s$ subscript on the discrete time sequence. The continuous and discrete signals can be differentiated by the letter used in their argument; $x(k)$ represents an element of the sequence and $x(t)$ denotes the continuous time signal.

Using the first-order derivative approximation

$$\dot{x}(t_0 + kT) \approx \frac{x(k+1) - x(k)}{T}$$

and substituting into the continuous time LTI system of equation (J.8) yields

$$\frac{x(k+1) - x(k)}{T} \approx Ax(k) + Bu(k),$$

which leads to

$$x(k+1) \approx x(k) + TAx(k) + TBu(k).$$

Defining $F = I_{n\times n} + TA$, $G = TB$, and $H = C$, we can then write a discrete time approximation of the continuous system:

$$\begin{aligned} \dot{x}(k+1) &= Fx(k) + Gu(k); \qquad x(0) = x_0 \\ y(k) &= Hx(k) \end{aligned} \tag{J.10}$$

Most of the concepts from continuous LTI systems have direct analogs in discrete time LTI systems. We discuss them briefly here.

### J.5.1 Stability

The discrete time notions of stability, asymptotic stability, and instability follow directly from the continuous time definitions. As in the case of continuous systems, the stability of the unforced system $x(k+1) = Fx(k)$ can be evaluated in terms of the eigenvalues of $F$:

THEOREM J.5.1 (Discrete Time LTI Stability) *Consider the unforced discrete time LTI system described by the equation $x(k+1) = Fx(k)$, and let $\lambda_i$, $i \in \{1, 2, \ldots, n\}$ denote the eigenvalues of $F$. Then the following hold:*

1. *$x_e = 0$ is stable if and only if $|\lambda_i| \leq 1$ for all $i$.*

2. *$x_e = 0$ is asymptotically stable if and only if $|\lambda_i| < 1$ for all $i$.*

3. *$x_e = 0$ is unstable if and only if $|\lambda_i| > 1$ for some $i$.*

### J.5.2 Controllability and Observability

The properties of controllability and observability for the discrete time LTI system follow from the properties of the continuous time system. The controllability test is the same for both: the pair $(F, G)$ is controllable if and only if the matrix $[G \;\; FG \;\; F^2G \;\; \cdots \;\; F^{n-1}G]$ has rank $n$. The pair $(F, H)$ is observable if and only if the pair $(F^T, H^T)$ is controllable. As in the case of continuous systems, construction of linear state feedback control laws or linear observers results in a pole placement problem which can be solved if the system is controllable or observable, respectively.

# *Bibliography*

[1] http://www.accuray.com/ck/how9.htm and http://www.cksociety.org/.

[2] http://www.intuitivesurgical.com/about_intuitive/index.html.

[3] http://computermotion.wwwa.com/productsandsolutions/products/zeus/index.cfm.

[4] http://www.aemdesign.com.

[5] http://www.sbsi-sol-optimize.com/NPSOL.htm.

[6] http://www.vni.com.

[7] http://www.nag.com.

[8] *Webster's Ninth New Collegiate Dictionary*. Merriam-Webster, Inc., Springfield, MA, 1990.

[9] R. Abraham, J. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications*. Springer-Verlag, New York, 2 edition, 1988.

[10] R. Abraham and J. E. Marsden. *Foundations of Mechanics*. Addison-Wesley, 1985.

[11] E. U. Acar and H. Choset. Sensor-based coverage of unknown environments: Incremental construction of Morse decompositions. *International Journal of Robotics Research,* 21:345–366, April 2002.

[12] E. U. Acar, H. Choset, A. A. Rizzi, P. Atkar, and D. Hull. Morse decompositions for coverage tasks. *International Journal of Robotics Research,* 21:331–344, April 2002.

[13] S. Akella, W. Huang, K. Lynch, and M. Mason. Parts feeding on a conveyor with a one joint robot. *Algorithmica (Special Issue on Robotics),* 26(3/4):313–344, 2000.

[14] M. Akinc, K. E. Bekris, B. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In *International Symposium on Robotics Research,* 2003. Book to appear.

[15] R. Alami, J. Laumond, and T. Siméon. Two manipulation planning algorithms. In K. Goldberg, D. Halperin, J. C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics,* pages 109–125. A.K. Peters, 1995.

[16] R. Alami, T. Siméon, and J. P. Laumond. A geometrical approach to planning manipulation tasks. In *International Symposium on Robotics Research,* pages 113–119, 1989.

[17] P. Allen and I. Stamos. Integration of range and image sensing for photorealistic 3D modeling. In *IEEE International Conference on Robotics and Automation,* 2000.

[18] N. M. Amato, B. Bayazit, L. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3d workspaces. In P. Agarwal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective,* pages 156–168. AK Peters, 1998.

[19] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE International Conference on Robotics and Automation,* pages 630–637, 1998.

[20] N. M. Amato, K. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. In *International Conference on Research in Computational Molecular Biology,* pages 2–11, April 2002.

[21] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. In *International Conference on Research in Computational Molecular Biology,* pages 287–296, 2001.

[22] E. Anshelevich, S. Owens, F. Lamiraux, and L. E. Kavraki. Deformable volumes in path planning applications. In *IEEE International Conference on Robotics and Automation,* pages 2290–2295, 2000.

[23] M. Apaydin, D. Brutlag, C. Guestrin, D. Hsu, J. C. Latombe, and C. Varm. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. *Journal of Computational Biology,* 10:257–281, 2003.

[24] M. Apaydin, C. Guestrin, C. Varma, D. Brutlag, and J. C. Latombe. Studying protein-ligand interactions with stochastic roadmap simulation. *Bioinformatics,* 18(2):18–26, 2002.

[25] M. S. Apaydin, D. L. Brutlag, C. Guestrin, D. Hsu, and J. C. Latombe. Stochastic roadmap simulation: An efficient representation and algorithm for analyzing molecular motion. In *International Conference on Research in Computational Molecular Biology,* pages 12–21, April 2002.

[26] V. I. Arnold. *Mathematical Methods of Classical Mechanics*. Springer-Verlag, 1989.

[27] K. Arras, N. Tomatis, B. Jensen, and R. Siegwart. Multisensor on-the-fly localization: Precision and reliability for applications. *Robotics and Autonomous Systems,* 34(2-3):131–143, 2001.

[28] K. Arras and S. Vestli. Hybrid, high-precision localization for the mail distributing mobile robot system MOPS. In *IEEE International Conference on Robotics and Automation,* 1998.

[29] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing,* 50(2):174–188, 2002.

[30] S. Arya, D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *47th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA),* pages 271–280, 1993.

[31] F. Aurenhammer. Voronoi diagrams—A survey of a fundamental geometric structure. *ACM Computing Surveys,* 23:345–405, 1991.

[32] D. Avots, E. Lim, R. Thibaux, and S. Thrun. A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2002.

[33] B. Baginski. Motion planning for manipulators with many degrees of freedom—The BB Method. Ph.D. Thesis, Technische Universität München, 1998.

[34] J. Baillieul and B. Lehman. Open-loop control using oscillatory inputs. In *CRC Control Handbook,* pages 967–980. CRC Press, Boca Raton, FL, 1996.

[35] D. J. Balkcom and M. T. Mason. Time optimal trajectories for differential drive vehicles. *International Journal of Robotics Research,* 21(3):199–217, Mar. 2002.

[36] J. Barraquand and P. Ferbach. A penalty function method for constrained motion planning. In *IEEE International Conference on Robotics and Automation,* pages 1235–1242, 1994.

[37] J. Barraquand, L. E. Kavraki, J. C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. *International Journal of Robotics Research,* 16(6):759–774, 1997.

[38] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Man and Cybernetics,* 22(2):224–241, Mar/Apr 1992.

[39] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. Technical Report STAN-CS-89-1257, Stanford University, Stanford CA, 1989.

[40] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research,* 10(6):628–649, Dec. 1991.

[41] J. Barraquand and J. C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica,* 10:121–155, 1993.

[42] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer-Verlag, 2003.

[43] K. E. Bekris, B. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Multiple query motion planning using single query primitives. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 656–661, 2003.

[44] J. Bentley. Multidimensional divide and conquer. *Communications of the ACM,* 23(4), 1980.

[45] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, second edition, 1999.

[46] P. Besl and N. McKay. A method for registration of 3D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 18(14):239–256, 1992.

[47] P. Bessiere, E. Mazer, and J.-M. Ahuactzin. Planning in continuous space with forbidden regions: The Ariadne's clew algorithm. In K. Goldberg, K. Goldberg, R. Wilson, and D. Halperin, editors, *Algorithmic Foundations of Robotics (WAFR),* pages 39–47. A.K. Peters, Wellsley MA, 1995.

[48] P. Bessiere, E. Mazer, and J.-M. Ahuactzin. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research (JAIR),* 9:295–316, 1998.

[49] J. T. Betts. Survey of numerical methods for trajectory optimization. *AIAA Journal of Guidance, Control, and Dynamics,* 21(2):193–207, March-April 1998.

[50] A. M. Bloch. *Nonholonomic Mechanics and Control*. Springer, New York, 2003.

[51] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research,* 4(3):3–17, Fall 1985.

[52] R. Bohlin. Path planning in practice: Lazy evaluation on a multi-resolution grid. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2001.

[53] R. Bohlin and L. E. Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation,* pages 521–528, 2000.

[54] R. Bohlin and L. E. Kavraki. A randomized algorithm for robot path planning based on lazy evaluation. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Handbook on Randomized Computing,* pages 221–249. Kluwer Academic Publishers, 2001.

[55] K.-F. Böhringer, B. R. Donald, L. E. Kavraki, and F. Lamiraux. Part orientation to one or two stable equilibria using programmable force fields. *IEEE Transactions on Robotics and Automation*, 16(2):731–747, 2000.

[56] K. Böhringer, B. R. Donald, and N. MacDonald. Programmable vector fields for distributed manipulation, with application to mems actuator arrays and vibratory part feeders. *International Journal of Robotics Research*, 18:168–200, Feb. 1999.

[57] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. John Wiley and Sons Inc., New York, NY, 2000.

[58] B. Bonnard. Contrôlabilité des systèmes nonlinéaires. *C. R. Acad. Sci. Paris,* 292:535–537, 1981.

[59] V. Boor, N. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation,* pages 1018–1023, 1999.

[60] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Academic Press, 1986.

[61] J. Borenstein, B. Everett, and L. Feng. *Navigating Mobile Robots: Systems and Techniques*. A.K. Peters, Ltd., Wellesley, MA, 1996.

[62] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation,* pages 1481–1487, 2001.

[63] G. E. Bredon. *Topology and Geometry*. Springer-Verlag, New York, NY, 1993.

[64] T. Bretl, J. C. Latombe, and S. Rock. Toward autonomous free climbing robots. In *International Symposium on Robotics Research,* 2003. Book to appear.

[65] R. W. Brockett. Nonlinear systems and differential geometry. *Proceedings of the IEEE,* 64(1):61–72, Jan. 1976.

[66] R. W. Brockett. Control theory and singular Riemannian geometry. In P. J. Hilton and G. S. Young, editors, *New Directions in Applied Mathematics,* pages 11–27. Springer-Verlag, 1982.

[67] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions Systems, Man, and Cybernetics,* 15:224–233, 1985.

[68] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics,* 13(3):190–197, 1983.

[69] R. C. Brost. *Analysis and Planning of Planar Manipulation Tasks*. PhD thesis, Carnegie Mellon University, Jan. 1991. Available as Technical Report CMU-CS-91-149.

[70] R. C. Brost. Computing the possible rest configurations of two interacting polygons. In *IEEE International Conference on Robotics and Automation,* pages 686–693, Apr. 1991.

[71] A. E. Bryson. *Dynamic Optimization*. Addison-Wesley, 1998.

[72] A. E. Bryson and Y. C. Ho. *Applied Optimal Control*. Hemisphere Publishing, New York, 1975.

[73] J. Buhmann, W. Burgard, A. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine,* 16(2):31–38, Summer 1995.

[74] F. Bullo. Series expansions for the evolution of mechanical control systems. *SIAM Journal on Control and Optimization,* 40(1):166–190, 2001.

[75] F. Bullo. Averaging and vibrational control of mechanical systems. *SIAM Journal on Control and Optimization,* 41:542–562, 2002.

[76] F. Bullo, N. E. Leonard, and A. D. Lewis. Controllability and motion algorithms for underactuated Lagrangian systems on Lie groups. *IEEE Transactions on Automatic Control,* 45(8):1437–1454, 2000.

[77] F. Bullo and A. D. Lewis. *Geometric Control of Mechanical Systems*. Springer, 2004.

[78] F. Bullo, A. D. Lewis, and K. M. Lynch. Controllable kinematic reductions for mechanical systems: Concepts, computational tools, and examples. In *2002 International Symposium on the Mathematical Theory of Networks and Systems,* Aug. 2002.

[79] F. Bullo and K. M. Lynch. Kinematic controllability for decoupled trajectory planning of underactuated mechanical systems. *IEEE Transactions on Robotics and Automation,* 17(4):402–412, Aug. 2001.

[80] F. Bullo and M. Žefran. On mechanical control systems with nonholonomic constraints and symmetries. *Systems and Control Letters,* 45(2):133–143, Jan. 2002.

[81] W. Burgard, A. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence,* 114(1-2), 2000.

[82] W. Burgard, A. Derr, D. Fox, and A. Cremers. Integrating global position estimation and position tracking for mobile robots: the dynamic Markov localization approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 1998.

[83] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the National Conference on Artificial Intelligence (AAAI),* 1996.

[84] W. Burgard, D. Fox, H. Jans, C. Matenar, and S. Thrun. Sonar-based mapping of large-scale mobile robot environments using EM. In *Proc. of the International Conference on Machine Learning (ICML),* 1999.

[85] L. Bushnell, D. Tilbury, and S. Sastry. Steering three-input nonholonomic systems: The fire-truck example. *International Journal of Robotics Research,* 14(4):366–381, 1995.

[86] Z. J. Butler, A. A. Rizzi, and R. L. Hollis. Contact sensor-based coverage of rectilinear environments. In *Proc. of IEEE Int'l Symposium on Intelligent Control,* Sept. 1999.

[87] P. E. Caines and E. S. Lemch. On the global controllability of Hamiltonian and other nonlinear systems: Fountains and recurrence. In *IEEE International Conference on Decision and Control,* pages 3575–3580, 1998.

[88] S. Cameron. Collision detection by four-dimensional intersection testing. *IEEE Transactions on Robotics and Automation,* pages 291–302, 1990.

[89] S. Cameron. Enhancing GJK: Computing minimum distance and penetration distanses between convex polyhedra. In *IEEE International Conference on Robotics and Automation,* pages 3112–3117, 1997.

[90] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[91] J. F. Canny. Constructing roadmaps of semi-algebraic sets I: Completeness. *Artificial Intelligence,* 37:203–222, 1988.

[92] J. F. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal,* 35(5):504–514, 1993.

[93] J. F. Canny and M. Lin. An opportunistic global path planner. *Algorithmica,* 10:102–120, 1993.

[94] J. F. Canny, J. Reif, B. Donald, and P. Xavier. On the complexity of kinodynamic planning. In *IEEE Symposium on the Foundations of Computer Science,* pages 306–316, White Plains, NY, 1988.

[95] J. F. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. 20th ACM Symposium on the Theory of Computing,* pages 460–469, 1998.

[96] Z. L. Cao, Y. Huang, and E. Hall. Region filling operations with random obstacle avoidance for mobile robots. *Journal of Robotic systems,* pages 87–102, February 1988.

[97] J. Carpenter, P. Clifford, and P. Fernhead. An improved particle filter for nonlinear problems. *IEE Proceedings on Radar and Sonar Navigation,* 146(2-7), 1999.

[98] A. Casal. *Reconfiguration Planning for Modular Self-Reconfigurable Robots*. PhD thesis, Stanford University, Stanford, CA, 2002.

[99] J. Castellanos, J. Montiel, J. Neira, and J. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation,* 15(5):948–953, 1999.

[100] J. Castellanos and J. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Kluwer Academic Publishers, Boston, MA, 2000.

[101] P. C. Chen and Y. K. Hwang. SANDROS: A motion planner with performance proportional to task difficulty. *IEEE International Conference on Robotics and Automation,* pages 2346–2353, 1992.

[102] P. C. Chen and Y. K. Hwang. SANDROS: A dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation,* 14(3):390–403, June 1998.

[103] P. Cheng and S. M. LaValle. Reducing metric sensitivity in randomized trajectory design. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 43–48, 2001.

[104] H. Choset. Nonsmooth analysis, convex analysis, and their applications to motion planning. *Special Issue of the Int. Jour. of Comp. Geom. and Apps.,* 1998.

[105] H. Choset and J. Burdick. Sensor based motion planning: Incremental construction of the hierarchical generalized Voronoi graph. *International Journal of Robotics Research,* 19(2):126–148, February 2000.

[106] H. Choset and J. Burdick. Sensor based motion planning: The hierarchical generalized Voronoi graph. *International Journal of Robotics Research,* 19(2):96–125, February 2000.

[107] H. Choset and J. Y. Lee. Sensor-based construction of a retract-like structure for a planar rod robot. *IEEE Transaction of Robotics and Automation,* 17, 2001.

[108] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (T-SLAM). *IEEE Transactions on Robotics Automation,* 17, April 2001.

[109] H. Choset, K. Nagatani, and A. Rizzi. Sensor based planning: Using a honing strategy and local map method to implement the generalized Voronoi graph. In *SPIE Conference on Systems and Manufacturing,* Pittsburgh, PA, 1997.

[110] H. Choset and P. Pignon. Coverage path planning: The boustrophedon decomposition. In *Proceedings of the International Conference on Field and Service Robotics,* Canberra, Australia, December 1997.

[111] P. Choudhury and K. M. Lynch. Trajectory planning for second-order underactuated mechanical systems in the presence of obstacles. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V,* pages 559–575. Springer-Verlag, 2002.

[112] W.-L. Chow. Uber systemen von linearen partiellen differentialgleichungen erster ordnung. *Math. Ann.,* 117:98–105, 1939.

[113] S. Ciarcia. An ultrasonic ranging system. *Byte Magazine,* pages 113–123, October 1984.

[114] F. H. Clarke. *Optimization and Nonsmooth Analysis*. Society of Industrial and Applied Mathematics, Philadelphia, PA, 1990.

[115] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Symposium on Interactive 3D Graphics,* pages 189–196, 218, 1995.

[116] J. Colegrave and A. Branch. A case study of autonomous household vacuum cleaner. In *AIAA/NASA CIRFFSS,* 1994.

[117] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Lecture Notes in Computer Science,* volume 33, pages 134–183. Springer-Verlag, 1975.

[118] H. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[119] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2002.

[120] J. Cortes, S. Martinez, J. P. Ostrowski, and H. Zhang. Simple mechanical control systems with constraints and symmetry. *SIAM Journal on Control and Optimization,* 41(3):851–874, 2002.

[121] J. Cortés, T. Simeon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains. In *IEEE International Conference on Robotics and Automation,* pages 2141–2146, 2002.

[122] J. Crowley. World modeling and position estimation for a mobile robot using ultrasound ranging. In *IEEE International Conference on Robotics and Automation,* 1989.

[123] T. Danner and L. E. Kavraki. Randomized planning for short inspection paths. In *IEEE International Conference on Robotics and Automation,* pages 971–976, San Fransisco, CA, April 2000. IEEE Press.

[124] M. de Berg, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, Berlin, 1997.

[125] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo Localization for mobile robots. In *IEEE International Conference on Robotics and Automation,* 1999.

[126] F. Dellaert, S. Seitz, C. Thorpe, and S. Thrun. Structure from motion without correspondence. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR),* 2000.

[127] A. O. Dempster, A. N. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B,* 39(1):1–38, 1977.

[128] G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics and Automation,* 2001.

[129] A. W. Divelbiss and J. Wen. Nonholonomic path planning with inequality constraints. In *IEEE International Conference on Decision and Control,* pages 2712–2717, 1993.

[130] A. W. Divelbiss and J.-T. Wen. A path space approach to nonholonomic motion planning in the presence of obstacles. *IEEE Transactions on Robotics and Automation,* 13(3):443–451, 1997.

[131] M. P. do Carmo. *Riemannian Geometry*. Birkhäuser, Boston, MA, 1992.

[132] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence,* 31:295–353, 1987.

[133] B. R. Donald, P. Xavier, J. Canny, and J. Reif. Kinodynamic motion planning. *Journal of the Association for Computing Machinery,* 40(5):1048–1066, Nov. 1993.

[134] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators. *Algorithmica,* 4(6):480–530, 1995.

[135] B. R. Donald and P. Xavier. Provably good approximation algorithms for optimal kinodynamic planning: robots with decoupled dynamics bounds. *Algorithmica,* 4(6):443–479, 1995.

[136] A. Doucet. On sequential simulation-based methods for Bayesian filtering. Technical report, Department of Engeneering, University of Cambridge, 1998.

[137] A. Doucet, J. de Freitas, K. Murphy, and S. Russel. Rao-Blackwellised particle filtering for dynamic Bayesian networks. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI),* 2000.

[138] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[139] D. Duff, M. Yim, and K. Roufas. Evolution of polybot: A modular reconfigurable robot. In *Proc. of the Harmonic Drive Intl. Symposium,* Nagano, Japan, 2001.

[140] S. Ehmann and M. C. Lin. Swift: Accelerated distance computation between convex polyhedra by multi-level Voronoi marching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2000.

[141] S. A. Ehmann and M. C. Lin. Geometric algorithims: Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum—Proc. of Eurographics,* 20:500–510, 2001.

[142] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation,* RA-3:249–265, June 1987.

[143] A. Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Percepti on and Navigation*. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 1989.

[144] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer,* pages 46–57, 1989.

[145] S. Engelson. *Passive Map Learning and Visual Place Recognition*. PhD thesis, Department of Computer Science, Yale University, 1994.

[146] M. Erdmann and M. Mason. An exploration of sensorless manipulation. *IEEE Tr. on Rob. and Autom.,* 4(4):369–379, 1988.

[147] C. Fernandes, L. Gurvits, and Z. Li. Optimal nonholonomic motion planning for a falling cat. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning*. Kluwer Academic, 1993.

[148] C. Fernandes, L. Gurvits, and Z. Li. Near-optimal nonholonomic motion planning for a system of coupled rigid bodies. *IEEE Transactions on Automatic Control,* 30(3):450–463, Mar. 1994.

[149] R. Fitch, Z. Butler, and D. Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2003.

[150] S. Fleury, P. Souères, J.-P. Laumond, and R. Chatila. Primitives for smoothing paths of mobile robots. In *IEEE International Conference on Robotics and Automation,* volume 1, pages 832–839, 1993.

[151] S. Fleury, P. Souères, J.-P. Laumond, and R. Chatila. Primitives for smoothing mobile robot trajectories. *IEEE Transactions on Robotics and Automation,* 11:441–448, 1995.

[152] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. On differentially flat nonlinear systems. In *IFAC Symposium NOLCOS,* pages 408–412, 1992.

[153] M. Fliess, J. Lévine, P. Martin, and P. Rouchon. Flatness and defect of nonlinear systems: Introductory theory and examples. *International Journal of Control,* 61(6):1327–1361, 1995.

[154] A. T. Fomenko and T. L. Kunii. *Topological Modeling for Visualization*. Springer-Verlag, Tokyo, 1997.

[155] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2001.

[156] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proc. of the National Conference on Artificial Intelligence (AAAI),* 1999.

[157] D. Fox, W. Burgard, H. Kruppa, and S. Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous Robots,* 8(3), 2000.

[158] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research (JAIR),* 11:391–427, 1999.

[159] T. Fraichard and J.-M. Ahuactzin. Smooth path planning for cars. In *IEEE International Conference on Robotics and Automation,* pages 3722–3727, Seoul, Korea, 2001.

[160] E. Frazzoli, M. A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidance, Control, and Dynamics,* 25(1):116–129, 2002.

[161] C. Früh and A. Zakhor. 3D model generation for cities using aerial photographs and ground level laser scans. In *Proc. of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR),* 2001.

[162] R. Geraerts and M. Overmars. A comparative study of probabilistic roadmap planners. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V,* pages 43–58. Springer-Verlag, 2003.

[163] E. Gilbert, D. Johnson, and S. Keerthi. A fast procedure for computing distance between complex objects in three-dimensional space. *IEEE Transactions on Robotics and Automation,* 4:193–203, 1988.

[164] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, New York, 1981.

[165] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *IEEE International Conference on Robotics and Automation,* pages 1718–1723, 1990.

[166] K. Y. Goldberg. Orienting polygonal parts without sensors. *Algorithmica,* 10:201–225, 1993.

[167] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Procedings F,* 140(2):107–113, 1993.

[168] S. Gottschalk, M. C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. *Computer Graphics,* 30(Annual Conference Series):171–180, 1996.

[169] P. Grandjean and A. Robert de Saint Vincent. 3-D modeling of indoor scenes by fusion of noisy range and stereo data. In *IEEE International Conference on Robotics and Automation,* 1989.

[170] F. Gravoit, S. Cambon, and R. Alami. Asymov: a planner that deals with intricate symbolic and geometric problems. In *International Symposium on Robotics Research,* 2003. Book to appear.

[171] L. J. Guibas, C. Holleman, and L. E. Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 254–260, 1999.

[172] L. J. Guibas, J. C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *International Journal of Computational Geometry and Applications,* 9(4/5):471–512, August/October 1999.

[173] V. Guillemin and A. Pollack, editors. *Differential Topology*. Prentice-Hall, Inc., New Jersey, 1974.

[174] K. Gupta and Z. Guo. Motion planning with many degrees of freedom: sequential search with backtracking. *IEEE Transactions on Robotics and Automation,* 6(11):897–906, 1995.

[175] L. Gurvits. Averaging approach to nonholonomic motion planning. In *IEEE International Conference on Robotics and Automation,* pages 2541–2546, 1992.

[176] J. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2002.

[177] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 1998.

[178] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA),* 1999.

[179] J.-S. Gutmann and C. Schlegel. AMOS: Comparison of scan matching approaches for self-localization in indoor environments. In *Proc. of the 1st Euromicro Workshop on Advanced Mobile Robots*. IEEE Computer Society Press, 1996.

[180] J.-S. Gutmann, T. Weigel, and B. Nebel. A fast, accurate, and robust method for self-localization in polygonal environments using laser-range-finders. *Advanced Robotics Journal,* 14(8):651–668, 2001.

[181] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. Submitted for publication.

[182] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* 2002.

[183] D. Halperin and M. Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete Computational Geometry,* 16:121–134, 1996.

[184] L. Han and N. M. Amato. A kinematics-based probabilistic roadmap for closed chain systems. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics,* pages 233–246. AK Peters, 2001.

[185] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden. Time-optimal trajectories for a robot manipulator: A provably good approximation algorithm. In *IEEE International Conference on Robotics and Automation,* pages 150–156, 1989.

[186] G. Heinzinger and B. Paden. Bounds on robot dynamics. In *IEEE International Conference on Robotics and Automation,* pages 1227–1232, Scottsdale, Arizona, 1989.

[187] S. Hert, S. Tiwari, and V. Lumelsky. A Terrain-Covering Algorithm for an AUV. *Autonomous Robots,* 3:91–119, 1996.

[188] J. Hertzberg and F. Kirchner. Landmark-based autonomous navigation in sewerage pipes. In *Proc. of the First Euromicro Workshop on Advanced Mobile Robots,* 1996.

[189] H. Hirukawa, B. Mourrain, and Y. Papegay. A symbolic-numeric silhouette algorithm. In *Intelligent Robots and Systems,* pages 2358–2365, Nov 2000.

[190] C. Hofner and G. Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. *Robotics and Autonomous Systems,* 14:199–212, 1995.

[191] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *IEEE International Conference on Robotics and Automation,* pages 1408–1413, 2000.

[192] D. Hsu. *Randomized Single-Query Motion Planning In Expansive Spaces*. PhD thesis, Department of Computer Science, Stanford University, 2000.

[193] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation,* 2003.

[194] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In e. a. P. Agarwal, editor, *Robotics: The Algorithmic Perspective,* pages 141–154. A.K. Peters, Wellesley, MA, 1998.

[195] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research,* 21(3):233–255, 2002.

[196] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *IEEE International Conference on Robotics and Automation,* pages 2719–2726, 1997.

[197] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal of Computational Geometry and Applications,* 9(4/5):495–512, 1998.

[198] Y. Y. Huang, Z. L. Cao, and E. Hall. Region filling operations for mobile robot using computer graphics. In *Proceedings of the IEEE Conference on Robotics and Automation,* pages 1607–1614, 1986.

[199] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, and D. Manocha. V-COLLIDE: Accelerated collision detection for VRML. In R. Carey and P. Strauss, editors, *VRML 97: Second Symposium on the Virtual Reality Modeling Language,* pages 119–125, New York City, NY, 1997. ACM Press.

[200] S. Iannitti and K. M. Lynch. Exact minimum control switch motion planning for the snakeboard. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2003.

[201] M. Isard and A. Blake. Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision,* 29(1), 1998.

[202] A. Isidori. *Nonlinear Control Systems: An Introduction*. Springer-Verlag, 1985.

[203] P. Isto. A two-level search algorithm for motion planning. In *IEEE International Conference on Robotics and Automation,* pages 2025–2031, 1997.

[204] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 2323–2328, 2002.

[205] P. Jacobs, G. Heinzinger, J. Canny, and B. Paden. Planning guaranteed near-time-optimal trajectories for a manipulator in a cluttered workspace. In *International Workshop on Sensorial Integration for Industrial Robots: Architectures and Applications,* Zaragoza, Spain, 1989.

[206] P. Jacobs, G. Heinzinger, J. Canny, and B. Paden. Planning guaranteed near-time-optimal trajectories for a manipulator in a cluttered workspace. Technical Report RAMP 89-15, University of California, Berkeley, Engineering Systems Research Center, Sept. 1989.

[207] K. Janich. *Topology*. Spring-Verlag, New York, NY, 1984.

[208] R. Jarvis. Collision free trajectory planning using distance transforms. *Mech Eng Trans of the IE Aust,* ME10:197–191, 1985.

[209] P. Jensfelt and S. Kristensen. Active global localisation for a mobile robot using multiple hypothesis tracking. *IEEE Transactions on Robotics and Automation,* 17(5):748–760, Oct. 2001.

[210] X. Ji and J. Xiao. Planning motion compliant to complex contact states. *International Journal of Robotics Research,* 20(6):446–465, 2001.

[211] V. Jurdjevic. *Geometric Control Theory*. Cambridge University Press, 1997.

[212] V. Jurdjevic and H. J. Sussmann. Control systems on Lie groups. *Journal of Differential Equations,* 12:313–329, 1972.

[213] L. Kaelbling, A. Cassandra, and J. Kurien. Acting under uncertainty: Discrete Bayesian models for mobile-robot navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 1996.

[214] T. Kailath. *Linear Systems*. Prentice-Hall, 1980.

[215] R. Kalman. A new approach to linear filtering and prediction problems. *Trans. of the ASME, Journal of basic engineering,* 82:35–45, March 1960.

[216] I. Kamon, E. Rimon, and E. Rivlin. Tangentbug: A range-sensor based navigation algorithm. *Int. Journal of Robotics Research,* 17(9):934–953, 1998.

[217] I. Kamon, E. Rivlin, and E. Rimon. A new range-sensor based globally convergent navigation for mobile robots. In *IEEE Int'l. Conf. on Robotics and Automation,* Minneapolis, MN, April 1996.

[218] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proc. of the 11th Annual Conference on Uncertainty in AI (UAI),* 1995.

[219] K. Kant and S. Zucker. Toward efficient trajectory planning: Path velocity decomposition. *International Journal of Robotics Research,* 5:72–89, 1986.

[220] L. E. Kavraki. Part orientation with programmable vector fields: Two stable equilibria for most parts. In *IEEE International Conference on Robotics and Automation*, pages 20–25, Albuquerque, New Mexico, Apr. 1997.

[221] L. E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford University, 1995.

[222] L. E. Kavraki, M. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE International Conference on Robotics and Automation,* pages 3020–3026, 1996.

[223] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation,* 14(1):166–171, February 1998.

[224] L. E. Kavraki, F. Lamiraux, and C. Holleman. Towards planning for elastic objects. In P. Agrawal, L. E. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective,* pages 313–325. A.K. Peters, 1998.

[225] L. E. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. Technical Report STAN-CS-93-1490, Dept. Comput. Sci., Stanford Univ., Stanford, CA, 1993.

[226] L. E. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for path planning. In *IEEE International Conference on Robotics and Automation,* pages 2138–2139, 1994.

[227] L. E. Kavraki and J. C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Challenges,* pages 33–53. John Wiley, West Sussex, England, 1998.

[228] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot motion planning. In *Proc. ACM Symp. on Theory of Computing,* pages 353–362, 1995.

[229] L. E. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences,* 57(1):50–60, August 1998.

[230] L. E. Kavraki, J. C. Latombe, and R. Wilson. On the complexity of assembly partitioning. *Information Processing Letters,* 48:229–235, 1993.

[231] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation,* 12(4):566–580, June 1996.

[232] H. Keller. *Lectures on Numerical Methods in Bifurcation Problems*. Tata Institute of Fundamental Research, Bombay, India, 1987.

[233] S. D. Kelly and R. M. Murray. Geometric phases and robotic locomotion. *Journal of Robotic Systems,* 12(6):417–431, 1995.

[234] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research,* 5:90–98, 1986.

[235] R. Kindel, D. Hsu, J. C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *IEEE International Conference on Robotics and Automation,* pages 537–543, 2000.

[236] D. E. Kirk. *Optimal Control Theory*. Prentice-Hall Inc., 1970.

[237] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, O. E., and H. Matsubara. RoboCup: A challenge problem for AI. *AI Magazine,* 18(1):73–85, 1997.

[238] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of $k$-DOPs. *IEEE Transactions on Visualization and Computer Graphics,* 4(1):21–36, 1998.

[239] D. E. Koditschek and E. Rimon. Robot navigation functions on manifolds with boundary. *Advances in Applied Mathematics,* 11:412–442, 1990.

[240] S. Koenig and R. Simmons. A robot navigation architecture based on partially observable Markov decision process models. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, Cambridge, MA, 1998.

[241] Y. Koga, K. Kondo, J. Kuffner, and J. C. Latombe. Planning motions with intentions. *Computer Graphics (SIGGRAPH'94),* pages 395–408, 1994.

[242] Y. Koga and J. C. Latombe. Experiments in dual-arm manipulation planning. In *IEEE International Conference on Robotics and Automation,* pages 2238–2245, 1992.

[243] Y. Koga and J. C. Latombe. On multi-arm manipulation planning. In *IEEE International Conference on Robotics and Automation,* pages 945–952, 1994.

[244] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Transactions on Robotics and Automation,* 7:267–277, 1991.

[245] K. Konolige. Markov localization using correlation. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI),* 1999.

[246] J. J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation,* 2004.

[247] J. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *IEEE International Conference on Robotics and Automation,* pages 692–698, Seoul, Korea, May 2001.

[248] J. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots. In *International Symposium on Robotics Research,* 2003. Book to appear.

[249] J. J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation,* pages 995–1001, 2000.

[250] B. Kuipers and Y. Byan. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems,* 8:47–63, 1991.

[251] A. M. Ladd and L. E. Kavraki. Motion planning for knot untangling. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V,* pages 7–24. Springer-Verlag, 2002.

[252] A. M. Ladd and L. E. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation,* 20(2):229–242, 2004.

[253] G. Lafferriere and H. Sussmann. Motion planning for controllable systems without drift. In *IEEE International Conference on Robotics and Automation,* pages 1148–1153, Sacramento, CA, 1991.

[254] G. Lafferriere and H. J. Sussmann. A differential geometric approach to motion planning. In Z. Li and J. Canny, editors, *Nonholonomic Motion Planning*. Kluwer Academic, 1993.

[255] F. Lamiraux and L. E. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research,* 20(3):188–208, 2001.

[256] F. Lamiraux and L. E. Kavraki. Positioning of symmetric and nonsymmetric parts using radial and constant fields: Computation of al equilibrium configurations. *International Journal of Robotics Research,* 20(8):635–659, 2001.

[257] F. Lamiraux and J.-P. Laumond. On the expected complexity of random path planning. In *IEEE International Conference on Robotics and Automation,* pages 3014–3019, 1996.

[258] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation,* 17(4):498–502, Aug. 2001.

[259] F. Lamiraux, S. Sekhavat, and J.-P. Laumond. Motion planning and control for Hilare pulling a trailer. *IEEE Transactions on Robotics and Automation,* 15(4):640–652, Aug. 1999.

[260] S. Land and H. Choset. Coverage path planning for landmine location. In *Third International Symposium on Technology and the Mine Problem,* Monterey, CA, April 1998.

[261] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of North Carolina at Chapel Hill, North Carolina, 1999.

[262] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[263] J. C. Latombe. Personal communication.

[264] J.-P. Laumond and R. Alami. A geometrical approach to planning manipulation tasks: The case of a circular robot and a movable circular object amidst polygonal obstacles. Report 88314, LAAS/CNRS, Toulouse, France, 1989.

[265] J.-P. Laumond. Controllability of a multibody mobile robot. *IEEE Transactions on Robotics and Automation,* 9(6):755–763, Dec. 1993.

[266] J.-P. Laumond. *Robot motion planning and control*. Springer, 1998.

[267] J.-P. Laumond, P. E. Jacobs, M. Taïx, and R. M. Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation,* 10(5):577–593, Oct. 1994.

[268] S. M. LaValle, J. Yakey, and L. E. Kavraki. Randomized path planning for linkages with closed kinematics chains. *IEEE Transactions on Robotics and Automation,* 17(6):951–959, 2001.

[269] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V,* pages 59–76. Springer-Verlag, 2002.

[270] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *IEEE International Conference on Robotics and Automation,* pages 473–479, 1999.

[271] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research,* 20(5):378–400, May 2001.

[272] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics,* pages 293–308. AK Peters, 2001.

[273] S. M. Lavalle, D. Lin, L. J. Guibas, J. C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *IEEE International Conference on Robotics and Automation,* pages 1677–1682, 1997.

[274] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *Computer Graphics,* 24(4):327–335, 1990.

[275] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *IEEE International Conference on Robotics and Automation,* 2000.

[276] J. J. Leonard and H. Durrant-Whyte. *Directed Sonar Sensing for Mobile Robot Navigation*. Kluwer Academic, Boston, MA, 1992.

[277] J. J. Leonard and H. Feder. A computationally efficient method for large-scale concurrent mapping and localization. In J. Hollerbach and D. Koditschek, editors, *Proceedings of the Ninth International Symposium on Robotics Research,* Salt Lake City, Utah, 1999.

[278] J. J. Leonard and H. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ International Workshop on Intelligent Robots and Systems,* pages 1442–1447, May 1991.

[279] N. E. Leonard. Control synthesis and adaptation for an underactuated autonomous underwater vehicle. *IEEE Journal of Oceanic Engineering,* 20(3):211–220, July 1995.

[280] N. E. Leonard and P. S. Krishnaprasad. Motion control of drift-free, left-invariant systems on Lie groups. *IEEE Transactions on Automatic Control,* 40(9):1539–1554, Sept. 1995.

[281] P. Leven and S. Hutchinson. Real-time path planning in changing environments. *International Journal of Robotics Research,* 21(12):999–1030, Dec. 2002.

[282] P. Leven and S. Hutchinson. Using manipulability to bias sampling during the construction of probabilistic roadmaps. *IEEE Transactions on Robotics and Automation,* 19(6):1020–1026, Dec. 2003.

[283] A. D. Lewis. When is a mechanical control system kinematic? In *IEEE Conference on Decision and Control,* pages 1162–1167, Dec. 1999.

[284] A. D. Lewis. Simple mechanical control systems with constraints. *IEEE Transactions on Automatic Control,* 45(8):1420–1436, 2000.

[285] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Journal on Control and Optimization,* 35(3):766–790, May 1997.

[286] A. D. Lewis and R. M. Murray. Configuration controllability of simple mechanical control systems. *SIAM Review,* 41(3):555–574, 1999.

[287] F. L. Lewis and V. L. Syrmos. *Optimal Control*. John Wiley and Sons, Inc., 1995.

[288] Z. Li and J. Canny. *Nonholonomic Motion Planning*. Kluwer Academic, 1993.

[289] K. Lian, L. Wang, and L. Fu. Controllability of spacecraft systems in a central gravitational field. *IEEE Transactions on Automatic Control,* 39(12):2426–2440, Dec. 1994.

[290] M. C. Lin, D. Manocha, J. Cohen, and S. Gottschalk. Collision detection: Algorithms and applications. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation,* pages 129–142. A K Peters, Wellesley, MA, 1997.

[291] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *IEEE International Conference on Robotics and Automation,* pages 2920–2927, 2003.

[292] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In *International Symposium on Robotics Research,* 2003. Book to appear.

[293] G. Liu and Z. Li. A unified geometric approach to modeling and control of constrained mechanical systems. *IEEE Transactions on Robotics and Automation,* 18(4):574–587, Aug. 2002.

[294] Y. Liu and S. Arimoto. Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *International Journal of Robotics Research,* 11(4):376–382, 1992.

[295] C. Lobry. Controllability of nonlinear systems on compact manifolds. *SIAM Journal on Control,* 12(1):1–4, 1974.

[296] I. Lotan, F. Schwarzer, D. Halperin, and J. C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proceedings of the 18th annual Symposium on Computational geometry,* pages 43–52. ACM Press, 2002.

[297] T. Lozano-Pérez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal of Robotics and Automation,* RA-3(3):224–238, 1987.

[298] T. Lozano-Perez and M. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM,* 22(10):560–570, 1979.

[299] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots,* 4:333–349, 1997.

[300] V. Lumelsky, S. Mukhopadhyay, and K. Sun. Dynamic path planning in sensor-based terrain acquisition. *IEEE Transactions on Robotics and Automation,* 6(4):462–472, August 1990.

[301] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica,* 2:403–430, 1987.

[302] J. E. Luntz, W. Messner, and H. Choset. Distributed manipulation using discrete actuator arrays. *International Journal of Robotics Research,* 20(7):553–582, 2001.

[303] K. M. Lynch. Controllability of a planar body with unilateral thrusters. *IEEE Transactions on Automatic Control,* 44(6):1206–1211, June 1999.

[304] K. M. Lynch and C. K. Black. Recurrence, controllability, and stabilization of juggling. *IEEE Transactions on Robotics and Automation,* 17(2):113–124, Apr. 2001.

[305] K. M. Lynch, N. Shiroma, H. Arai, and K. Tanie. Collision-free trajectory planning for a 3-DOF robot with a passive joint. *International Journal of Robotics Research,* 19(12):1171–1184, Dec. 2000.

[306] D. K. M. Ben-Or and J. Reif. The complexity of elementary algebra and geometry. *Journal of Computational Sciences,* 32:251–264, 1986.

[307] J. Marsden. *Elementary Classical Analysis*. W. H. Freeman and Company, New York, 1974.

[308] J. Marsden and T. Ratiu. *Introduction to Mechanics and Symmetry*. Springer-Verlag, New York, 1994.

[309] P. Martin, R. M. Murray, and P. Rouchon. Flat systems. In G. Bastin and M. Gevers, editors, *1997 European Control Conference Plenary Lectures and Mini-Courses*. 1997.

[310] S. Martinez, J. Cortés, and F. Bullo. A catalog of inverse-kinematics planners for underactuated systems on matrix Lie groups. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2003.

[311] M. T. Mason. *Manipulation by grasping and pushing operations*. PhD thesis, MIT, Artificial Intelligence Laboratory, 1982.

[312] M. T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[313] P. Maybeck. The Kalman filter: An introduction to concepts. In *Autonomous Robot Vehicles*. Springer verlag, 1990.

[314] M. B. Milam, K. Mushambi, and R. M. Murray. A new computational approach to real-time trajectory generation for constrained mechanical systems. In *IEEE International Conference on Decision and Control,* 2000.

[315] J. Milnor. *Morse Theory*. Princeton University Press, Princeton, NJ, 1963.

[316] B. Mirtich. V-clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics,* 17(3):177–208, 1998.

[317] M. Moll, K. Goldberg, M. A. Erdmann, and R. Fearing. Aligning parts for micro assemblies. *Assembly Automation*, 22(1):46–54, Feb. 2002.

[318] M. Moll and L. E. Kavraki. Path planning for minimal energy curves of constant length. In *IEEE International Conference on Robotics and Automation,* pages 2826–2831, 2004.

[319] M. Moll and L. E. Kavraki. Path planning for variable resolution minimal energy curves of constant length. In *IEEE International Conference on Robotics and Automation,* 2005.

[320] M. Montemerlo and S. Thrun. Simultaneous localization and mapping problem with unknown data association using FastSLAM. In *IEEE International Conference on Robotics and Automation,* 2003.

[321] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the National Conference on Artificial Intelligence (AAAI),* 2002.

[322] M. Montemerlo, S. Thrun, and W. Whittaker. Conditional particle filters for simultaneous mobile robot localization and people tracking. In *IEEE International Conference on Robotics and Automation,* 2002.

[323] M. Morales, S. Rodriguez, and N. M. Amato. Improving the connectivity of prm roadmaps. In *IEEE International Conference on Robotics and Automation,* pages 4427–4432, 2003.

[324] H. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine,* pages 61–74, Summer 1988.

[325] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation,* 1985.

[326] J. J. Moré and S. J. Wright. *Optimization Software Guide*. SIAM, Philadelphia, PA, 1993.

[327] K. A. Morgansen. *Temporal patterns in learning and control*. PhD thesis, Harvard University, 1999.

[328] K. A. Morgansen, P. A. Vela, and J. W. Burdick. Trajectory stabilization for a planar carangiform robot fish. In *IEEE International Conference on Robotics and Automation,* 2002.

[329] K. Murphy. Bayesian map learning in dynamic environments. In *Neural Info. Proc. Systems (NIPS),* 1999.

[330] R. M. Murray, Z. Li, and S. S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[331] R. M. Murray, M. Rathinam, and W. Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. In *ASME Int Mech Eng Congress and Expo,* 1995.

[332] R. M. Murray and S. S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control,* 38(5):700–716, 1993.

[333] Y. Nakamura, T. Suzuki, and M. Koinuma. Nonlinear behavior and control of a nonholonomic free-joint manipulator. *IEEE Transactions on Robotics and Automation,* 13(6):853–862, 1997.

[334] P. Newman, J. Leonard, J. Neira, and J. Tardós. Explore and return: Experimental validation of real time concurrent mapping and localization. In *IEEE International Conference on Robotics and Automation,* 2002.

[335] C. Nielsen and L. E. Kavraki. A two level fuzzy PRM for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 1716–1722, Japan, 2000.

[336] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation,* pages 446–452, 2004.

[337] C. Nissoux, T. Simeon, and J.-P. Laumond. Visibility based probabilistic roadmaps. *Advanced Robotics Journal,* 14(6), 2000.

[338] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Verlag, 1999.

[339] I. Nourbakhsh, R. Powers, and S. Birchfield. DERVISH an office-navigating robot. *AI Magazine,* 16(2), 1995.

[340] C. Ó'Dúnlaing and C. Yap. A "retraction" method for planning the motion of a disc. *Algorithmica,* 6:104–111, 1985.

[341] M. Ollis and A. Stentz. First results in vision-based crop line tracking. In *IEEE International Conference on Robotics and Automation,* 1996.

[342] J. P. Ostrowski and J. W. Burdick. The geometric mechanics of undulatory robotic locomotion. *International Journal of Robotics Research,* 17(7):683–701, July 1998.

[343] J. P. Ostrowski, J. P. Desai, and V. Kumar. Optimal gait selection for nonholonomic locomotion systems. *International Journal of Robotics Research,* 19(3):225–237, Mar. 2000.

[344] M. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, Oct. 1992.

[345] M. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K. Goldberg, D. Halperin, J. C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics (WAFR),* pages 19–37. A. K. Peters, Ltd, 1995.

[346] R. Parr and A. Eliazar. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI),* 2003.

[347] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., 1988.

[348] F. Pfeiffer and R. Johanni. A concept for manipulator trajectory planning. *IEEE Journal of Robotics and Automation,* RA-3(2):115–123, 1987.

[349] J. M. Phillips, N. Bedrossian, and L. E. Kavraki. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *IEEE International Conference on Robotics and Automation,* pages 3968–3973, 2004.

[350] J. Phillips, L. Kavraki, and N. Bedrossian. Spacecraft rendezvous and docking with real-time, randomized optimization. In *AIAA Guidance, Navigation, and Control,* 2003.

[351] A. Piazza, M. Romano, and C. G. L. Bianco. $G^3$-splines for the path planning of wheeled mobile robots. In *European Control Conference,* 2003.

[352] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelarated Voronoi sampling. In B. R. Donald, K. Lynch, and D. Rus, editors, *New Directions in Algorithmic and Computational Robotics*. AK Peters, 2001.

[353] E. Plaku and L. E. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *IEEE International Conference on Robotics and Automation,* 2005.

[354] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The Mathematical Theory of Optimal Processes*. Interscience Publishers, 1962.

[355] C. Pradalier, J. Hermosillo, C. Koike, C. Braillon, P. P. Bessière, and C. Laugier. Safe and autonomous navigation for a car-like robot among pedestrian. In *IARP Int. Workshop on Service, Assistive and Personal Robots,* 2003.

[356] F. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985. p198–257.

[357] S. Quinlan. Efficient distance computation between nonconvex objects. In *IEEE International Conference on Robotics and Automation,* pages 3324–3329, 1994.

[358] A. Rao and K. Goldberg. Manipulating algebraic parts in the plane. *IEEE Tr. on Rob. and Autom.*, 11:598–602, 1995.

[359] N. Rao, N. Stolzfus, and S. Iyengar. A retraction method for learned navigation in unknown terrains for a circular robot. *IEEE Transactions on Robotics and Automation,* 7:699–707, October 1991.

[360] J. A. Reeds and L. A. Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics,* 145(2):367–393, 1990.

[361] J. Reif. Complexity of the mover's problem and generalizations. In *Proc. 20th IEEE Symposium on Foundations of Computer Science,* pages 421–427, 1979.

[362] J. H. Reif and H. Wang. Nonuniform discretization for kinodynamic motion planning and its applications. *SIAM Journal of Computing,* 30(1):161–190, 2000.

[363] D. Reznik, E. Moshkivich, and J. F. Canny. Building a universal planar manipulator. In K.-F. Böhringer and H. Choset, editors, *Distributed Manipulation,* pages 147–171. Kluwer Academic Publishers, Boston, 2000.

[364] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation,* 8(5):501–518, October 1992.

[365] T. Röfer. Using histogram correlation to create consistent laser scan maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2002.

[366] H. Rohnert. Shortest path in the plane with convex polygonal obstacles. *Information Processing Letters,* 23:71–76, 1986.

[367] G. Sánchez and J. C. Latombe. On delaying collision checking in prm planning: Application to multi-robot coor dination. *International Journal of Robotics Research,* 21(1):5–26, 2002.

[368] S. S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, New York, 1999.

[369] D. H. Sattinger and O. L. Weaver. *Lie Groups and Algebras with Applications to Physics, Geometry, and Mechanics*. Springer-Verlag, 1986.

[370] A. Scheuer and T. Fraichard. Collision-free and continuous-curvature path planning for car-like robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 1304–1311, Osaka, Japan, 1997.

[371] B. Schiele and J. Crowley. A comparison of position estimation techniques using occupancy grids. In *IEEE International Conference on Robotics and Automation,* 1994.

[372] B. Schutz. *Geometrical methods of mathematical physics*. Cambridge University Press, 1980.

[373] J. T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics,* 4:298–351, 1983.

[374] J. T. Schwartz and M. Sharir. On the piano movers' problem: V. The case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Communications on Pure and Applied Mathematics,* 37:815–848, 1984.

[375] J. T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence.,* 37:157–169, 1988.

[376] F. Schwarzer, M. Saha, and J. C. Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V,* pages 25–42. Springer-Verlag, 2002.

[377] S. Sekhavat and J.-P. Laumond. Topological property of trajectories computed from sinusoidal inputs for nonholonomic chained form systems. In *IEEE International Conference on Robotics and Automation,* pages 3383–3388, 1996.

[378] S. Sekhavat and J.-P. Laumond. Topological property for collision-free nonholonomic motion planning: the case of sinusoidal inputs for chained form systems. *IEEE Transactions on Robotics and Automation,* 14(5):671–680, Oct. 1998.

[379] S. Sekhavat, P. Švestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *International Journal of Robotics Research,* 17(8):840–857, Aug. 1998.

[380] J.-P. Serre. *Lie Algebras and Lie Groups*. W. A. Benjamin, New York, 1965.

[381] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, 1999.

[382] H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *Proceedings of IJCAI-97*. IJCAI, Inc., 1997. 1997.

[383] Z. Shiller and S. Dubowsky. On computing the global time-optimal motions of robotic manipulators in the presence of obstacles. *IEEE Transactions on Robotics and Automation,* 7(6): 785–797, Dec. 1991.

[384] Z. Shiller and H.-H. Lu. Computation of path constrained time optimal motions with dynamic singularities. *ASME Journal of Dynamic Systems, Measurement, and Control,* 114:34–40, Mar. 1992.

[385] K. G. Shin and N. D. McKay. Minimum-time control of robotic manipulators with geometric path constraints. *IEEE Transactions on Automatic Control,* 30(6):531–541, June 1985.

[386] R. Simmons and S. Koenig. Probabilistic robot navigation in partially observable environments. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI),* 1995.

[387] A. Singh, J. C. Latombe, and D. Brutlag. A motion planning approach to flexible ligand binding. In *Intelligent Systems for Molecular Biology,* pages 252–261, 1999.

[388] J.-J. E. Slotine and H. S. Yang. Improving the efficiency of time-optimal path-following algorithms. *IEEE Transactions on Robotics and Automation,* 5(1):118–124, Feb. 1989.

[389] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The International Journal of Robotics Research,* 5(4):56–68, 1986.

[390] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*. Springer Verlag, 1990.

[391] E. Sontag. Gradient techniques for systems with no drift: A classical idea revisited. In *IEEE International Conference on Decision and Control,* pages 2706–2711, 1993.

[392] E. D. Sontag. Control of systems without drift via generic loops. *IEEE Transactions on Automatic Control,* 40(7):1210–1219, July 1995.

[393] O. J. Sørdalen. Conversion of a car with *n* trailers into a chained form. In *IEEE International Conference on Robotics and Automation,* pages 1382–1387, 1993.

[394] P. Souères and J.-D. Boissonnat. Optimal trajectories for nonholonomic mobile robots. In J.-P. Laumond, editor, *Robot Motion Planning and Control*. Springer, 1998.

[395] P. Souères and J.-P. Laumond. Shortest paths synthesis for a car-like robot. *IEEE Transactions on Automatic Control,* 41(5):672–688, May 1996.

[396] R. F. Stengel. *Optimal control and estimation*. Dover, New York, 1994.

[397] A. Stentz. Optimal and efficient path planning for unknown and dynamic environments. *International Journal of Robotics and Automation,* 10, 1995.

[398] G. Strang. *Linear Algebra and Its Applications*. Orlando: Academic Press, 1980.

[399] A. Sudsang and L. Kavraki. A geometric approach to designing a programmable force field with a unique stable equilibrium for parts in the plane. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1079–1085, Seoul, 2001.

[400] H. Sussmann. A continuation method for nonholonomic path-finding problems. In *IEEE International Conference on Decision and Control,* pages 2718–2723, 1993.

[401] H. J. Sussmann. A general theorem on local controllability. *SIAM Journal on Control and Optimization,* 25(1):158–194, Jan. 1987.

[402] H. J. Sussmann and W. Tang. Shortest paths for the Reeds-Shepp car: a worked out example of the use of geometric techniques in nonlinear optimal control. Technical Report SYCON-91-10, Rutgers University, 1991.

[403] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal of Computing,* 21(5):863–888, October 1992.

[404] P. Švestka. A probabilistic approach to motion planning for car-like robots. Technical Report RUU-CS-93-18, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, 1993.

[405] P. Švestka and M. H. Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation,* pages 1631–1636, 1995.

[406] P. Švestka and J. Vleugels. Exact motion planning for tractor-trailer robots. In *IEEE International Conference on Robotics and Automation,* pages 2445–2450, 1995.

[407] K. R. Symon. *Mechanics*. Addison-Wesley, 1971.

[408] X. Tang, B. Kirkpatrick, S. Thomas, G. Song, and N. M. Amato. Using motion planning to study rna folding kinetics. In *International Conference on Research in Computational Molecular Biology,* 2004.

[409] M. Teodoro, G. N. Phillips, and L. E. Kavraki. Molecular docking: A problem with thousands of degrees of freedom. In *IEEE International Conference on Robotics and Automation,* pages 960–966, 2001.

[410] J. Thorpe. *Elementary Topics in Differential Geometry*. Springer-Verlag, 1985.

[411] S. Thrun. Exploration and model building in mobile robot domains. In *Proc. of the IEEE International Conference on Neural Networks,* 1993.

[412] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *International Journal of Robotics Research,* 20(5):335–363, 2001.

[413] S. Thrun. Learning occupancy grids with forward sensor models. *Autonomous Robots,* 2002.

[414] S. Thrun, M. Bennewitz, W. Burgard, A. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A second generation mobile tour-guide robot. In *IEEE International Conference on Robotics and Automation,* 1999.

[415] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, to appear.

[416] S. Thrun, W. Burgard, and D. Fox. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *IEEE International Conference on Robotics and Automation,* 2000.

[417] S. Thrun, D. Fox, and W. Burgard. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning and Autonomous Robots (joint issue),* 31(1–3):29–53, 1998.

[418] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. In *Proc. of the National Conference on Artificial Intelligence (AAAI),* 1998.

[419] D. Tilbury, R. Murray, and S. Sastry. Trajectory generation for the n-trailer problem using Goursat normal form. In *IEEE International Conference on Decision and Control,* 1993.

[420] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools: JGT,* 2(4):1–14, 1997.

[421] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools: JGT,* 4(2):7–25, 1999.

[422] P. Vela and J. W. Burdick. Control of biomimetic locomotion via averaging theory. In *IEEE International Conference on Robotics and Automation,* 2003.

[423] P. A. Vela, K. A. Morgansen, and J. W. Burdick. Underwater locomotion from oscillatory shape deformations. In *IEEE International Conference on Decision and Control,* 2002.

[424] G. Weiß, C. Wetzler, and E. von Puttkamer. Keeping track of position and orientation of moving indoor systems by correlation of range-finder scans. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,* pages 595–601, 1994.

[425] J. T. Wen. Control of nonholonomic systems. In W. S. Levine, editor, *The Control Handbook,* pages 1359–1368. CRC Press, 1996.

[426] J. Wiegley, K. Goldberg, M. Peshkin, and M. Brokowski. A complete algorithm for designing passive fences to orient parts. In *Proc. Int. Conf. on Rob. and Autom.,* pages 1133–1139, 1996.

[427] S. Wilmarth, N. M. Amato, and P. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of th e free space. In *IEEE International Conference on Robotics and Automation,* pages 1024–1031, 1999.

[428] R. Wilson and J. C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence,* 71:371–396, 1995.

[429] R. H. Wilson, L. E. Kavraki, J. C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *International Journal of Robotics Research,* 14:335–350, 1995.

[430] B. Yamauchi and P. Langley. Place recognition in dynamic environments. *Journal of Robotic Systems,* 14(2):107–120, 1997.

[431] M. Yim and A. Berlin. Two approaches to distributed manipulation. In K.-F. Böhringer and H. Choset, editors, *Distributed Manipulation,* pages 237–261. Kluwer Academic Publishers, Boston, 2000.

[432] T. Yoshikawa. Manipulability of robotic mechanisms. *International Journal of Robotics Research,* 4(2):3–9, Apr. 1985.

[433] M. Zhang and L. E. Kavraki. A new method for fast and accurate derivation of molecular conformations. *Journal of Chemical Information and Computer Sciences,* 42(1):64–70, 2002.

# *Index*