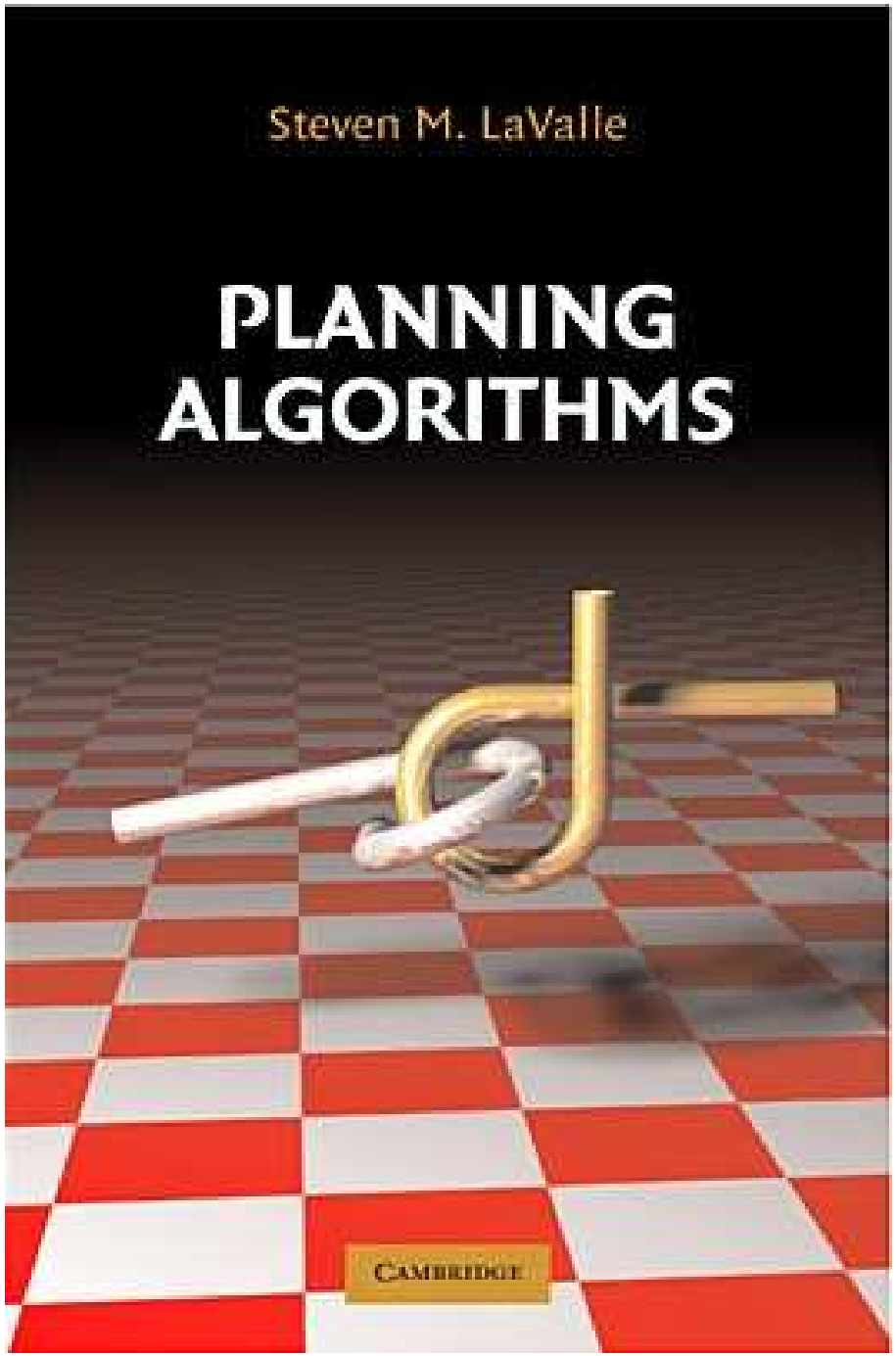


Part II

Motion Planning

Steven M. LaValle

University of Illinois

Available for downloading at **http://planning.cs.uiuc.edu/**

Published by Cambridge University Press

Overview of Part II: Motion Planning

Planning in Continuous Spaces

Part II makes the transition from discrete to continuous state spaces. Two alternative titles are appropriate for this part: 1) *motion planning*, or 2) *planning in continuous state spaces*. Chapters 3–8 are based on research from the field of motion planning, which has been building since the 1970s; therefore, the name *motion planning* is widely known to refer to the collection of models and algorithms that will be covered. On the other hand, it is convenient to also think of Part II as *planning in continuous spaces* because this is the primary distinction with respect to most other forms of planning.

In addition, motion planning will frequently refer to motions of a *robot* in a 2D or 3D *world* that contains *obstacles*. The robot could model an actual robot, or any other collection of moving bodies, such as humans or flexible molecules. A *motion plan* involves determining what motions are appropriate for the robot so that it reaches a goal state without colliding into obstacles. Recall the examples from Section 1.2.

Many issues that arose in Chapter 2 appear once again in motion planning. Two themes that may help to see the connection are as follows.

1. Implicit representations

A familiar theme from Chapter 2 is that planning algorithms must deal with *implicit* representations of the state space. In motion planning, this will become even more important because the state space is uncountably infinite. Furthermore, a complicated transformation exists between the world in which the models are defined and the space in which the planning occurs. Chapter 3 covers ways to model motion planning problems, which includes defining 2D and 3D geometric models and transforming them. Chapter 4 introduces the state space that arises for these problems. Following motion planning literature [344, 304], we will refer to this state space as the *configuration space*. The dimension of the configuration space corresponds to the number of degrees of freedom of the robot. Using the configuration space, motion planning will be viewed as a kind of search in a high-dimensional configuration space that contains implicitly represented obstacles. One additional complication is that configuration spaces have unusual topological structure that must be correctly characterized to ensure correct operation of planning algorithms. A motion plan will then be defined as a continuous path in the configuration space.

2. Continuous $\rightarrow$ discrete

A central theme throughout motion planning is to transform the continuous model into a discrete one. Due to this transformation, many algorithms from Chapter

2 are embedded in motion planning algorithms. There are two alternatives to achieving this transformation, which are covered in Chapters 5 and 6, respectively. Chapter 6 covers *combinatorial motion planning*, which means that from the input model the algorithms build a discrete representation that *exactly* represents the original problem. This leads to *complete* planning approaches, which are guaranteed to find a solution when it exists, or correctly report failure if one does not exist. Chapter 5 covers *sampling-based motion planning*, which refers to algorithms that use collision detection methods to sample the configuration space and conduct discrete searches that utilize these samples. In this case, completeness is sacrificed, but it is often replaced with a weaker notion, such as *resolution completeness* or *probabilistic completeness*. It is important to study both Chapters 5 and 6 because each methodology has its strengths and weaknesses. Combinatorial methods can solve virtually any motion planning problem, and in some restricted cases, very elegant solutions may be efficiently constructed in practice. However, for the majority of "industrial-grade" motion planning problems, the running times and implementation difficulties of these algorithms make them unappealing. Sampling-based algorithms have fulfilled much of this need in recent years by solving challenging problems in several settings, such as automobile assembly, humanoid robot planning, and conformational analysis in drug design. Although the completeness guarantees are weaker, the efficiency and ease of implementation of these methods have bolstered interest in applying motion planning algorithms to a wide variety of applications.

Two additional chapters appear in Part II. Chapter 7 covers several extensions of the basic motion planning problem from the earlier chapters. These extensions include avoiding moving obstacles, multiple robot coordination, manipulation planning, and planning with closed kinematic chains. Algorithms that solve these problems build on the principles of earlier chapters, but each extension involves new challenges.

Chapter 8 is a transitional chapter that involves many elements of motion planning but is additionally concerned with gracefully recovering from unexpected deviations during execution. Although uncertainty in predicting the future is not explicitly modeled until Part III, Chapter 8 redefines the notion of a plan to be a function over state space, as opposed to being a path through it. The function gives the appropriate actions to take during exection, regardless of what configuration is entered. This allows the true configuration to drift away from the commanded configuration. In Part III such uncertainties will be explicitly modeled, but this comes at greater modeling and computational costs. It is worthwhile to develop effective ways to avoid this.

Chapter 3

Geometric Representations and Transformations

This chapter provides important background material that will be needed for Part II. Formulating and solving motion planning problems require defining and manipulating complicated geometric models of a system of bodies in space. Section 3.1 introduces geometric modeling, which focuses mainly on semi-algebraic modeling because it is an important part of Chapter 6. If your interest is mainly in Chapter 5, then understanding semi-algebraic models is not critical. Sections 3.2 and 3.3 describe how to transform a single body and a chain of bodies, respectively. This will enable the robot to "move." These sections are essential for understanding all of Part II and many sections beyond. It is expected that many readers will already have some or all of this background (especially Section 3.2, but it is included for completeness). Section 3.4 extends the framework for transforming chains of bodies to transforming trees of bodies, which allows modeling of complicated systems, such as humanoid robots and flexible organic molecules. Finally, Section 3.5 briefly covers transformations that do not assume each body is rigid.

3.1 Geometric Modeling

A wide variety of approaches and techniques for geometric modeling exist, and the particular choice usually depends on the application and the difficulty of the problem. In most cases, there are generally two alternatives: 1) a *boundary representation*, and 2) a *solid representation*. Suppose we would like to define a model of a planet. Using a boundary representation, we might write the equation of a sphere that roughly coincides with the planet's surface. Using a solid representation, we would describe the set of all points that are contained in the sphere. Both alternatives will be considered in this section.

The first step is to define the *world* $\mathcal{W}$ for which there are two possible choices: 1) a 2D world, in which $\mathcal{W} = \mathbb{R}^2$, and 2) a 3D world, in which $\mathcal{W} = \mathbb{R}^3$. These

choices should be sufficient for most problems; however, one might also want to allow more complicated worlds, such as the surface of a sphere or even a higher dimensional space. Such generalities are avoided in this book because their current applications are limited. Unless otherwise stated, the world generally contains two kinds of entities:

1. **Obstacles:** Portions of the world that are "permanently" occupied, for example, as in the walls of a building.

2. **Robots:** Bodies that are modeled geometrically and are controllable via a motion plan.

Based on the terminology, one obvious application is to model a robot that moves around in a building; however, many other possibilities exist. For example, the robot could be a flexible molecule, and the obstacles could be a folded protein. As another example, the robot could be a virtual human in a graphical simulation that involves obstacles (imagine the family of Doom-like video games).

This section presents a method for systematically constructing representations of obstacles and robots using a collection of primitives. Both obstacles and robots will be considered as (closed) subsets of $\mathcal{W}$. Let the *obstacle region* $\mathcal{O}$ denote the set of all points in $\mathcal{W}$ that lie in one or more obstacles; hence, $\mathcal{O} \subseteq \mathcal{W}$. The next step is to define a systematic way of representing $\mathcal{O}$ that has great expressive power while being computationally efficient. Robots will be defined in a similar way; however, this will be deferred until Section 3.2, where transformations of geometric bodies are defined.

3.1.1 Polygonal and Polyhedral Models

In this and the next subsection, a solid representation of $\mathcal{O}$ will be developed in terms of a combination of *primitives*. Each primitive H_i represents a subset of $\mathcal{W}$ that is easy to represent and manipulate in a computer. A complicated obstacle region will be represented by taking finite, Boolean combinations of primitives. Using set theory, this implies that $\mathcal{O}$ can also be defined in terms of a finite number of unions, intersections, and set differences of primitives.

Convex polygons First consider $\mathcal{O}$ for the case in which the obstacle region is a convex, polygonal subset of a 2D world, $\mathcal{W} = \mathbb{R}^2$. A subset $X \subset \mathbb{R}^n$ is called *convex* if and only if, for any pair of points in X, all points along the line segment that connects them are contained in X. More precisely, this means that for any $x_1, x_2 \in X$ and $\lambda \in [0, 1]$,

$$\lambda x_1 + (1 - \lambda) x_2 \in X. \tag{3.1}$$

Thus, interpolation between x_1 and x_2 always yields points in X. Intuitively, X contains no pockets or indentations. A set that is not convex is called *nonconvex* (as opposed to *concave*, which seems better suited for lenses).

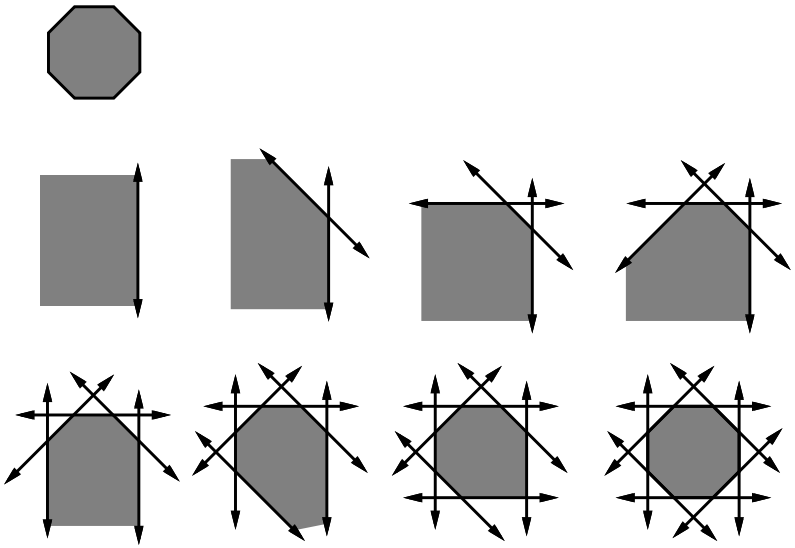


Figure 3.1: A convex polygonal region can be identified by the intersection of half-planes.

A boundary representation of $\mathcal{O}$ is an m-sided polygon, which can be described using two kinds of *features*: vertices and edges. Every *vertex* corresponds to a "corner" of the polygon, and every *edge* corresponds to a line segment between a pair of vertices. The polygon can be specified by a sequence, (x_1, y_1), (x_2, y_2), ..., (x_m, y_m), of m points in $\mathbb{R}^2$, given in counterclockwise order.

A solid representation of $\mathcal{O}$ can be expressed as the intersection of m half-planes. Each half-plane corresponds to the set of all points that lie to one side of a line that is common to a polygon edge. Figure 3.1 shows an example of an octagon that is represented as the intersection of eight half-planes.

An edge of the polygon is specified by two points, such as (x_1, y_1) and (x_2, y_2). Consider the equation of a line that passes through (x_1, y_1) and (x_2, y_2). An equation can be determined of the form $ax + by + c = 0$, in which $a, b, c \in \mathbb{R}$ are constants that are determined from x_1, y_1, x_2, and y_2. Let $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ be the function given by $f(x, y) = ax + by + c$. Note that $f(x, y) < 0$ on one side of the line, and $f(x, y) > 0$ on the other. (In fact, f may be interpreted as a signed Euclidean distance from (x, y) to the line.) The sign of $f(x, y)$ indicates a half-plane that is bounded by the line, as depicted in Figure 3.2. Without loss of generality, assume that $f(x, y)$ is defined so that $f(x, y) < 0$ for all points to the left of the edge from (x_1, y_1) to (x_2, y_2) (if it is not, then multiply $f(x, y)$ by -1).

Let $f_i(x, y)$ denote the f function derived from the line that corresponds to the edge from (x_i, y_i) to (x_{i+1}, y_{i+1}) for $1 \leq i < m$. Let $f_m(x, y)$ denote the line equation that corresponds to the edge from (x_m, y_m) to (x_1, y_1). Let a *half-plane*

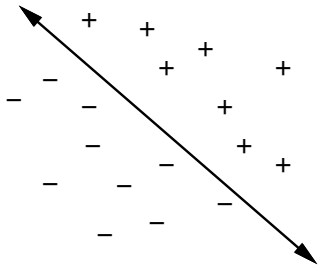


Figure 3.2: The sign of the $f(x, y)$ partitions $\mathbb{R}^2$ into three regions: two half-planes given by $f(x, y) < 0$ and $f(x, y) > 0$, and the line $f(x, y) = 0$.

H_i for $1 \leq i \leq m$ be defined as a subset of $\mathcal{W}$:

$$H_i = \{(x, y) \in \mathcal{W} \mid f_i(x, y) \leq 0\}. \tag{3.2}$$

Above, H_i is a primitive that describes the set of all points on one side of the line $f_i(x, y) = 0$ (including the points on the line). A convex, m-sided, polygonal obstacle region $\mathcal{O}$ is expressed as

$$\mathcal{O} = H_1 \cap H_2 \cap \cdots \cap H_m. \tag{3.3}$$

Nonconvex polygons The assumption that $\mathcal{O}$ is convex is too limited for most applications. Now suppose that $\mathcal{O}$ is a nonconvex, polygonal subset of $\mathcal{W}$. In this case $\mathcal{O}$ can be expressed as

$$\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2 \cup \cdots \cup \mathcal{O}_n, \tag{3.4}$$

in which each $\mathcal{O}_i$ is a convex, polygonal set that is expressed in terms of half-planes using (3.3). Note that $\mathcal{O}_i$ and $\mathcal{O}_j$ for $i \neq j$ need not be disjoint. Using this representation, very complicated obstacle regions in $\mathcal{W}$ can be defined. Although these regions may contain multiple components and holes, if $\mathcal{O}$ is bounded (i.e., $\mathcal{O}$ will fit inside of a big enough rectangular box), its boundary will consist of linear segments.

In general, more complicated representations of $\mathcal{O}$ can be defined in terms of any finite combination of unions, intersections, and set differences of primitives; however, it is always possible to simplify the representation into the form given by (3.3) and (3.4). A set difference can be avoided by redefining the primitive. Suppose the model requires removing a set defined by a primitive H_i that contains[1] $f_i(x, y) < 0$. This is equivalent to keeping all points such that $f_i(x, y) \geq 0$, which is equivalent to $-f_i(x, y) \leq 0$. This can be used to define a new primitive H_i', which

[1] In this section, we want the resulting set to include all of the points along the boundary. Therefore, $<$ is used to model a set for removal, as opposed to $\leq$.

when taken in union with other sets, is equivalent to the removal of H_i. Given a complicated combination of primitives, once set differences are removed, the expression can be simplified into a finite union of finite intersections by applying Boolean algebra laws.

Note that the representation of a nonconvex polygon is not unique. There are many ways to decompose $\mathcal{O}$ into convex components. The decomposition should be carefully selected to optimize computational performance in whatever algorithms that model will be used. In most cases, the components may even be allowed to overlap. Ideally, it seems that it would be nice to represent $\mathcal{O}$ with the minimum number of primitives, but automating such a decomposition may lead to an NP-hard problem (see Section 6.5.1 for a brief overview of NP-hardness). One efficient, practical way to decompose $\mathcal{O}$ is to apply the vertical cell decomposition algorithm, which will be presented in Section 6.2.2

Defining a logical predicate What is the value of the previous representation? As a simple example, we can define a logical predicate that serves as a collision detector. Recall from Section 2.4.1 that a predicate is a Boolean-valued function. Let ϕ be a predicate defined as $\phi : \mathcal{W} \rightarrow \{\text{TRUE}, \text{FALSE}\}$, which returns TRUE for a point in $\mathcal{W}$ that lies in $\mathcal{O}$, and FALSE otherwise. For a line given by $f(x, y) = 0$, let $e(x, y)$ denote a logical predicate that returns TRUE if $f(x, y) \leq 0$, and FALSE otherwise.

A predicate that corresponds to a convex polygonal region is represented by a logical conjunction,

$$\alpha(x, y) = e_1(x, y) \wedge e_2(x, y) \wedge \cdots \wedge e_m(x, y). \tag{3.5}$$

The predicate $\alpha(x, y)$ returns TRUE if the point (x, y) lies in the convex polygonal region, and FALSE otherwise. An obstacle region that consists of n convex polygons is represented by a logical disjunction of conjuncts,

$$\phi(x, y) = \alpha_1(x, y) \vee \alpha_2(x, y) \vee \cdots \vee \alpha_n(x, y). \tag{3.6}$$

Although more efficient methods exist, ϕ can check whether a point (x, y) lies in $\mathcal{O}$ in time $O(n)$, in which n is the number of primitives that appear in the representation of $\mathcal{O}$ (each primitive is evaluated in constant time).

Note the convenient connection between a logical predicate representation and a set-theoretic representation. Using the logical predicate, the unions and intersections of the set-theoretic representation are replaced by logical ORs and ANDs. It is well known from Boolean algebra that any complicated logical sentence can be reduced to a logical disjunction of conjunctions (this is often called "sum of products" in computer engineering). This is equivalent to our previous statement that $\mathcal{O}$ can always be represented as a union of intersections of primitives.

Polyhedral models For a 3D world, $\mathcal{W} = \mathbb{R}^3$, and the previous concepts can be nicely generalized from the 2D case by replacing polygons with polyhedra and

replacing half-plane primitives with half-space primitives. A boundary representation can be defined in terms of three features: vertices, edges, and faces. Every face is a "flat" polygon embedded in $\mathbb{R}^3$. Every edge forms a boundary between two faces. Every vertex forms a boundary between three or more edges.

Several data structures have been proposed that allow one to conveniently "walk" around the polyhedral features. For example, the *doubly connected edge list* [146] data structure contains three types of records: faces, half-edges, and vertices. Intuitively, a half-edge is a directed edge. Each vertex record holds the point coordinates and a pointer to an arbitrary half-edge that touches the vertex. Each face record contains a pointer to an arbitrary half-edge on its boundary. Each face is bounded by a circular list of half-edges. There is a pair of directed half-edge records for each edge of the polyhedon. Each half-edge is shown as an arrow in Figure 3.3b. Each half-edge record contains pointers to five other records: 1) the vertex from which the half-edge originates; 2) the "twin" half-edge, which bounds the neighboring face, and has the opposite direction; 3) the face that is bounded by the half-edge; 4) the next element in the circular list of edges that bound the face; and 5) the previous element in the circular list of edges that bound the face. Once all of these records have been defined, one can conveniently traverse the structure of the polyhedron.

Now consider a solid representation of a polyhedron. Suppose that $\mathcal{O}$ is a convex polyhedron, as shown in Figure 3.3. A solid representation can be constructed from the vertices. Each face of $\mathcal{O}$ has at least three vertices along its boundary. Assuming these vertices are not collinear, an equation of the plane that passes through them can be determined of the form

$$ax + by + cz + d = 0, \tag{3.7}$$

in which $a, b, c, d \in \mathbb{R}$ are constants.

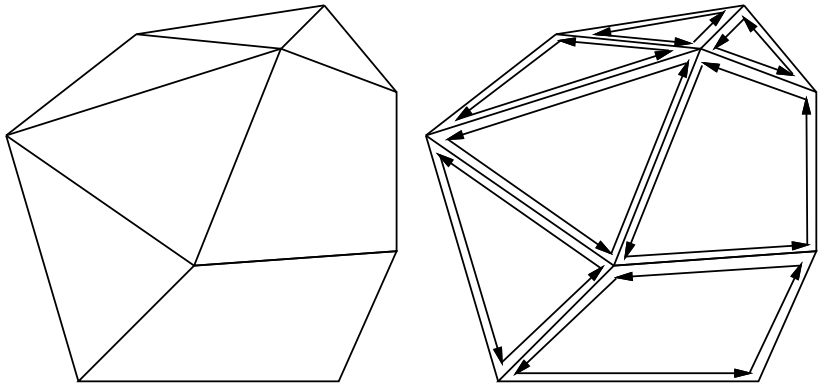
Once again, f can be constructed, except now $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ and

$$f(x, y, z) = ax + by + cz + d. \tag{3.8}$$

Let m be the number of faces. For each face of $\mathcal{O}$, a *half-space* H_i is defined as a subset of $\mathcal{W}$:

$$H_i = \{(x, y, z) \in \mathcal{W} \mid f_i(x, y, z) \leq 0\}. \tag{3.9}$$

It is important to choose f_i so that it takes on negative values inside of the polyhedron. In the case of a polygonal model, it was possible to consistently define f_i by proceeding in counterclockwise order around the boundary. In the case of a polyhedron, the half-edge data structure can be used to obtain for each face the list of edges that form its boundary in counterclockwise order. Figure 3.3b shows the edge ordering for each face. For every edge, the arrows point in opposite directions, as required by the half-edge data structure. The equation for each face can be consistently determined as follows. Choose three consecutive vertices, p_1, p_2, p_3 (they must not be collinear) in counterclockwise order on the



(a) (b)

Figure 3.3: (a) A polyhedron can be described in terms of faces, edges, and vertices. (b) The edges of each face can be stored in a circular list that is traversed in counterclockwise order with respect to the outward normal vector of the face.

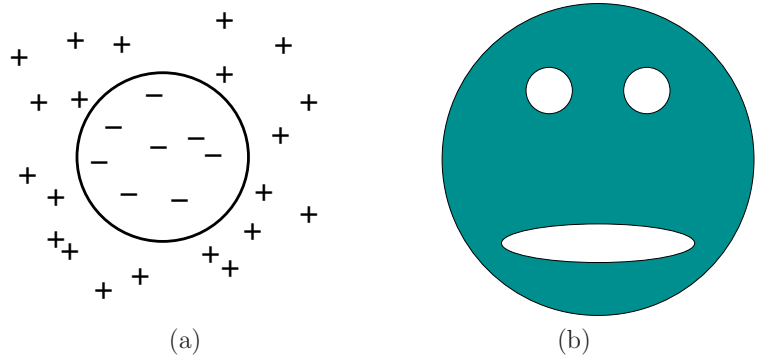
boundary of the face. Let v_{12} denote the vector from p_1 to p_2, and let v_{23} denote the vector from p_2 to p_3. The cross product $v = v_{12} \times v_{23}$ always yields a vector that points out of the polyhedron and is normal to the face. Recall that the vector $[a \;\; b \;\; c]$ is parallel to the normal to the plane. If its components are chosen as $a = v[1]$, $b = v[2]$, and $c = v[3]$, then $f(x, y, z) \leq 0$ for all points in the half-space that contains the polyhedron.

As in the case of a polygonal model, a convex polyhedron can be defined as the intersection of a finite number of half-spaces, one for each face. A nonconvex polyhedron can be defined as the union of a finite number of convex polyhedra. The predicate $\phi(x, y, z)$ can be defined in a similar manner, in this case yielding TRUE if $(x, y, z) \in \mathcal{O}$, and FALSE otherwise.

3.1.2 Semi-Algebraic Models

In both the polygonal and polyhedral models, f was a linear function. In the case of a semi-algebraic model for a 2D world, f can be any polynomial with real-valued coefficients and variables x and y. For a 3D world, f is a polynomial with variables x, y, and z. The class of semi-algebraic models includes both polygonal and polyhedral models, which use first-degree polynomials. A point set determined by a single polynomial primitive is called an *algebraic set*; a point set that can be obtained by a finite number of unions and intersections of algebraic sets is called a *semi-algebraic set*.

Consider the case of a 2D world. A solid representation can be defined using



(a) (b)

Figure 3.4: (a) Once again, f is used to partition $\mathbb{R}^2$ into two regions. In this case, the algebraic primitive represents a disc-shaped region. (b) The shaded "face" can be exactly modeled using only four algebraic primitives.

algebraic primitives of the form

$$H = \{(x, y) \in \mathcal{W} \mid f(x, y) \leq 0\}. \tag{3.10}$$

As an example, let $f = x^2 + y^2 - 4$. In this case, H represents a disc of radius 2 that is centered at the origin. This corresponds to the set of points (x, y) for which $f(x, y) \leq 0$, as depicted in Figure 3.4a.

Example 3.1 (Gingerbread Face) Consider constructing a model of the shaded region shown in Figure 3.4b. Let the center of the outer circle have radius r_1 and be centered at the origin. Suppose that the "eyes" have radius r_2 and r_3 and are centered at (x_2, y_2) and (x_3, y_3), respectively. Let the "mouth" be an ellipse with major axis a and minor axis b and is centered at $(0, y_4)$. The functions are defined as

$$\begin{aligned} f_1 &= x^2 + y^2 - r_1^2, \\ f_2 &= -\big((x - x_2)^2 + (y - y_2)^2 - r_2^2\big), \\ f_3 &= -\big((x - x_3)^2 + (y - y_3)^2 - r_3^2\big), \\ f_4 &= -\big(x^2/a^2 + (y - y_4)^2/b^2 - 1\big). \end{aligned} \tag{3.11}$$

For f_2, f_3, and f_4, the familiar circle and ellipse equations were multiplied by -1 to yield algebraic primitives for all points outside of the circle or ellipse. The shaded region $\mathcal{O}$ is represented as

$$\mathcal{O} = H_1 \cap H_2 \cap H_3 \cap H_4. \tag{3.12}$$

■

In the case of semi-algebraic models, the intersection of primitives does not necessarily result in a convex subset of $\mathcal{W}$. In general, however, it might be necessary to form $\mathcal{O}$ by taking unions and intersections of algebraic primitives.

A logical predicate, $\phi(x, y)$, can once again be formed, and collision checking is still performed in time that is linear in the number of primitives. Note that it is still very efficient to evaluate every primitive; f is just a polynomial that is evaluated on the point (x, y, z).

The semi-algebraic formulation generalizes easily to the case of a 3D world. This results in algebraic primitives of the form

$$H = \{(x, y, z) \in \mathcal{W} \mid f(x, y, z) \leq 0\}, \tag{3.13}$$

which can be used to define a solid representation of a 3D obstacle $\mathcal{O}$ and a logical predicate ϕ.

Equations (3.10) and (3.13) are sufficient to express any model of interest. One may define many other primitives based on different relations, such as $f(x, y, z) \geq 0$, $f(x, y, z) = 0$, $f(x, y, z) < 0$, $f(x, y, z) = 0$, and $f(x, y, z) \neq 0$; however, most of them do not enhance the set of models that can be expressed. They might, however, be more convenient in certain contexts. To see that some primitives do not allow new models to be expressed, consider the primitive

$$H = \{(x, y, z) \in \mathcal{W} \mid f(x, y, z) \geq 0\}. \tag{3.14}$$

The right part may be alternatively represented as $-f(x, y, z) \leq 0$, and $-f$ may be considered as a new polynomial function of x, y, and z. For an example that involves the $=$ relation, consider the primitive

$$H = \{(x, y, z) \in \mathcal{W} \mid f(x, y, z) = 0\}. \tag{3.15}$$

It can instead be constructed as $H = H_1 \cap H_2$, in which

$$H_1 = \{(x, y, z) \in \mathcal{W} \mid f(x, y, z) \leq 0\} \tag{3.16}$$

and

$$H_2 = \{(x, y, z) \in \mathcal{W} \mid -f(x, y, z) \leq 0\}. \tag{3.17}$$

The relation $<$ does add some expressive power if it is used to construct primitives.[2] It is needed to construct models that do not include the outer boundary (for example, the set of all points *inside* of a sphere, which does not include points *on* the sphere). These are generally called *open sets* and are defined Chapter 4.

[2]An alternative that yields the same expressive power is to still use $\leq$, but allow set complements, in addition to unions and intersections.

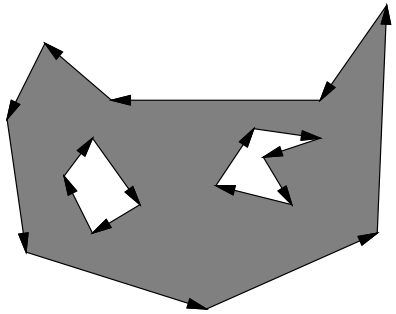


Figure 3.5: A polygon with holes can be expressed by using different orientations: counterclockwise for the outer boundary and clockwise for the hole boundaries. Note that the shaded part is always to the left when following the arrows.

3.1.3 Other Models

The choice of a model often depends on the types of operations that will be performed by the planning algorithm. For combinatorial motion planning methods, to be covered in Chapter 6, the particular representation is critical. On the other hand, for sampling-based planning methods, to be covered in Chapter 5, the particular representation is important only to the collision detection algorithm, which is treated as a "black box" as far as planning is concerned. Therefore, the models given in the remainder of this section are more likely to appear in sampling-based approaches and may be invisible to the designer of a planning algorithm (although it is never wise to forget completely about the representation).

Nonconvex polygons and polyhedra The method in Section 3.1.1 required nonconvex polygons to be represented as a union of convex polygons. Instead, a boundary representation of a nonconvex polygon may be directly encoded by listing vertices in a specific order; assume that counterclockwise order is used. Each polygon of m vertices may be encoded by a list of the form (x_1, y_1), (x_2, y_2), ..., (x_m, y_m). It is assumed that there is an edge between each (x_i, y_i) and (x_{i+1}, y_{i+1}) for each i from 1 to $m - 1$, and also an edge between (x_m, y_m) and (x_1, y_1). Ordinarily, the vertices should be chosen in a way that makes the polygon *simple*, meaning that no edges intersect. In this case, there is a well-defined interior of the polygon, which is to the left of every edge, if the vertices are listed in counterclockwise order.

What if a polygon has a hole in it? In this case, the boundary of the hole can be expressed as a polygon, but with its vertices appearing in the clockwise direction. To the left of each edge is the interior of the outer polygon, and to the right is the hole, as shown in Figure 3.5

Although the data structures are a little more complicated for three dimensions, boundary representations of nonconvex polyhedra may be expressed in a

Figure 3.6: Triangle strips and triangle fans can reduce the number of redundant points.

similar manner. In this case, instead of an edge list, one must specify faces, edges, and vertices, with pointers that indicate their incidence relations. Consistent orientations must also be chosen, and holes may be modeled once again by selecting opposite orientations.

3D triangles Suppose $\mathcal{W} = \mathbb{R}^3$. One of the most convenient geometric models to express is a set of triangles, each of which is specified by three points, (x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3). This model has been popular in computer graphics because graphics acceleration hardware primarily uses triangle primitives. It is assumed that the interior of the triangle is part of the model. Thus, two triangles are considered as "colliding" if one pokes into the interior of another. This model offers great flexibility because there are no constraints on the way in which triangles must be expressed; however, this is also one of the drawbacks. There is no coherency that can be exploited to easily declare whether a point is "inside" or "outside" of a 3D obstacle. If there is at least some coherency, then it is sometimes preferable to reduce redundancy in the specification of triangle coordinates (many triangles will share the same corners). Representations that remove this redundancy are called a *triangle strip*, which is a sequence of triangles such that each adjacent pair shares a common edge, and a *triangle fan*, which is a triangle strip in which all triangles share a common vertex. See Figure 3.6.

Nonuniform rational B-splines (NURBS) These are used in many engineering design systems to allow convenient design and adjustment of curved surfaces, in applications such as aircraft or automobile body design. In contrast to semi-algebraic models, which are implicit equations, NURBS and other splines are parametric equations. This makes computations such as rendering easier; however, others, such as collision detection, become more difficult. These models may be defined in any dimension. A brief 2D formulation is given here.

A curve can be expressed as

$$C(u) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,k}(u)}{\sum_{i=0}^{n} w_i N_{i,k}(u)}, \tag{3.18}$$

in which $w_i \in \mathbb{R}$ are *weights* and P_i are control points. The $N_{i,k}$ are normalized

basis functions of degree k, which can be expressed recursively as

$$N_{i,k}(u) = \left(\frac{u - t_i}{t_{i+k} - t_i}\right) N_{i,k-1}(u) + \left(\frac{t_{i+k+1} - u}{t_{i+k+1} - t_{i+1}}\right) N_{i+1,k-1}(u). \tag{3.19}$$

The basis of the recursion is $N_{i,0}(u) = 1$ if $t_i \leq u < t_{i+1}$, and $N_{i,0}(u) = 0$ otherwise. A *knot vector* is a nondecreasing sequence of real values, $\{t_0, t_1, \ldots, t_m\}$, that controls the intervals over which certain basic functions take effect.

Bitmaps For either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, it is possible to discretize a bounded portion of the world into rectangular cells that may or may not be occupied. The resulting model looks very similar to Example 2.1. The resolution of this discretization determines the number of cells per axis and the quality of the approximation. The representation may be considered as a binary image in which each "1" in the image corresponds to a rectangular region that contains at least one point of $\mathcal{O}$, and "0" represents those that do not contain any of $\mathcal{O}$. Although bitmaps do not have the elegance of the other models, they often arise in applications. One example is a digital map constructed by a mobile robot that explores an environment with its sensors. One generalization of bitmaps is a *gray-scale map* or *occupancy grid*. In this case, a numerical value may be assigned to each cell, indicating quantities such as "the probability that an obstacle exists" or the "expected difficulty of traversing the cell." The latter interpretation is often used in terrain maps for navigating planetary rovers.

Superquadrics Instead of using polynomials to define f_i, many generalizations can be constructed. One popular primitive is a *superquadric*, which generalizes quadric surfaces. One example is a superellipsoid, which is given for $\mathcal{W} = \mathbb{R}^3$ by

$$\left(|x/a|^{n_1} + |y/b|^{n_2}\right)^{n_1/n_2} + |z/c|^{n_1} - 1 \leq 0, \tag{3.20}$$

in which $n_1 \geq 2$ and $n_2 \geq 2$. If $n_1 = n_2 = 2$, an ellipse is generated. As n_1 and n_2 increase, the superellipsoid becomes shaped like a box with rounded corners.

Generalized cylinders A *generalized cylinder* is a generalization of an ordinary cylinder. Instead of being limited to a line, the center axis is a continuous *spine curve*, $(x(s), y(s), z(s))$, for some parameter $s \in [0, 1]$. Instead of a constant radius, a radius function $r(s)$ is defined along the spine. The value $r(s)$ is the radius of the circle obtained as the cross section of the generalized cylinder at the point $(x(s), y(s), z(s))$. The normal to the cross-section plane is the tangent to the spine curve at s.

3.2 Rigid-Body Transformations

Any of the techniques from Section 3.1 can be used to define both the obstacle region and the robot. Let $\mathcal{O}$ refer to the *obstacle region*, which is a subset of $\mathcal{W}$.

Let $\mathcal{A}$ refer to the robot, which is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$, matching the dimension of $\mathcal{W}$. Although $\mathcal{O}$ remains fixed in the world, $\mathcal{W}$, motion planning problems will require "moving" the robot, $\mathcal{A}$.

3.2.1 General Concepts

Before giving specific transformations, it will be helpful to define them in general to avoid confusion in later parts when intuitive notions might fall apart. Suppose that a rigid robot, $\mathcal{A}$, is defined as a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$. A *rigid-body transformation* is a function, $h : \mathcal{A} \to \mathcal{W}$, that maps every point of $\mathcal{A}$ into $\mathcal{W}$ with two requirements: 1) The distance between any pair of points of $\mathcal{A}$ must be preserved, and 2) the orientation of $\mathcal{A}$ must be preserved (no "mirror images").

Using standard function notation, $h(a)$ for some $a \in \mathcal{A}$ refers to the point in $\mathcal{W}$ that is "occupied" by a. Let

$$h(\mathcal{A}) = \{h(a) \in \mathcal{W} \mid a \in \mathcal{A}\}, \tag{3.21}$$

which is the image of h and indicates all points in $\mathcal{W}$ occupied by the transformed robot.

Transforming the robot model Consider transforming a robot model. If $\mathcal{A}$ is expressed by naming specific points in $\mathbb{R}^2$, as in a boundary representation of a polygon, then each point is simply transformed from a to $h(a) \in \mathcal{W}$. In this case, it is straightforward to transform the entire model using h. However, there is a slight complication if the robot model is expressed using primitives, such as

$$H_i = \{a \in \mathbb{R}^2 \mid f_i(a) \leq 0\}. \tag{3.22}$$

This differs slightly from (3.2) because the robot is defined in $\mathbb{R}^2$ (which is not necessarily $\mathcal{W}$), and also a is used to denote a point $(x, y) \in \mathcal{A}$. Under a transformation h, the primitive is transformed as

$$h(H_i) = \{h(a) \in \mathcal{W} \mid f_i(a) \leq 0\}. \tag{3.23}$$

To transform the primitive completely, however, it is better to directly name points in $w \in \mathcal{W}$, as opposed to $h(a) \in \mathcal{W}$. Using the fact that $a = h^{-1}(w)$, this becomes

$$h(H_i) = \{w \in \mathcal{W} \mid f_i(h^{-1}(w)) \leq 0\}, \tag{3.24}$$

in which the inverse of h appears in the right side because the original point $a \in A$ needs to be recovered to evaluate f_i. Therefore, it is important to be careful because either h or h^{-1} may be required to transform the model. This will be observed in more specific contexts in some coming examples.

A parameterized family of transformations It will become important to study families of transformations, in which some parameters are used to select the particular transformation. Therefore, it makes sense to generalize h to accept two variables: a parameter vector, $q \in \mathbb{R}^n$, along with $a \in \mathcal{A}$. The resulting transformed point a is denoted by $h(q, a)$, and the entire robot is transformed to $h(q, \mathcal{A}) \subset \mathcal{W}$.

The coming material will use the following shorthand notation, which requires the specific h to be inferred from the context. Let $h(q, a)$ be shortened to $a(q)$, and let $h(q, \mathcal{A})$ be shortened to $\mathcal{A}(q)$. This notation makes it appear that by adjusting the parameter q, the robot $\mathcal{A}$ travels around in $\mathcal{W}$ as different transformations are selected from the predetermined family. This is slightly abusive notation, but it is convenient. The expression $\mathcal{A}(q)$ can be considered as a set-valued function that yields the set of points in $\mathcal{W}$ that are occupied by $\mathcal{A}$ when it is transformed by q. Most of the time the notation does not cause trouble, but when it does, it is helpful to remember the definitions from this section, especially when trying to determine whether h or h^{-1} is needed.

Defining frames It was assumed so far that $\mathcal{A}$ is defined in $\mathbb{R}^2$ or $\mathbb{R}^3$, but before it is transformed, it is not considered to be a subset of $\mathcal{W}$. The transformation h *places* the robot in $\mathcal{W}$. In the coming material, it will be convenient to indicate this distinction using coordinate frames. The origin and coordinate basis vectors of $\mathcal{W}$ will be referred to as the *world frame*.[3] Thus, any point $w \in \mathcal{W}$ is expressed in terms of the world frame.

The coordinates used to define $\mathcal{A}$ are initially expressed in the *body frame*, which represents the origin and coordinate basis vectors of $\mathbb{R}^2$ or $\mathbb{R}^3$. In the case of $\mathcal{A} \subset \mathbb{R}^2$, it can be imagined that the body frame is painted on the robot. Transforming the robot is equivalent to converting its model from the body frame to the world frame. This has the effect of *placing*[4] $\mathcal{A}$ into $\mathcal{W}$ at some position and orientation. When multiple bodies are covered in Section 3.3, each body will have its own body frame, and transformations require expressing all bodies with respect to the world frame.

3.2.2 2D Transformations

Translation A rigid robot $\mathcal{A} \subset \mathbb{R}^2$ is *translated* by using two parameters, $x_t, y_t \in \mathbb{R}$. Using definitions from Section 3.2.1, $q = (x_t, y_t)$, and h is defined as

$$h(x, y) = (x + x_t, y + y_t). \tag{3.25}$$

[3]The world frame serves the same purpose as an inertial frame in Newtonian mechanics. Intuitively, it is a frame that remains fixed and from which all measurements are taken. See Section 13.3.1.

[4]Technically, this placement is a function called an *orientation-preserving isometric embedding*.

A boundary representation of $\mathcal{A}$ can be translated by transforming each vertex in the sequence of polygon vertices using (3.25). Each point, (x_i, y_i), in the sequence is replaced by $(x_i + x_t, y_i + y_t)$.

Now consider a solid representation of $\mathcal{A}$, defined in terms of primitives. Each primitive of the form

$$H_i = \{(x, y) \in \mathbb{R}^2 \mid f(x, y) \leq 0\} \tag{3.26}$$

is transformed to

$$h(H_i) = \{(x, y) \in \mathcal{W} \mid f(x - x_t, y - y_t) \leq 0\}. \tag{3.27}$$

Example 3.2 (Translating a Disc) For example, suppose the robot is a disc of unit radius, centered at the origin. It is modeled by a single primitive,

$$H_i = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 - 1 \leq 0\}. \tag{3.28}$$

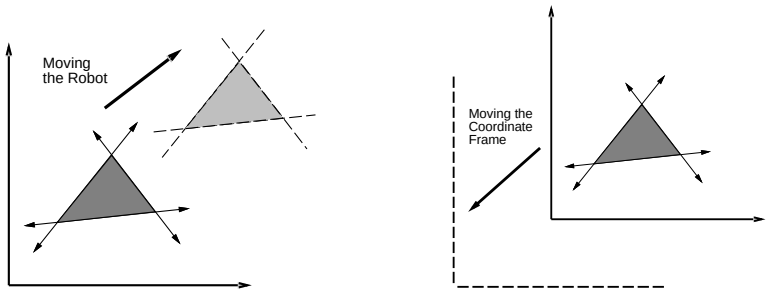
Suppose $\mathcal{A} = H_i$ is translated x_t units in the x direction and y_t units in the y direction. The transformed primitive is

$$h(H_i) = \{(x, y) \in \mathcal{W} \mid (x - x_t)^2 + (y - y_t)^2 - 1 \leq 0\}, \tag{3.29}$$

which is the familiar equation for a disc centered at (x_t, y_t). In this example, the inverse, h^{-1} is used, as described in Section 3.2.1. ■

The translated robot is denoted as $\mathcal{A}(x_t, y_t)$. Translation by $(0, 0)$ is the *identity transformation*, which results in $\mathcal{A}(0, 0) = \mathcal{A}$, if it is assumed that $\mathcal{A} \subset \mathcal{W}$ (recall that $\mathcal{A}$ does not necessarily have to be initially embedded in $\mathcal{W}$). It will be convenient to use the term *degrees of freedom* to refer to the maximum number of independent parameters that are needed to completely characterize the transformation applied to the robot. If the set of allowable values for x_t and y_t forms a two-dimensional subset of $\mathbb{R}^2$, then the degrees of freedom is two.

Suppose that $\mathcal{A}$ is defined directly in $\mathcal{W}$ with translation. As shown in Figure 3.7, there are two interpretations of a rigid-body transformation applied to $\mathcal{A}$: 1) The world frame remains fixed and the robot is transformed; 2) the robot remains fixed and the world frame is translated. The first one characterizes the effect of the transformation from a fixed world frame, and the second one indicates how the transformation appears from the robot's perspective. Unless stated otherwise, the first interpretation will be used when we refer to motion planning problems because it often models a robot moving in a physical world. Numerous books cover coordinate transformations under the second interpretation. This has been known to cause confusion because the transformations may sometimes appear "backward" from what is desired in motion planning.



(a) Translation of the robot (b) Translation of the frame

Figure 3.7: Every transformation has two interpretations.

Rotation The robot, $\mathcal{A}$, can be *rotated* counterclockwise by some angle $\theta \in [0, 2\pi)$ by mapping every $(x, y) \in \mathcal{A}$ as

$$(x, y) \mapsto (x \cos\theta - y \sin\theta,\ x \sin\theta + y \cos\theta). \tag{3.30}$$

Using a 2×2 rotation matrix,

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{3.31}$$

the transformation can be written as

$$\begin{pmatrix} x \cos\theta - y \sin\theta \\ x \sin\theta + y \cos\theta \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}. \tag{3.32}$$

Using the notation of Section 3.2.1, $R(\theta)$ becomes $h(q)$, for which $q = \theta$. For linear transformations, such as the one defined by (3.32), recall that the column vectors represent the basis vectors of the new coordinate frame. The column vectors of $R(\theta)$ are unit vectors, and their inner product (or dot product) is zero, indicating that they are orthogonal. Suppose that the x and y coordinate axes, which represent the body frame, are "painted" on $\mathcal{A}$. The columns of $R(\theta)$ can be derived by considering the resulting directions of the x- and y-axes, respectively, after performing a counterclockwise rotation by the angle θ. This interpretation generalizes nicely for higher dimensional rotation matrices.

Note that the rotation is performed about the origin. Thus, when defining the model of $\mathcal{A}$, the origin should be placed at the intended axis of rotation. Using the semi-algebraic model, the entire robot model can be rotated by transforming each primitive, yielding $\mathcal{A}(\theta)$. The inverse rotation, $R(-\theta)$, must be applied to each primitive.

Combining translation and rotation Suppose a rotation by θ is performed, followed by a translation by x_t, y_t. This can be used to place the robot in any desired position and orientation. Note that translations and rotations do not commute! If the operations are applied successively, each $(x, y) \in \mathcal{A}$ is transformed to

$$\begin{pmatrix} x\cos\theta - y\sin\theta + x_t \\ x\sin\theta + y\cos\theta + y_t \end{pmatrix}. \tag{3.33}$$

The following matrix multiplication yields the same result for the first two vector components:

$$\begin{pmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & \cos\theta & y_t \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x\cos\theta - y\sin\theta + x_t \\ x\sin\theta + y\cos\theta + y_t \\ 1 \end{pmatrix}. \tag{3.34}$$

This implies that the 3×3 matrix,

$$T = \begin{pmatrix} \cos\theta & -\sin\theta & x_t \\ \sin\theta & \cos\theta & y_t \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.35}$$

represents a rotation followed by a translation. The matrix T will be referred to as a *homogeneous transformation matrix*. It is important to remember that T represents a rotation *followed by* a translation (not the other way around). Each primitive can be transformed using the inverse of T, resulting in a transformed solid model of the robot. The transformed robot is denoted by $\mathcal{A}(x_t, y_t, \theta)$, and in this case there are three degrees of freedom. The homogeneous transformation matrix is a convenient representation of the combined transformations; therefore, it is frequently used in robotics, mechanics, computer graphics, and elsewhere. It is called homogeneous because over $\mathbb{R}^3$ it is just a linear transformation without any translation. The trick of increasing the dimension by one to absorb the translational part is common in projective geometry [404].

3.2.3 3D Transformations

Rigid-body transformations for the 3D case are conceptually similar to the 2D case; however, the 3D case appears more difficult because rotations are significantly more complicated.

3D translation The robot, $\mathcal{A}$, is *translated* by some $x_t, y_t, z_t \in \mathbb{R}$ using

$$(x, y, z) \mapsto (x + x_t, y + y_t, z + z_t). \tag{3.36}$$

A primitive of the form

$$H_i = \{(x, y, z) \in \mathcal{W} \mid f_i(x, y, z) \leq 0\} \tag{3.37}$$

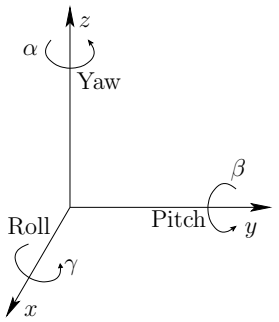


Figure 3.8: Any three-dimensional rotation can be described as a sequence of yaw, pitch, and roll rotations.

is transformed to

$$\{(x, y, z) \in \mathcal{W} \mid f_i(x - x_t, y - y_t, z - z_t) \leq 0\}. \tag{3.38}$$

The translated robot is denoted as $\mathcal{A}(x_t, y_t, z_t)$.

Yaw, pitch, and roll rotations A 3D body can be rotated about three orthogonal axes, as shown in Figure 3.8. Borrowing aviation terminology, these rotations will be referred to as yaw, pitch, and roll:

1. A *yaw* is a counterclockwise rotation of α about the z-axis. The rotation matrix is given by

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.39}$$

   Note that the upper left entries of $R_z(\alpha)$ form a 2D rotation applied to the x and y coordinates, whereas the z coordinate remains constant.

2. A *pitch* is a counterclockwise rotation of β about the y-axis. The rotation matrix is given by

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}. \tag{3.40}$$

3. A *roll* is a counterclockwise rotation of γ about the x-axis. The rotation matrix is given by

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}. \tag{3.41}$$

Each rotation matrix is a simple extension of the 2D rotation matrix, (3.31). For example, the yaw matrix, $R_z(\alpha)$, essentially performs a 2D rotation with respect to the x and y coordinates while leaving the z coordinate unchanged. Thus, the third row and third column of $R_z(\alpha)$ look like part of the identity matrix, while the upper right portion of $R_z(\alpha)$ looks like the 2D rotation matrix.

The yaw, pitch, and roll rotations can be used to place a 3D body in any orientation. A single rotation matrix can be formed by multiplying the yaw, pitch, and roll rotation matrices to obtain

$$R(\alpha,\beta,\gamma) = R_z(\alpha)\, R_y(\beta)\, R_x(\gamma) = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}. \tag{3.42}$$

It is important to note that $R(\alpha, \beta, \gamma)$ performs the roll first, then the pitch, and finally the yaw. If the order of these operations is changed, a different rotation matrix would result. Be careful when interpreting the rotations. Consider the final rotation, a yaw by α. Imagine sitting inside of a robot $\mathcal{A}$ that looks like an aircraft. If $\beta = \gamma = 0$, then the yaw turns the plane in a way that feels like turning a car to the left. However, for arbitrary values of β and γ, the final rotation axis will not be vertically aligned with the aircraft because the aircraft is left in an unusual orientation before α is applied. The yaw rotation occurs about the z-axis of the world frame, not the body frame of $\mathcal{A}$. Each time a new rotation matrix is introduced from the left, it has no concern for original body frame of $\mathcal{A}$. It simply rotates every point in $\mathbb{R}^3$ in terms of the world frame. Note that 3D rotations depend on three parameters, α, β, and γ, whereas 2D rotations depend only on a single parameter, θ. The primitives of the model can be transformed using $R(\alpha, \beta, \gamma)$, resulting in $\mathcal{A}(\alpha, \beta, \gamma)$.

Determining yaw, pitch, and roll from a rotation matrix It is often convenient to determine the α, β, and γ parameters directly from a given rotation matrix. Suppose an arbitrary rotation matrix

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{3.43}$$

is given. By setting each entry equal to its corresponding entry in (3.42), equations are obtained that must be solved for α, β, and γ. Note that $r_{21}/r_{11} = \tan\alpha$ and $r_{32}/r_{33} = \tan\gamma$. Also, $r_{31} = -\sin\beta$ and $\sqrt{r_{32}^2 + r_{33}^2} = \cos\beta$. Solving for each angle yields

$$\alpha = \tan^{-1}(r_{21}/r_{11}), \tag{3.44}$$

$$\beta = \tan^{-1}\Big(-r_{31}/\sqrt{r_{32}^2 + r_{33}^2}\Big), \tag{3.45}$$

and

$$\gamma = \tan^{-1}(r_{32}/r_{33}). \tag{3.46}$$

There is a choice of four quadrants for the inverse tangent functions. How can the correct quadrant be determined? Each quadrant should be chosen by using the signs of the numerator and denominator of the argument. The numerator sign selects whether the direction will be above or below the x-axis, and the denominator selects whether the direction will be to the left or right of the y-axis. This is the same as the atan2 function in the C programming language, which nicely expands the range of the arctangent to $[0, 2\pi)$. This can be applied to express (3.44), (3.45), and (3.46) as

$$\alpha = \text{atan2}(r_{21}, r_{11}), \tag{3.47}$$

$$\beta = \text{atan2}\Big(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\Big), \tag{3.48}$$

and

$$\gamma = \text{atan2}(r_{32}, r_{33}). \tag{3.49}$$

Note that this method assumes $r_{11} \neq 0$ and $r_{33} \neq 0$.

The homogeneous transformation matrix for 3D bodies As in the 2D case, a homogeneous transformation matrix can be defined. For the 3D case, a 4×4 matrix is obtained that performs the rotation given by $R(\alpha, \beta, \gamma)$, followed by a translation given by x_t, y_t, z_t. The result is

$$T = \begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma & x_t \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & y_t \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma & z_t \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.50}$$

Once again, the order of operations is critical. The matrix T in (3.50) represents the following sequence of transformations:

1. Roll by γ
2. Pitch by β
3. Yaw by α
4. Translate by (x_t, y_t, z_t).

The robot primitives can be transformed to yield $\mathcal{A}(x_t, y_t, z_t, \alpha, \beta, \gamma)$. A 3D rigid body that is capable of translation and rotation therefore has six degrees of freedom.

3.3 Transforming Kinematic Chains of Bodies

The transformations become more complicated for a chain of attached rigid bodies. For convenience, each rigid body is referred to as a *link*. Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$ denote a set of m links. For each i such that $1 \leq i < m$, link $\mathcal{A}_i$ is "attached" to

link $\mathcal{A}_{i+1}$ in a way that allows $\mathcal{A}_{i+1}$ some constrained motion with respect to $\mathcal{A}_i$. The motion constraint must be explicitly given, and will be discussed shortly. As an example, imagine a trailer that is attached to the back of a car by a hitch that allows the trailer to rotate with respect to the car. In general, a set of attached bodies will be referred to as a *linkage*. This section considers bodies that are attached in a single chain. This leads to a particular linkage called a *kinematic chain*.

3.3.1 A 2D Kinematic Chain

Before considering a kinematic chain, suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are unattached rigid bodies, each of which is capable of translating and rotating in $\mathcal{W} = \mathbb{R}^2$. Since each body has three degrees of freedom, there is a combined total of six degrees of freedom; the independent parameters are x_1, y_1, θ_1, x_2, y_2, and θ_2.

Attaching bodies When bodies are attached in a kinematic chain, degrees of freedom are removed. Figure 3.9 shows two different ways in which a pair of 2D links can be attached. The place at which the links are attached is called a *joint*. For a *revolute joint*, one link is capable only of rotation with respect to the other. For a *prismatic joint* is shown, one link slides along the other. Each type of joint removes two degrees of freedom from the pair of bodies. For example, consider a revolute joint that connects $\mathcal{A}_1$ to $\mathcal{A}_2$. Assume that the point $(0, 0)$ in the body frame of $\mathcal{A}_2$ is permanently fixed to a point (x_a, y_a) in the body frame of $\mathcal{A}_1$. This implies that the translation of $\mathcal{A}_2$ is completely determined once x_a and y_a are given. Note that x_a and y_a depend on x_1, y_1, and θ_1. This implies that $\mathcal{A}_1$ and $\mathcal{A}_2$ have a total of four degrees of freedom when attached. The independent parameters are x_1, y_1, θ_1, and θ_2. The task in the remainder of this section is to determine exactly how the models of $\mathcal{A}_1$, $\mathcal{A}_2$, ..., $\mathcal{A}_m$ are transformed when they are attached in a chain, and to give the expressions in terms of the independent parameters.

Consider the case of a kinematic chain in which each pair of links is attached by a revolute joint. The first task is to specify the geometric model for each link, $\mathcal{A}_i$. Recall that for a single rigid body, the origin of the body frame determines the axis of rotation. When defining the model for a link in a kinematic chain, excessive complications can be avoided by carefully placing the body frame. Since rotation occurs about a revolute joint, a natural choice for the origin is the joint between $\mathcal{A}_i$ and $\mathcal{A}_{i-1}$ for each $i > 1$. For convenience that will soon become evident, the x_i-axis for the body frame of $\mathcal{A}_i$ is defined as the line through the two joints that lie in $\mathcal{A}_i$, as shown in Figure 3.10. For the last link, $\mathcal{A}_m$, the x_m-axis can be placed arbitrarily, assuming that the origin is placed at the joint that connects $\mathcal{A}_m$ to $\mathcal{A}_{m-1}$. The body frame for the first link, $\mathcal{A}_1$, can be placed using the same considerations as for a single rigid body.

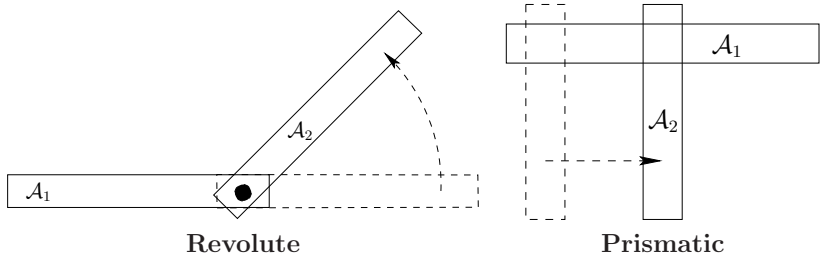


Figure 3.9: Two types of 2D joints: a revolute joint allows one link to rotate with respect to the other, and a prismatic joint allows one link to translate with respect to the other.

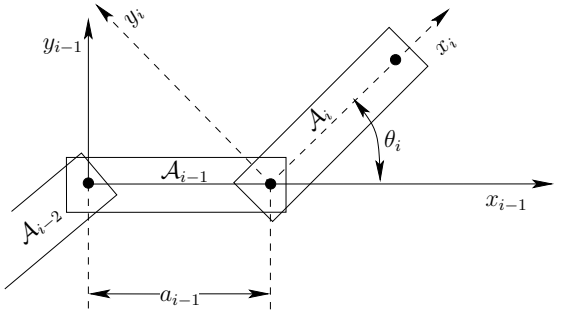


Figure 3.10: The body frame of each $\mathcal{A}_i$, for $1 < i < m$, is based on the joints that connect $\mathcal{A}_i$ to $\mathcal{A}_{i-1}$ and $\mathcal{A}_{i+1}$.

Homogeneous transformation matrices for 2D chains We are now prepared to determine the location of each link. The location in $\mathcal{W}$ of a point in $(x, y) \in \mathcal{A}_1$ is determined by applying the 2D homogeneous transformation matrix (3.35),

$$T_1 = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & x_t \\ \sin\theta_1 & \cos\theta_1 & y_t \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.51}$$

As shown in Figure 3.10, let a_{i-1} be the distance between the joints in $\mathcal{A}_{i-1}$. The orientation difference between $\mathcal{A}_i$ and $\mathcal{A}_{i-1}$ is denoted by the angle θ_i. Let T_i represent a 3×3 homogeneous transformation matrix (3.35), specialized for link $\mathcal{A}_i$ for $1 < i \leq m$,

$$T_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & a_{i-1} \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.52}$$

This generates the following sequence of transformations:

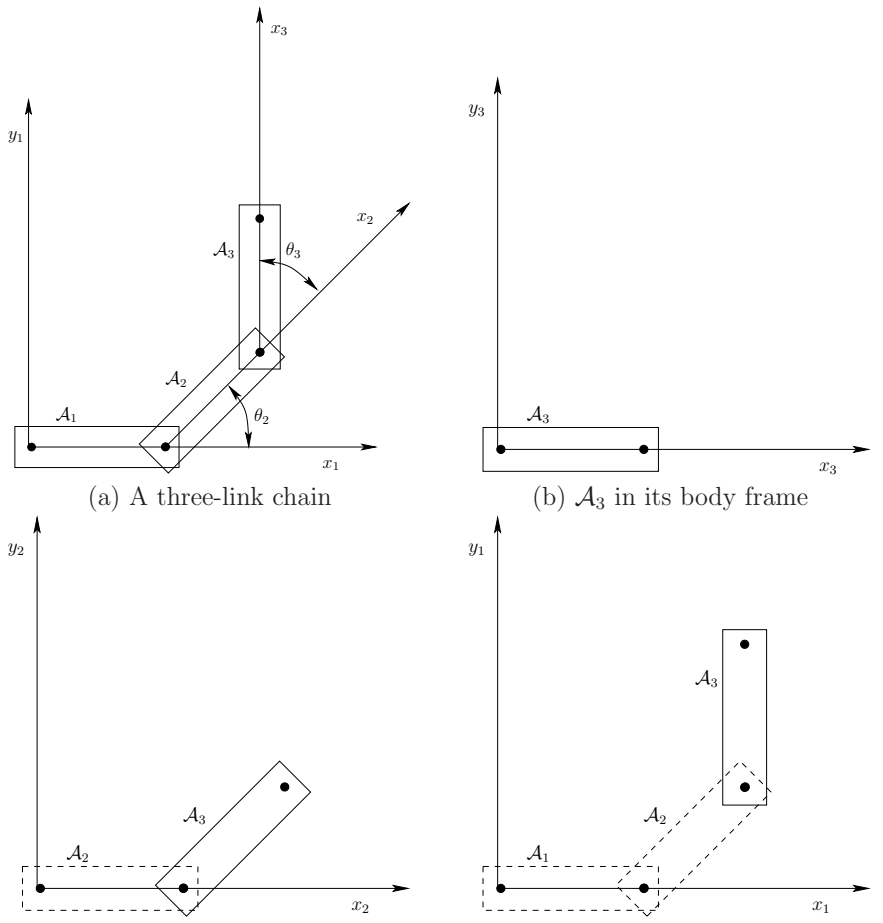
1. Rotate counterclockwise by θ_i.
2. Translate by a_{i-1} along the x-axis.

The transformation T_i expresses the difference between the body frame of $\mathcal{A}_i$ and the body frame of $\mathcal{A}_{i-1}$. The application of T_i moves $\mathcal{A}_i$ from its body frame to the body frame of $\mathcal{A}_{i-1}$. The application of $T_{i-1}T_i$ moves both $\mathcal{A}_i$ and $\mathcal{A}_{i-1}$ to the body frame of $\mathcal{A}_{i-2}$. By following this procedure, the location in $\mathcal{W}$ of any point $(x, y) \in \mathcal{A}_m$ is determined by multiplying the transformation matrices to obtain

$$T_1 T_2 \cdots T_m \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{3.53}$$

Example 3.3 (A 2D Chain of Three Links) To gain an intuitive understanding of these transformations, consider determining the configuration for link $\mathcal{A}_3$, as shown in Figure 3.11. Figure 3.11a shows a three-link chain in which $\mathcal{A}_1$ is at its initial configuration and the other links are each offset by $\pi/4$ from the previous link. Figure 3.11b shows the frame in which the model for $\mathcal{A}_3$ is initially defined. The application of T_3 causes a rotation of θ_3 and a translation by a_2. As shown in Figure 3.11c, this places $\mathcal{A}_3$ in its appropriate configuration. Note that $\mathcal{A}_2$ can be placed in its initial configuration, and it will be attached correctly to $\mathcal{A}_3$. The application of T_2 to the previous result places both $\mathcal{A}_3$ and $\mathcal{A}_2$ in their proper configurations, and $\mathcal{A}_1$ can be placed in its initial configuration. ■

For revolute joints, the a_i parameters are constants, and the θ_i parameters are variables. The transformed mth link is represented as $\mathcal{A}_m(x_t, y_t, \theta_1, \ldots, \theta_m)$. In some cases, the first link might have a fixed location in the world. In this case, the



(a) A three-link chain (b) $\mathcal{A}_3$ in its body frame

(c) T_3 puts $\mathcal{A}_3$ in $\mathcal{A}_2$'s body frame (d) T_2T_3 puts $\mathcal{A}_3$ in $\mathcal{A}_1$'s body frame

Figure 3.11: Applying the transformation T_2T_3 to the model of $\mathcal{A}_3$. If T_1 is the identity matrix, then this yields the location in $\mathcal{W}$ of points in $\mathcal{A}_3$.

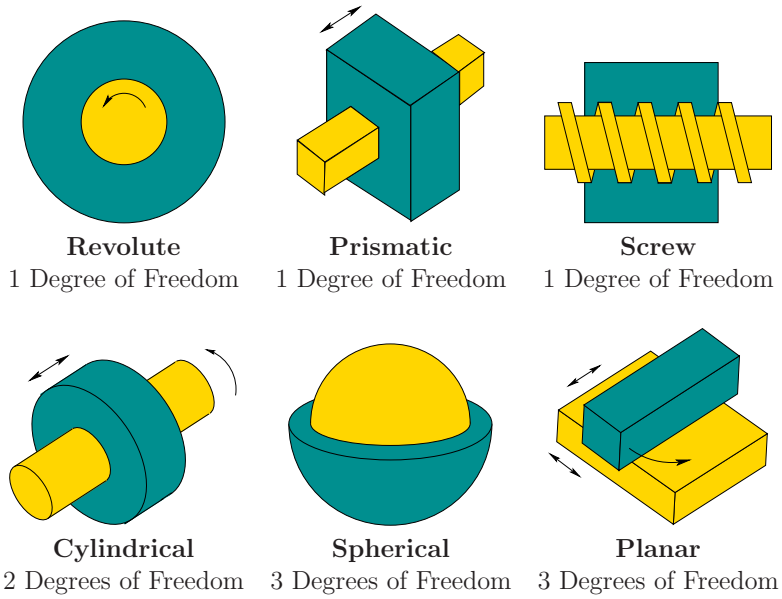


Figure 3.12: Types of 3D joints arising from the 2D surface contact between two bodies.

revolute joints account for all degrees of freedom, yielding $\mathcal{A}_m(\theta_1, \ldots, \theta_m)$. For prismatic joints, the a_i parameters are variables, instead of the θ_i parameters. It is straightforward to include both types of joints in the same kinematic chain.

3.3.2 A 3D Kinematic Chain

As for a single rigid body, the 3D case is significantly more complicated than the 2D case due to 3D rotations. Also, several more types of joints are possible, as shown in Figure 3.12. Nevertheless, the main ideas from the transformations of 2D kinematic chains extend to the 3D case. The following steps from Section 3.3.1 will be recycled here:

1. The body frame must be carefully placed for each $\mathcal{A}_i$.
2. Based on joint relationships, several parameters are measured.
3. The parameters define a homogeneous transformation matrix, T_i.
4. The location in $\mathcal{W}$ of any point in $\mathcal{A}_m$ is given by applying the matrix $T_1 T_2 \cdots T_m$.

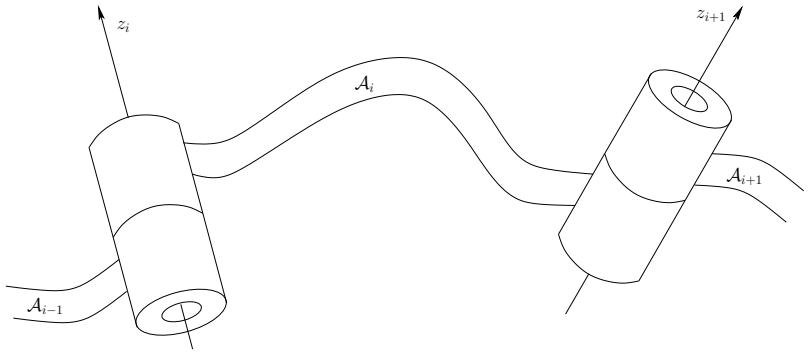


Figure 3.13: The rotation axes for a generic link attached by revolute joints.

Consider a kinematic chain of m links in $\mathcal{W} = \mathbb{R}^3$, in which each $\mathcal{A}_i$ for $1 \leq i < m$ is attached to $\mathcal{A}_{i+1}$ by a revolute joint. Each link can be a complicated, rigid body as shown in Figure 3.13. For the 2D problem, the coordinate frames were based on the points of attachment. For the 3D problem, it is convenient to use the axis of rotation of each revolute joint (this is equivalent to the point of attachment for the 2D case). The axes of rotation will generally be skew lines in $\mathbb{R}^3$, as shown in Figure 3.14. Let the z_i-axis be the axis of rotation for the revolute joint that holds $\mathcal{A}_i$ to $\mathcal{A}_{i-1}$. Between each pair of axes in succession, let the x_i-axis join the closest pair of points between the z_i- and z_{i+1}-axes, with the origin on the z_i-axis and the direction pointing towards the nearest point of the z_{i+1}-axis. This axis is uniquely defined if the z_i- and z_{i+1}-axes are not parallel. The recommended body frame for each $\mathcal{A}_i$ will be given with respect to the z_i- and x_i-axes, which are shown in Figure 3.14. Assuming a right-handed coordinate system, the y_i-axis points away from us in Figure 3.14. In the transformations that will appear shortly, the coordinate frame given by x_i, y_i, and z_i will be most convenient for defining the model for $\mathcal{A}_i$. It might not always appear convenient because the origin of the frame may even lie outside of $\mathcal{A}_i$, but the resulting transformation matrices will be easy to understand.

In Section 3.3.1, each T_i was defined in terms of two parameters, a_{i-1} and θ_i. For the 3D case, four parameters will be defined: d_i, θ_i, a_{i-1}, and α_{i-1}. These are referred to as *Denavit-Hartenberg (DH) parameters* [223]. The definition of each parameter is indicated in Figure 3.15. Figure 3.15a shows the definition of d_i. Note that the x_{i-1}- and x_i-axes contact the z_i-axis at two different places. Let d_i denote signed distance between these points of contact. If the x_i-axis is above the x_{i-1}-axis along the z_i-axis, then d_i is positive; otherwise, d_i is negative. The parameter θ_i is the angle between the x_i- and x_{i-1}-axes, which corresponds to the rotation about the z_i-axis that moves the x_{i-1}-axis to coincide with the x_i-axis. The parameter a_i is the distance between the z_i- and z_{i-1}-axes; recall these are generally skew lines in $\mathbb{R}^3$. The parameter α_{i-1} is the angle between the z_i- and

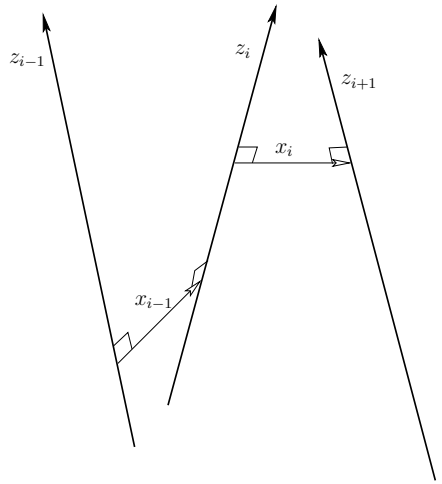


Figure 3.14: The rotation axes of the generic links are skew lines in $\mathbb{R}^3$.

z_{i-1}-axes.

Two screws The homogeneous transformation matrix T_i will be constructed by combining two simpler transformations. The transformation

$$R_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.54}$$

causes a rotation of θ_i about the z_i-axis, and a translation of d_i along the z_i-axis. Notice that the rotation by θ_i and translation by d_i commute because both operations occur with respect to the same axis, z_i. The combined operation of a translation and rotation with respect to the same axis is referred to as a *screw* (as in the motion of a screw through a nut). The effect of R_i can thus be considered as a screw about the z_i-axis. The second transformation is

$$Q_{i-1} = \begin{pmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.55}$$

which can be considered as a screw about the x_{i-1}-axis. A rotation of α_{i-1} about the x_{i-1}-axis and a translation of a_{i-1} are performed.

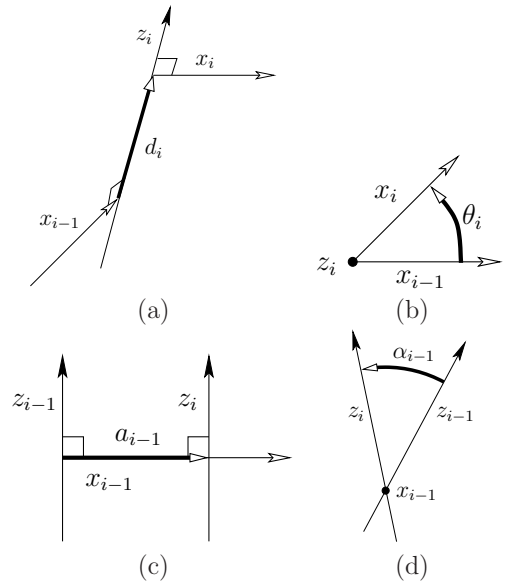


Figure 3.15: Definitions of the four DH parameters: d_i, θ_i, a_{i-1}, α_{i-1}. The z_i- and x_{i-1}-axes in (b) and (d), respectively, are pointing outward. Any parameter may be positive, zero, or negative.

The homogeneous transformation matrix The transformation T_i, for each i such that $1 < i \leq m$, is

$$T_i = Q_{i-1}R_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.56}$$

This can be considered as the 3D counterpart to the 2D transformation matrix, (3.52). The following four operations are performed in succession:

1. Translate by d_i along the z_i-axis.
2. Rotate counterclockwise by θ_i about the z_i-axis.
3. Translate by a_{i-1} along the x_{i-1}-axis.
4. Rotate counterclockwise by α_{i-1} about the x_{i-1}-axis.

As in the 2D case, the first matrix, T_1, is special. To represent any position and orientation of $\mathcal{A}_1$, it could be defined as a general rigid-body homogeneous

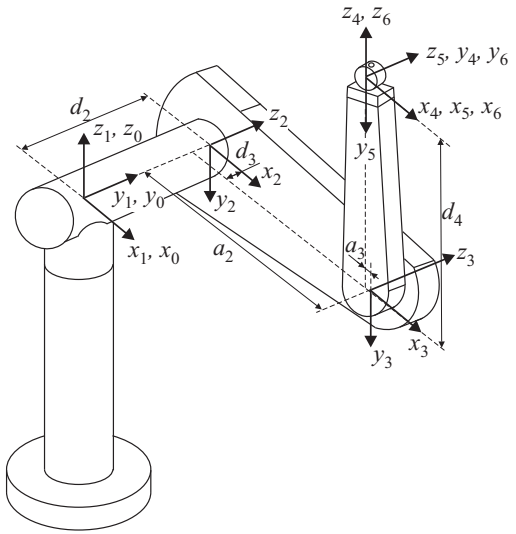


Figure 3.16: The Puma 560 is shown along with the DH parameters and body frames for each link in the chain. This figure is borrowed from [291] by courtesy of the authors.

transformation matrix, (3.50). If the first body is only capable of rotation via a revolute joint, then a simple convention is usually followed. Let the a_0, α_0 parameters of T_1 be assigned as $a_0 = \alpha_0 = 0$ (there is no z_0-axis). This implies that Q_0 from (3.55) is the identity matrix, which makes $T_1 = R_1$.

The transformation T_i for $i > 1$ gives the relationship between the body frame of $\mathcal{A}_i$ and the body frame of $\mathcal{A}_{i-1}$. The position of a point (x, y, z) on $\mathcal{A}_m$ is given by

$$T_1 T_2 \cdots T_m \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{3.57}$$

For each revolute joint, θ_i is treated as the only variable in T_i. Prismatic joints can be modeled by allowing a_i to vary. More complicated joints can be modeled as a sequence of degenerate joints. For example, a spherical joint can be considered as a sequence of three zero-length revolute joints; the joints perform a roll, a pitch, and a yaw. Another option for more complicated joints is to abandon the DH representation and directly develop the homogeneous transformation matrix. This might be needed to preserve topological properties that become important in Chapter 4.

Example 3.4 (Puma 560) This example demonstrates the 3D chain kinematics on a classic robot manipulator, the PUMA 560, shown in Figure 3.16. The

Matrix	α_{i-1}	a_{i-1}	θ_i	d_i
$T_1(\theta_1)$	0	0	θ_1	0
$T_2(\theta_2)$	$-\pi/2$	0	θ_2	d_2
$T_3(\theta_3)$	0	a_2	θ_3	d_3
$T_4(\theta_4)$	$\pi/2$	a_3	θ_4	d_4
$T_5(\theta_5)$	$-\pi/2$	0	θ_5	0
$T_6(\theta_6)$	$\pi/2$	0	θ_6	0

Figure 3.17: The DH parameters are shown for substitution into each homogeneous transformation matrix (3.56). Note that a_3 and d_3 are negative in this example (they are signed displacements, not distances).

current parameterization here is based on [29, 291]. The procedure is to determine appropriate body frames to represent each of the links. The first three links allow the hand (called an end-effector) to make large movements in $\mathcal{W}$, and the last three enable the hand to achieve a desired orientation. There are six degrees of freedom, each of which arises from a revolute joint. The body frames are shown in Figure 3.16, and the corresponding DH parameters are given in Figure 3.17. Each transformation matrix T_i is a function of θ_i; hence, it is written $T_i(\theta_i)$. The other parameters are fixed for this example. Only θ_1, θ_2, ..., θ_6 are allowed to vary.

The parameters from Figure 3.17 may be substituted into the homogeneous transformation matrices to obtain

$$T_1(\theta_1) = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.58}$$

$$T_2(\theta_2) = \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ -\sin\theta_2 & -\cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.59}$$

$$T_3(\theta_3) = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.60}$$

$$T_4(\theta_4) = \begin{pmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & a_3 \\ 0 & 0 & -1 & -d_4 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.61}$$

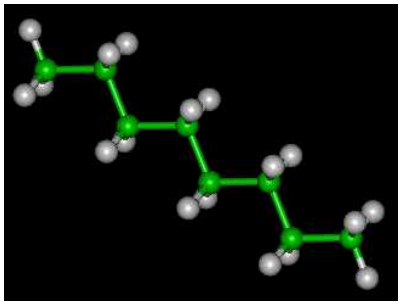


Figure 3.18: A hydrocarbon (octane) molecule with 8 carbon atoms and 18 hydrogen atoms (courtesy of the New York University MathMol Library).

$$T_5(\theta_5) = \begin{pmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin\theta_5 & -\cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{3.62}$$

and

$$T_6(\theta_6) = \begin{pmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.63}$$

A point (x, y, z) in the body frame of the last link $\mathcal{A}_6$ appears in $\mathcal{W}$ as

$$T_1(\theta_1)T_2(\theta_2)T_3(\theta_3)T_4(\theta_4)T_5(\theta_5)T_6(\theta_6) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \tag{3.64}$$

■

Example 3.5 (Transforming Octane) Figure 3.18 shows a ball-and-stick model of an octane molecule. Each "ball" is an atom, and each "stick" represents a bond between a pair of atoms. There is a linear chain of eight carbon atoms, and a bond exists between each consecutive pair of carbons in the chain. There are also numerous hydrogen atoms, but we will ignore them. Each bond between a pair of carbons is capable of twisting, as shown in Figure 3.19. Studying the configurations (called *conformations*) of molecules is an important part of computational biology. It is assumed that there are seven degrees of freedom, each of which

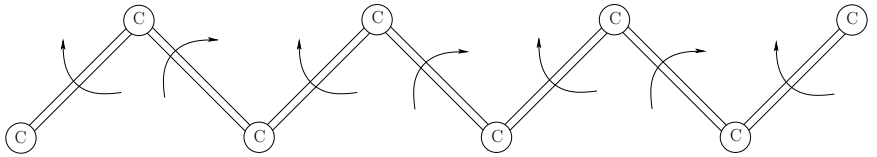


Figure 3.19: Consider transforming the spine of octane by ignoring the hydrogen atoms and allowing the bonds between carbons to rotate. This can be easily constructed with balls and sticks (e.g., Tinkertoys). If the first link is held fixed, then there are six degrees of freedom. The rotation of the last link is ignored.

arises from twisting a bond. The techniques from this section can be applied to represent these transformations.

Note that the bonds correspond exactly to the axes of rotation. This suggests that the z_i axes should be chosen to coincide with the bonds. Since consecutive bonds meet at atoms, there is no distance between them. From Figure 3.15c, observe that this makes $a_i = 0$ for all i. From Figure 3.15a, it can be seen that each d_i corresponds to a *bond length*, the distance between consecutive carbon atoms. See Figure 3.20. This leaves two angular parameters, θ_i and α_i. Since the only possible motion of the links is via rotation of the z_i-axes, the angle between two consecutive axes, as shown in Figure 3.15d, must remain constant. In chemistry, this is referred to as the *bond angle* and is represented in the DH parameterization as α_i. The remaining θ_i parameters are the variables that represent the degrees of freedom. However, looking at Figure 3.15b, observe that the example is degenerate because each x_i-axis has no frame of reference because each $a_i = 0$. This does not, however, cause any problems. For visualization purposes, it may be helpful to replace x_{i-1} and x_i by z_{i-1} and z_{i+1}, respectively. This way it is easy to see that as the bond for the z_i-axis is twisted, the observed angle changes accordingly. Each bond is interpreted as a link, $\mathcal{A}_i$. The origin of each $\mathcal{A}_i$ must be chosen to coincide with the intersection point of the z_i- and z_{i+1}-axes. Thus, most of the points in $\mathcal{A}_i$ will lie in the $-z_i$ direction; see Figure 3.20.

The next task is to write down the matrices. Attach a world frame to the first bond, with the second atom at the origin and the bond aligned with the z-axis, in the negative direction; see Figure 3.20. To define T_1, recall that $T_1 = R_1$ from (3.54) because Q_0 is dropped. The parameter d_1 represents the distance between the intersection points of the x_0- and x_1-axes along the z_1 axis. Since there is no x_0-axis, there is freedom to choose d_1; hence, let $d_1 = 0$ to obtain

$$T_1(\theta_1) = R_1(\theta_1) = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.65}$$

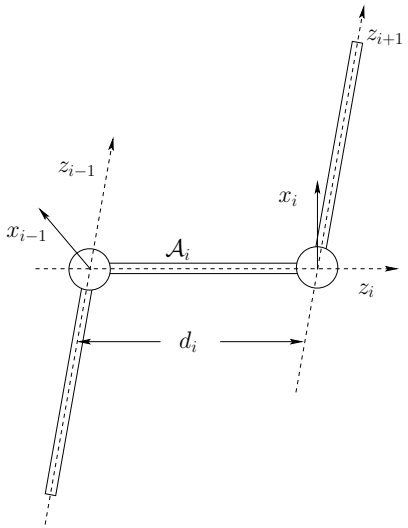


Figure 3.20: Each bond may be interpreted as a "link" of length d_i that is aligned with the z_i-axis. Note that most of $\mathcal{A}_i$ appears in the $-z_i$ direction.

The application of T_1 to points in $\mathcal{A}_1$ causes them to rotate around the z_1-axis, which appears correct.

The matrices for the remaining six bonds are

$$T_i(\theta_i) = \begin{pmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i \cos\alpha_{i-1} & \cos\theta_i \cos\alpha_{i-1} & -\sin\alpha_{i-1} & -\sin\alpha_{i-1} d_i \\ \sin\theta_i \sin\alpha_{i-1} & \cos\theta_i \sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1} d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad (3.66)$$

for $i \in \{2, \ldots, 7\}$. The position of any point, $(x, y, z) \in \mathcal{A}_7$, is given by

$$T_1(\theta_1) T_2(\theta_2) T_3(\theta_3) T_4(\theta_4) T_5(\theta_5) T_6(\theta_6) T_7(\theta_7) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}. \qquad (3.67)$$

■

3.4 Transforming Kinematic Trees

Motivation For many interesting problems, the linkage is arranged in a "tree" as shown in Figure 3.21a. Assume here that the links are not attached in ways that

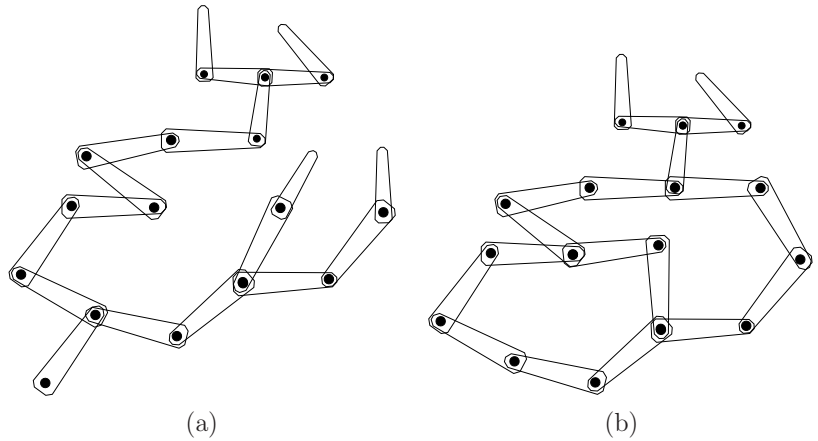


Figure 3.21: General linkages: (a) Instead of a chain of rigid bodies, a "tree" of rigid bodies can be considered. (b) If there are loops, then parameters must be carefully assigned to ensure that the loops are closed.

form loops (i.e., Figure 3.21b); that case is deferred until Section 4.4, although some comments are also made at the end of this section. The human body, with its joints and limbs attached to the torso, is an example that can be modeled as a tree of rigid links. Joints such as knees and elbows are considered as revolute joints. A shoulder joint is an example of a spherical joint, although it cannot achieve any orientation (without a visit to the emergency room!). As mentioned in Section 1.4, there is widespread interest in animating humans in virtual environments and also in developing humanoid robots. Both of these cases rely on formulations of kinematics that mimic the human body.

Another problem that involves kinematic trees is the conformational analysis of molecules. Example 3.5 involved a single chain; however, most organic molecules are more complicated, as in the familiar drugs shown in Figure 1.14a (Section 1.2). The bonds may twist to give degrees of freedom to the molecule. Moving through the space of conformations requires the formulation of a kinematic tree. Studying these conformations is important because scientists need to determine for some candidate drug whether the molecule can twist the right way so that it docks nicely (i.e., requires low energy) with a protein cavity; this induces a pharmacological effect, which hopefully is the desired one. Another important problem is determining how complicated protein molecules fold into certain configurations. These molecules are orders of magnitude larger (in terms of numbers of atoms and degrees of freedom) than typical drug molecules. For more information, see Section 7.5.

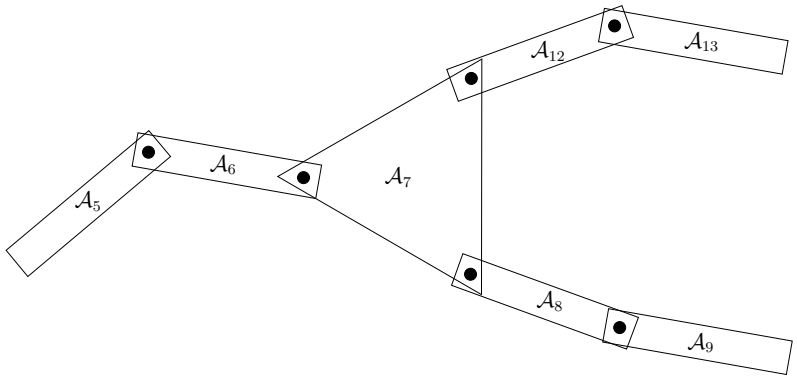


Figure 3.22: Now it is possible for a link to have more than two joints, as in $\mathcal{A}_7$.

Common joints for $\mathcal{W} = \mathbb{R}^2$ First consider the simplest case in which there is a 2D tree of links for which every link has only two points at which revolute joints may be attached. This corresponds to Figure 3.21a. A single link is designated as the *root*, $\mathcal{A}_1$, of the tree. To determine the transformation of a body, $\mathcal{A}_i$, in the tree, the tools from Section 3.3.1 are directly applied to the chain of bodies that connects $\mathcal{A}_i$ to $\mathcal{A}_1$ while ignoring all other bodies. Each link contributes a θ_i to the total degrees of freedom of the tree. This case seems quite straightforward; unfortunately, it is not this easy in general.

Junctions with more than two rotation axes Now consider modeling a more complicated collection of attached links. The main novelty is that one link may have joints attached to it in more than two locations, as in $\mathcal{A}_7$ in Figure 3.22. A link with more than two joints will be referred to as a *junction*.

If there is only one junction, then most of the complications arising from junctions can be avoided by choosing the junction as the root. For example, for a simple humanoid model, the torso would be a junction. It would be sensible to make this the root of the tree, as opposed to the right foot. The legs, arms, and head could all be modeled as independent chains. In each chain, the only concern is that the first link of each chain does not attach to the same point on the torso. This can be solved by inserting a fixed, fictitious link that connects from the origin of the torso to the attachment point of the limb.

The situation is more interesting if there are multiple junctions. Suppose that Figure 3.22 represents part of a 2D system of links for which the root, $\mathcal{A}_1$, is attached via a chain of links to $\mathcal{A}_5$. To transform link $\mathcal{A}_9$, the tools from Section

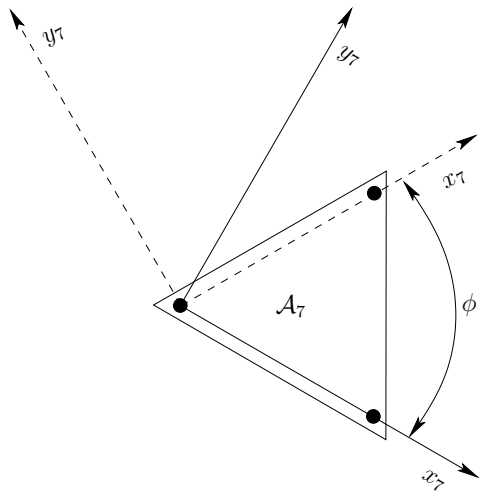


Figure 3.23: The junction is assigned two different frames, depending on which chain was followed. The solid axes were obtained from transforming $\mathcal{A}_9$, and the dashed axes were obtained from transforming $\mathcal{A}_{13}$.

3.3.1 may be directly applied to yield a sequence of transformations,

$$T_1 \cdots T_5 T_6 T_7 T_8 T_9 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.68}$$

for a point $(x, y) \in \mathcal{A}_9$. Likewise, to transform T_{13}, the sequence

$$T_1 \cdots T_5 T_6 T_7 T_{12} T_{13} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{3.69}$$

can be used by ignoring the chain formed by $\mathcal{A}_8$ and $\mathcal{A}_9$. So far everything seems to work well, but take a close look at $\mathcal{A}_7$. As shown in Figure 3.23, its body frame was defined in two different ways, one for each chain. If both are forced to use the same frame, then at least one must abandon the nice conventions of Section 3.3.1 for choosing frames. This situation becomes worse for 3D trees because this would suggest abandoning the DH parameterization. The Khalil-Kleinfinger parameterization is an elegant extension of the DH parameterization and solves these frame assignment issues [272].

Constraining parameters Fortunately, it is fine to use different frames when following different chains; however, one extra piece of information is needed. Imag-

ine transforming the whole tree. The variable θ_7 will appear twice, once from each of the upper and lower chains. Let θ_{7u} and θ_{7l} denote these θ's. Can θ really be chosen two different ways? This would imply that the tree is instead as pictured in Figure 3.24, in which there are two independently moving links, $\mathcal{A}_{7u}$ and $\mathcal{A}_{7l}$. To fix this problem, a constraint must be imposed. Suppose that θ_{7l} is treated as

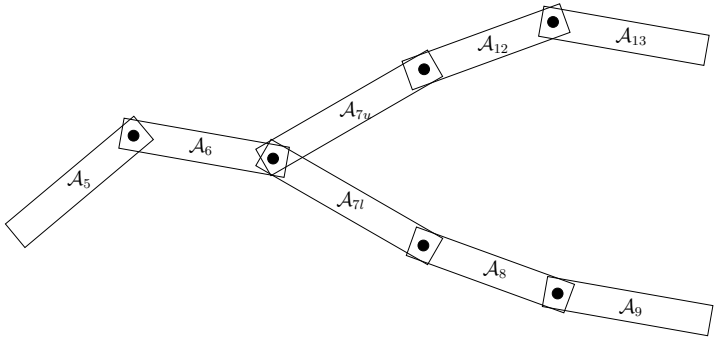


Figure 3.24: Choosing each θ_7 independently would result in a tree that ignores that fact that $\mathcal{A}_7$ is rigid.

an independent variable. The parameter θ_{7u} must then be chosen as $\theta_{7l} + \phi$, in which ϕ is as shown in Figure 3.23.

Example 3.6 (A 2D Tree of Bodies) Figure 3.25 shows a 2D example that involves six links. To transform $(x, y) \in \mathcal{A}_6$, the only relevant links are $\mathcal{A}_5$, $\mathcal{A}_2$, and $\mathcal{A}_1$. The chain of transformations is

$$T_1 T_{2l} T_5 T_6 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.70}$$

in which

$$T_1 = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & x_t \\ \sin\theta_1 & \cos\theta_1 & y_t \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.71}$$

$$T_{2l} = \begin{pmatrix} \cos\theta_{2l} & -\sin\theta_{2l} & a_1 \\ \sin\theta_{2l} & \cos\theta_{2l} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 1 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.72}$$

$$T_5 = \begin{pmatrix} \cos\theta_5 & -\sin\theta_5 & a_2 \\ \sin\theta_5 & \cos\theta_5 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_5 & -\sin\theta_5 & \sqrt{2} \\ \sin\theta_5 & \cos\theta_5 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.73}$$

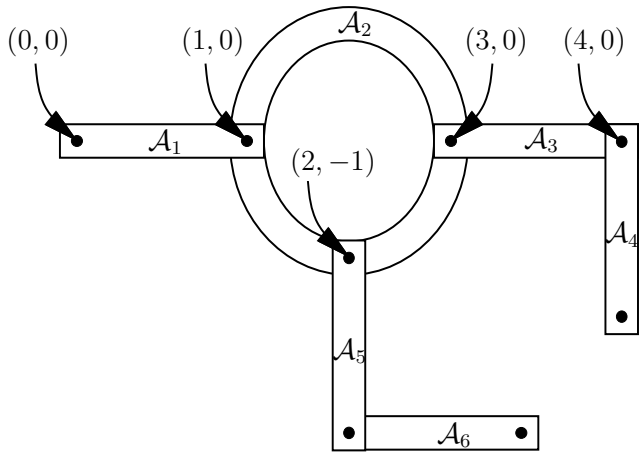


Figure 3.25: A tree of bodies in which the joints are attached in different places.

and

$$T_6 = \begin{pmatrix} \cos\theta_6 & -\sin\theta_6 & a_5 \\ \sin\theta_6 & \cos\theta_6 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_6 & -\sin\theta_6 & 1 \\ \sin\theta_6 & \cos\theta_6 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.74}$$

The matrix T_{2l} in (3.72) denotes the fact that the lower chain was followed. The transformation for points in $\mathcal{A}_4$ is

$$T_1 T_{2u} T_4 T_5 \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{3.75}$$

in which T_1 is the same as in (3.71), and

$$T_3 = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & a_2 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & \sqrt{2} \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.76}$$

and

$$T_4 = \begin{pmatrix} \cos\theta_4 & -\sin\theta_4 & a_4 \\ \sin\theta_4 & \cos\theta_4 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_4 & -\sin\theta_4 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{3.77}$$

The interesting case is

$$T_{2u} = \begin{pmatrix} \cos\theta_{2u} & -\sin\theta_{2u} & a_1 \\ \sin\theta_{2u} & \cos\theta_{2u} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_{2l} + \pi/4) & -\sin(\theta_{2l} + \pi/4) & a_1 \\ \sin(\theta_{2l} + \pi/4) & \cos(\theta_{2l} + \pi/4) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{3.78}$$

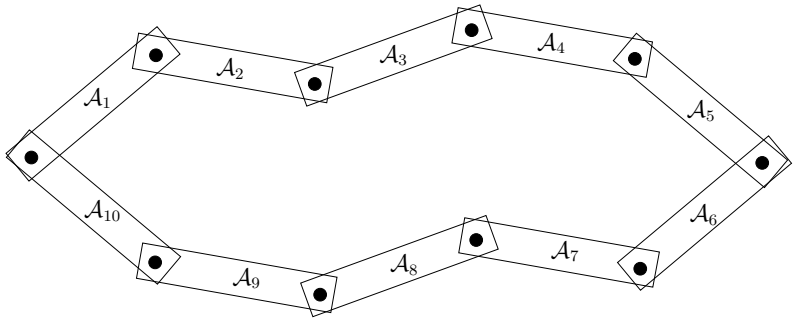


Figure 3.26: There are ten links and ten revolute joints arranged in a loop. This is an example of a closed kinematic chain.

in which the constraint $\theta_{2u} = \theta_{2l} + \pi/4$ is imposed to enforce the fact that $\mathcal{A}_2$ is a junction. ■

For a 3D tree of bodies the same general principles may be followed. In some cases, there will not be any complications that involve special considerations of junctions and constraints. One example of this is the transformation of flexible molecules because all consecutive rotation axes intersect, and junctions occur directly at these points of intersection. In general, however, the DH parameter technique may be applied for each chain, and then the appropriate constraints have to be determined and applied to represent the true degrees of freedom of the tree. The Khalil-Kleinfinger parameterization conveniently captures the resulting solution [272].

What if there are loops? The most general case includes links that are connected in loops, as shown in Figure 3.26. These are generally referred to as *closed kinematic chains*. This arises in many applications. For example, with humanoid robotics or digital actors, a loop is formed when both feet touch the ground. As another example, suppose that two robot manipulators, such as the Puma 560 from Example 3.4, cooperate together to carry an object. If each robot grasps the same object with its hand, then a loop will be formed. A complicated example of this was shown in Figure 1.5, in which mobile robots moved a piano. Outside of robotics, a large fraction of organic molecules have flexible loops. Exploring the space of their conformations requires careful consideration of the difficulties imposed by these loops.

The main difficulty of working with closed kinematic chains is that it is hard to determine which parameter values are within an acceptable range to ensure closure. If these values are given, then the transformations are handled in the

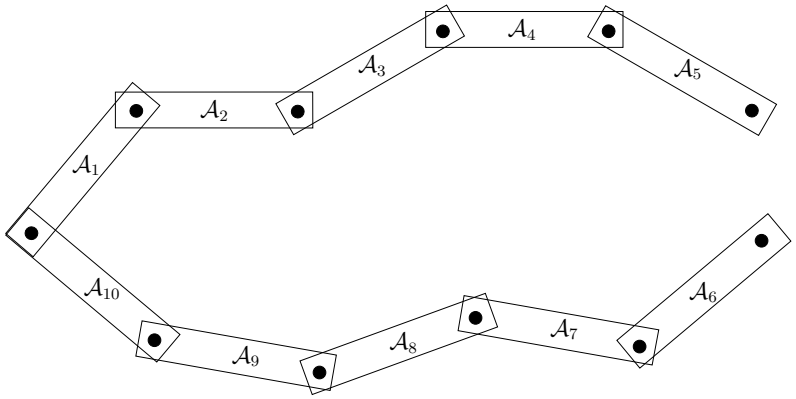


Figure 3.27: Loops may be opened to enable tree-based transformations to be applied; however, a closure constraint must still be satisfied.

same way as the case of trees. For example, the links in Figure 3.26 may be transformed by breaking the loop into two different chains. Suppose we forget that the joint between $\mathcal{A}_5$ and $\mathcal{A}_6$ exists, as shown in Figure 3.27. Consider two different kinematic chains that start at the joint on the extreme left. There is an upper chain from $\mathcal{A}_1$ to $\mathcal{A}_5$ and a lower chain from $\mathcal{A}_{10}$ to $\mathcal{A}_6$. The transformations for any of these bodies can be obtained directly from the techniques of Section 3.3.1. Thus, it is easy to transform the bodies, but how do we choose parameter values that ensure $\mathcal{A}_5$ and $\mathcal{A}_6$ are connected at their common joint? Using the upper chain, the position of this joint may be expressed as

$$T_1(\theta_1)T_2(\theta_2)T_3(\theta_3)T_4(\theta_4)T_5(\theta_5) \begin{pmatrix} a_5 \\ 0 \\ 1 \end{pmatrix}, \tag{3.79}$$

in which $(a_5, 0) \in \mathcal{A}_5$ is the location of the joint of $\mathcal{A}_5$ that is supposed to connect to $\mathcal{A}_6$. The position of this joint may also be expressed using the lower chain as

$$T_{10}(\theta_{10})T_9(\theta_9)T_8(\theta_8)T_7(\theta_7)T_6(\theta_6) \begin{pmatrix} a_6 \\ 0 \\ 1 \end{pmatrix}, \tag{3.80}$$

with $(a_6, 0)$ representing the position of the joint in the body frame of $\mathcal{A}_6$. If the loop does not have to be maintained, then any values for $\theta_1, \ldots, \theta_{10}$ may be selected, resulting in ten degrees of freedom. However, if a loop must maintained,

then (3.79) and (3.80) must be equal,

$$T_1(\theta_1)T_2(\theta_2)T_3(\theta_3)T_4(\theta_4)T_5(\theta_5)\begin{pmatrix} a_5 \\ 0 \\ 1 \end{pmatrix} = T_{10}(\theta_{10})T_9(\theta_9)T_8(\theta_8)T_7(\theta_7)T_6(\theta_6)\begin{pmatrix} a_6 \\ 0 \\ 1 \end{pmatrix}, \tag{3.81}$$

which is quite a mess of nonlinear, trigonometric equations that must be solved. The set of solutions to (3.81) could be very complicated. For the example, the true degrees of freedom is eight because two were removed by making the joint common. Since the common joint allows the links to rotate, exactly two degrees of freedom are lost. If $\mathcal{A}_5$ and $\mathcal{A}_6$ had to be rigidly attached, then the total degrees of freedom would be only seven. For most problems that involve loops, it will not be possible to obtain a nice parameterization of the set of solutions. This a form of the well-known *inverse kinematics problem* [140, 360, 395, 485].

In general, a complicated arrangement of links can be imagined in which there are many loops. Each time a joint along a loop is "ignored," as in the procedure just described, then one less loop exists. This process can be repeated iteratively until there are no more loops in the graph. The resulting arrangement of links will be a tree for which the previous techniques of this section may be applied. However, for each joint that was "ignored" an equation similar to (3.81) must be introduced. All of these equations must be satisfied simultaneously to respect the original loop constraints. Suppose that a set of value parameters is already given. This could happen, for example, using motion capture technology to measure the position and orientation of every part of a human body in contact with the ground. From this the solution parameters could be computed, and all of the transformations are easy to represent. However, as soon as the model moves, it is difficult to ensure that the new transformations respect the closure constraints. The foot of the digital actor may push through the floor, for example. Further information on this problem appears in Section 4.4.

3.5 Nonrigid Transformations

One can easily imagine motion planning for nonrigid bodies. This falls outside of the families of transformations studied so far in this chapter. Several kinds of nonrigid transformations are briefly surveyed here.

Linear transformations A rotation is a special case of a linear transformation, which is generally expressed by an $n \times n$ matrix, M, assuming the transformations are performed over $\mathbb{R}^n$. Consider transforming a point (x, y) in a 2D robot, $\mathcal{A}$, as

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}. \tag{3.82}$$

If M is a rotation matrix, then the size and shape of $\mathcal{A}$ will remain the same. In some applications, however, it may be desirable to distort these. The robot can

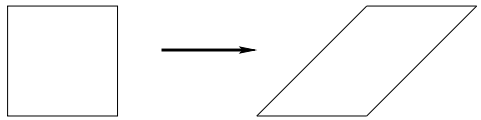


Figure 3.28: Shearing transformations may be performed.

be *scaled* by m_{11} along the x-axis and m_{22} along the y-axis by applying

$$\begin{pmatrix} m_{11} & 0 \\ 0 & m_{22} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}, \tag{3.83}$$

for positive real values m_{11} and m_{22}. If one of them is negated, then a mirror image of $\mathcal{A}$ is obtained. In addition to scaling, $\mathcal{A}$ can be *sheared* by applying

$$\begin{pmatrix} 1 & m_{12} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} \tag{3.84}$$

for $m_{12} \neq 0$. The case of $m_{12} = 1$ is shown in Figure 3.28.

The scaling, shearing, and rotation matrices may be multiplied together to yield a general transformation matrix that explicitly parameterizes each effect. It is also possible to extend the M from $n \times n$ to $(n+1) \times (n+1)$ to obtain a homogeneous transformation matrix that includes translation. Also, the concepts extend in a straightforward way to $\mathbb{R}^3$ and beyond. This enables the additional effects of scaling and shearing to be incorporated directly into the concepts from Sections 3.2-3.4.

Flexible materials In some applications there is motivation to move beyond linear transformations. Imagine trying to warp a flexible material, such as a mattress, through a doorway. The mattress could be approximated by a 2D array of links; however, the complexity and degrees of freedom would be too cumbersome. For another example, suppose that a snake-like robot is designed by connecting 100 revolute joints together in a chain. The tools from Section 3.3 may be used to transform it with 100 rotation parameters, θ_1, ..., θ_{100}, but this may become unwieldy for use in a planning algorithm. An alternative is to approximate the snake with a deformable curve or shape.

For problems such as these, it is desirable to use a parameterized family of curves or surfaces. Spline models are often most appropriate because they are designed to provide easy control over the shape of a curve through the adjustment of a small number of parameters. Other possibilities include the generalized-cylinder and superquadric models that were mentioned in Section 3.1.3.

One complication is that complicated constraints may be imposed on the space of allowable parameters. For example, each joint of a snake-like robot could have a

small range of rotation. This would be easy to model using a kinematic chain; however, determining which splines from a spline family satisfy this extra constraint may be difficult. Likewise, for manipulating flexible materials, there are usually complicated constraints based on the elasticity of the material. Even determining its correct shape under the application of some forces requires integration of an elastic energy function over the material [300].

Further Reading

Section 3.1 barely scratches the surface of geometric modeling. Most literature focuses on parametric curves and surfaces [190, 374, 399]. These models are not as popular for motion planning because obtaining efficient collision detection is most important in practice, and processing implicit algebraic surfaces is most important in theoretical methods. A thorough coverage of solid and boundary representations, including semi-algebraic models, can be found in [234]. Theoretical algorithm issues regarding semi-algebraic models are covered in [369, 370]. For a comparison of the doubly connected edge list to its variants, see [270].

The material of Section 3.2 appears in virtually any book on robotics, computer vision, or computer graphics. Consulting linear algebra texts may be helpful to gain more insight into rotations. There are many ways to parameterize the set of all 3D rotation matrices. The yaw-pitch-roll formulation was selected because it is the easiest to understand. There are generally 12 different variants of the yaw-pitch-roll formulation (also called *Euler angles*) based on different rotation orderings and axis selections. This formulation, however, is not well suited for the development of motion planning algorithms. It is easy (and safe) to use for making quick 3D animations of motion planning output, but it incorrectly captures the structure of the state space for planning algorithms. Section 4.2 introduces the quaternion parameterization, which correctly captures this state space; however, it is harder to interpret when constructing examples. Therefore, it is helpful to understand both. In addition to Euler angles and quaternions, there is still motivation for using many other parameterizations of rotations, such as spherical coordinates, Cayley-Rodrigues parameters, and stereographic projection. Chapter 5 of [113] provides extensive coverage of 3D rotations and different parameterizations.

The coverage in Section 3.3 of transformations of chains of bodies was heavily influenced by two classic robotics texts [140, 395]. The DH parameters were introduced in [223] and later extended to trees and loops in [272]. An alternative to DH parameters is exponential coordinates [378], which simplify some computations; however, determining the parameters in the modeling stage may be less intuitive. A fascinating history of mechanisms appears in [224]. Other texts on kinematics include [23, 163, 277, 359]. The standard approach in many robotics books [184, 432, 447, 485] is to introduce the kinematic chain formulations and DH parameters in the first couple of chapters, and then move on to topics that are crucial for controlling robot manipulators, including dynamics modeling, singularities, manipulability, and control. Since this book is concerned instead with planning algorithms, we depart at the point where dynamics would usually be covered and move into a careful study of the configuration space in Chapter 4.

Exercises

1. Define a semi-algebraic model that removes a triangular "nose" from the region shown in Figure 3.4.

2. For distinct values of yaw, pitch, and roll, it is possible to generate the same rotation. In other words, $R(\alpha, \beta, \gamma) = R(\alpha', \beta', \gamma')$ for some cases in which at least $\alpha \neq \alpha$, $\beta \neq \beta'$, or $\gamma \neq \gamma'$. Characterize the sets of angles for which this occurs.

3. Using rotation matrices, prove that 2D rotation is commutative but 3D rotation is not.

4. An alternative to the yaw-pitch-roll formulation from Section 3.2.3 is considered here. Consider the following Euler angle representation of rotation (there are many other variants). The first rotation is $R_z(\gamma)$, which is just (3.39) with α replaced by γ. The next two rotations are identical to the yaw-pitch-roll formulation: $R_y(\beta)$ is applied, followed by $R_z(\alpha)$. This yields $R_{euler}(\alpha, \beta, \gamma) = R_z(\alpha)R_y(\beta)R_z(\gamma)$.

   (a) Determine the matrix R_{euler}.

   (b) Show that $R_{euler}(\alpha, \beta, \gamma) = R_{euler}(\alpha - \pi, -\beta, \gamma - \pi)$.

   (c) Suppose that a rotation matrix is given as shown in (3.43). Show that the Euler angles are

$$\alpha = \text{atan2}(r_{23}, r_{13}), \tag{3.85}$$

$$\beta = \text{atan2}(\sqrt{1 - r_{33}^2}, r_{33}), \tag{3.86}$$

   and

$$\gamma = \text{atan2}(r_{32}, -r_{31}). \tag{3.87}$$

5. There are 12 different variants of yaw-pitch-roll (or Euler angles), depending on which axes are used and the order of these axes. Determine all of the possibilities, using only notation such as $R_z(\alpha)R_y(\beta)R_z(\gamma)$ for each one. Give brief arguments that support why or why not specific combinations of rotations are included in your list of 12.

6. Let $\mathcal{A}$ be a unit disc, centered at the origin, and $\mathcal{W} = \mathbb{R}^2$. Assume that $\mathcal{A}$ is represented by a single, algebraic primitive, $H = \{(x, y) \mid x^2 + y^2 \leq 1\}$. Show that the transformed primitive is unchanged after any rotation is applied.

7. Consider the articulated chain of bodies shown in Figure 3.29. There are three identical rectangular bars in the plane, called $\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3$. Each bar has width 2 and length 12. The distance between the two points of attachment is 10. The first bar, $\mathcal{A}_1$, is attached to the origin. The second bar, $\mathcal{A}_2$, is attached to $\mathcal{A}_1$, and $\mathcal{A}_3$ is attached to $\mathcal{A}_2$. Each bar is allowed to rotate about its point of attachment. The configuration of the chain can be expressed with three angles,

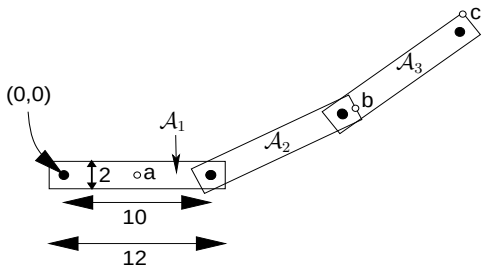


Figure 3.29: A chain of three bodies.

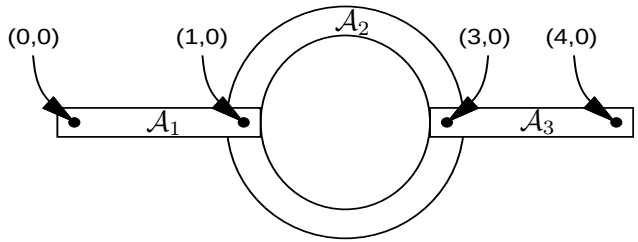


Figure 3.30: Another exercise involving a chain of bodies.

$(\theta_1, \theta_2, \theta_3)$. The first angle, θ_1, represents the angle between the segment drawn between the two points of attachment of $\mathcal{A}_1$ and the x-axis. The second angle, θ_2, represents the angle between $\mathcal{A}_2$ and $\mathcal{A}_1$ ($\theta_2 = 0$ when they are parallel). The third angle, θ_3, represents the angle between $\mathcal{A}_3$ and $\mathcal{A}_2$. Suppose the configuration is $(\pi/4, \pi/2, -\pi/4)$.

(a) Use the homogeneous transformation matrices to determine the locations of points a, b, and c.

(b) Characterize the set of all configurations for which the final point of attachment (near the end of $\mathcal{A}_3$) is at $(0, 0)$ (you should be able to figure this out without using the matrices).

8. A three-link chain of bodies that moves in a 2D world is shown Figure 3.30. The first link, $\mathcal{A}_1$, is attached at $(0, 0)$ but can rotate. Each remaining link is attached to another link with a revolute joint. The second link, $\mathcal{A}_2$, is a rigid ring, and the other two links are rectangular bars.

   Assume that the structure is shown in the zero configuration. Suppose that the linkage is moved to the configuration $(\theta_1, \theta_2, \theta_3) = (\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{4})$, in which θ_1 is the angle of $\mathcal{A}_1$, θ_2 is the angle of $\mathcal{A}_2$ with respect to $\mathcal{A}_1$, and θ_3 is the angle of $\mathcal{A}_3$ with respect to $\mathcal{A}_2$. Using homogeneous transformation matrices, compute the position of the point at $(4, 0)$ in Figure 3.30, when the linkage is at configuration $(\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{4})$ (the point is attached to $\mathcal{A}_3$).

9. Approximate a spherical joint as a chain of three short, perpendicular links that are attached by revolute joints and give the sequence of transformation matrices. Show that as the link lengths approach zero, the resulting sequence of transformation matrices converges to exactly representing the freedom of a spherical joint. Compare this approach to directly using a full rotation matrix, (3.42), to represent the joint in the homogeneous transformation matrix.

10. Figure 3.12 showed six different ways in which 2D surfaces can slide with respect to each other to produce a joint.

    (a) Suppose that two bodies contact each other along a one-dimensional curve. Characterize as many different kinds of "joints" as possible, and indicate the degrees of freedom of each.

    (b) Suppose that the two bodies contact each other at a point. Indicate the types of rolling and sliding that are possible, and their corresponding degrees of freedom.

11. Suppose that two bodies form a screw joint in which the axis of the central axis of the screw aligns with the x-axis of the first body. Determine an appropriate homogeneous transformation matrix to use in place of the DH matrix. Define the matrix with the screw radius, r, and displacement-per-revolution, d, as parameters.

12. Recall Example 3.6. How should the transformations be modified so that the links are in the positions shown in Figure 3.25 at the zero configuration ($\theta_i = 0$ for every revolute joint whose angle can be independently chosen)?

13. Generalize the shearing transformation of (3.84) to enable shearing of 3D models.

Implementations

14. Develop and implement a kinematic model for 2D linkages. Enable the user to display the arrangement of links in the plane.

15. Implement the kinematics of molecules that do not have loops and show them graphically as a "ball and stick" model. The user should be able to input the atomic radii, bond connections, bond lengths, and rotation ranges for each bond.

16. Design and implement a software system in which the user can interactively attach various links to make linkages that resemble those possible from using Tinkertoys (or another popular construction set that allows pieces to move). There are several rods of various lengths, which fit into holes in the center and around the edge of several coin-shaped pieces. Assume that all joints are revolute. The user should be allowed to change parameters and see the resulting positions of all of the links.

17. Construct a model of the human body as a tree of links in a 3D world. For simplicity, the geometric model may be limited to spheres and cylinders. Design and implement a system that displays the virtual human and allows the user to click on joints of the body to enable them to rotate.

18. Develop a simulator with 3D graphics for the Puma 560 model shown in Figure 3.4.

Chapter 4

The Configuration Space

Chapter 3 only covered how to model and transform a collection of bodies; however, for the purposes of planning it is important to define the state space. The state space for motion planning is a set of possible transformations that could be applied to the robot. This will be referred to as the *configuration space*, based on Lagrangian mechanics and the seminal work of Lozano-Pérez [343, 347, 344], who extensively utilized this notion in the context of planning (the idea was also used in early collision avoidance work by Udupa [462]). The motion planning literature was further unified around this concept by Latombe's book [304]. Once the configuration space is clearly understood, many motion planning problems that appear different in terms of geometry and kinematics can be solved by the same planning algorithms. This level of abstraction is therefore very important.

This chapter provides important foundational material that will be very useful in Chapters 5 to 8 and other places where planning over continuous state spaces occurs. Many concepts introduced in this chapter come directly from mathematics, particularly from topology. Therefore, Section 4.1 gives a basic overview of topological concepts. Section 4.2 uses the concepts from Chapter 3 to define the configuration space. After reading this, you should be able to precisely characterize the configuration space of a robot and understand its structure. In Section 4.3, obstacles in the world are transformed into obstacles in the configuration space, but it is important to understand that this transformation may not be explicitly constructed. The implicit representation of the state space is a recurring theme throughout planning. Section 4.4 covers the important case of kinematic chains that have loops, which was mentioned in Section 3.4. This case is so difficult that even the space of transformations usually cannot be explicitly characterized (i.e., parameterized).

4.1 Basic Topological Concepts

This section introduces basic topological concepts that are helpful in understanding configuration spaces. Topology is a challenging subject to understand in depth.

The treatment given here provides only a brief overview and is designed to stimulate further study (see the literature overview at the end of the chapter). To advance further in this chapter, it is not necessary to understand all of the material of this section; however, the more you understand, the deeper will be your understanding of motion planning in general.

4.1.1 Topological Spaces

Recall the concepts of open and closed intervals in the set of real numbers $\mathbb{R}$. The open interval $(0, 1)$ includes all real numbers between 0 and 1, *except* 0 and 1. However, for either endpoint, an infinite sequence may be defined that converges to it. For example, the sequence $1/2$, $1/4$, ..., $1/2^i$ converges to 0 as i tends to infinity. This means that we can choose a point in $(0, 1)$ within any small, positive distance from 0 or 1, but we cannot pick one exactly on the boundary of the interval. For a closed interval, such as $[0, 1]$, the boundary points are included.

The notion of an open set lies at the heart of topology. The open set definition that will appear here is a substantial generalization of the concept of an open interval. The concept applies to a very general collection of subsets of some larger space. It is general enough to easily include any kind of configuration space that may be encountered in planning.

A set X is called a *topological space* if there is a collection of subsets of X called *open sets* for which the following axioms hold:

1. The union of any number of open sets is an open set.
2. The intersection of a finite number of open sets is an open set.
3. Both X and $\emptyset$ are open sets.

Note that in the first axiom, the union of an infinite number of open sets may be taken, and the result must remain an open set. Intersecting an infinite number of open sets, however, does not necessarily lead to an open set.

For the special case of $X = \mathbb{R}$, the open sets include open intervals, as expected. Many sets that are not intervals are open sets because taking unions and intersections of open intervals yields other open sets. For example, the set

$$\bigcup_{i=1}^{\infty} \left(\frac{1}{3^i}, \frac{2}{3^i} \right), \tag{4.1}$$

which is an infinite union of pairwise-disjoint intervals, is an open set.

Closed sets Open sets appear directly in the definition of a topological space. It next seems that closed sets are needed. Suppose X is a topological space. A subset $C \subset X$ is defined to be a *closed set* if and only if $X \setminus C$ is an open set. Thus, the complement of any open set is closed, and the complement of any

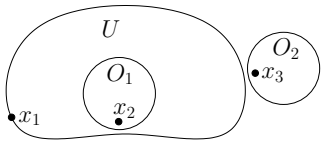


Figure 4.1: An illustration of the boundary definition. Suppose $X = \mathbb{R}^2$, and U is a subset as shown. Three kinds of points appear: 1) x_1 is a boundary point, 2) x_2 is an interior point, and 3) x_3 is an exterior point. Both x_1 and x_2 are limit points of U.

closed set is open. Any closed interval, such as $[0, 1]$, is a closed set because its complement, $(-\infty, 0) \cup (1, \infty)$, is an open set. For another example, $(0, 1)$ is an open set; therefore, $\mathbb{R} \setminus (0, 1) = (-\infty, 0] \cup [1, \infty)$ is a closed set. The use of "(" may seem wrong in the last expression, but "[" cannot be used because $-\infty$ and ∞ do not belong to $\mathbb{R}$. Thus, the use of "(" is just a notational quirk.

Are all subsets of X either closed or open? Although it appears that open sets and closed sets are opposites in some sense, the answer is *no*. For $X = \mathbb{R}$, the interval $[0, 2\pi)$ is neither open nor closed (consider its complement: $[2\pi, \infty)$ is closed, and $(-\infty, 0)$ is open). Note that for any topological space, X and $\emptyset$ are both open and closed!

Special points From the definitions and examples so far, it should seem that points on the "edge" or "border" of a set are important. There are several terms that capture where points are relative to the border. Let X be a topological space, and let U be any subset of X. Furthermore, let x be any point in X. The following terms capture the position of point x relative to U (see Figure 4.1):

- If there exists an open set O_1 such that $x \in O_1$ and $O_1 \subseteq U$, then x is called an *interior point* of U. The set of all interior points in U is called the *interior of* U and is denoted by $\text{int}(U)$.

- If there exists an open set O_2 such that $x \in O_2$ and $O_2 \subseteq X \setminus U$, then x is called an *exterior point* with respect to U.

- If x is neither an interior point nor an exterior point, then it is called a *boundary point* of U. The set of all boundary points in X is called the *boundary of* U and is denoted by ∂U.

- All points in $x \in X$ must be one of the three above; however, another term is often used, even though it is redundant given the other three. If x is either an interior point or a boundary point, then it is called a *limit point* (or *accumulation point*) of U. The set of all limit points of U is a closed set called the *closure* of U, and it is denoted by $\text{cl}(U)$. Note that $\text{cl}(U) = \text{int}(U) \cup \partial U$.

For the case of $X = \mathbb{R}$, the boundary points are the endpoints of intervals. For example, 0 and 1 are boundary points of intervals, $(0, 1)$, $[0, 1]$, $[0, 1)$, and $(0, 1]$. Thus, U may or may not include its boundary points. All of the points in $(0, 1)$ are interior points, and all of the points in $[0, 1]$ are limit points. The motivation of the name "limit point" comes from the fact that such a point might be the limit of an infinite sequence of points in U. For example, 0 is the limit point of the sequence generated by $1/2^i$ for each $i \in \mathbb{N}$, the natural numbers.

There are several convenient consequences of the definitions. A closed set C contains the limit point of any sequence that is a subset of C. This implies that it contains all of its boundary points. The closure, cl, always results in a closed set because it adds all of the boundary points to the set. On the other hand, an open set contains none of its boundary points. These interpretations will come in handy when considering obstacles in the configuration space for motion planning.

Some examples The definition of a topological space is so general that an incredible variety of topological spaces can be constructed.

Example 4.1 (The Topology of $\mathbb{R}^n$) We should expect that $X = \mathbb{R}^n$ for any integer n is a topological space. This requires characterizing the open sets. An *open ball* $B(x, \rho)$ is the set of points in the interior of a sphere of radius ρ, centered at x. Thus,

$$B(x, \rho) = \{x' \in \mathbb{R}^n \mid \|x' - x\| < \rho\}, \tag{4.2}$$

in which $\| \cdot \|$ denotes the Euclidean norm (or magnitude) of its argument. The open balls are open sets in $\mathbb{R}^n$. Furthermore, all other open sets can be expressed as a countable union of open balls.[1] For the case of $\mathbb{R}$, this reduces to representing any open set as a union of intervals, which was done so far.

Even though it is possible to express open sets of $\mathbb{R}^n$ as unions of balls, we prefer to use other representations, with the understanding that one could revert to open balls if necessary. The primitives of Section 3.1 can be used to generate many interesting open and closed sets. For example, any algebraic primitive expressed in the form $H = \{x \in \mathbb{R}^n \mid f(x) \leq 0\}$ produces a closed set. Taking finite unions and intersections of these primitives will produce more closed sets. Therefore, all of the models from Sections 3.1.1 and 3.1.2 produce an obstacle region $\mathcal{O}$ that is a closed set. As mentioned in Section 3.1.2, sets constructed only from primitives that use the $<$ relation are open. ■

Example 4.2 (Subspace Topology) A new topological space can easily be constructed from a subset of a topological space. Let X be a topological space, and let $Y \subset X$ be a subset. The *subspace topology* on Y is obtained by defining the open sets to be every subset of Y that can be represented as $U \cap Y$ for some open set $U \subseteq X$. Thus, the open sets for Y are almost the same as for X, except

[1] Such a collection of balls is often referred to as a *basis*.

that the points that do not lie in Y are trimmed away. New subspaces can be constructed by intersecting open sets of $\mathbb{R}^n$ with a complicated region defined by semi-algebraic models. This leads to many interesting topological spaces, some of which will appear later in this chapter. ■

Example 4.3 (The Trivial Topology) For any set X, there is always one trivial example of a topological space that can be constructed from it. Declare that X and $\emptyset$ are the only open sets. Note that all of the axioms are satisfied. ■

Example 4.4 (A Strange Topology) It is important to keep in mind the almost absurd level of generality that is allowed by the definition of a topological space. A topological space can be defined for any set, as long as the declared open sets obey the axioms. Suppose a four-element set is defined as

$$X = \{\text{CAT}, \text{DOG}, \text{TREE}, \text{HOUSE}\}. \tag{4.3}$$

In addition to $\emptyset$ and X, suppose that $\{\text{CAT}\}$ and $\{\text{DOG}\}$ are open sets. Using the axioms, $\{\text{CAT}, \text{DOG}\}$ must also be an open set. Closed sets and boundary points can be derived for this topology once the open sets are defined. ■

After the last example, it seems that topological spaces are so general that not much can be said about them. Most spaces that are considered in topology and analysis satisfy more axioms. For $\mathbb{R}^n$ and any configuration spaces that arise in this book, the following is satisfied:

Hausdorff axiom: For any distinct $x_1, x_2 \in X$, there exist open sets O_1 and O_2 such that $x_1 \in O_1$, $x_2 \in O_2$, and $O_1 \cap O_2 = \emptyset$.

In other words, it is possible to separate x_1 and x_2 into nonoverlapping open sets. Think about how to do this for $\mathbb{R}^n$ by selecting small enough open balls. Any topological space X that satisfies the Hausdorff axiom is referred to as a *Hausdorff space*. Section 4.1.2 will introduce manifolds, which happen to be Hausdorff spaces and are general enough to capture the vast majority of configuration spaces that arise. We will have no need in this book to consider topological spaces that are not Hausdorff spaces.

Continuous functions A very simple definition of continuity exists for topological spaces. It nicely generalizes the definition from standard calculus. Let $f : X \to Y$ denote a function between topological spaces X and Y. For any set $B \subseteq Y$, let the *preimage* of B be denoted and defined by

$$f^{-1}(B) = \{x \in X \mid f(x) \in B\}. \tag{4.4}$$

Note that this definition does not require f to have an inverse.

The function f is called *continuous* if $f^{-1}(O)$ is an open set for every open set $O \subseteq Y$. Analysis is greatly simplified by this definition of continuity. For example, to show that any composition of continuous functions is continuous requires only a one-line argument that the preimage of the preimage of any open set always yields an open set. Compare this to the cumbersome classical proof that requires a mess of δ's and ϵ's. The notion is also so general that continuous functions can even be defined on the absurd topological space from Example 4.4.

Homeomorphism: Making a donut into a coffee cup You might have heard the expression that to a topologist, a donut and a coffee cup appear the same. In many branches of mathematics, it is important to define when two basic objects are equivalent. In graph theory (and group theory), this equivalence relation is called an *isomorphism*. In topology, the most basic equivalence is a homeomorphism, which allows spaces that appear quite different in most other subjects to be declared equivalent in topology. The surfaces of a donut and a coffee cup (with one handle) are considered equivalent because both have a single hole. This notion needs to be made more precise!

Suppose $f : X \to Y$ is a bijective (one-to-one and onto) function between topological spaces X and Y. Since f is bijective, the inverse f^{-1} exists. If both f and f^{-1} are continuous, then f is called a *homeomorphism*. Two topological spaces X and Y are said to be *homeomorphic*, denoted by $X \cong Y$, if there exists a homeomorphism between them. This implies an equivalence relation on the set of topological spaces (verify that the reflexive, symmetric, and transitive properties are implied by the homeomorphism).

Example 4.5 (Interval Homeomorphisms) Any open interval of $\mathbb{R}$ is homeomorphic to any other open interval. For example, $(0, 1)$ can be mapped to $(0, 5)$ by the continuous mapping $x \mapsto 5x$. Note that $(0, 1)$ and $(0, 5)$ are each being interpreted here as topological subspaces of $\mathbb{R}$. This kind of homeomorphism can be generalized substantially using linear algebra. If a subset, $X \subset \mathbb{R}^n$, can be mapped to another, $Y \subset \mathbb{R}^n$, via a nonsingular linear transformation, then X and Y are homeomorphic. For example, the rigid-body transformations of the previous chapter were examples of homeomorphisms applied to the robot. Thus, the topology of the robot does not change when it is translated or rotated. (In this example, note that the robot itself is the topological space. This will not be the case for the rest of the chapter.)

Be careful when mixing closed and open sets. The space $[0, 1]$ is not homeomorphic to $(0, 1)$, and neither is homeomorphic to $[0, 1)$. The endpoints cause trouble when trying to make a bijective, continuous function. Surprisingly, a bounded and unbounded set may be homeomorphic. A subset X of $\mathbb{R}^n$ is called *bounded* if there exists a ball $B \subset \mathbb{R}^n$ such that $X \subset B$. The mapping $x \mapsto 1/x$ establishes that $(0, 1)$ and $(1, \infty)$ are homeomorphic. The mapping $x \mapsto 2 \tan^{-1}(x)/\pi$ establishes that $(-1, 1)$ and all of $\mathbb{R}$ are homeomorphic! ■

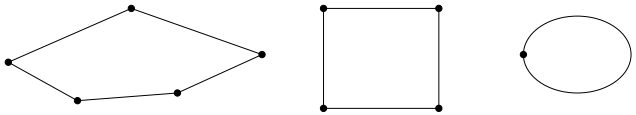


Figure 4.2: Even though the graphs are not isomorphic, the corresponding topological spaces may be homeomorphic due to useless vertices. The example graphs map into $\mathbb{R}^2$, and are all homeomorphic to a circle.

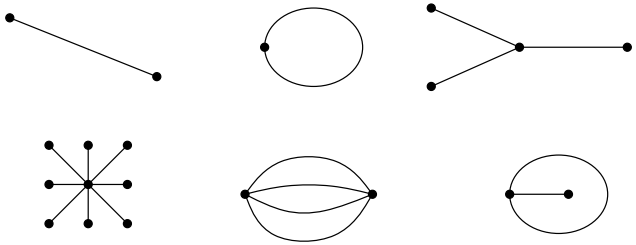


Figure 4.3: These topological graphs map into subsets of $\mathbb{R}^2$ that are not homeomorphic to each other.

Example 4.6 (Topological Graphs) Let X be a topological space. The previous example can be extended nicely to make homeomorphisms look like graph isomorphisms. Let a *topological graph*[2] be a graph for which every vertex corresponds to a point in X and every edge corresponds to a continuous, injective (one-to-one) function, $\tau : [0, 1] \rightarrow X$. The image of τ connects the points in X that correspond to the endpoints (vertices) of the edge. The images of different edge functions are not allowed to intersect, except at vertices. Recall from graph theory that two graphs, $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, are called *isomorphic* if there exists a bijective mapping, $f : V_1 \rightarrow V_2$ such that there is an edge between v_1 and v_1' in G_1, if and only if there exists an edge between $f(v_1)$ and $f(v_1')$ in G_2.

The bijective mapping used in the graph isomorphism can be extended to produce a homeomorphism. Each edge in E_1 is mapped continuously to its corresponding edge in E_2. The mappings nicely coincide at the vertices. Now you should see that two topological graphs are homeomorphic if they are isomorphic under the standard definition from graph theory.[3] What if the graphs are not isomorphic? There is still a chance that the topological graphs may be homeomorphic, as shown in Figure 4.2. The problem is that there appear to be "useless" vertices in the graph. By removing vertices of degree two that can be deleted without affecting the connectivity of the graph, the problem is fixed. In this case, graphs that are not isomorphic produce topological graphs that are not homeomorphic. This allows many distinct, interesting topological spaces to be constructed. A few are shown in Figure 4.3. ■

[2] In topology this is called a 1-complex [226].

[3] Technically, the images of the topological graphs, as subspaces of X, are homeomorphic, not the graphs themselves.

4.1.2 Manifolds

In motion planning, efforts are made to ensure that the resulting configuration space has nice properties that reflect the true structure of the space of transformations. One important kind of topological space, which is general enough to include most of the configuration spaces considered in Part II, is called a manifold. Intuitively, a manifold can be considered as a "nice" topological space that behaves at every point like our intuitive notion of a surface.

Manifold definition A topological space $M \subseteq \mathbb{R}^m$ is a *manifold*[4] if for every $x \in M$, an open set $O \subset M$ exists such that: 1) $x \in O$, 2) O is homeomorphic to $\mathbb{R}^n$, and 3) n is fixed for all $x \in M$. The fixed n is referred to as the *dimension* of the manifold, M. The second condition is the most important. It states that in the vicinity of any point, $x \in M$, the space behaves just like it would in the vicinity of any point $y \in \mathbb{R}^n$; intuitively, the set of directions that one can move appears the same in either case. Several simple examples that may or may not be manifolds are shown in Figure 4.4.

One natural consequence of the definitions is that $m \geq n$. According to Whitney's embedding theorem [231], $m \leq 2n + 1$. In other words, $\mathbb{R}^{2n+1}$ is "big enough" to hold any n-dimensional manifold.[5] Technically, it is said that the n-dimensional manifold M is *embedded* in $\mathbb{R}^m$, which means that an injective mapping exists from M to $\mathbb{R}^m$ (if it is not injective, then the topology of M could change).

As it stands, it is impossible for a manifold to include its boundary points because they are not contained in open sets. A *manifold with boundary* can be

[4] Manifolds that are not subsets of $\mathbb{R}^m$ may also be defined. This requires that M is a Hausdorff space and is second countable, which means that there is a countable number of open sets from which any other open set can be constructed by taking a union of some of them. These conditions are automatically satisfied when assuming $M \subseteq \mathbb{R}^m$; thus, it avoids these extra complications and is still general enough for our purposes. Some authors use the term *manifold* to refer to a *smooth manifold*. This requires the definition of a smooth structure, and the homeomorphism is replaced by diffeomorphism. This extra structure is not needed here but will be introduced when it is needed in Section 8.3.

[5] One variant of the theorem is that for smooth manifolds, $\mathbb{R}^{2n}$ is sufficient. This bound is tight because $\mathbb{RP}^n$ (n-dimensional projective space, which will be introduced later in this section), cannot be embedded in $\mathbb{R}^{2n-1}$.

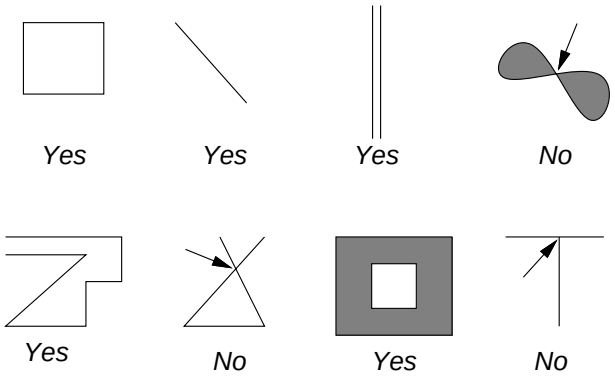


Figure 4.4: Some subsets of $\mathbb{R}^2$ that may or may not be manifolds. For the three that are not, the point that prevents them from being manifolds is indicated.

defined requiring that the neighborhood of each boundary point of M is homeomorphic to a half-space of dimension n (which was defined for $n = 2$ and $n = 3$ in Section 3.1) and that the interior points must be homeomorphic to $\mathbb{R}^n$.

The presentation now turns to ways of constructing some manifolds that frequently appear in motion planning. It is important to keep in mind that two manifolds will be considered equivalent if they are homeomorphic (recall the donut and coffee cup).

Cartesian products There is a convenient way to construct new topological spaces from existing ones. Suppose that X and Y are topological spaces. The *Cartesian product*, $X \times Y$, defines a new topological space as follows. Every $x \in X$ and $y \in Y$ generates a point (x, y) in $X \times Y$. Each open set in $X \times Y$ is formed by taking the Cartesian product of one open set from X and one from Y. Exactly one open set exists in $X \times Y$ for every pair of open sets that can be formed by taking one from X and one from Y. Furthermore, these new open sets are used as a basis for forming the remaining open sets of $X \times Y$ by allowing any unions and finite intersections of them.

A familiar example of a Cartesian product is $\mathbb{R} \times \mathbb{R}$, which is equivalent to $\mathbb{R}^2$. In general, $\mathbb{R}^n$ is equivalent to $\mathbb{R} \times \mathbb{R}^{n-1}$. The Cartesian product can be taken over many spaces at once. For example, $\mathbb{R} \times \mathbb{R} \times \cdots \times \mathbb{R} = \mathbb{R}^n$. In the coming text, many important manifolds will be constructed via Cartesian products.

1D manifolds The set $\mathbb{R}$ of reals is the most obvious example of a 1D manifold because $\mathbb{R}$ certainly looks like (via homeomorphism) $\mathbb{R}$ in the vicinity of every point. The range can be restricted to the unit interval to yield the manifold $(0, 1)$ because they are homeomorphic (recall Example 4.5).

Another 1D manifold, which is not homeomorphic to $(0, 1)$, is a circle, $\mathbb{S}^1$. In this case $\mathbb{R}^m = \mathbb{R}^2$, and let

$$\mathbb{S}^1 = \{(x, y) \in \mathbb{R}^2 \mid x^2 + y^2 = 1\}. \tag{4.5}$$

If you are thinking like a topologist, it should appear that this particular circle is not important because there are numerous ways to define manifolds that are homeomorphic to $\mathbb{S}^1$. For any manifold that is homeomorphic to $\mathbb{S}^1$, we will sometimes say that the manifold *is* $\mathbb{S}^1$, just represented in a different way. Also, $\mathbb{S}^1$ will be called a *circle*, but this is meant only in the topological sense; it only needs to be homeomorphic to the circle that we learned about in high school geometry. Also, when referring to $\mathbb{R}$, we might instead substitute $(0, 1)$ without any trouble. The alternative representations of a manifold can be considered as changing *parameterizations*, which are formally introduced in Section 8.3.2.

Identifications A convenient way to represent $\mathbb{S}^1$ is obtained by *identification*, which is a general method of declaring that some points of a space are identical, even though they originally were distinct.[6] For a topological space X, let $X/\sim$ denote that X has been redefined through some form of identification. The open sets of X become redefined. Using identification, $\mathbb{S}^1$ can be defined as $[0, 1]/\sim$, in which the identification declares that 0 and 1 are equivalent, denoted as $0 \sim 1$. This has the effect of "gluing" the ends of the interval together, forming a closed loop. To see the homeomorphism that makes this possible, use polar coordinates to obtain $\theta \mapsto (\cos 2\pi\theta, \sin 2\pi\theta)$. You should already be familiar with 0 and 2π leading to the same point in polar coordinates; here they are just normalized to 0 and 1. Letting θ run from 0 up to 1, and then "wrapping around" to 0 is a convenient way to represent $\mathbb{S}^1$ because it does not need to be curved as in (4.5).

It might appear that identifications are cheating because the definition of a manifold requires it to be a subset of $\mathbb{R}^m$. This is not a problem because Whitney's theorem, as mentioned previously, states that any n-dimensional manifold can be embedded in $\mathbb{R}^{2n+1}$. The identifications just reduce the number of dimensions needed for visualization. They are also convenient in the implementation of motion planning algorithms.

2D manifolds Many important, 2D manifolds can be defined by applying the Cartesian product to 1D manifolds. The 2D manifold $\mathbb{R}^2$ is formed by $\mathbb{R} \times \mathbb{R}$. The product $\mathbb{R} \times \mathbb{S}^1$ defines a manifold that is equivalent to an infinite cylinder. The product $\mathbb{S}^1 \times \mathbb{S}^1$ is a manifold that is equivalent to a torus (the surface of a donut).

Can any other 2D manifolds be defined? See Figure 4.5. The identification idea can be applied to generate several new manifolds. Start with an open square $M = (0, 1) \times (0, 1)$, which is homeomorphic to $\mathbb{R}^2$. Let (x, y) denote a point in

[6] This is usually defined more formally and called a *quotient topology*.

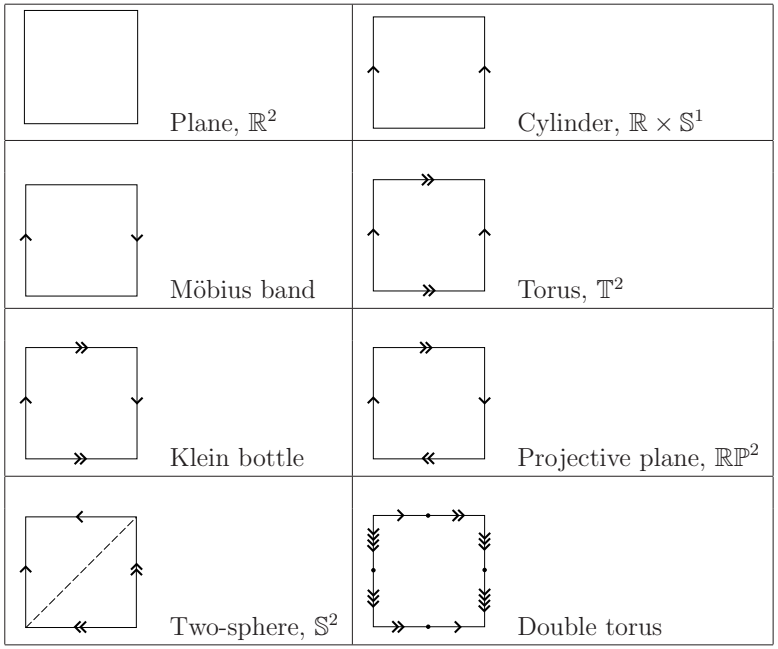


Figure 4.5: Some 2D manifolds that can be obtained by identifying pairs of points along the boundary of a square region.

the plane. A *flat cylinder* is obtained by making the identification $(0, y) \sim (1, y)$ for all $y \in (0, 1)$ and adding all of these points to M. The result is depicted in Figure 4.5 by drawing arrows where the identification occurs.

A *Möbius band* can be constructed by taking a strip of paper and connecting the ends after making a 180-degree twist. This result is not homeomorphic to the cylinder. The Möbius band can also be constructed by putting the twist into the identification, as $(0, y) \sim (1, 1 - y)$ for all $y \in (0, 1)$. In this case, the arrows are drawn in opposite directions. The Möbius band has the famous properties that it has only one side (trace along the paper strip with a pencil, and you will visit both sides of the paper) and is nonorientable (if you try to draw it in the plane, without using identification tricks, it will always have a twist).

For all of the cases so far, there has been a boundary to the set. The next few manifolds will not even have a boundary, even though they may be bounded. If you were to live in one of them, it means that you could walk forever along any trajectory and never encounter the edge of your universe. It might seem like our physical universe is unbounded, but it would only be an illusion. Furthermore, there are several distinct possibilities for the universe that are not homeomorphic to each other. In higher dimensions, such possibilities are the subject of cosmology,

which is a branch of astrophysics that uses topology to characterize the structure of our universe.

A *torus* can be constructed by performing identifications of the form $(0, y) \sim (1, y)$, which was done for the cylinder, and also $(x, 0) \sim (x, 1)$, which identifies the top and bottom. Note that the point $(0, 0)$ must be included and is identified with three other points. Double arrows are used in Figure 4.5 to indicate the top and bottom identification. All of the identification points must be added to M. Note that there are no twists. A funny interpretation of the resulting *flat torus* is as the universe appears for a spacecraft in some 1980s-style *Asteroids*-like video games. The spaceship flies off of the screen in one direction and appears somewhere else, as prescribed by the identification.

Two interesting manifolds can be made by adding twists. Consider performing all of the identifications that were made for the torus, except put a twist in the side identification, as was done for the Möbius band. This yields a fascinating manifold called the *Klein bottle*, which can be embedded in $\mathbb{R}^4$ as a closed 2D surface in which the inside and the outside are the same! (This is in a sense similar to that of the Möbius band.) Now suppose there are twists in both the sides and the top and bottom. This results in the most bizarre manifold yet: the real projective plane, $\mathbb{RP}^2$. This space is equivalent to the set of all lines in $\mathbb{R}^3$ that pass through the origin. The 3D version, $\mathbb{RP}^3$, happens to be one of the most important manifolds for motion planning!

Let $\mathbb{S}^2$ denote the unit sphere, which is defined as

$$\mathbb{S}^2 = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = 1\}. \tag{4.6}$$

Another way to represent $\mathbb{S}^2$ is by making the identifications shown in the last row of Figure 4.5. A dashed line is indicated where the equator might appear, if we wanted to make a distorted wall map of the earth. The poles would be at the upper left and lower right corners. The final example shown in Figure 4.5 is a *double torus*, which is the surface of a two-holed donut.

Higher dimensional manifolds The construction techniques used for the 2D manifolds generalize nicely to higher dimensions. Of course, $\mathbb{R}^n$, is an n-dimensional manifold. An n-dimensional torus, $\mathbb{T}^n$, can be made by taking a Cartesian product of n copies of $\mathbb{S}^1$. Note that $\mathbb{S}^1 \times \mathbb{S}^1 \neq \mathbb{S}^2$. Therefore, the notation $\mathbb{T}^n$ is used for $(\mathbb{S}^1)^n$. Different kinds of n-dimensional cylinders can be made by forming a Cartesian product $\mathbb{R}^i \times \mathbb{T}^j$ for positive integers i and j such that $i + j = n$. Higher dimensional spheres are defined as

$$\mathbb{S}^n = \{x \in \mathbb{R}^{n+1} \mid \|x\| = 1\}, \tag{4.7}$$

in which $\|x\|$ denotes the Euclidean norm of x, and n is a positive integer. Many interesting spaces can be made by identifying faces of the cube $(0, 1)^n$ (or even faces of a polyhedron or polytope), especially if different kinds of twists are allowed. An

n-dimensional projective space can be defined in this way, for example. *Lens spaces* are a family of manifolds that can be constructed by identification of polyhedral faces [419].

Due to its coming importance in motion planning, more details are given on projective spaces. The standard definition of an *n-dimensional real projective space* $\mathbb{RP}^n$ is the set of all lines in $\mathbb{R}^{n+1}$ that pass through the origin. Each line is considered as a point in $\mathbb{RP}^n$. Using the definition of $\mathbb{S}^n$ in (4.7), note that each of these lines in $\mathbb{R}^{n+1}$ intersects $\mathbb{S}^n \subset \mathbb{R}^{n+1}$ in exactly two places. These intersection points are called *antipodal*, which means that they are as far from each other as possible on $\mathbb{S}^n$. The pair is also unique for each line. If we identify all pairs of antipodal points of $\mathbb{S}^n$, a homeomorphism can be defined between each line through the origin of $\mathbb{R}^{n+1}$ and each antipodal pair on the sphere. This means that the resulting manifold, $\mathbb{S}^n / \sim$, is homeomorphic to $\mathbb{RP}^n$.

Another way to interpret the identification is that $\mathbb{RP}^n$ is just the upper half of $\mathbb{S}^n$, but with every equatorial point identified with its antipodal point. Thus, if you try to walk into the southern hemisphere, you will find yourself on the other side of the world walking north. It is helpful to visualize the special case of $\mathbb{RP}^2$ and the upper half of $\mathbb{S}^2$. Imagine warping the picture of $\mathbb{RP}^2$ from Figure 4.5 from a square into a circular disc, with opposite points identified. The result still represents $\mathbb{RP}^2$. The center of the disc can now be lifted out of the plane to form the upper half of $\mathbb{S}^2$.

4.1.3 Paths and Connectivity

Central to motion planning is determining whether one part of a space is reachable from another. In Chapter 2, one state was reached from another by applying a sequence of actions. For motion planning, the analog to this is connecting one point in the configuration space to another by a continuous path. Graph connectivity is important in the discrete planning case. An analog to this for topological spaces is presented in this section.

Paths Let X be a topological space, which for our purposes will also be a manifold. A *path* is a continuous function, $\tau : [0,1] \rightarrow X$. Alternatively, $\mathbb{R}$ may be used for the domain of τ. Keep in mind that a path is a function, not a set of points. Each point along the path is given by $\tau(s)$ for some $s \in [0,1]$. This makes it appear as a nice generalization to the sequence of states visited when a plan from Chapter 2 is applied. Recall that there, a countable set of stages was defined, and the states visited could be represented as x_1, x_2, ... In the current setting $\tau(s)$ is used, in which s replaces the stage index. To make the connection clearer, we could use x instead of τ to obtain $x(s)$ for each $s \in [0,1]$.

Connected vs. path connected A topological space X is said to be *connected* if it cannot be represented as the union of two disjoint, nonempty, open sets. While

this definition is rather elegant and general, if X is connected, it does not imply that a path exists between any pair of points in X thanks to crazy examples like the *topologist's sine curve*:

$$X = \{(x, y) \in \mathbb{R}^2 \mid x = 0 \text{ or } y = \sin(1/x)\}. \tag{4.8}$$

Consider plotting X. The $\sin(1/x)$ part creates oscillations near the y-axis in which the frequency tends to infinity. After union is taken with the y-axis, this space is connected, but there is no path that reaches the y-axis from the sine curve.

How can we avoid such problems? The standard way to fix this is to use the path definition directly in the definition of connectedness. A topological space X is said to be *path connected* if for all $x_1, x_2 \in X$, there exists a path τ such that $\tau(0) = x_1$ and $\tau(1) = x_2$. It can be shown that if X is path connected, then it is also connected in the sense defined previously.

Another way to fix it is to make restrictions on the kinds of topological spaces that will be considered. This approach will be taken here by assuming that all topological spaces are manifolds. In this case, no strange things like (4.8) can happen,[7] and the definitions of connected and path connected coincide [232]. Therefore, we will just say a space is *connected*. However, it is important to remember that this definition of connected is sometimes inadequate, and one should really say that X is *path connected*.

Simply connected Now that the notion of connectedness has been established, the next step is to express different kinds of connectivity. This may be done by using the notion of homotopy, which can intuitively be considered as a way to continuously "warp" or "morph" one path into another, as depicted in Figure 4.6a.

Two paths τ_1 and τ_2 are called *homotopic* (with endpoints fixed) if there exists a continuous function $h : [0,1] \times [0,1] \rightarrow X$ for which the following four conditions are met:

1. (**Start with first path**) $h(s,0) = \tau_1(s)$ for all $s \in [0,1]$.
2. (**End with second path**) $h(s,1) = \tau_2(s)$ for all $s \in [0,1]$.
3. (**Hold starting point fixed**) $h(0,t) = h(0,0)$ for all $t \in [0,1]$.
4. (**Hold ending point fixed**) $h(1,t) = h(1,0)$ for all $t \in [0,1]$.

The parameter t can be interpreted as a knob that is turned to gradually deform the path from τ_1 into τ_2. The first two conditions indicate that $t = 0$ yields τ_1

[7] The topologist's sine curve is not a manifold because all open sets that contain the point $(0,0)$ contain some of the points from the sine curve. These open sets are not homeomorphic to $\mathbb{R}$.

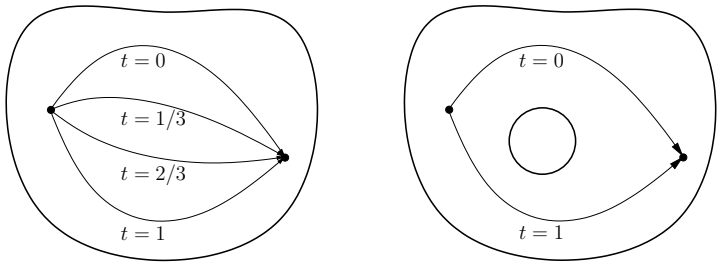


Figure 4.6: (a) Homotopy continuously warps one path into another. (b) The image of the path cannot be continuously warped over a hole in $\mathbb{R}^2$ because it causes a discontinuity. In this case, the two paths are not homotopic.

and $t = 1$ yields τ_2, respectively. The remaining two conditions indicate that the path endpoints are held fixed.

During the warping process, the path image cannot make a discontinuous jump. In $\mathbb{R}^2$, this prevents it from moving over holes, such as the one shown in Figure 4.6b. The key to preventing homotopy from jumping over some holes is that h must be continuous. In higher dimensions, however, there are many different kinds of holes. For the case of $\mathbb{R}^3$, for example, suppose the space is like a block of Swiss cheese that contains air bubbles. Homotopy can go around the air bubbles, but it cannot pass through a hole that is drilled through the entire block of cheese. Air bubbles and other kinds of holes that appear in higher dimensions can be characterized by generalizing homotopy to the warping of higher dimensional surfaces, as opposed to paths [226].

It is straightforward to show that homotopy defines an equivalence relation on the set of all paths from some $x_1 \in X$ to some $x_2 \in X$. The resulting notion of "equivalent paths" appears frequently in motion planning, control theory, and many other contexts. Suppose that X is path connected. If all paths fall into the same equivalence class, then X is called *simply connected*; otherwise, X is called *multiply connected*.

Groups The equivalence relation induced by homotopy starts to enter the realm of algebraic topology, which is a branch of mathematics that characterizes the structure of topological spaces in terms of algebraic objects, such as groups. These resulting groups have important implications for motion planning. Therefore, we give a brief overview. First, the notion of a group must be precisely defined. A *group* is a set, G, together with a binary operation, $\circ$, such that the following *group axioms* are satisfied:

1. (**Closure**) For any $a, b \in G$, the product $a \circ b \in G$.

2. (**Associativity**) For all $a, b, c \in G$, $(a \circ b) \circ c = a \circ (b \circ c)$. Hence, parentheses are not needed, and the product may be written as $a \circ b \circ c$.

3. (**Identity**) There is an element $e \in G$, called the *identity*, such that for all $a \in G$, $e \circ a = a$ and $a \circ e = a$.

4. (**Inverse**) For every element $a \in G$, there is an element a^{-1}, called the *inverse* of a, for which $a \circ a^{-1} = e$ and $a^{-1} \circ a = e$.

Here are some examples.

Example 4.7 (Simple Examples of Groups) The set of integers $\mathbb{Z}$ is a group with respect to addition. The identity is 0, and the inverse of each i is $-i$. The set $\mathbb{Q} \setminus 0$ of rational numbers with 0 removed is a group with respect to multiplication. The identity is 1, and the inverse of every element, q, is $1/q$ (0 was removed to avoid division by zero). ■

An important property, which only some groups possess, is *commutativity*: $a \circ b = b \circ a$ for any $a, b \in G$. The group in this case is called *commutative* or *Abelian*. We will encounter examples of both kinds of groups, both commutative and noncommutative. An example of a commutative group is vector addition over $\mathbb{R}^n$. The set of all 3D rotations is an example of a noncommutative group.

The fundamental group Now an interesting group will be constructed from the space of paths and the equivalence relation obtained by homotopy. The *fundamental group*, $\pi_1(X)$ (or *first homotopy group*), is associated with any topological space, X. Let a (continuous) path for which $f(0) = f(1)$ be called a *loop*. Let some $x_b \in X$ be designated as a *base point*. For some arbitrary but fixed base point, x_b, consider the set of all loops such that $f(0) = f(1) = x_b$. This can be made into a group by defining the following binary operation. Let $\tau_1 : [0, 1] \to X$ and $\tau_2 : [0, 1] \to X$ be two loop paths with the same base point. Their product $\tau = \tau_1 \circ \tau_2$ is defined as

$$\tau(t) = \begin{cases} \tau_1(2t) & \text{if } t \in [0, 1/2) \\ \tau_2(2t - 1) & \text{if } t \in [1/2, 1]. \end{cases} \tag{4.9}$$

This results in a continuous loop path because τ_1 terminates at x_b, and τ_2 begins at x_b. In a sense, the two paths are concatenated end-to-end.

Suppose now that the equivalence relation induced by homotopy is applied to the set of all loop paths through a fixed point, x_b. It will no longer be important which particular path was chosen from a class; any representative may be used. The equivalence relation also applies when the set of loops is interpreted as a group. The group operation actually occurs over the set of equivalences of paths.

Consider what happens when two paths from different equivalence classes are concatenated using $\circ$. Is the resulting path homotopic to either of the first two?

Is the resulting path homotopic if the original two are from the same homotopy class? The answers in general are *no* and *no*, respectively. The fundamental group describes how the equivalence classes of paths are related and characterizes the connectivity of X. Since fundamental groups are based on paths, there is a nice connection to motion planning.

Example 4.8 (A Simply Connected Space) Suppose that a topological space X is simply connected. In this case, all loop paths from a base point x_b are homotopic, resulting in one equivalence class. The result is $\pi_1(X) = \mathbf{1_G}$, which is the group that consists of only the identity element. ■

Example 4.9 (The Fundamental Group of $\mathbb{S}^1$) Suppose $X = \mathbb{S}^1$. In this case, there is an equivalence class of paths for each $i \in \mathbb{Z}$, the set of integers. If $i > 0$, then it means that the path winds i times around $\mathbb{S}^1$ in the counterclockwise direction and then returns to x_b. If $i < 0$, then the path winds around i times in the clockwise direction. If $i = 0$, then the path is equivalent to one that remains at x_b. The fundamental group is $\mathbb{Z}$, with respect to the operation of addition. If τ_1 travels i_1 times counterclockwise, and τ_2 travels i_2 times counterclockwise, then $\tau = \tau_1 \circ \tau_2$ belongs to the class of loops that travel around $i_1 + i_2$ times counterclockwise. Consider additive inverses. If a path travels seven times around $\mathbb{S}^1$, and it is combined with a path that travels seven times in the opposite direction, the result is homotopic to a path that remains at x_b. Thus, $\pi_1(\mathbb{S}^1) = \mathbb{Z}$. ■

Example 4.10 (The Fundamental Group of $\mathbb{T}^n$) For the torus, $\pi_1(\mathbb{T}^n) = \mathbb{Z}^n$, in which the ith component of $\mathbb{Z}^n$ corresponds to the number of times a loop path wraps around the ith component of $\mathbb{T}^n$. This makes intuitive sense because $\mathbb{T}^n$ is just the Cartesian product of n circles. The fundamental group $\mathbb{Z}^n$ is obtained by starting with a simply connected subset of the plane and drilling out n disjoint, bounded holes. This situation arises frequently when a mobile robot must avoid collision with n disjoint obstacles in the plane. ■

By now it seems that the fundamental group simply keeps track of how many times a path travels around holes. This next example yields some very bizarre behavior that helps to illustrate some of the interesting structure that arises in algebraic topology.

Example 4.11 (The Fundamental Group of $\mathbb{RP}^2$) Suppose $X = \mathbb{RP}^2$, the projective plane. In this case, there are only two equivalence classes on the space of loop paths. All paths that "wrap around" an even number of times are homotopic. Likewise, all paths that wrap around an odd number of times are homotopic. This strange behavior is illustrated in Figure 4.7. The resulting fundamental group

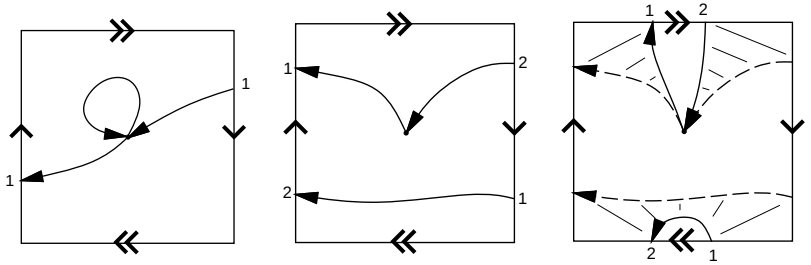


Figure 4.7: An illustration of why $\pi_1(\mathbb{RP}^2) = \mathbb{Z}_2$. The integers 1 and 2 indicate precisely where a path continues when it reaches the boundary. (a) Two paths are shown that are not equivalent. (b) A path that winds around twice is shown. (c) This is homotopic to a loop path that does not wind around at all. Eventually, the part of the path that appears at the bottom is pulled through the top. It finally shrinks into an arbitrarily small loop.

therefore has only two elements: $\pi_1(\mathbb{RP}^2) = \mathbb{Z}_2$, the cyclic group of order 2, which corresponds to addition mod 2. This makes intuitive sense because the group keeps track of whether a sum of integers is odd or even, which in this application corresponds to the total number of traversals over the square representation of $\mathbb{RP}^2$. The fundamental group is the same for $\mathbb{RP}^3$, which arises in Section 4.2.2 because it is homeomorphic to the set of 3D rotations. Thus, there are surprisingly only two path classes for the set of 3D rotations. ■

Unfortunately, two topological spaces may have the same fundamental group even if the spaces are not homeomorphic. For example, $\mathbb{Z}$ is the fundamental group of $\mathbb{S}^1$, the cylinder, $\mathbb{R} \times \mathbb{S}^1$, and the Möbius band. In the last case, the fundamental group does not indicate that there is a "twist" in the space. Another problem is that spaces with interesting connectivity may be declared as simply connected. The fundamental group of the sphere $\mathbb{S}^2$ is just $\mathbf{1_G}$, the same as for $\mathbb{R}^2$. Try envisioning loop paths on the sphere; it can be seen that they all fall into one equivalence class. Hence, $\mathbb{S}^2$ is simply connected. The fundamental group also neglects bubbles in $\mathbb{R}^3$ because the homotopy can warp paths around them. Some of these troubles can be fixed by defining second-order homotopy groups. For example, a continuous function, $[0,1] \times [0,1] \rightarrow X$, of two variables can be used instead of a path. The resulting homotopy generates a kind of sheet or surface that can be warped through the space, to yield a homotopy group $\pi_2(X)$ that wraps around bubbles in $\mathbb{R}^3$. This idea can be extended beyond two dimensions to detect many different kinds of holes in higher dimensional spaces. This leads to the *higher order homotopy groups*. A stronger concept than simply connected for

a space is that its homotopy groups of all orders are equal to the identity group. This prevents all kinds of holes from occurring and implies that a space, X, is *contractible*, which means a kind of homotopy can be constructed that shrinks X to a point [226]. In the plane, the notions of *contractible* and *simply connected* are equivalent; however, in higher dimensional spaces, such as those arising in motion planning, the term *contractible* should be used to indicate that the space has no interior obstacles (holes).

An alternative to basing groups on homotopy is to derive them using *homology*, which is based on the structure of cell complexes instead of homotopy mappings. This subject is much more complicated to present, but it is more powerful for proving theorems in topology. See the literature overview at the end of the chapter for suggested further reading on algebraic topology.

4.2 Defining the Configuration Space

This section defines the manifolds that arise from the transformations of Chapter 3. If the robot has n degrees of freedom, the set of transformations is usually a manifold of dimension n. This manifold is called the *configuration space* of the robot, and its name is often shortened to *C-space*. In this book, the C-space may be considered as a special state space. To solve a motion planning problem, algorithms must conduct a search in the C-space. The C-space provides a powerful abstraction that converts the complicated models and transformations of Chapter 3 into the general problem of computing a path that traverses a manifold. By developing algorithms directly for this purpose, they apply to a wide variety of different kinds of robots and transformations. In Section 4.3 the problem will be complicated by bringing obstacles into the configuration space, but in Section 4.2 there will be no obstacles.

4.2.1 2D Rigid Bodies: $SE(2)$

Section 3.2.2 expressed how to transform a rigid body in $\mathbb{R}^2$ by a homogeneous transformation matrix, T, given by (3.35). The task in this chapter is to characterize the set of all possible rigid-body transformations. Which manifold will this be? Here is the answer and brief explanation. Since any $x_t, y_t \in \mathbb{R}$ can be selected for translation, this alone yields a manifold $M_1 = \mathbb{R}^2$. Independently, any rotation, $\theta \in [0, 2\pi)$, can be applied. Since 2π yields the same rotation as 0, they can be identified, which makes the set of 2D rotations into a manifold, $M_2 = \mathbb{S}^1$. To obtain the manifold that corresponds to all rigid-body motions, simply take $\mathcal{C} = M_1 \times M_2 = \mathbb{R}^2 \times \mathbb{S}^1$. The answer to the question is that the C-space is a kind of cylinder.

Now we give a more detailed technical argument. The main purpose is that such a simple, intuitive argument will not work for the 3D case. Our approach is

to introduce some of the technical machinery here for the 2D case, which is easier to understand, and then extend it to the 3D case in Section 4.2.2.

Matrix groups The first step is to consider the set of transformations as a group, in addition to a topological space.[8] We now derive several important groups from sets of matrices, ultimately leading to $SO(n)$, the group of $n \times n$ rotation matrices, which is very important for motion planning. The set of all nonsingular $n \times n$ real-valued matrices is called the *general linear group*, denoted by $GL(n)$, with respect to matrix multiplication. Each matrix $A \in GL(n)$ has an inverse $A^{-1} \in GL(n)$, which when multiplied yields the identity matrix, $AA^{-1} = I$. The matrices must be nonsingular for the same reason that 0 was removed from $\mathbb{Q}$. The analog of division by zero for matrix algebra is the inability to invert a singular matrix.

Many interesting groups can be formed from one group, G_1, by removing some elements to obtain a *subgroup*, G_2. To be a subgroup, G_2 must be a subset of G_1 and satisfy the group axioms. We will arrive at the set of rotation matrices by constructing subgroups. One important subgroup of $GL(n)$ is the *orthogonal group*, $O(n)$, which is the set of all matrices $A \in GL(n)$ for which $AA^T = I$, in which A^T denotes the matrix *transpose* of A. These matrices have orthogonal columns (the inner product of any pair is zero) and the determinant is always 1 or -1. Thus, note that AA^T takes the inner product of every pair of columns. If the columns are different, the result must be 0; if they are the same, the result is 1 because $AA^T = I$. The *special orthogonal group*, $SO(n)$, is the subgroup of $O(n)$ in which every matrix has determinant 1. Another name for $SO(n)$ is the *group of n-dimensional rotation matrices*.

A chain of groups, $SO(n) \leq O(n) \leq GL(n)$, has been described in which $\leq$ denotes "a subgroup of." Each group can also be considered as a topological space. The set of all $n \times n$ matrices (which is not a group with respect to multiplication) with real-valued entries is homeomorphic to $\mathbb{R}^{n^2}$ because n^2 entries in the matrix can be independently chosen. For $GL(n)$, singular matrices are removed, but an n^2-dimensional manifold is nevertheless obtained. For $O(n)$, the expression $AA^T = I$ corresponds to n^2 algebraic equations that have to be satisfied. This should substantially drop the dimension. Note, however, that many of the equations are redundant (pick your favorite value for n, multiply the matrices, and see what happens). There are only $\binom{n}{2}$ ways (pairwise combinations) to take the inner product of pairs of columns, and there are n equations that require the magnitude of each column to be 1. This yields a total of $n(n+1)/2$ independent equations. Each independent equation drops the manifold dimension by one, and

[8] The groups considered in this section are actually Lie groups because they are smooth manifolds [45]. We will not use that name here, however, because the notion of a smooth structure has not yet been defined. Readers familiar with Lie groups, however, will recognize most of the coming concepts. Some details on Lie groups appear later in Sections 15.4.3 and 15.5.1.

the resulting dimension of $O(n)$ is $n^2 - n(n+1)/2 = n(n-1)/2$, which is easily remembered as $\binom{n}{2}$. To obtain $SO(n)$, the constraint $\det A = 1$ is added, which eliminates exactly half of the elements of $O(n)$ but keeps the dimension the same.

Example 4.12 (Matrix Subgroups) It is helpful to illustrate the concepts for $n = 2$. The set of all 2×2 matrices is

$$\left\{ \begin{pmatrix} a & b \\ c & d \end{pmatrix} \;\middle|\; a, b, c, d \in \mathbb{R} \right\}, \tag{4.10}$$

which is homeomorphic to $\mathbb{R}^4$. The group $GL(2)$ is formed from the set of all nonsingular 2×2 matrices, which introduces the constraint that $ad - bc \neq 0$. The set of singular matrices forms a 3D manifold with boundary in $\mathbb{R}^4$, but all other elements of $\mathbb{R}^4$ are in $GL(2)$; therefore, $GL(2)$ is a 4D manifold.

Next, the constraint $AA^T = I$ is enforced to obtain $O(2)$. This becomes

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a & c \\ b & d \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{4.11}$$

which directly yields four algebraic equations:

$$a^2 + b^2 = 1 \tag{4.12}$$
$$ac + bd = 0 \tag{4.13}$$
$$ca + db = 0 \tag{4.14}$$
$$c^2 + d^2 = 1. \tag{4.15}$$

Note that (4.14) is redundant. There are two kinds of equations. One equation, given by (4.13), forces the inner product of the columns to be 0. There is only one because $\binom{n}{2} = 1$ for $n = 2$. Two other constraints, (4.12) and (4.15), force the rows to be unit vectors. There are two because $n = 2$. The resulting dimension of the manifold is $\binom{n}{2} = 1$ because we started with $\mathbb{R}^4$ and lost three dimensions from (4.12), (4.13), and (4.15). What does this manifold look like? Imagine that there are two different two-dimensional unit vectors, (a, b) and (c, d). Any value can be chosen for (a, b) as long as $a^2 + b^2 = 1$. This looks like $\mathbb{S}^1$, but the inner product of (a, b) and (c, d) must also be 0. Therefore, for each value of (a, b), there are two choices for c and d: 1) $c = b$ and $d = -a$, or 2) $c = -b$ and $d = a$. It appears that there are two circles! The manifold is $\mathbb{S}^1 \sqcup \mathbb{S}^1$, in which $\sqcup$ denotes the union of disjoint sets. Note that this manifold is not connected because no path exists from one circle to the other.

The final step is to require that $\det A = ad - bc = 1$, to obtain $SO(2)$, the set of all 2D rotation matrices. Without this condition, there would be matrices that produce a rotated mirror image of the rigid body. The constraint simply forces the choice for c and d to be $c = -b$ and $a = d$. This throws away one of the circles from $O(2)$, to obtain a single circle for $SO(2)$. We have finally obtained what you already knew: $SO(2)$ is homeomorphic to $\mathbb{S}^1$. The circle can be parameterized using polar coordinates to obtain the standard 2D rotation matrix, (3.31), given in Section 3.2.2. ■

Special Euclidean group Now that the group of rotations, $SO(n)$, is characterized, the next step is to allow both rotations and translations. This corresponds to the set of all $(n+1) \times (n+1)$ transformation matrices of the form

$$\left\{ \begin{pmatrix} R & v \\ 0 & 1 \end{pmatrix} \;\middle|\; R \in SO(n) \text{ and } v \in \mathbb{R}^n \right\}. \tag{4.16}$$

This should look like a generalization of (3.52) and (3.56), which were for $n = 2$ and $n = 3$, respectively. The R part of the matrix achieves rotation of an n-dimensional body in $\mathbb{R}^n$, and the v part achieves translation of the same body. The result is a group, $SE(n)$, which is called the *special Euclidean group*. As a topological space, $SE(n)$ is homeomorphic to $\mathbb{R}^n \times SO(n)$, because the rotation matrix and translation vectors may be chosen independently. In the case of $n = 2$, this means $SE(2)$ is homeomorphic to $\mathbb{R}^2 \times \mathbb{S}^1$, which verifies what was stated at the beginning of this section. Thus, the C-space of a 2D rigid body that can translate and rotate in the plane is

$$\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1. \tag{4.17}$$

To be more precise, $\cong$ should be used in the place of $=$ to indicate that $\mathcal{C}$ could be any space homeomorphic to $\mathbb{R}^2 \times \mathbb{S}^1$; however, this notation will mostly be avoided.

Interpreting the C-space It is important to consider the topological implications of $\mathcal{C}$. Since $\mathbb{S}^1$ is multiply connected, $\mathbb{R} \times \mathbb{S}^1$ and $\mathbb{R}^2 \times \mathbb{S}^1$ are multiply connected. It is difficult to visualize $\mathcal{C}$ because it is a 3D manifold; however, there is a nice interpretation using identification. Start with the open unit cube, $(0, 1)^3 \subset \mathbb{R}^3$. Include the boundary points of the form $(x, y, 0)$ and $(x, y, 1)$, and make the identification $(x, y, 0) \sim (x, y, 1)$ for all $x, y \in (0, 1)$. This means that when traveling in the x and y directions, there is a "frontier" to the C-space; however, traveling in the z direction causes a wraparound.

It is very important for a motion planning algorithm to understand that this wraparound exists. For example, consider $\mathbb{R} \times \mathbb{S}^1$ because it is easier to visualize. Imagine a path planning problem for which $\mathcal{C} = \mathbb{R} \times \mathbb{S}^1$, as depicted in Figure 4.8. Suppose the top and bottom are identified to make a cylinder, and there is an obstacle across the middle. Suppose the task is to find a path from q_I to q_G. If the top and bottom were not identified, then it would not be possible to connect q_I to q_G; however, if the algorithm realizes it was given a cylinder, the task is straightforward. In general, it is very important to understand the topology of $\mathcal{C}$; otherwise, potential solutions will be lost.

The next section addresses $SE(n)$ for $n = 3$. The main difficulty is determining the topology of $SO(3)$. At least we do not have to consider $n > 3$ in this book.

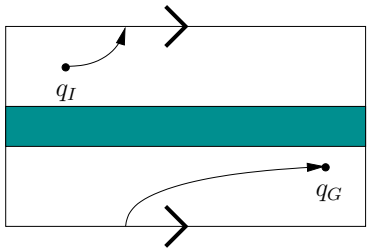


Figure 4.8: A planning algorithm may have to cross the identification boundary to find a solution path.

4.2.2 3D Rigid Bodies: $SE(3)$

One might expect that defining $\mathcal{C}$ for a 3D rigid body is an obvious extension of the 2D case; however, 3D rotations are significantly more complicated. The resulting C-space will be a six-dimensional manifold, $\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3$. Three dimensions come from translation and three more come from rotation.

The main quest in this section is to determine the topology of $SO(3)$. In Section 3.2.3, yaw, pitch, and roll were used to generate rotation matrices. These angles are convenient for visualization, performing transformations in software, and also for deriving the DH parameters. However, these were concerned with applying a single rotation, whereas the current problem is to characterize the set of all rotations. It is possible to use α, β, and γ to parameterize the set of rotations, but it causes serious troubles. There are some cases in which nonzero angles yield the identity rotation matrix, which is equivalent to $\alpha = \beta = \gamma = 0$. There are also cases in which a continuum of values for yaw, pitch, and roll angles yield the same rotation matrix. These problems destroy the topology, which causes both theoretical and practical difficulties in motion planning.

Consider applying the matrix group concepts from Section 4.2.1. The general linear group $GL(3)$ is homeomorphic to $\mathbb{R}^9$. The orthogonal group, $O(3)$, is determined by imposing the constraint $AA^T = I$. There are $\binom{3}{2} = 3$ independent equations that require distinct columns to be orthogonal, and three independent equations that force the magnitude of each column to be 1. This means that $O(3)$ has three dimensions, which matches our intuition since there were three rotation parameters in Section 3.2.3. To obtain $SO(3)$, the last constraint, $\det A = 1$, is added. Recall from Example 4.12 that $SO(2)$ consists of two circles, and the constraint $\det A = 1$ selects one of them. In the case of $O(3)$, there are two three-spheres, $\mathbb{S}^3 \sqcup \mathbb{S}^3$, and $\det A = 1$ selects one of them. However, there is one additional complication: Antipodal points on these spheres generate the same rotation matrix. This will be seen shortly when quaternions are used to parameterize $SO(3)$.

Using complex numbers to represent SO(2) Before introducing quaternions to parameterize 3D rotations, consider using complex numbers to parameterize 2D rotations. Let the term *unit complex number* refer to any complex number, $a + bi$, for which $a^2 + b^2 = 1$.

The set of all unit complex numbers forms a group under multiplication. It will be seen that it is "the same" group as $SO(2)$. This idea needs to be made more precise. Two groups, G and H, are considered "the same" if they are *isomorphic*, which means that there exists a bijective function $f : G \to H$ such that for all $a, b \in G$, $f(a) \circ f(b) = f(a \circ b)$. This means that we can perform some calculations in G, map the result to H, perform more calculations, and map back to G without any trouble. The sets G and H are just two alternative ways to express the same group.

The unit complex numbers and $SO(2)$ are isomorphic. To see this clearly, recall that complex numbers can be represented in polar form as $re^{i\theta}$; a unit complex number is simply $e^{i\theta}$. A bijective mapping can be made between 2D rotation matrices and unit complex numbers by letting $e^{i\theta}$ correspond to the rotation matrix (3.31).

If complex numbers are used to represent rotations, it is important that they behave algebraically in the same way. If two rotations are combined, the matrices are multiplied. The equivalent operation is multiplication of complex numbers. Suppose that a 2D robot is rotated by θ_1, followed by θ_2. In polar form, the complex numbers are multiplied to yield $e^{i\theta_1} e^{i\theta_2} = e^{i(\theta_1 + \theta_2)}$, which clearly represents a rotation of $\theta_1 + \theta_2$. If the unit complex number is represented in Cartesian form, then the rotations corresponding to $a_1 + b_1 i$ and $a_2 + b_2 i$ are combined to obtain $(a_1 a_2 - b_1 b_2) + (a_1 b_2 + a_2 b_1)i$. Note that here we have not used complex numbers to express the solution to a polynomial equation, which is their more popular use; we simply borrowed their nice algebraic properties. At any time, a complex number $a + bi$ can be converted into the equivalent rotation matrix

$$R(a, b) = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}. \tag{4.18}$$

Recall that only one independent parameter needs to be specified because $a^2 + b^2 = 1$. Hence, it appears that the set of unit complex numbers is the same manifold as $SO(2)$, which is the circle $\mathbb{S}^1$ (recall, that "same" means in the sense of homeomorphism).

Quaternions The manner in which complex numbers were used to represent 2D rotations will now be adapted to using quaternions to represent 3D rotations. Let $\mathbb{H}$ represent the set of *quaternions*, in which each quaternion, $h \in \mathbb{H}$, is represented as $h = a + bi + cj + dk$, and $a, b, c, d \in \mathbb{R}$. A quaternion can be considered as a four-dimensional vector. The symbols i, j, and k are used to denote three "imaginary" components of the quaternion. The following relationships are defined: $i^2 = j^2 = k^2 = ijk = -1$, from which it follows that $ij = k$, $jk = i$, and

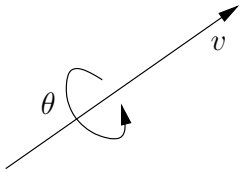


Figure 4.9: Any 3D rotation can be considered as a rotation by an angle θ about the axis given by the unit direction vector $v = [v_1 \;\; v_2 \;\; v_3]$.

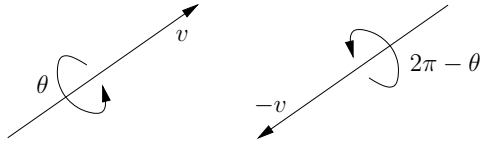


Figure 4.10: There are two ways to encode the same rotation.

$ki = j$. Using these, multiplication of two quaternions, $h_1 = a_1 + b_1 i + c_1 j + d_1 k$ and $h_2 = a_2 + b_2 i + c_2 j + d_2 k$, can be derived to obtain $h_1 \cdot h_2 = a_3 + b_3 i + c_3 j + d_3 k$, in which

$$\begin{aligned} a_3 &= a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \\ b_3 &= a_1 b_2 + a_2 b_1 + c_1 d_2 - c_2 d_1 \\ c_3 &= a_1 c_2 + a_2 c_1 + b_2 d_1 - b_1 d_2 \\ d_3 &= a_1 d_2 + a_2 d_1 + b_1 c_2 - b_2 c_1. \end{aligned} \tag{4.19}$$

Using this operation, it can be shown that $\mathbb{H}$ is a group with respect to quaternion multiplication. Note, however, that the multiplication is not commutative! This is also true of 3D rotations; there must be a good reason.

For convenience, quaternion multiplication can be expressed in terms of vector multiplications, a dot product, and a cross product. Let $v = [b \;\; c \;\; d]$ be a three-dimensional vector that represents the final three quaternion components. The first component of $h_1 \cdot h_2$ is $a_1 a_2 - v_1 \cdot v_2$. The final three components are given by the three-dimensional vector $a_1 v_2 + a_2 v_1 + v_1 \times v_2$.

In the same way that *unit* complex numbers were needed for $SO(2)$, *unit quaternions* are needed for $SO(3)$, which means that $\mathbb{H}$ is restricted to quaternions for which $a^2 + b^2 + c^2 + d^2 = 1$. Note that this forms a subgroup because the multiplication of unit quaternions yields a unit quaternion, and the other group axioms hold.

The next step is to describe a mapping from unit quaternions to $SO(3)$. Let the unit quaternion $h = a + bi + cj + dk$ map to the matrix

$$R(h) = \begin{pmatrix} 2(a^2+b^2)-1 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & 2(a^2+c^2)-1 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & 2(a^2+d^2)-1 \end{pmatrix}, \tag{4.20}$$

which can be verified as orthogonal and $\det R(h) = 1$. Therefore, it belongs to $SO(3)$. It is not shown here, but it conveniently turns out that h represents the rotation shown in Figure 4.9, by making the assignment

$$h = \cos\frac{\theta}{2} + \left(v_1 \sin\frac{\theta}{2}\right) i + \left(v_2 \sin\frac{\theta}{2}\right) j + \left(v_3 \sin\frac{\theta}{2}\right) k. \tag{4.21}$$

Unfortunately, this representation is not unique. It can be verified in (4.20) that $R(h) = R(-h)$. A nice geometric interpretation is given in Figure 4.10. The quaternions h and $-h$ represent the same rotation because a rotation of θ about the direction v is equivalent to a rotation of $2\pi - \theta$ about the direction $-v$. Consider the quaternion representation of the second expression of rotation with respect to the first. The real part is

$$\cos\left(\frac{2\pi-\theta}{2}\right) = \cos\left(\pi - \frac{\theta}{2}\right) = -\cos\left(\frac{\theta}{2}\right) = -a. \tag{4.22}$$

The i, j, and k components are

$$-v \sin\left(\frac{2\pi-\theta}{2}\right) = -v \sin\left(\pi - \frac{\theta}{2}\right) = -v \sin\left(\frac{\theta}{2}\right) = [-b \;\; -c \;\; -d]. \tag{4.23}$$

The quaternion $-h$ has been constructed. Thus, h and $-h$ represent the same rotation. Luckily, this is the only problem, and the mapping given by (4.20) is two-to-one from the set of unit quaternions to $SO(3)$.

This can be fixed by the identification trick. Note that the set of unit quaternions is homeomorphic to $\mathbb{S}^3$ because of the constraint $a^2 + b^2 + c^2 + d^2 = 1$. The algebraic properties of quaternions are not relevant at this point. Just imagine each h as an element of $\mathbb{R}^4$, and the constraint $a^2 + b^2 + c^2 + d^2 = 1$ forces the points to lie on $\mathbb{S}^3$. Using identification, declare $h \sim -h$ for all unit quaternions. This means that the antipodal points of $\mathbb{S}^3$ are identified. Recall from the end of Section 4.1.2 that when antipodal points are identified, $\mathbb{RP}^n \cong \mathbb{S}^n / \sim$. Hence, $SO(3) \cong \mathbb{RP}^3$, which can be considered as the set of all lines through the origin of $\mathbb{R}^4$, but this is hard to visualize. The representation of $\mathbb{RP}^2$ in Figure 4.5 can be extended to $\mathbb{RP}^3$. Start with $(0,1)^3 \subset \mathbb{R}^3$, and make three different kinds of identifications, one for each pair of opposite cube faces, and add all of the points to the manifold. For each kind of identification a twist needs to be made (without the twist, $\mathbb{T}^3$ would be obtained). For example, in the z direction, let $(x, y, 0) \sim (1 - x, 1 - y, 1)$ for all $x, y \in [0, 1]$.

One way to force uniqueness of rotations is to require staying in the "upper half" of $\mathbb{S}^3$. For example, require that $a \geq 0$, as long as the boundary case of $a = 0$ is handled properly because of antipodal points at the equator of $\mathbb{S}^3$. If $a = 0$, then require that $b \geq 0$. However, if $a = b = 0$, then require that $c \geq 0$ because points such as $(0, 0, -1, 0)$ and $(0, 0, 1, 0)$ are the same rotation. Finally, if $a = b = c = 0$, then only $d = 1$ is allowed. If such restrictions are made, it is

important, however, to remember the connectivity of $\mathbb{RP}^3$. If a path travels across the equator of $\mathbb{S}^3$, it must be mapped to the appropriate place in the "northern hemisphere." At the instant it hits the equator, it must move to the antipodal point. These concepts are much easier to visualize if you remove a dimension and imagine them for $\mathbb{S}^2 \subset \mathbb{R}^3$, as described at the end of Section 4.1.2.

Using quaternion multiplication The representation of rotations boiled down to picking points on $\mathbb{S}^3$ and respecting the fact that antipodal points give the same element of $SO(3)$. In a sense, this has nothing to do with the algebraic properties of quaternions. It merely means that $SO(3)$ can be parameterized by picking points in $\mathbb{S}^3$, just like $SO(2)$ was parameterized by picking points in $\mathbb{S}^1$ (ignoring the antipodal identification problem for $SO(3)$).

However, one important reason why the quaternion arithmetic was introduced is that the group of unit quaternions with h and $-h$ identified is also isomorphic to $SO(3)$. This means that a sequence of rotations can be multiplied together using quaternion multiplication instead of matrix multiplication. This is important because fewer operations are required for quaternion multiplication in comparison to matrix multiplication. At any point, (4.20) can be used to convert the result back into a matrix; however, this is not even necessary. It turns out that a point in the world, $(x, y, z) \in \mathbb{R}^3$, can be transformed by directly using quaternion arithmetic. An analog to the complex conjugate from complex numbers is needed. For any $h = a + bi + cj + dk \in \mathbb{H}$, let $h^* = a - bi - cj - dk$ be its *conjugate*. For any point $(x, y, z) \in \mathbb{R}^3$, let $p \in \mathbb{H}$ be the quaternion $0 + xi + yj + zk$. It can be shown (with a lot of algebra) that the rotated point (x, y, z) is given by $h \cdot p \cdot h^*$. The i, j, k components of the resulting quaternion are new coordinates for the transformed point. It is equivalent to having transformed (x, y, z) with the matrix $R(h)$.

Finding quaternion parameters from a rotation matrix Recall from Section 3.2.3 that given a rotation matrix (3.43), the yaw, pitch, and roll parameters could be directly determined using the atan2 function. It turns out that the quaternion representation can also be determined directly from the matrix. This is the inverse of the function in (4.20).[9]

For a given rotation matrix (3.43), the quaternion parameters $h = a + bi + cj + dk$ can be computed as follows [113]. The first component is

$$a = \tfrac{1}{2}\sqrt{r_{11} + r_{22} + r_{33} + 1}, \tag{4.24}$$

and if $a \neq 0$, then

$$b = \frac{r_{32} - r_{23}}{4a}, \tag{4.25}$$

[9]Since that function was two-to-one, it is technically not an inverse until the quaternions are restricted to the upper hemisphere, as described previously.

$$c = \frac{r_{13} - r_{31}}{4a}, \tag{4.26}$$

and

$$d = \frac{r_{21} - r_{12}}{4a}. \tag{4.27}$$

If $a = 0$, then the previously mentioned equator problem occurs. In this case,

$$b = \frac{r_{13} r_{12}}{\sqrt{r_{12}^2 r_{13}^2 + r_{12}^2 r_{23}^2 + r_{13}^2 r_{23}^2}}, \tag{4.28}$$

$$c = \frac{r_{12} r_{23}}{\sqrt{r_{12}^2 r_{13}^2 + r_{12}^2 r_{23}^2 + r_{13}^2 r_{23}^2}}, \tag{4.29}$$

and

$$d = \frac{r_{13} r_{23}}{\sqrt{r_{12}^2 r_{13}^2 + r_{12}^2 r_{23}^2 + r_{13}^2 r_{23}^2}}. \tag{4.30}$$

This method fails if $r_{12} = r_{23} = 0$ or $r_{13} = r_{23} = 0$ or $r_{12} = r_{23} = 0$. These correspond precisely to the cases in which the rotation matrix is a yaw, (3.39), pitch, (3.40), or roll, (3.41), which can be detected in advance.

Special Euclidean group Now that the complicated part of representing $SO(3)$ has been handled, the representation of $SE(3)$ is straightforward. The general form of a matrix in $SE(3)$ is given by (4.16), in which $R \in SO(3)$ and $v \in \mathbb{R}^3$. Since $SO(3) \cong \mathbb{RP}^3$, and translations can be chosen independently, the resulting C-space for a rigid body that rotates and translates in $\mathbb{R}^3$ is

$$\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3, \tag{4.31}$$

which is a six-dimensional manifold. As expected, the dimension of $\mathcal{C}$ is exactly the number of degrees of freedom of a free-floating body in space.

4.2.3 Chains and Trees of Bodies

If there are multiple bodies that are allowed to move independently, then their C-spaces can be combined using Cartesian products. Let $\mathcal{C}_i$ denote the C-space of $\mathcal{A}_i$. If there are n free-floating bodies in $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, then

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_n. \tag{4.32}$$

If the bodies are attached to form a kinematic chain or kinematic tree, then each C-space must be considered on a case-by-case basis. There is no general rule that simplifies the process. One thing to generally be careful about is that the full range of motion might not be possible for typical joints. For example, a revolute joint might not be able to swing all of the way around to enable any $\theta \in [0, 2\pi)$. If θ cannot wind around $\mathbb{S}^1$, then the C-space for this joint is homeomorphic to $\mathbb{R}$ instead of $\mathbb{S}^1$. A similar situation occurs for a spherical joint. A typical ball joint

cannot achieve any orientation in $SO(3)$ due to mechanical obstructions. In this case, the C-space is not $\mathbb{RP}^3$ because part of $SO(3)$ is missing.

Another complication is that the DH parameterization of Section 3.3.2 is designed to facilitate the assignment of coordinate frames and computation of transformations, but it neglects considerations of topology. For example, a common approach to representing a spherical robot wrist is to make three zero-length links that each behave as a revolute joint. If the range of motion is limited, this might not cause problems, but in general the problems would be similar to using yaw, pitch, and roll to represent $SO(3)$. There may be multiple ways to express the same arm configuration.

Several examples are given below to help in determining C-spaces for chains and trees of bodies. Suppose $\mathcal{W} = \mathbb{R}^2$, and there is a chain of n bodies that are attached by revolute joints. Suppose that the first joint is capable of rotation only about a fixed point (e.g., it spins around a nail). If each joint has the full range of motion $\theta_i \in [0, 2\pi)$, the C-space is

$$\mathcal{C} = \mathbb{S}^1 \times \mathbb{S}^1 \times \cdots \times \mathbb{S}^1 = \mathbb{T}^n. \tag{4.33}$$

However, if each joint is restricted to $\theta_i \in (-\pi/2, \pi/2)$, then $\mathcal{C} = \mathbb{R}^n$. If any transformation in $SE(2)$ can be applied to $\mathcal{A}_1$, then an additional $\mathbb{R}^2$ is needed. In the case of restricted joint motions, this yields $\mathbb{R}^{n+2}$. If the joints can achieve any orientation, then $\mathcal{C} = \mathbb{R}^2 \times \mathbb{T}^n$. If there are prismatic joints, then each joint contributes $\mathbb{R}$ to the C-space.

Recall from Figure 3.12 that for $\mathcal{W} = \mathbb{R}^3$ there are six different kinds of joints. The cases of revolute and prismatic joints behave the same as for $\mathcal{W} = \mathbb{R}^2$. Each screw joint contributes $\mathbb{R}$. A cylindrical joint contributes $\mathbb{R} \times \mathbb{S}^1$, unless its rotational motion is restricted. A planar joint contributes $\mathbb{R}^2 \times \mathbb{S}^1$ because any transformation in $SE(2)$ is possible. If its rotational motions are restricted, then it contributes $\mathbb{R}^3$. Finally, a spherical joint can theoretically contribute $\mathbb{RP}^3$. In practice, however, this rarely occurs. It is more likely to contribute $\mathbb{R}^2 \times \mathbb{S}^1$ or $\mathbb{R}^3$ after restrictions are imposed. Note that if the first joint is a free-floating body, then it contributes $\mathbb{R}^3 \times \mathbb{RP}^3$.

Kinematic trees can be handled in the same way as kinematic chains. One issue that has not been mentioned is that there might be collisions between the links. This has been ignored up to this point, but obviously this imposes very complicated restrictions. The concepts from Section 4.3 can be applied to handle this case and the placement of additional obstacles in $\mathcal{W}$. Reasoning about these kinds of restrictions and the path connectivity of the resulting space is indeed the main point of motion planning.

4.3 Configuration Space Obstacles

Section 4.2 defined $\mathcal{C}$, the manifold of robot transformations in the absence of any collision constraints. The current section removes from $\mathcal{C}$ the configurations

that either cause the robot to collide with obstacles or cause some specified links of the robot to collide with each other. The removed part of $\mathcal{C}$ is referred to as the obstacle region. The leftover space is precisely what a solution path must traverse. A motion planning algorithm must find a path in the leftover space from an initial configuration to a goal configuration. Finally, after the models of Chapter 3 and the previous sections of this chapter, the motion planning problem can be precisely described.

4.3.1 Definition of the Basic Motion Planning Problem

Obstacle region for a rigid body Suppose that the world, $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, contains an obstacle region, $\mathcal{O} \subset \mathcal{W}$. Assume here that a rigid robot, $\mathcal{A} \subset \mathcal{W}$, is defined; the case of multiple links will be handled shortly. Assume that both $\mathcal{A}$ and $\mathcal{O}$ are expressed as semi-algebraic models (which includes polygonal and polyhedral models) from Section 3.1. Let $q \in \mathcal{C}$ denote the *configuration* of $\mathcal{A}$, in which $q = (x_t, y_t, \theta)$ for $\mathcal{W} = \mathbb{R}^2$ and $q = (x_t, y_t, z_t, h)$ for $\mathcal{W} = \mathbb{R}^3$ (h represents the unit quaternion).

The *obstacle region*, $\mathcal{C}_{obs} \subseteq \mathcal{C}$, is defined as

$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}, \tag{4.34}$$

which is the set of all configurations, q, at which $\mathcal{A}(q)$, the transformed robot, intersects the obstacle region, $\mathcal{O}$. Since $\mathcal{O}$ and $\mathcal{A}(q)$ are closed sets in $\mathcal{W}$, the obstacle region is a closed set in $\mathcal{C}$.

The leftover configurations are called the *free space*, which is defined and denoted as $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$. Since $\mathcal{C}$ is a topological space and $\mathcal{C}_{obs}$ is closed, $\mathcal{C}_{free}$ must be an open set. This implies that the robot can come arbitrarily close to the obstacles while remaining in $\mathcal{C}_{free}$. If $\mathcal{A}$ "touches" $\mathcal{O}$,

$$\text{int}(\mathcal{O}) \cap \text{int}(\mathcal{A}(q)) = \emptyset \text{ and } \mathcal{O} \cap \mathcal{A}(q) \neq \emptyset, \tag{4.35}$$

then $q \in \mathcal{C}_{obs}$ (recall that int means the interior). The condition above indicates that only their boundaries intersect.

The idea of getting arbitrarily close may be nonsense in practical robotics, but it makes a clean formulation of the motion planning problem. Since $\mathcal{C}_{free}$ is open, it becomes impossible to formulate some optimization problems, such as finding the shortest path. In this case, the closure, $\text{cl}(\mathcal{C}_{free})$, should instead be used, as described in Section 7.7.

Obstacle region for multiple bodies If the robot consists of multiple bodies, the situation is more complicated. The definition in (4.34) only implies that the robot does not collide with the obstacles; however, if the robot consists of multiple bodies, then it might also be appropriate to avoid collisions between different links of the robot. Let the robot be modeled as a collection, $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m\}$, of m

links, which may or may not be attached together by joints. A single configuration vector q is given for the entire collection of links. We will write $\mathcal{A}_i(q)$ for each link, i, even though some of the parameters of q may be irrelevant for moving link $\mathcal{A}_i$. For example, in a kinematic chain, the configuration of the second body does not depend on the angle between the ninth and tenth bodies.

Let P denote the set of *collision pairs*, in which each collision pair, $(i, j) \in P$, represents a pair of link indices $i, j \in \{1, 2, \ldots, m\}$, such that $i \neq j$. If (i, j) appears in P, it means that A_i and A_j are not allowed to be in a configuration, q, for which $\mathcal{A}_i(q) \cap \mathcal{A}_j(q) \neq \emptyset$. Usually, P does not represent all pairs because consecutive links are in contact all of the time due to the joint that connects them. One common definition for P is that each link must avoid collisions with any links to which it is not attached by a joint. For m bodies, P is generally of size $O(m^2)$; however, in practice it is often possible to eliminate many pairs by some geometric analysis of the linkage. Collisions between some pairs of links may be impossible over all of $\mathcal{C}$, in which case they do not need to appear in P.

Using P, the consideration of robot self-collisions is added to the definition of $\mathcal{C}_{obs}$ to obtain

$$\mathcal{C}_{obs} = \left(\bigcup_{i=1}^{m} \{q \in \mathcal{C} \mid \mathcal{A}_i(q) \cap \mathcal{O} \neq \emptyset\} \right) \bigcup \left(\bigcup_{[i,j] \in P} \{q \in \mathcal{C} \mid \mathcal{A}_i(q) \cap \mathcal{A}_j(q) \neq \emptyset\} \right). \tag{4.36}$$

Thus, a configuration $q \in \mathcal{C}$ is in $\mathcal{C}_{obs}$ if at least one link collides with $\mathcal{O}$ or a pair of links indicated by P collide with each other.

Definition of basic motion planning Finally, enough tools have been introduced to precisely define the motion planning problem. The problem is conceptually illustrated in Figure 4.11. The main difficulty is that it is neither straightforward nor efficient to construct an explicit boundary or solid representation of either $\mathcal{C}_{free}$ or $\mathcal{C}_{obs}$. The components are as follows:

Formulation 4.1 (The Piano Mover's Problem)

1. A *world* $\mathcal{W}$ in which either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.

2. A semi-algebraic *obstacle region* $\mathcal{O} \subset \mathcal{W}$ in the world.

3. A semi-algebraic *robot* is defined in $\mathcal{W}$. It may be a rigid robot $\mathcal{A}$ or a collection of m links, $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.

4. The *configuration space* $\mathcal{C}$ determined by specifying the set of all possible transformations that may be applied to the robot. From this, $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are derived.

5. A configuration, $q_I \in \mathcal{C}_{free}$ designated as the *initial configuration*.

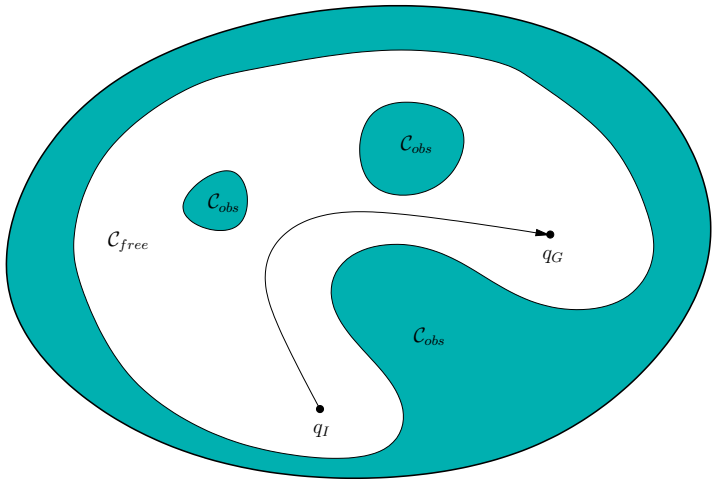


Figure 4.11: The basic motion planning problem is conceptually very simple using C-space ideas. The task is to find a path from q_I to q_G in $\mathcal{C}_{free}$. The entire blob represents $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$.

6. A configuration $q_G \in \mathcal{C}_{free}$ designated as the *goal configuration*. The initial and goal configurations together are often called a *query pair* (or *query*) and designated as (q_I, q_G).

7. A complete algorithm must compute a (continuous) *path*, $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, such that $\tau(0) = q_I$ and $\tau(1) = q_G$, or correctly report that such a path does not exist.

It was shown by Reif [409] that this problem is PSPACE-hard, which implies NP-hard. The main problem is that the dimension of $\mathcal{C}$ is unbounded.

4.3.2 Explicitly Modeling $\mathcal{C}_{obs}$: The Translational Case

It is important to understand how to construct a representation of $\mathcal{C}_{obs}$. In some algorithms, especially the combinatorial methods of Chapter 6, this represents an important first step to solving the problem. In other algorithms, especially the sampling-based planning algorithms of Chapter 5, it helps to understand why such constructions are avoided due to their complexity.

The simplest case for characterizing $\mathcal{C}_{obs}$ is when $\mathcal{C} = \mathbb{R}^n$ for $n = 1$, 2, and 3, and the robot is a rigid body that is restricted to translation only. Under these conditions, $\mathcal{C}_{obs}$ can be expressed as a type of convolution. For any two sets $X, Y \subset \mathbb{R}^n$, let their *Minkowski difference*[10] be defined as

$$X \ominus Y = \{x - y \in \mathbb{R}^n \mid x \in X \text{ and } y \in Y\}, \tag{4.37}$$

[10]In some contexts, which include mathematics and image processing, the Minkowski differ-

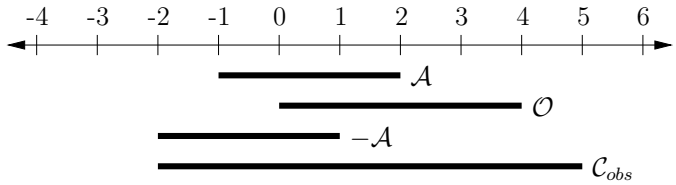


Figure 4.12: A one-dimensional C-space obstacle.

in which $x - y$ is just vector subtraction on $\mathbb{R}^n$. The Minkowski difference between X and Y can also be considered as the Minkowski sum of X and $-Y$. The *Minkowski sum* $\oplus$ is obtained by simply adding elements of X and Y in (4.37), as opposed to subtracting them. The set $-Y$ is obtained by replacing each $y \in Y$ by $-y$.

In terms of the Minkowski difference, $\mathcal{C}_{obs} = \mathcal{O} \ominus \mathcal{A}(0)$. To see this, it is helpful to consider a one-dimensional example.

Example 4.13 (One-Dimensional C-Space Obstacle) In Figure 4.12, both the robot $\mathcal{A} = [-1, 2]$ and obstacle region $\mathcal{O} = [0, 4]$ are intervals in a one-dimensional world, $\mathcal{W} = \mathbb{R}$. The negation, $-\mathcal{A}$, of the robot is shown as the interval $[-2, 1]$. Finally, by applying the Minkowski sum to $\mathcal{O}$ and $-\mathcal{A}$, the C-space obstacle, $\mathcal{C}_{obs} = [-2, 5]$, is obtained. ■

The Minkowski difference is often considered as a *convolution*. It can even be defined to appear the same as studied in differential equations and system theory. For a one-dimensional example, let $f : \mathbb{R} \to \{0, 1\}$ be a function such that $f(x) = 1$ if and only if $x \in \mathcal{O}$. Similarly, let $g : \mathbb{R} \to \{0, 1\}$ be a function such that $g(x) = 1$ if and only if $x \in \mathcal{A}$. The convolution

$$h(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau) d\tau, \tag{4.38}$$

yields a function h, for which $h(x) > 0$ if $x \in \text{int}(\mathcal{C}_{obs})$, and $h(x) = 0$ otherwise.

A polygonal C-space obstacle A simple algorithm for computing $\mathcal{C}_{obs}$ exists in the case of a 2D world that contains a convex polygonal obstacle $\mathcal{O}$ and a convex polygonal robot $\mathcal{A}$ [344]. This is often called the *star algorithm*. For this problem, $\mathcal{C}_{obs}$ is also a convex polygon. Recall that nonconvex obstacles and robots can be modeled as the union of convex parts. The concepts discussed below can also be applied in the nonconvex case by considering $\mathcal{C}_{obs}$ as the union of convex components, each of which corresponds to a convex component of $\mathcal{A}$ colliding with a convex component of $\mathcal{O}$.

ence or *Minkowski subtraction* is defined differently (instead, it is a kind of "erosion"). For this reason, some authors prefer to define all operations in terms of the Minkowski sum, $\oplus$, which is consistently defined in all contexts. Following this convention, we would define $X \oplus (-Y)$, which is equivalent to $X \ominus Y$.

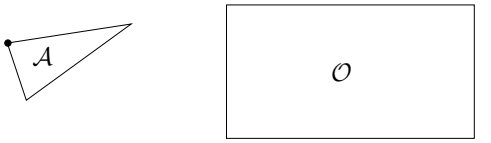


Figure 4.13: A triangular robot and a rectangular obstacle.

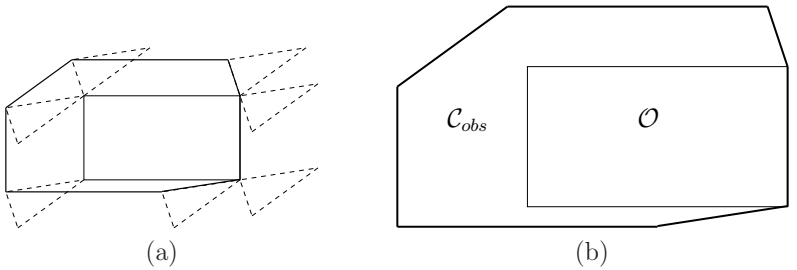


Figure 4.14: (a) Slide the robot around the obstacle while keeping them both in contact. (b) The edges traced out by the origin of $\mathcal{A}$ form $\mathcal{C}_{obs}$.

The method is based on sorting normals to the edges of the polygons on the basis of angles. The key observation is that every edge of $\mathcal{C}_{obs}$ is a translated edge from either $\mathcal{A}$ or $\mathcal{O}$. In fact, every edge from $\mathcal{O}$ and $\mathcal{A}$ is used exactly once in the construction of $\mathcal{C}_{obs}$. The only problem is to determine the ordering of these edges of $\mathcal{C}_{obs}$. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ denote the angles of the inward edge normals in counterclockwise order around $\mathcal{A}$. Let $\beta_1, \beta_2, \ldots, \beta_n$ denote the outward edge normals to $\mathcal{O}$. After sorting both sets of angles in circular order around $\mathbb{S}^1$, $\mathcal{C}_{obs}$ can be constructed incrementally by using the edges that correspond to the sorted normals, in the order in which they are encountered.

Example 4.14 (A Triangular Robot and Rectangular Obstacle) To gain an understanding of the method, consider the case of a triangular robot and a rectangular obstacle, as shown in Figure 4.13. The black dot on $\mathcal{A}$ denotes the origin of its body frame. Consider sliding the robot around the obstacle in such a way that they are always in contact, as shown in Figure 4.14a. This corresponds to the traversal of all of the configurations in $\partial\mathcal{C}_{obs}$ (the boundary of $\mathcal{C}_{obs}$). The origin of $\mathcal{A}$ traces out the edges of $\mathcal{C}_{obs}$, as shown in Figure 4.14b. There are seven edges, and each edge corresponds to either an edge of $\mathcal{A}$ or an edge of $\mathcal{O}$. The directions of the normals are defined as shown in Figure 4.15a. When sorted as shown in Figure 4.15b, the edges of $\mathcal{C}_{obs}$ can be incrementally constructed. ■

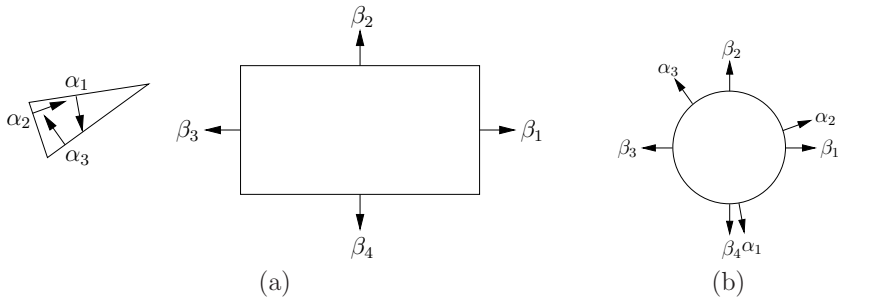


Figure 4.15: (a) Take the inward edge normals of $\mathcal{A}$ and the outward edge normals of $\mathcal{O}$. (b) Sort the edge normals around $\mathbb{S}^1$. This gives the order of edges in $\mathcal{C}_{obs}$.

The running time of the algorithm is $O(n+m)$, in which n is the number of edges defining $\mathcal{A}$, and m is the number of edges defining $\mathcal{O}$. Note that the angles can be sorted in linear time because they already appear in counterclockwise order around $\mathcal{A}$ and $\mathcal{O}$; they only need to be merged. If two edges are collinear, then they can be placed end-to-end as a single edge of $\mathcal{C}_{obs}$.

Computing the boundary of $\mathcal{C}_{obs}$ So far, the method quickly identifies each edge that contributes to $\mathcal{C}_{obs}$. It can also construct a solid representation of $\mathcal{C}_{obs}$ in terms of half-planes. This requires defining $n+m$ linear equations (assuming there are no collinear edges).

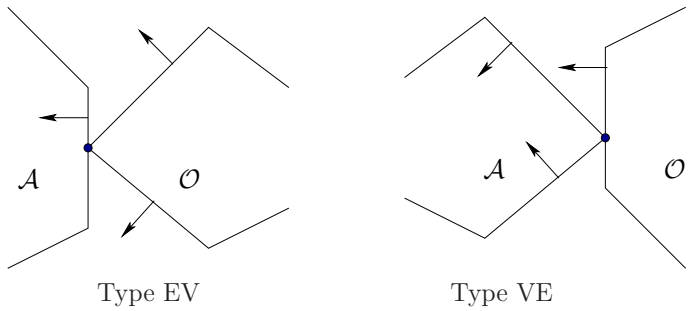


Figure 4.16: Two different types of contact, each of which generates a different kind of $\mathcal{C}_{obs}$ edge [152, 344].

There are two different ways in which an edge of $\mathcal{C}_{obs}$ is generated, as shown in Figure 4.16 [153, 344]. *Type EV* contact refers to the case in which an edge of $\mathcal{A}$ is in contact with a vertex of $\mathcal{O}$. Type EV contacts contribute to n edges of $\mathcal{C}_{obs}$, once for each edge of $\mathcal{A}$. *Type VE* contact refers to the case in which

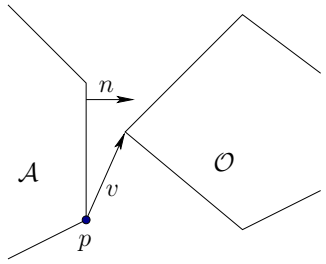


Figure 4.17: Contact occurs when n and v are perpendicular.

a vertex of $\mathcal{A}$ is in contact with an edge of $\mathcal{O}$. This contributes to m edges of $\mathcal{C}_{obs}$. The relationships between the edge normals are also shown in Figure 4.16. For Type EV, the inward edge normal points between the outward edge normals of the obstacle edges that share the contact vertex. Likewise for Type VE, the outward edge normal of $\mathcal{O}$ points between the inward edge normals of $\mathcal{A}$.

Using the ordering shown in Figure 4.15b, Type EV contacts occur precisely when an edge normal of $\mathcal{A}$ is encountered, and Type VE contacts occur when an edge normal of $\mathcal{O}$ is encountered. The task is to determine the line equation for each occurrence. Consider the case of a Type EV contact; the Type VE contact can be handled in a similar manner. In addition to the constraint on the directions of the edge normals, the contact vertex of $\mathcal{O}$ must lie on the contact edge of $\mathcal{A}$. Recall that convex obstacles were constructed by the intersection of half-planes. Each edge of $\mathcal{C}_{obs}$ can be defined in terms of a supporting half-plane; hence, it is only necessary to determine whether the vertex of $\mathcal{O}$ lies on the line through the contact edge of $\mathcal{A}$. This condition occurs precisely as n and v are perpendicular, as shown in Figure 4.17, and yields the constraint $n \cdot v = 0$.

Note that the normal vector n does not depend on the configuration of $\mathcal{A}$ because the robot cannot rotate. The vector v, however, depends on the translation $q = (x_t, y_t)$ of the point p. Therefore, it is more appropriate to write the condition as $n \cdot v(x_t, y_t) = 0$. The transformation equations are linear for translation; hence, $n \cdot v(x_t, y_t) = 0$ is the equation of a line in $\mathcal{C}$. For example, if the coordinates of p are $(1, 2)$ for $\mathcal{A}(0, 0)$, then the expression for p at configuration (x_t, y_t) is $(1 + x_t, 2 + y_t)$. Let $f(x_t, y_t) = n \cdot v(x_t, y_t)$. Let $H = \{(x_t, y_t) \in \mathcal{C} \mid f(x_t, y_t) \leq 0\}$. Observe that any configurations not in H must lie in $\mathcal{C}_{free}$. The half-plane H is used to define one edge of $\mathcal{C}_{obs}$. The obstacle region $\mathcal{C}_{obs}$ can be completely characterized by intersecting the resulting half-planes for each of the Type EV and Type VE contacts. This yields a convex polygon in $\mathcal{C}$ that has $n+m$ sides, as expected.

Example 4.15 (The Boundary of $\mathcal{C}_{obs}$) Consider building a geometric model of $\mathcal{C}_{obs}$ for the robot and obstacle shown in Figure 4.18. Suppose that the orientation of $\mathcal{A}$ is fixed as shown, and $\mathcal{C} = \mathbb{R}^2$. In this case, $\mathcal{C}_{obs}$ will be a convex

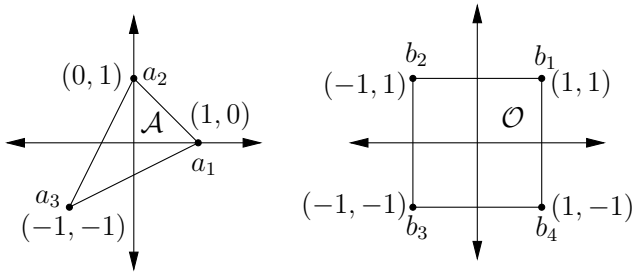


Figure 4.18: Consider constructing the obstacle region for this example.

Type	Vtx.	Edge	n	v	Half-Plane
VE	a_3	b_4-b_1	$[1, 0]$	$[x_t - 2, y_t]$	$\{q \in \mathcal{C} \mid x_t - 2 \leq 0\}$
VE	a_3	b_1-b_2	$[0, 1]$	$[x_t - 2, y_t - 2]$	$\{q \in \mathcal{C} \mid y_t - 2 \leq 0\}$
EV	b_2	a_3-a_1	[1,-2]	$[-x_t, 2 - y_t]$	$\{q \in \mathcal{C} \mid -x_t + 2y_t - 4 \leq 0\}$
VE	a_1	b_2-b_3	$[-1, 0]$	$[2 + x_t, y_t - 1]$	$\{q \in \mathcal{C} \mid -x_t - 2 \leq 0\}$
EV	b_3	a_1-a_2	$[1, 1]$	$[-1 - x_t, -y_t]$	$\{q \in \mathcal{C} \mid -x_t - y_t - 1 \leq 0\}$
VE	a_2	b_3-b_4	$[0, -1]$	$[x_t + 1, y_t + 2]$	$\{q \in \mathcal{C} \mid -y_t - 2 \leq 0\}$
EV	b_4	a_2-a_3	$[-2, 1]$	$[2 - x_t, -y_t]$	$\{q \in \mathcal{C} \mid 2x_t - y_t - 4 \leq 0\}$

Figure 4.19: The various contact conditions are shown in the order as the edge normals appear around $\mathbb{S}^1$ (using inward normals for $\mathcal{A}$ and outward normals for $\mathcal{O}$).

polygon with seven sides. The contact conditions that occur are shown in Figure 4.19. The ordering as the normals appear around $\mathbb{S}^1$ (using inward edge normals for $\mathcal{A}$ and outward edge normals for $\mathcal{O}$). The $\mathcal{C}_{obs}$ edges and their corresponding contact types are shown in Figure 4.19. ■

A polyhedral C-space obstacle Most of the previous ideas generalize nicely for the case of a polyhedral robot that is capable of translation only in a 3D world that contains polyhedral obstacles. If $\mathcal{A}$ and $\mathcal{O}$ are convex polyhedra, the resulting $\mathcal{C}_{obs}$ is a convex polyhedron.

There are three different kinds of contacts that each lead to half-spaces in $\mathcal{C}$:

1. **Type FV:** A face of $\mathcal{A}$ and a vertex of $\mathcal{O}$
2. **Type VF:** A vertex of $\mathcal{A}$ and a face of $\mathcal{O}$
3. **Type EE:** An edge of $\mathcal{A}$ and an edge of $\mathcal{O}$.

These are shown in Figure 4.20. Each half-space defines a face of the polyhedron, $\mathcal{C}_{obs}$. The representation of $\mathcal{C}_{obs}$ can be constructed in $O(n+m+k)$ time, in which

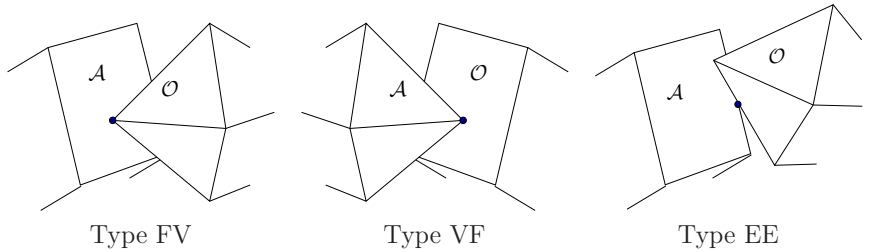


Figure 4.20: Three different types of contact, each of which generates a different kind of $\mathcal{C}_{obs}$ face.

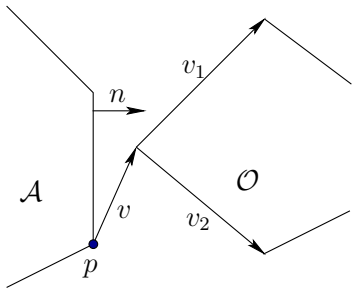


Figure 4.21: An illustration to help in constructing $\mathcal{C}_{obs}$ when rotation is allowed.

n is the number of faces of $\mathcal{A}$, m is the number of faces of $\mathcal{O}$, and k is the number of faces of $\mathcal{C}_{obs}$, which is at most nm [211].

4.3.3 Explicitly Modeling $\mathcal{C}_{obs}$: The General Case

Unfortunately, the cases in which $\mathcal{C}_{obs}$ is polygonal or polyhedral are quite limited. Most problems yield extremely complicated C-space obstacles. One good point is that $\mathcal{C}_{obs}$ can be expressed using semi-algebraic models, for any robots and obstacles defined using semi-algebraic models, even after applying any of the transformations from Sections 3.2 to 3.4. It might not be true, however, for other kinds of transformations, such as warping a flexible material [25, 300].

Consider the case of a convex polygonal robot and a convex polygonal obstacle in a 2D world. Assume that any transformation in $SE(2)$ may be applied to $\mathcal{A}$; thus, $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$ and $q = (x_t, y_t, \theta)$. The task is to define a set of algebraic primitives that can be combined to define $\mathcal{C}_{obs}$. Once again, it is important to distinguish between Type EV and Type VE contacts. Consider how to construct the algebraic primitives for the Type EV contacts; Type VE can be handled in a similar manner.

For the translation-only case, we were able to determine all of the Type EV

contacts by sorting the edge normals. With rotation, the ordering of edge normals depends on θ. This implies that the applicability of a Type EV contact depends on θ, the robot orientation. Recall the constraint that the inward normal of $\mathcal{A}$ must point between the outward normals of the edges of $\mathcal{O}$ that contain the vertex of contact, as shown in Figure 4.21. This constraint can be expressed in terms of inner products using the vectors v_1 and v_2. The statement regarding the directions of the normals can equivalently be formulated as the statement that the angle between n and v_1, and between n and v_2, must each be less than $\pi/2$. Using inner products, this implies that $n \cdot v_1 \geq 0$ and $n \cdot v_2 \geq 0$. As in the translation case, the condition $n \cdot v = 0$ is required for contact. Observe that n now depends on θ. For any $q \in \mathcal{C}$, if $n(\theta) \cdot v_1 \geq 0$, $n(\theta) \cdot v_2 \geq 0$, and $n(\theta) \cdot v(q) > 0$, then $q \in \mathcal{C}_{free}$. Let H_f denote the set of configurations that satisfy these conditions. These conditions imply that a point is in $\mathcal{C}_{free}$. Furthermore, any other Type EV and Type VE contacts could imply that more points are in $\mathcal{C}_{free}$. Ordinarily, $H_f \subset \mathcal{C}_{free}$, which implies that the complement, $\mathcal{C} \setminus H_f$, is a superset of $\mathcal{C}_{obs}$ (thus, $\mathcal{C}_{obs} \subset \mathcal{C} \setminus H_f$). Let $H_A = \mathcal{C} \setminus H_f$. Using the primitives

$$H_1 = \{q \in \mathcal{C} \mid n(\theta) \cdot v_1 \leq 0\}, \tag{4.39}$$

$$H_2 = \{q \in \mathcal{C} \mid n(\theta) \cdot v_2 \leq 0\}, \tag{4.40}$$

and

$$H_3 = \{q \in \mathcal{C} \mid n(\theta) \cdot v(q) \leq 0\}, \tag{4.41}$$

let $H_A = H_1 \cup H_2 \cup H_3$.

It is known that $\mathcal{C}_{obs} \subseteq H_A$, but H_A may contain points in $\mathcal{C}_{free}$. The situation is similar to what was explained in Section 3.1.1 for building a model of a convex polygon from half-planes. In the current setting, it is only known that any configuration outside of H_A must be in $\mathcal{C}_{free}$. If H_A is intersected with all other corresponding sets for each possible Type EV and Type VE contact, then the result is $\mathcal{C}_{obs}$. Each contact has the opportunity to remove a portion of $\mathcal{C}_{free}$ from consideration. Eventually, enough pieces of $\mathcal{C}_{free}$ are removed so that the only configurations remaining must lie in $\mathcal{C}_{obs}$. For any Type EV contact, $(H_1 \cup H_2) \setminus H_3 \subseteq \mathcal{C}_{free}$. A similar statement can be made for Type VE contacts. A logical predicate, similar to that defined in Section 3.1.1, can be constructed to determine whether $q \in \mathcal{C}_{obs}$ in time that is linear in the number of $\mathcal{C}_{obs}$ primitives.

One important issue remains. The expression $n(\theta)$ is not a polynomial because of the $\cos\theta$ and $\sin\theta$ terms in the rotation matrix of $SO(2)$. If polynomials could be substituted for these expressions, then everything would be fixed because the expression of the normal vector (not a unit normal) and the inner product are both linear functions, thereby transforming polynomials into polynomials. Such a substitution can be made using stereographic projection (see [304]); however, a simpler approach is to use complex numbers to represent rotation. Recall that when $a + bi$ is used to represent rotation, each rotation matrix in $SO(2)$ is represented as (4.18), and the 3×3 homogeneous transformation matrix becomes

$$T(a, b, x_t, y_t) = \begin{pmatrix} a & -b & x_t \\ b & a & y_t \\ 0 & 0 & 1 \end{pmatrix}. \tag{4.42}$$

Using this matrix to transform a point $[x \; y \; 1]$ results in the point coordinates $(ax - by + x_t, bx + ay + y_t)$. Thus, any transformed point on $\mathcal{A}$ is a linear function of a, b, x_t, and y_t.

This was a simple trick to make a nice, linear function, but what was the cost? The dependency is now on a and b instead of θ. This appears to increase the dimension of $\mathcal{C}$ from 3 to 4, and $\mathcal{C} = \mathbb{R}^4$. However, an algebraic primitive must be added that constrains a and b to lie on the unit circle.

By using complex numbers, primitives in $\mathbb{R}^4$ are obtained for each Type EV and Type VE contact. By defining $\mathcal{C} = \mathbb{R}^4$, the following algebraic primitives are obtained for a Type EV contact:

$$H_1 = \{(x_t, y_t, a, b) \in \mathcal{C} \mid n(x_t, y_t, a, b) \cdot v_1 \leq 0\}, \tag{4.43}$$

$$H_2 = \{(x_t, y_t, a, b) \in \mathcal{C} \mid n(x_t, y_t, a, b) \cdot v_2 \leq 0\}, \tag{4.44}$$

and

$$H_3 = \{(x_t, y_t, a, b) \in \mathcal{C} \mid n(x_t, y_t, a, b) \cdot v(x_t, y_t, a, b) \leq 0\}. \tag{4.45}$$

This yields $H_A = H_1 \cup H_2 \cup H_3$. To preserve the correct $\mathbb{R}^2 \times \mathbb{S}^1$ topology of $\mathcal{C}$, the set

$$H_s = \{(x_t, y_t, a, b) \in \mathcal{C} \mid a^2 + b^2 - 1 = 0\} \tag{4.46}$$

is intersected with H_A. The set H_s remains fixed over all Type EV and Type VE contacts; therefore, it only needs to be considered once.

Example 4.16 (A Nonlinear Boundary for $\mathcal{C}_{obs}$) Consider adding rotation to the model described in Example 4.15. In this case, all possible contacts between pairs of edges must be considered. For this example, there are 12 Type EV contacts and 12 Type VE contacts. Each contact produces 3 algebraic primitives. With the inclusion of H_s, this simple example produces 73 primitives! Rather than construct all of these, we derive the primitives for a single contact. Consider the Type VE contact between a_3 and b_4-b_1. The outward edge normal n remains fixed at $n = [1, 0]$. The vectors v_1 and v_2 are derived from the edges adjacent to a_3, which are a_3-a_2 and a_3-a_1. Note that each of a_1, a_2, and a_3 depend on the configuration. Using the 2D homogeneous transformation (3.35), a_1 at configuration (x_t, y_t, θ) is $(\cos\theta + x_t, \sin\theta + y_t)$. Using $a + bi$ to represent rotation, the expression of a_1 becomes $(a + x_t, b + y_t)$. The expressions of a_2 and a_3 are $(-b + x_t, a + y_t)$ and $(-a + b + x_t, -b - a + y_t)$, respectively. It follows that $v_1 = a_2 - a_3 = [a - 2b, 2a + b]$ and $v_2 = a_1 - a_3 = [2a - b, a + 2b]$. Note that v_1 and v_2 depend only on the orientation of $\mathcal{A}$, as expected. Assume that v is drawn from b_4 to a_3. This yields

$v = a_3 - b_4 = [-a + b + x_t - 1, -a - b + y_t + 1]$. The inner products $v_1 \cdot n$, $v_2 \cdot n$, and $v \cdot n$ can easily be computed to form H_1, H_2, and H_3 as algebraic primitives.

One interesting observation can be made here. The only nonlinear primitive is $a^2 + b^2 = 1$. Therefore, $\mathcal{C}_{obs}$ can be considered as a linear polytope (like a polyhedron, but one dimension higher) in $\mathbb{R}^4$ that is intersected with a cylinder. ■

3D rigid bodies For the case of a 3D rigid body to which any transformation in $SE(3)$ may be applied, the same general principles apply. The quaternion parameterization once again becomes the right way to represent $SO(3)$ because using (4.20) avoids all trigonometric functions in the same way that (4.18) avoided them for $SO(2)$. Unfortunately, (4.20) is not linear in the configuration variables, as it was for (4.18), but it is at least polynomial. This enables semi-algebraic models to be formed for $\mathcal{C}_{obs}$. Type FV, VF, and EE contacts arise for the $SE(3)$ case. From all of the contact conditions, polynomials that correspond to each patch of $\mathcal{C}_{obs}$ can be made. These patches are polynomials in seven variables: x_t, y_t, z_t, a, b, c, and d. Once again, a special primitive must be intersected with all others; here, it enforces the constraint that unit quaternions are used. This reduces the dimension from 7 back down to 6. Also, constraints should be added to throw away half of $\mathbb{S}^3$, which is redundant because of the identification of antipodal points on $\mathbb{S}^3$.

Chains and trees of bodies For chains and trees of bodies, the ideas are conceptually the same, but the algebra becomes more cumbersome. Recall that the transformation for each link is obtained by a product of homogeneous transformation matrices, as given in (3.53) and (3.57) for the 2D and 3D cases, respectively. If the rotation part is parameterized using complex numbers for $SO(2)$ or quaternions for $SO(3)$, then each matrix consists of polynomial entries. After the matrix product is formed, polynomial expressions in terms of the configuration variables are obtained. Therefore, a semi-algebraic model can be constructed. For each link, all of the contact types need to be considered. Extrapolating from Examples 4.15 and 4.16, you can imagine that no human would ever want to do all of that by hand, but it can at least be automated. The ability to construct this representation automatically is also very important for the existence of theoretical algorithms that solve the motion planning problem combinatorially; see Section 6.4.

If the kinematic chains were formulated for $\mathcal{W} = \mathbb{R}^3$ using the DH parameterization, it may be inconvenient to convert to the quaternion representation. One way to avoid this is to use complex numbers to represent each of the θ_i and α_i variables that appear as configuration variables. This can be accomplished because only cos and sin functions appear in the transformation matrices. They can be replaced by the real and imaginary parts, respectively, of a complex number. The dimension will be increased, but this will be appropriately reduced after imposing the constraints that all complex numbers must have unit magnitude.

4.4 Closed Kinematic Chains

This section continues the discussion from Section 3.4. Suppose that a collection of links is arranged in a way that forms loops. In this case, the C-space becomes much more complicated because the joint angles must be chosen to ensure that the loops remain closed. This leads to constraints such as that shown in (3.80) and Figure 3.26, in which some links must maintain specified positions relative to each other. Consider the set of all configurations that satisfy such constraints. Is this a manifold? It turns out, unfortunately, that the answer is generally *no*. However, the C-space belongs to a nice family of spaces from algebraic geometry called *varieties*. Algebraic geometry deals with characterizing the solution sets of polynomials. As seen so far in this chapter, all of the kinematics can be expressed as polynomials. Therefore, it may not be surprising that the resulting constraints are a system of polynomials whose solution set represents the C-space for closed kinematic linkages. Although the algebraic varieties considered here need not be manifolds, they can be decomposed into a finite collection of manifolds that fit together nicely.[11]

Unfortunately, a parameterization of the variety that arises from closed chains is available in only a few simple cases. Even the topology of the variety is extremely difficult to characterize. To make matters worse, it was proved in [253] that for every closed, bounded real algebraic variety that can be embedded in $\mathbb{R}^n$, there exists a linkage whose C-space is homeomorphic to it. These troubles imply that most of the time, motion planning algorithms need to work directly with implicit polynomials. For the algebraic methods of Section 6.4.2, this does not pose any conceptual difficulty because the methods already work directly with polynomials. Sampling-based methods usually rely on the ability to efficiently sample configurations, which cannot be easily adapted to a variety without a parameterization. Section 7.4 covers recent methods that extend sampling-based planning algorithms to work for varieties that arise from closed chains.

4.4.1 Mathematical Concepts

To understand varieties, it will be helpful to have definitions of polynomials and their solutions that are more formal than the presentation in Chapter 3.

Fields Polynomials are usually defined over a *field*, which is another object from algebra. A field is similar to a group, but it has more operations and axioms. The definition is given below, and while reading it, keep in mind several familiar

[11] This is called a Whitney stratification [92, 473].

examples of fields: the rationals, $\mathbb{Q}$; the reals, $\mathbb{R}$; and the complex plane, $\mathbb{C}$. You may verify that these fields satisfy the following six axioms.

A *field* is a set $\mathbb{F}$ that has two binary operations, $\cdot : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (called *multiplication*) and $+ : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ (called *addition*), for which the following axioms are satisfied:

1. (**Associativity**) For all $a, b, c \in \mathbb{F}$, $(a + b) + c = a + (b + c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
2. (**Commutativity**) For all $a, b \in \mathbb{F}$, $a + b = b + a$ and $a \cdot b = b \cdot a$.
3. (**Distributivity**) For all $a, b, c \in \mathbb{F}$, $a \cdot (b + c) = a \cdot b + a \cdot c$.
4. (**Identities**) There exist $0, 1 \in \mathbb{F}$, such that $a + 0 = a \cdot 1 = a$ for all $a \in \mathbb{F}$.
5. (**Additive Inverses**) For every $a \in \mathbb{F}$, there exists some $b \in \mathbb{F}$ such that $a + b = 0$.
6. (**Multiplicative Inverses**) For every $a \in F$, except $a = 0$, there exists some $c \in \mathbb{F}$ such that $a \cdot c = 1$.

Compare these axioms to the group definition from Section 4.2.1. Note that a field can be considered as two different kinds of groups, one with respect to multiplication and the other with respect to addition. Fields additionally require commutativity; hence, we cannot, for example, build a field from quaternions. The distributivity axiom appears because there is now an interaction between two different operations, which was not possible with groups.

Polynomials Suppose there are n variables, $x_1, x_2, \ldots, x_n$. A *monomial* over a field $\mathbb{F}$ is a product of the form

$$x_1^{d_1} \cdot x_2^{d_2} \cdot \cdots \cdot x_n^{d_n}, \tag{4.47}$$

in which all of the exponents d_1, d_2, ..., d_n are positive integers. The *total degree* of the monomial is $d_1 + \cdots + d_n$.

A *polynomial* f in variables $x_1, \ldots, x_n$ with coefficients in $\mathbb{F}$ is a finite linear combination of monomials that have coefficients in $\mathbb{F}$. A polynomial can be expressed as

$$\sum_{i=1}^{m} c_i m_i, \tag{4.48}$$

in which m_i is a monomial as shown in (4.47), and $c_i \in \mathbb{F}$ is a *coefficient*. If $c_i \neq 0$, then each $c_i m_i$ is called a *term*. Note that the exponents d_i may be different for every term of f. The *total degree of* f is the maximum total degree among the monomials of the terms of f. The set of all polynomials in $x_1, \ldots, x_n$ with coefficients in $\mathbb{F}$ is denoted by $\mathbb{F}[x_1, \ldots, x_n]$.

Example 4.17 (Polynomials) The definitions correspond exactly to our intuitive notion of a polynomial. For example, suppose $\mathbb{F} = \mathbb{Q}$. An example of a polynomial in $\mathbb{Q}[x_1, x_2, x_3]$ is

$$x_1^4 - \tfrac{1}{2} x_1 x_2 x_3^3 + x_1^2 x_2^2 + 4. \tag{4.49}$$

Note that 1 is a valid monomial; hence, any element of $\mathbb{F}$ may appear alone as a term, such as the $4 \in \mathbb{Q}$ in the polynomial above. The total degree of (4.49) is 5 due to the second term. An equivalent polynomial may be written using nicer variables. Using x, y, and z as variables yields

$$x^4 - \tfrac{1}{2} x y z^3 + x^2 y^2 + 4, \tag{4.50}$$

which belongs to $\mathbb{Q}[x, y, z]$. ■

The set $\mathbb{F}[x_1, \ldots, x_n]$ of polynomials is actually a group with respect to addition; however, it is not a field. Even though polynomials can be multiplied, some polynomials do not have a multiplicative inverse. Therefore, the set $\mathbb{F}[x_1, \ldots, x_n]$ is often referred to as a *commutative ring* of polynomials. A commutative ring is a set with two operations for which every axiom for fields is satisfied except the last one, which would require a multiplicative inverse.

Varieties For a given field $\mathbb{F}$ and positive integer n, the n-dimensional *affine space* over $\mathbb{F}$ is the set

$$\mathbb{F}^n = \{(c_1, \ldots, c_n) \mid c_1, \ldots, c_n \in \mathbb{F}\}. \tag{4.51}$$

For our purposes in this section, an affine space can be considered as a vector space (for an exact definition, see [225]). Thus, $\mathbb{F}^n$ is like a vector version of the scalar field $\mathbb{F}$. Familiar examples of this are $\mathbb{Q}^n$, $\mathbb{R}^n$, and $\mathbb{C}^n$.

A polynomial in $f \in \mathbb{F}[x_1, \ldots, x_n]$ can be converted into a function,

$$f : \mathbb{F}^n \to \mathbb{F}, \tag{4.52}$$

by substituting elements of $\mathbb{F}$ for each variable and evaluating the expression using the field operations. This can be written as $f(a_1, \ldots, a_n) \in \mathbb{F}$, in which each a_i denotes an element of $\mathbb{F}$ that is substituted for the variable x_i.

We now arrive at an interesting question. For a given f, what are the elements of $\mathbb{F}^n$ such that $f(a_1, \ldots, a_n) = 0$? We could also ask the question for some nonzero element, but notice that this is not necessary because the polynomial may be redefined to formulate the question using 0. For example, what are the elements of $\mathbb{R}^2$ such that $x^2 + y^2 = 1$? This familiar equation for $\mathbb{S}^1$ can be reformulated to yield: What are the elements of $\mathbb{R}^2$ such that $x^2 + y^2 - 1 = 0$?

Let $\mathbb{F}$ be a field and let $\{f_1, \ldots, f_k\}$ be a set of polynomials in $\mathbb{F}[x_1, \ldots, x_n]$. The set

$$V(f_1, \ldots, f_k) = \{(a_1, \ldots, a_n) \in \mathbb{F} \mid f_i(a_1, \ldots, a_n) = 0 \text{ for all } 1 \leq i \leq k\} \tag{4.53}$$

is called the *(affine) variety* defined by $f_1, \ldots, f_k$. One interesting fact is that unions and intersections of varieties are varieties. Therefore, they behave like the semi-algebraic sets from Section 3.1.2, but for varieties only equality constraints are allowed. Consider the varieties $V(f_1, \ldots, f_k)$ and $V(g_1, \ldots, g_l)$. Their intersection is given by

$$V(f_1, \ldots, f_k) \cap V(g_1, \ldots, g_l) = V(f_1, \ldots, f_k, g_1, \ldots, g_l), \tag{4.54}$$

because each element of $\mathbb{F}^n$ must produce a 0 value for each of the polynomials in $\{f_1, \ldots, f_k, g_1, \ldots, g_l\}$.

To obtain unions, the polynomials simply need to be multiplied. For example, consider the varieties $V_1, V_2 \subset \mathbb{F}$ defined as

$$V_1 = \{(a_1, \ldots, a_n) \in \mathbb{F} \mid f_1(a_1, \ldots, a_n) = 0\} \tag{4.55}$$

and

$$V_2 = \{(a_1, \ldots, a_n) \in \mathbb{F} \mid f_2(a_1, \ldots, a_n) = 0\}. \tag{4.56}$$

The set $V_1 \cup V_2 \subset \mathbb{F}$ is obtained by forming the polynomial $f = f_1 f_2$. Note that $f(a_1, \ldots, a_n) = 0$ if either $f_1(a_1, \ldots, a_n) = 0$ or $f_2(a_1, \ldots, a_n) = 0$. Therefore, $V_1 \cup V_2$ is a variety. The varieties V_1 and V_2 were defined using a single polynomial, but the same idea applies to any variety. All pairs of the form $f_i g_j$ must appear in the argument of $V(\cdot)$ if there are multiple polynomials.

4.4.2 Kinematic Chains in $\mathbb{R}^2$

To illustrate the concepts it will be helpful to study a simple case in detail. Let $\mathcal{W} = \mathbb{R}^2$, and suppose there is a chain of links, $\mathcal{A}_1$, ..., $\mathcal{A}_n$, as considered in Example 3.3 for $n = 3$. Suppose that the first link is attached at the origin of $\mathcal{W}$ by a revolute joint, and every other link, $\mathcal{A}_i$ is attached to $\mathcal{A}_{i-1}$ by a revolute joint. This yields the C-space

$$\mathcal{C} = \mathbb{S}^1 \times \mathbb{S}^1 \times \cdots \times \mathbb{S}^1 = \mathbb{T}^n, \tag{4.57}$$

which is the n-dimensional torus.

Two links If there are two links, $\mathcal{A}_1$ and $\mathcal{A}_2$, then the C-space can be nicely visualized as a square with opposite faces identified. Each coordinate, θ_1 and θ_2, ranges from 0 to 2π, for which $0 \sim 2\pi$. Suppose that each link has length 1. This yields $a_1 = 1$. A point $(x, y) \in \mathcal{A}_2$ is transformed as

$$\begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 1 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{4.58}$$

To obtain polynomials, the technique from Section 4.2.2 is applied to replace the trigonometric functions using $a_i = \cos\theta_i$ and $b_i = \sin\theta_i$, subject to the constraint $a_i^2 + b_i^2 = 1$. This results in

$$\begin{pmatrix} a_1 & -b_1 & 0 \\ b_1 & a_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_2 & -b_2 & 1 \\ b_2 & a_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{4.59}$$

for which the constraints $a_i^2 + b_i^2 = 1$ for $i = 1, 2$ must be satisfied. This preserves the torus topology of $\mathcal{C}$, but now the C-space is embedded in $\mathbb{R}^4$. The coordinates of each point are $(a_1, b_1, a_2, b_2) \in \mathbb{R}^4$; however, there are only two degrees of freedom because each a_i, b_i pair must lie on a unit circle.

Multiplying the matrices in (4.59) yields the polynomials, $f_1, f_2 \in \mathbb{R}[a_1, b_1, a_2, b_2]$,

$$f_1 = x a_1 a_2 - y a_1 b_2 - x b_1 b_2 + y a_2 b_1 + a_1 \tag{4.60}$$

and

$$f_2 = -y a_1 a_2 + x a_1 b_2 + x a_2 b_1 - y b_1 b_2 + b_1, \tag{4.61}$$

for the x and y coordinates, respectively. Note that the polynomial variables are configuration parameters; x and y are not polynomial variables. For a given point $(x, y) \in \mathcal{A}_2$, all coefficients are determined.

A zero-dimensional variety Now a kinematic closure constraint will be imposed. Fix the point $(1, 0)$ in the body frame of $\mathcal{A}_2$ at $(1, 1)$ in $\mathcal{W}$. This yields the constraints

$$f_1 = a_1 a_2 - b_1 b_2 + a_1 = 1 \tag{4.62}$$

and

$$f_2 = a_1 b_2 + a_2 b_1 + b_1 = 1, \tag{4.63}$$

by substituting $x = 1$ and $y = 0$ into (4.60) and (4.61). This yields the variety

$$V(a_1 a_2 - b_1 b_2 + a_1 - 1, a_1 b_2 + a_2 b_1 + b_1 - 1, a_1^2 + b_1^2 - 1, a_2^2 + b_2^2 - 1), \tag{4.64}$$

which is a subset of $\mathbb{R}^4$. The polynomials are slightly modified because each constraint must be written in the form $f = 0$.

Although (4.64) represents the constrained configuration space for the chain of two links, it is not very explicit. Without an explicit characterization (i.e., a parameterization), it complicates motion planning. From Figure 4.22 it can be seen that there are only two solutions. These occur for $\theta_1 = 0$, $\theta_2 = \pi/2$ and $\theta_1 = \pi/2$, $\theta_2 = -\pi/2$. In terms of the polynomial variables, (a_1, b_1, a_2, b_2), the two solutions are $(1, 0, 0, 1)$ and $(0, 1, 0, -1)$. These may be substituted into each polynomial in (4.64) to verify that 0 is obtained. Thus, the variety represents two points in $\mathbb{R}^4$. This can also be interpreted as two points on the torus, $\mathbb{S}^1 \times \mathbb{S}^1$.

It might not be surprising that the set of solutions has dimension zero because there are four independent constraints, shown in (4.64), and four variables. Depending on the choices, the variety may be empty. For example, it is physically impossible to bring the point $(1, 0) \in \mathcal{A}_2$ to $(1000, 0) \in \mathcal{W}$.

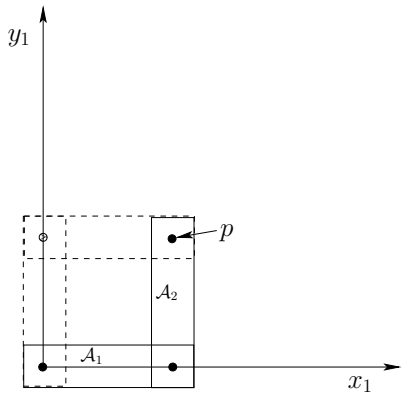


Figure 4.22: Two configurations hold the point p at $(1, 1)$.

A one-dimensional variety The most interesting and complicated situations occur when there is a continuum of solutions. For example, if one of the constraints is removed, then a one-dimensional set of solutions can be obtained. Suppose only one variable is constrained for the example in Figure 4.22. Intuitively, this should yield a one-dimensional variety. Set the x coordinate to 0, which yields

$$a_1a_2 - b_1b_2 + a_1 = 0, \tag{4.65}$$

and allow any possible value for y. As shown in Figure 4.23a, the point p must follow the y-axis. (This is equivalent to a three-bar linkage that can be constructed by making a third joint that is prismatic and forced to stay along the y-axis.) Figure 4.23b shows the resulting variety $V(a_1a_2 - b_1b_2 + a_1)$ but plotted in $\theta_1 - \theta_2$ coordinates to reduce the dimension from 4 to 2 for visualization purposes. To correctly interpret the figures in Figure 4.23, recall that the topology is $\mathbb{S}^1 \times \mathbb{S}^1$, which means that the top and bottom are identified, and also the sides are identified. The center of Figure 4.23b, which corresponds to $(\theta_1, \theta_2) = (\pi, \pi)$, prevents the variety from being a manifold. The resulting space is actually homeomorphic to two circles that touch at a point. Thus, even with such a simple example, the nice manifold structure may disappear. Observe that at (π, π) the links are completely overlapped, and the point p of $\mathcal{A}_2$ is placed at $(0, 0)$ in $\mathcal{W}$. The horizontal line in Figure 4.23b corresponds to keeping the two links overlapping and swinging them around together by varying θ_1. The diagonal lines correspond to moving along configurations such as the one shown in Figure 4.23a. Note that the links and the y-axis always form an isosceles triangle, which can be used to show that the solution set is any pair of angles, θ_1, θ_2 for which $\theta_2 = \pi - \theta_1$. This is the reason why the diagonal curves in Figure 4.23b are linear. Figures 4.23c and

4.23d show the varieties for the constraints

$$a_1a_2 - b_1b_2 + a_1 = \tfrac{1}{8}, \tag{4.66}$$

and

$$a_1a_2 - b_1b_2 + a_1 = 1, \tag{4.67}$$

respectively. In these cases, the point $(0, 1)$ in $\mathcal{A}_2$ must follow the $x = 1/8$ and $x = 1$ axes, respectively. The varieties are manifolds, which are homeomorphic to $\mathbb{S}^1$. The sequence from Figure 4.23b to 4.23d can be imagined as part of an animation in which the variety shrinks into a small circle. Eventually, it shrinks to a point for the case $a_1a_2 - b_1b_2 + a_1 = 2$, because the only solution is when $\theta_1 = \theta_2 = 0$. Beyond this, the variety is the empty set because there are no solutions. Thus, by allowing one constraint to vary, four different topologies are obtained: 1) two circles joined at a point, 2) a circle, 3) a point, and 4) the empty set.

Three links Since visualization is still possible with one more dimension, suppose there are three links, $\mathcal{A}_1$, $\mathcal{A}_2$, and $\mathcal{A}_3$. The C-space can be visualized as a 3D cube with opposite faces identified. Each coordinate θ_i ranges from 0 to 2π, for which $0 \sim 2\pi$. Suppose that each link has length 1 to obtain $a_1 = a_2 = 1$. A point $(x, y) \in \mathcal{A}_3$ is transformed as

$$\begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 10 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_3 & -\sin\theta_3 & 10 \\ \sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{4.68}$$

To obtain polynomials, let $a_i = \cos\theta_i$ and $b_i = \sin\theta_i$, which results in

$$\begin{pmatrix} a_1 & -b_1 & 0 \\ b_1 & a_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_2 & -b_2 & 1 \\ b_2 & a_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_3 & -b_3 & 1 \\ b_3 & a_3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{4.69}$$

for which the constraints $a_i^2 + b_i^2 = 1$ for $i = 1, 2, 3$ must also be satisfied. This preserves the torus topology of $\mathcal{C}$, but now it is embedded in $\mathbb{R}^6$. Multiplying the matrices yields the polynomials $f_1, f_2 \in \mathbb{R}[a_1, b_1, a_2, b_2, a_3, b_3]$, defined as

$$f_1 = 2a_1a_2a_3 - a_1b_2b_3 + a_1a_2 - 2b_1b_2a_3 - b_1a_2b_3 + a_1, \tag{4.70}$$

and

$$f_2 = 2b_1a_2a_3 - b_1b_2b_3 + b_1a_2 + 2a_1b_2a_3 + a_1a_2b_3, \tag{4.71}$$

for the x and y coordinates, respectively.

Again, consider imposing a single constraint,

$$2a_1a_2a_3 - a_1b_2b_3 + a_1a_2 - 2b_1b_2a_3 - b_1a_2b_3 + a_1 = 0, \tag{4.72}$$

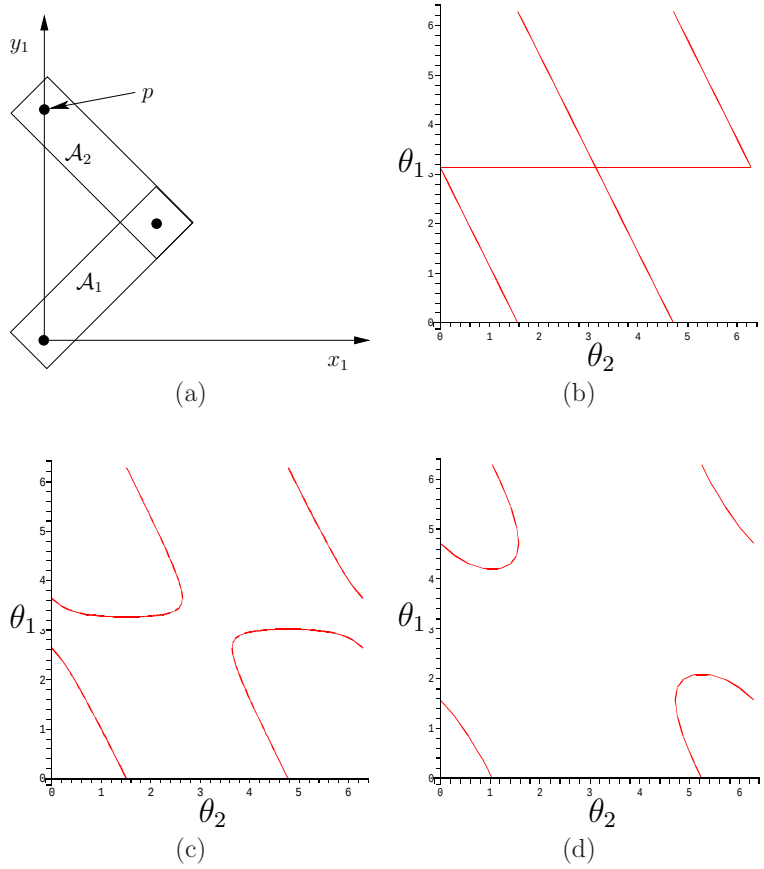


Figure 4.23: A single constraint was added to the point p on $\mathcal{A}_2$, as shown in (a). The curves in (b), (c), and (d) depict the variety for the cases of $f_1 = 0$, $f_1 = 1/8$, and $f_1 = 1$, respectively.

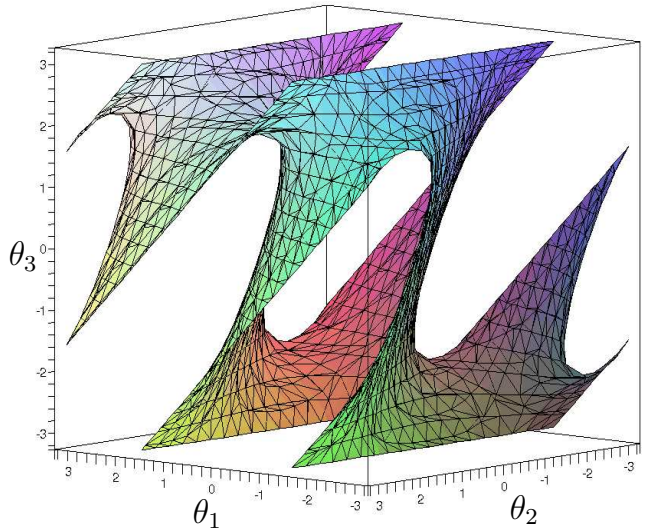


Figure 4.24: The variety for the three-link chain with $f_1 = 0$ is a 2D manifold.

which constrains the point $(1, 0) \in \mathcal{A}_3$ to traverse the y-axis. The resulting variety is an interesting manifold, depicted in Figure 4.24 (remember that the sides of the cube are identified).

Increasing the required f_1 value for the constraint on the final point causes the variety to shrink. Snapshots for $f_1 = 7/8$ and $f_1 = 2$ are shown in Figure 4.25. At $f_1 = 1$, the variety is not a manifold, but it then changes to $\mathbb{S}^2$. Eventually, this sphere is reduced to a point at $f_1 = 3$, and then for $f_1 > 3$ the variety is empty.

Instead of the constraint $f_1 = 0$, we could instead constrain the y coordinate of p to obtain $f_2 = 0$. This yields another 2D variety. If both constraints are enforced simultaneously, then the result is the intersection of the two original varieties. For example, suppose $f_1 = 1$ and $f_2 = 0$. This is equivalent to a kind of *four-bar mechanism* [163], in which the fourth link, $\mathcal{A}_4$, is fixed along the x-axis from 0 to 1. The resulting variety,

$$\begin{aligned} V(&2a_1a_2a_3 - a_1b_2b_3 + a_1a_2 - 2b_1b_2a_3 - b_1a_2b_3 + a_1 - 1, \\ &2b_1a_2a_3 - b_1b_2b_3 + b_1a_2 + 2a_1b_2a_3 + a_1a_2b_3), \end{aligned} \tag{4.73}$$

is depicted in Figure 4.26. Using the $\theta_1, \theta_2, \theta_3$ coordinates, the solution may be easily parameterized as a collection of line segments. For all $t \in [0, \pi]$, there exist solution points at $(0, 2t, \pi)$, $(t, 2\pi - t, \pi + t)$, $(2\pi - t, t, \pi - t)$, $(2\pi - t, \pi, \pi + t)$, and $(t, \pi, \pi - t)$. Note that once again the variety is not a manifold. A family of interesting varieties can be generated for the four-bar mechanism by selecting

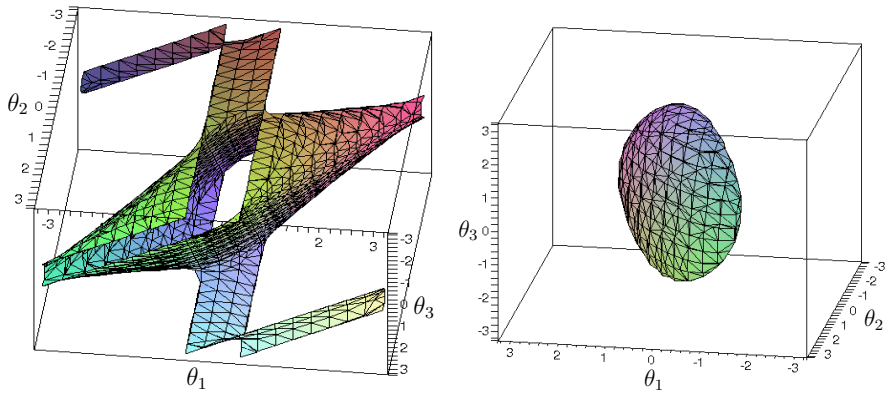


Figure 4.25: If $f_1 > 0$, then the variety shrinks. If $1 < p < 3$, the variety is a sphere. At $f_1 = 0$ it is a point, and for $f_1 > 3$ it completely vanishes.

different lengths for the links. The topologies of these mechanisms have been determined for 2D and a 3D extension that uses spherical joints (see [364]).

4.4.3 Defining the Variety for General Linkages

We now describe a general methodology for defining the variety. Keeping the previous examples in mind will help in understanding the formulation. In the general case, each constraint can be thought of as a statement of the form:

> *The* ith *coordinate of a point* $p \in \mathcal{A}_j$ *needs to be held at the value* x *in the body frame of* $\mathcal{A}_k$.

For the variety in Figure 4.23b, the first coordinate of a point $p \in \mathcal{A}_2$ was held at the value 0 in $\mathcal{W}$ in the body frame of $\mathcal{A}_1$. The general form must also allow a point to be fixed with respect to the body frames of links other than $\mathcal{A}_1$; this did not occur for the example in Section 4.4.2

Suppose that n links, $\mathcal{A}_1, \ldots, \mathcal{A}_n$, move in $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. One link, $\mathcal{A}_1$ for convenience, is designated as the root as defined in Section 3.4. Some links are attached in pairs to form joints. A *linkage graph*, $\mathcal{G}(V, E)$, is constructed from the links and joints. Each vertex of $\mathcal{G}$ represents a link in L. Each edge in $\mathcal{G}$ represents a joint. This definition may seem somewhat backward, especially in the plane because links often look like edges and joints look like vertices. This alternative assignment is also possible, but it is not easy to generalize to the case of a single link that has more than two joints. If more than two links are attached at the same point, each generates an edge of $\mathcal{G}$.

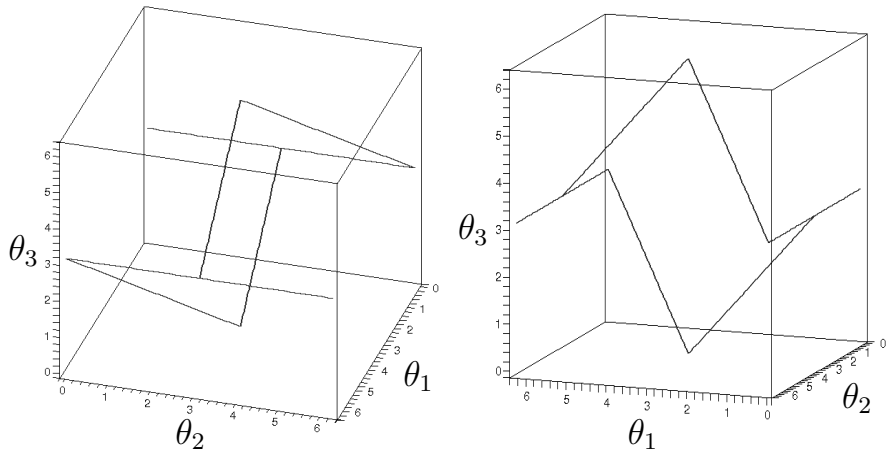


Figure 4.26: If two constraints, $f_1 = 1$ and $f_2 = 0$, are imposed, then the varieties are intersected to obtain a 1D set of solutions. The example is equivalent to a well-studied four-bar mechanism.

The steps to determine the polynomial constraints that express the variety are as follows:

1. Define the linkage graph $\mathcal{G}$ with one vertex per link and one edge per joint. If a joint connects more than two bodies, then one body must be designated as a junction. See Figures 4.27 and 4.28a. In Figure 4.28, links 4, 13, and 23 are designated as junctions in this way.

2. Designate one link as the root, $\mathcal{A}_1$. This link may either be fixed in $\mathcal{W}$, or transformations may be applied. In the latter case, the set of transformations could be $SE(2)$ or $SE(3)$, depending on the dimension of $\mathcal{W}$. This enables the entire linkage to move independently of its internal motions.

3. Eliminate the loops by constructing a spanning tree T of the linkage graph, $\mathcal{G}$. This implies that every vertex (or link) is reachable by a path from the root). Any spanning tree may be used. Figure 4.28b shows a resulting spanning tree after deleting the edges shown with dashed lines.

4. Apply the techniques of Section 3.4 to assign body frames and transformations to the resulting tree of links.

5. For each edge of $\mathcal{G}$ that does not appear in T, write a set of constraints between the two corresponding links. In Figure 4.28b, it can be seen that constraints are needed between four pairs of links: 14–15, 21–22, 23–24, and 19–23.

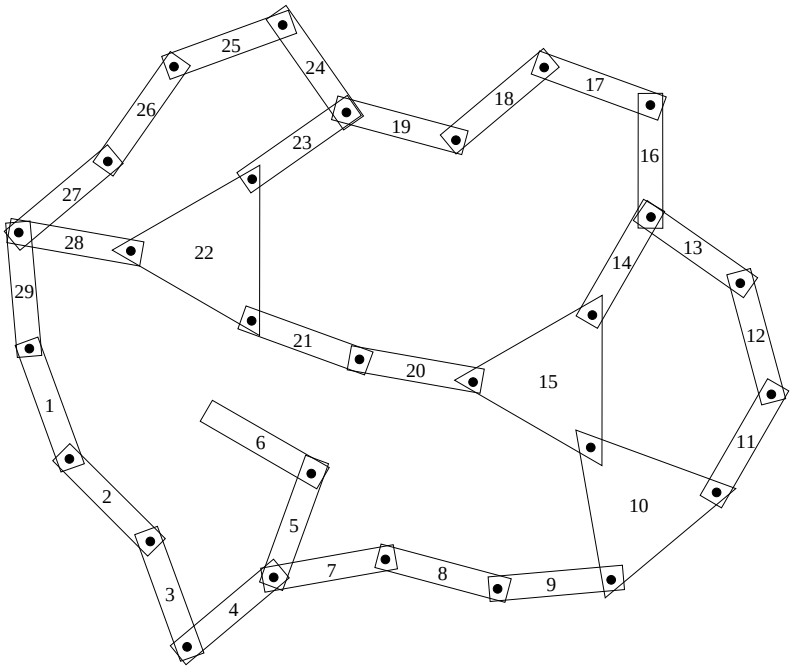


Figure 4.27: A complicated linkage that has 29 links, several loops, links with more than two bodies, and bodies with more than two links. Each integer i indicates link $\mathcal{A}_i$.

This is perhaps the trickiest part. For examples like the one shown in Figure 3.27, the constraint may be formulated as in (3.81). This is equivalent to what was done to obtain the example in Figure 4.26, which means that there are actually two constraints, one for each of the x and y coordinates. This will also work for the example shown in Figure 4.27 if all joints are revolute. Suppose instead that two bodies, $\mathcal{A}_j$ and $\mathcal{A}_k$, must be rigidly attached. This requires adding one more constraint that prevents mutual rotation. This could be achieved by selecting another point on $\mathcal{A}_j$ and ensuring that one of its coordinates is in the correct position in the body frame of $\mathcal{A}_k$. If four equations are added, two from each point, then one of them would be redundant because there are only three degrees of freedom possible for $\mathcal{A}_j$ relative to $\mathcal{A}_k$ (which comes from the dimension of $SE(2)$).

A similar but more complicated situation occurs for $\mathcal{W} = \mathbb{R}^3$. Holding a single point fixed produces three constraints. If a single point is held fixed, then $\mathcal{A}_j$ may achieve any rotation in $SO(3)$ with respect to $\mathcal{A}_k$. This implies

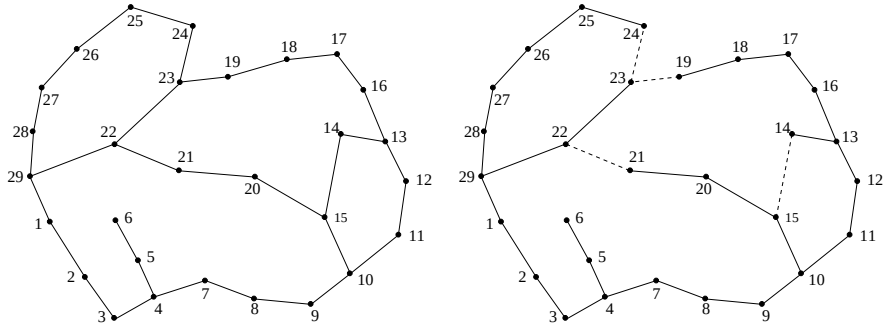


Figure 4.28: (a) One way to make the linkage graph that corresponds to the linkage in Figure 4.27. (b) A spanning tree is indicated by showing the removed edges with dashed lines.

that $\mathcal{A}_j$ and $\mathcal{A}_k$ are attached by a spherical joint. If they are attached by a revolute joint, then two more constraints are needed, which can be chosen from the coordinates of a second point. If $\mathcal{A}_j$ and $\mathcal{A}_k$ are rigidly attached, then one constraint from a third point is needed. In total, however, there can be no more than six independent constraints because this is the dimension of $SE(3)$.

6. Convert the trigonometric functions to polynomials. For any 2D transformation, the familiar substitution of complex numbers may be made. If the DH parameterization is used for the 3D case, then each of the $\cos\theta_i$, $\sin\theta_i$ expressions can be parameterized with one complex number, and each of the $\cos\alpha_i$, $\sin\alpha_i$ expressions can be parameterized with another. If the rotation matrix for $SO(3)$ is directly used in the parameterization, then the quaternion parameterization should be used. In all of these cases, polynomial expressions are obtained.

7. List the constraints as polynomial equations of the form $f = 0$. To write the description of the variety, all of the polynomials must be set equal to zero, as was done for the examples in Section 4.4.2.

Is it possible to determine the dimension of the variety from the number of independent constraints? The answer is generally *no*, which can be easily seen from the chains of links in Section 4.4.2; they produced varieties of various dimensions, depending on the particular equations. Techniques for computing the dimension exist but require much more machinery than is presented here (see the literature overview at the end of the chapter). However, there is a way to provide a simple upper bound on the number of degrees of freedom. Suppose the total

degrees of freedom of the linkage in spanning tree form is m. Each independent constraint can remove at most one degree of freedom. Thus, if there are l independent constraints, then the variety can have no more than $m - l$ dimensions. One expression of this for a general class of mechanisms is the Kutzbach criterion; the planar version of this is called Grübler's formula [163].

One final concern is the obstacle region, $\mathcal{C}_{obs}$. Once the variety has been identified, the obstacle region and motion planning definitions in (4.34) and Formulation 4.1 do not need to be changed. The configuration space $\mathcal{C}$ must be redefined, however, to be the set of configurations that satisfy the closure constraints.

Further Reading

Section 4.1 introduced the basic definitions and concepts of topology. Further study of this fascinating subject can provide a much deeper understanding of configuration spaces. There are many books on topology, some of which may be intimidating, depending on your level of math training. For a heavily illustrated, gentle introduction to topology, see [281]. Another gentle introduction appears in [256]. An excellent text at the graduate level is available on-line: [226]. Other sources include [30, 232]. To understand the motivation for many technical definitions in topology, [449] is helpful. The manifold coverage in Section 4.1.2 was simpler than that found in most sources because most sources introduce *smooth manifolds*, which are complicated by differentiability requirements (these were not needed in this chapter); see Section 8.3.2 for smooth manifolds. For the configuration spaces of points moving on a topological graph, see [3].

Section 4.2 provided basic C-space definitions. For further reading on matrix groups and their topological properties, see [45], which provides a transition into more advanced material on Lie group theory. For more about quaternions in engineering, see [113, 294]. The remainder of Section 4.2 and most of Section 4.3 were inspired by the coverage in [304]. C-spaces are also covered in [118]. For further reading on computing representations of $\mathcal{C}_{obs}$, see [262, 380] for bitmaps, and Chapter 6 and [435] for combinatorial approaches.

Much of the presentation in Section 4.4 was inspired by the nice introduction to algebraic varieties in [138], which even includes robotics examples; methods for determining the dimension of a variety are also covered. More algorithmic coverage appears in [369]. See [360] for detailed coverage of robots that are designed with closed kinematic chains.

Exercises

1. Consider the set $X = \{1, 2, 3, 4, 5\}$. Let X, $\emptyset$, $\{1, 3\}$, $\{1, 2\}$, $\{2, 3\}$, $\{1\}$, $\{2\}$, and $\{3\}$ be the collection of all subsets of X that are designated as *open sets*.

   (a) Is X a topological space?

   (b) Is it a topological space if $\{1, 2, 3\}$ is added to the collection of open sets? Explain.

   (c) What are the closed sets (assuming $\{1, 2, 3\}$ is included as an open set)?

   (d) Are any subsets of X neither open nor closed?

2. Continuous functions for the strange topology:

   (a) Give an example of a continuous function, $f : X \rightarrow X$, for the strange topology in Example 4.4.

   (b) Characterize the set of all possible continuous functions.

3. For the letters of the Russian alphabet, А, Б, В, Г, Д, Е, Ё, Ж, З, И, Й, К, Л, М, Н, О, П, Р, С, Т, У, Ф, Х, Ц, Ч, Ш, Щ, Ъ, Ы, Ь, Э, Ю, Я, determine which pairs are homeomorphic. Imagine each as a 1D subset of $\mathbb{R}^2$ and draw them accordingly before solving the problem.

4. Prove that homeomorphisms yield an equivalence relation on the collection of all topological spaces.

5. What is the dimension of the C-space for a cylindrical rod that can translate and rotate in $\mathbb{R}^3$? If the rod is rotated about its central axis, it is assumed that the rod's position and orientation are not changed in any detectable way. Express the C-space of the rod in terms of a Cartesian product of simpler spaces (such as $\mathbb{S}^1$, $\mathbb{S}^2$, $\mathbb{R}^n$, P^2, etc.). What is your reasoning?

6. Let $\tau_1 : [0, 1] \rightarrow \mathbb{R}^2$ be a loop path that traverses the unit circle in the plane, defined as $\tau_1(s) = (\cos(2\pi s), \sin(2\pi s))$. Let $\tau_2 : [0, 1] \rightarrow \mathbb{R}^2$ be another loop path: $\tau_1(s) = (-2 + 3\cos(2\pi s), \frac{1}{2}\sin(2\pi s))$. This path traverses an ellipse that is centered at $(-2, 0)$. Show that τ_1 and τ_2 are homotopic (by constructing a continuous function with an additional parameter that "morphs" τ_1 into τ_2).

7. Prove that homotopy yields an equivalence relation on the set of all paths from some $x_1 \in X$ to some $x_2 \in X$, in which x_1 and x_2 may be chosen arbitrarily.

8. Determine the C-space for a spacecraft that can translate and rotate in a 2D *Asteroids*-style video game. The sides of the screen are identified. The top and bottom are also identified. There are no "twists" in the identifications.

9. Repeat the derivation of H_A from Section 4.3.3, but instead consider Type VE contacts.

10. Determine the C-space for a car that drives around on a huge sphere (such as the earth with no mountains or oceans). Assume the sphere is big enough so that its curvature may be neglected (e.g., the car rests flatly on the earth without wobbling). [Hint: It is not $\mathbb{S}^2 \times \mathbb{S}^1$.]

11. Suppose that $\mathcal{A}$ and $\mathcal{O}$ are each defined as equilateral triangles, with coordinates $(0, 0)$, $(2, 0)$, and $(1, \sqrt{3})$. Determine the C-space obstacle. Specify the coordinates of all of its vertices and indicate the corresponding contact type for each edge.

12. Show that (4.20) is a valid rotation matrix for all unit quaternions.

13. Show that $\mathbb{F}[x_1, \ldots, x_n]$, the set of polynomials over a field $\mathbb{F}$ with variables $x_1, \ldots, x_n$, is a group with respect to addition.

14. Quaternions:

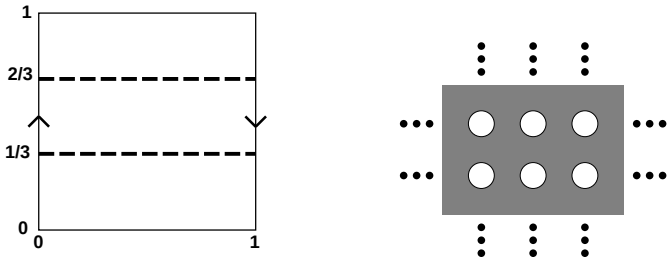


Figure 4.29: (a) What topological space is obtained after slicing the Möbius band? (b) Is a manifold obtained after tearing holes out of the plane?

(a) Define a unit quaternion h_1 that expresses a rotation of $-\frac{\pi}{2}$ around the axis given by the vector $[\frac{1}{\sqrt{3}} \ \frac{1}{\sqrt{3}} \ \frac{1}{\sqrt{3}}]$.

(b) Define a unit quaternion h_2 that expresses a rotation of π around the axis given by the vector $[0 \ 1 \ 0]$.

(c) Suppose the rotation represented by h_1 is performed, followed by the rotation represented by h_2. This combination of rotations can be represented as a single rotation around an axis given by a vector. Find this axis and the angle of rotation about this axis.

15. What topological space is contributed to the C-space by a spherical joint that achieves any orientation except the identity?

16. Suppose five polyhedral bodies float freely in a 3D world. They are each capable of rotating and translating. If these are treated as "one" composite robot, what is the topology of the resulting C-space (assume that the bodies are *not* attached to each other)? What is its dimension?

17. Suppose a goal region $G \subseteq \mathcal{W}$ is defined in the C-space by requiring that the *entire* robot is contained in G. For example, a car may have to be parked entirely within a space in a parking lot.

(a) Give a definition of $\mathcal{C}_{goal}$ that is similar to (4.34) but pertains to containment of $\mathcal{A}$ inside of G.

(b) For the case in which $\mathcal{A}$ and G are convex and polygonal, develop an algorithm for efficiently computing $\mathcal{C}_{goal}$.

18. Figure 4.29a shows the Möbius band defined by identification of sides of the unit square. Imagine that scissors are used to cut the band along the two dashed lines. Describe the resulting topological space. Is it a manifold? Explain.

19. Consider Figure 4.29b, which shows the set of points in $\mathbb{R}^2$ that are remaining after a closed disc of radius $1/4$ with center (x, y) is removed for every value of (x, y) such that x and y are both integers.

(a) Is the remaining set of points a manifold? Explain.

(b) Now remove discs of radius $1/2$ instead of $1/4$. Is a manifold obtained?

(c) Finally, remove disks of radius $2/3$. Is a manifold obtained?

20. Show that the solution curves shown in Figure 4.26 correctly illustrate the variety given in (4.73).

21. Find the number of faces of $\mathcal{C}_{obs}$ for a cube and regular tetrahedron, assuming $\mathcal{C}$ is $SE(3)$. How many faces of each contact type are obtained?

22. Following the analysis matrix subgroups from Section 4.2, determine the dimension of $SO(4)$, the group of 4×4 rotation matrices. Can you characterize this topological space?

23. Suppose that a kinematic chain of spherical joints is given. Show how to use (4.20) as the rotation part in each homogeneous transformation matrix, as opposed to using the DH parameterization. Explain why using (4.20) would be preferable for motion planning applications.

24. Suppose that the constraint that c is held to position $(10, 10)$ is imposed on the mechanism shown in Figure 3.29. Using complex numbers to represent rotation, express this constraint using polynomial equations.

25. The Tangle toy is made of 18 pieces of macaroni-shaped joints that are attached together to form a loop. Each attachment between joints forms a revolute joint. Each link is a curved tube that extends around $1/4$ of a circle. What is the dimension of the variety that results from maintaining the loop? What is its configuration space (accounting for internal degrees of freedom), assuming the toy can be placed anywhere in $\mathbb{R}^3$?

Implementations

26. Computing C-space obstacles:

(a) Implement the algorithm from Section 4.3.2 to construct a convex, polygonal C-space obstacle.

(b) Now allow the robot to rotate in the plane. For any convex robot and obstacle, compute the orientations at which the C-space obstacle fundamentally changes due to different Type EV and Type VE contacts becoming active.

(c) Animate the changing C-space obstacle by using the robot orientation as the time axis in the animation.

27. Consider "straight-line" paths that start at the origin (lower left corner) of the manifolds shown in Figure 4.5 and leave at a particular angle, which is input to the program. The lines must respect identifications; thus, as the line hits the edge of the square, it may continue onward. Study the conditions under which the lines fill the entire space versus forming a finite pattern (i.e., a segment, stripes, or a tiling).

Chapter 5

Sampling-Based Motion Planning

There are two main philosophies for addressing the motion planning problem, in Formulation 4.1 from Section 4.3.1. This chapter presents one of the philosophies, *sampling-based motion planning*, which is outlined in Figure 5.1. The main idea is to avoid the explicit construction of $\mathcal{C}_{obs}$, as described in Section 4.3, and instead conduct a search that probes the C-space with a sampling scheme. This probing is enabled by a collision detection module, which the motion planning algorithm considers as a "black box." This enables the development of planning algorithms that are independent of the particular geometric models. The collision detection module handles concerns such as whether the models are semi-algebraic sets, 3D triangles, nonconvex polyhedra, and so on. This general philosophy has been very successful in recent years for solving problems from robotics, manufacturing, and biological applications that involve thousands and even millions of geometric primitives. Such problems would be practically impossible to solve using techniques that explicitly represent $\mathcal{C}_{obs}$.

Notions of completeness It is useful to define several notions of completeness for sampling-based algorithms. These algorithms have the drawback that they result in weaker guarantees that the problem will be solved. An algorithm is considered *complete* if for any input it correctly reports whether there is a solution in a finite amount of time. If a solution exists, it must return one in finite time. The combinatorial motion planning methods of Chapter 6 will achieve this. Unfortunately, such completeness is not achieved with sampling-based planning. Instead, weaker notions of completeness are tolerated. The notion of denseness becomes important, which means that the samples come arbitrarily close to any configuration as the number of iterations tends to infinity. A deterministic approach that samples densely will be called *resolution complete*. This means that if a solution exists, the algorithm will find it in finite time; however, if a solution does not exist, the algorithm may run forever. Many sampling-based approaches are based on random sampling, which is dense with probability one. This leads to algorithms that are *probabilistically complete*, which means that with enough points, the probability that it finds an existing solution converges to one. The most relevant information, however, is the rate of convergence, which is usually very difficult to establish.

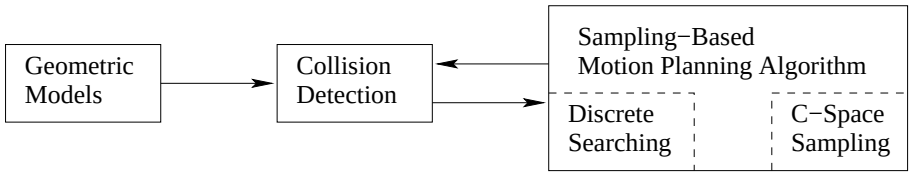


Figure 5.1: The sampling-based planning philosophy uses collision detection as a "black box" that separates the motion planning from the particular geometric and kinematic models. C-space sampling and discrete planning (i.e., searching) are performed.

Section 5.1 presents metric and measure space concepts, which are fundamental to nearly all sampling-based planning algorithms. Section 5.2 presents general sampling concepts and quality criteria that are effective for analyzing the performance of sampling-based algorithms. Section 5.3 gives a brief overview of collision detection algorithms, to gain an understanding of the information available to a planning algorithm and the computation price that must be paid to obtain it. Section 5.4 presents a framework that defines algorithms which solve motion planning problems by integrating sampling and discrete planning (i.e., searching) techniques. These approaches can be considered *single query* in the sense that a single pair, (q_I, q_G), is given, and the algorithm must search until it finds a solution (or it may report early failure). Section 5.5 focuses on *rapidly exploring random trees* (RRTs) and *rapidly exploring dense trees* (RDTs), which are used to develop efficient single-query planning algorithms. Section 5.6 covers *multiple-query* algorithms, which invest substantial preprocessing effort to build a data structure that is later used to obtain efficient solutions for many initial-goal pairs. In this case, it is assumed that the obstacle region $\mathcal{O}$ remains the same for every query.

5.1 Distance and Volume in C-Space

Virtually all sampling-based planning algorithms require a function that measures the distance between two points in $\mathcal{C}$. In most cases, this results in a *metric space*, which is introduced in Section 5.1.1. Useful examples for motion planning are given in Section 5.1.2. It will also be important in many of these algorithms to define the volume of a subset of $\mathcal{C}$. This requires a *measure space*, which is introduced in Section 5.1.3. Section 5.1.4 introduces invariant measures, which should be used whenever possible.

5.1.1 Metric Spaces

It is straightforward to define Euclidean distance in $\mathbb{R}^n$. To define a distance function over any $\mathcal{C}$, however, certain axioms will have to be satisfied so that it coincides with our expectations based on Euclidean distance.

The following definition and axioms are used to create a function that converts a topological space into a metric space.[1] A *metric space* (X, ρ) is a topological space X equipped with a function $\rho : X \times X \to \mathbb{R}$ such that for any $a, b, c \in X$:

1. **Nonnegativity:** $\rho(a, b) \geq 0$.
2. **Reflexivity:** $\rho(a, b) = 0$ if and only if $a = b$.
3. **Symmetry:** $\rho(a, b) = \rho(b, a)$.
4. **Triangle inequality:** $\rho(a, b) + \rho(b, c) \geq \rho(a, c)$.

The function ρ defines distances between points in the metric space, and each of the four conditions on ρ agrees with our intuitions about distance. The final condition implies that ρ is optimal in the sense that the distance from a to c will always be less than or equal to the total distance obtained by traveling through an intermediate point b on the way from a to c.

L_p metrics The most important family of metrics over $\mathbb{R}^n$ is given for any $p \geq 1$ as

$$\rho(x, x') = \left(\sum_{i=1}^{n} |x_i - x'_i|^p \right)^{1/p}. \tag{5.1}$$

For each value of p, (5.1) is called an L_p *metric* (pronounced "el pee"). The three most common cases are

1. L_2: The *Euclidean metric*, which is the familiar Euclidean distance in $\mathbb{R}^n$.
2. L_1: The *Manhattan metric*, which is often nicknamed this way because in $\mathbb{R}^2$ it corresponds to the length of a path that is obtained by moving along an axis-aligned grid. For example, the distance from $(0, 0)$ to $(2, 5)$ is 7 by traveling "east two blocks" and then "north five blocks".
3. L_∞: The L_∞ *metric* must actually be defined by taking the limit of (5.1) as p tends to infinity. The result is

$$L_\infty(x, x') = \max_{1 \leq i \leq n} \{|x_i - x'_i|\}, \tag{5.2}$$

which seems correct because the larger the value of p, the more the largest term of the sum in (5.1) dominates.

[1] Some topological spaces are not *metrizable*, which means that no function exists that satisfies the axioms. Many metrization theorems give sufficient conditions for a topological space to be metrizable [232], and virtually any space that has arisen in motion planning will be metrizable.

An L_p metric can be derived from a norm on a vector space. An L_p *norm* over $\mathbb{R}^n$ is defined as

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}. \tag{5.3}$$

The case of $p = 2$ is the familiar definition of the magnitude of a vector, which is called the *Euclidean norm*. For example, assume the vector space is $\mathbb{R}^n$, and let $\| \cdot \|$ be the standard Euclidean norm. The L_2 metric is $\rho(x, y) = \|x - y\|$. Any L_p metric can be written in terms of a vector subtraction, which is notationally convenient.

Metric subspaces By verifying the axioms, it can be shown that any subspace $Y \subset X$ of a metric space (X, ρ) itself becomes a metric space by restricting the domain of ρ to $Y \times Y$. This conveniently provides metrics on any of the manifolds and varieties from Chapter 4 by simply using any L_p metric on $\mathbb{R}^m$, the space in which the manifold or variety is embedded.

Cartesian products of metric spaces Metrics extend nicely across Cartesian products, which is very convenient because C-spaces are often constructed from Cartesian products, especially in the case of multiple bodies. Let (X, ρ_x) and (Y, ρ_y) be two metric spaces. A metric space (Z, ρ_z) can be constructed for the Cartesian product $Z = X \times Y$ by defining the metric ρ_z as

$$\rho_z(z, z') = \rho_z(x, y, x', y') = c_1 \rho_x(x, x') + c_2 \rho_y(y, y'), \tag{5.4}$$

in which $c_1 > 0$ and $c_2 > 0$ are any positive real constants, and $x, x' \in X$ and $y, y' \in Y$. Each $z \in Z$ is represented as $z = (x, y)$.

Other combinations lead to a metric for Z; for example,

$$\rho_z(z, z') = \Big(c_1 \big[\rho_x(x, x')\big]^p + c_2 \big[\rho_y(y, y')\big]^p \Big)^{1/p}, \tag{5.5}$$

is a metric for any positive integer p. Once again, two positive constants must be chosen. It is important to understand that many choices are possible, and there may not necessarily be a "correct" one.

5.1.2 Important Metric Spaces for Motion Planning

This section presents some metric spaces that arise frequently in motion planning.

Example 5.1 ($SO(2)$ Metric Using Complex Numbers) If $SO(2)$ is represented by unit complex numbers, recall that the C-space is the subset of $\mathbb{R}^2$ given by $\{(a, b) \in \mathbb{R}^2 \mid a^2 + b^2 = 1\}$. Any L_p metric from $\mathbb{R}^2$ may be applied. Using the Euclidean metric,

$$\rho(a_1, b_1, a_2, b_2) = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2}, \tag{5.6}$$

for any pair of points (a_1, b_1) and (a_2, b_2). ■

Example 5.2 ($SO(2)$ Metric by Comparing Angles) You might have noticed that the previous metric for $SO(2)$ does not give the distance traveling along the circle. It instead takes a shortcut by computing the length of the line segment in $\mathbb{R}^2$ that connects the two points. This distortion may be undesirable. An alternative metric is obtained by directly comparing angles, θ_1 and θ_2. However, in this case special care has to be given to the identification, because there are two ways to reach θ_2 from θ_1 by traveling along the circle. This causes a min to appear in the metric definition:

$$\rho(\theta_1, \theta_2) = \min\big\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\big\}, \tag{5.7}$$

for which $\theta_1, \theta_2 \in [0, 2\pi]/\sim$. This may alternatively be expressed using the complex number representation $a + bi$ as an angle between two vectors:

$$\rho(a_1, b_1, a_2, b_2) = \cos^{-1}(a_1 a_2 + b_1 b_2), \tag{5.8}$$

for two points (a_1, b_1) and (a_2, b_2). ■

Example 5.3 (An $SE(2)$ Metric) Again by using the subspace principle, a metric can easily be obtained for $SE(2)$. Using the complex number representation of $SO(2)$, each element of $SE(2)$ is a point $(x_t, y_t, a, b) \in \mathbb{R}^4$. The Euclidean metric, or any other L_p metric on $\mathbb{R}^4$, can be immediately applied to obtain a metric. ■

Example 5.4 ($SO(3)$ Metrics Using Quaternions) As usual, the situation becomes more complicated for $SO(3)$. The unit quaternions form a subset $\mathbb{S}^3$ of $\mathbb{R}^4$. Therefore, any L_p metric may be used to define a metric on $\mathbb{S}^3$, but this will not be a metric for $SO(3)$ because antipodal points need to be identified. Let $h_1, h_2 \in \mathbb{R}^4$ represent two unit quaternions (which are being interpreted here as elements of $\mathbb{R}^4$ by ignoring the quaternion algebra). Taking the identifications into account, the metric is

$$\rho(h_1, h_2) = \min\big\{\|h_1 - h_2\|, \|h_1 + h_2\|\big\}, \tag{5.9}$$

in which the two arguments of the min correspond to the distances from h_1 to h_2 and $-h_2$, respectively. The $h_1 + h_2$ appears because h_2 was negated to yield its antipodal point, $-h_2$.

As in the case of $SO(2)$, the metric in (5.9) may seem distorted because it measures the length of line segments that cut through the interior of $\mathbb{S}^3$, as opposed to traveling along the surface. This problem can be fixed to give a very natural

metric for $SO(3)$, which is based on *spherical linear interpolation*. This takes the line segment that connects the points and pushes it outward onto $\mathbb{S}^3$. It is easier to visualize this by dropping a dimension. Imagine computing the distance between two points on $\mathbb{S}^2$. If these points lie on the equator, then spherical linear interpolation yields a distance proportional to that obtained by traveling along the equator, as opposed to cutting through the interior of $\mathbb{S}^2$ (for points not on the equator, use the *great circle* through the points).

It turns out that this metric can easily be defined in terms of the inner product between the two quaternions. Recall that for unit vectors v_1 and v_2 in $\mathbb{R}^n$, $v_1 \cdot v_2 = \cos\theta$, in which θ is the angle between the vectors. This angle is precisely what is needed to give the proper distance along $\mathbb{S}^3$. The resulting metric is a surprisingly simple extension of (5.8). The distance along $\mathbb{S}^3$ between two quaternions is

$$\rho_s(h_1, h_2) = \cos^{-1}(a_1 a_2 + b_1 b_2 + c_1 c_2 + d_1 d_2), \tag{5.10}$$

in which each $h_i = (a_i, b_i, c_i, d_i)$. Taking identification into account yields the metric

$$\rho(h_1, h_2) = \min\big\{\rho_s(h_1, h_2), \rho_s(h_1, -h_2)\big\}. \tag{5.11}$$

■

Example 5.5 (Another $SE(2)$ Metric) For many C-spaces, the problem of relating different kinds of quantities arises. For example, any metric defined on $SE(2)$ must compare both distance in the plane and an angular quantity. For example, even if $c_1 = c_2 = 1$, the range for $\mathbb{S}^1$ is $[0, 2\pi)$ using radians but $[0, 360)$ using degrees. If the same constant c_2 is used in either case, two very different metrics are obtained. The units applied to $\mathbb{R}^2$ and $\mathbb{S}^1$ are completely incompatible. ■

Example 5.6 (Robot Displacement Metric) Sometimes this incompatibility problem can be fixed by considering the robot displacement. For any two configurations $q_1, q_2 \in \mathcal{C}$, a robot displacement metric can be defined as

$$\rho(q_1, q_2) = \max_{a \in \mathcal{A}} \big\{\|a(q_1) - a(q_2)\|\big\}, \tag{5.12}$$

in which $a(q_i)$ is the position of the point a in the world when the robot $\mathcal{A}$ is at configuration q_i. Intuitively, the robot displacement metric yields the maximum amount in $\mathcal{W}$ that any part of the robot is displaced when moving from configuration q_1 to q_2. The difficulty and efficiency with which this metric can be computed depend strongly on the particular robot geometric model and kinematics. For a convex polyhedral robot that can translate and rotate, it is sufficient to check only vertices. The metric may appear to be ideal, but efficient algorithms are not known for most situations. ■

Example 5.7 ($\mathbb{T}^n$ Metrics) Next consider making a metric over a torus $\mathbb{T}^n$. The Cartesian product rules such as (5.4) and (5.5) can be extended over every copy of $\mathbb{S}^1$ (one for each parameter θ_i). This leads to n arbitrary coefficients c_1, c_2, ..., c_n. Robot displacement could be used to determine the coefficients. For example, if the robot is a chain of links, it might make sense to weight changes in the first link more heavily because the entire chain moves in this case. When the last parameter is changed, only the last link moves; in this case, it might make sense to give it less weight. ■

Example 5.8 ($SE(3)$ Metrics) Metrics for $SE(3)$ can be formed by applying the Cartesian product rules to a metric for $\mathbb{R}^3$ and a metric for $SO(3)$, such as that given in (5.11). Again, this unfortunately leaves coefficients to be specified. These issues will arise again in Section 5.3.4, where more details appear on robot displacement. ■

Pseudometrics Many planning algorithms use functions that behave somewhat like a distance function but may fail to satisfy all of the metric axioms. If such distance functions are used, they will be referred to as *pseudometrics*. One general principle that can be used to derive pseudometrics is to define the distance to be the optimal cost-to-go for some criterion (recall discrete cost-to-go functions from Section 2.3). This will become more important when differential constraints are considered in Chapter 14.

In the continuous setting, the cost could correspond to the distance traveled by a robot or even the amount of energy consumed. Sometimes, the resulting pseudometric is not symmetric. For example, it requires less energy for a car to travel downhill as opposed to uphill. Alternatively, suppose that a car is only capable of driving forward. It might travel a short distance to go forward from q_1 to some q_2, but it might have to travel a longer distance to reach q_1 from q_2 because it cannot drive in reverse. These issues arise for the Dubins car, which is covered in Sections 13.1.2 and 15.3.1.

An important example of a pseudometric from robotics is a *potential function*, which is an important part of the randomized potential field method, which is discussed in Section 5.4.3. The idea is to make a scalar function that estimates the distance to the goal; however, there may be additional terms that attempt to repel the robot away from obstacles. This generally causes local minima to appear in the distance function, which may cause potential functions to violate the triangle inequality.

5.1.3 Basic Measure Theory Definitions

This section briefly indicates how to define volume in a metric space. This provides a basis for defining concepts such as integrals or probability densities. Measure theory is an advanced mathematical topic that is well beyond the scope of this book; however, it is worthwhile to briefly introduce some of the basic definitions because they sometimes arise in sampling-based planning.

Measure can be considered as a function that produces real values for subsets of a metric space, (X, ρ). Ideally, we would like to produce a nonnegative value, $\mu(A) \in [0, \infty]$, for any subset $A \subseteq X$. Unfortunately, due to the Banach-Tarski paradox, if $X = \mathbb{R}^n$, there are some subsets for which trying to assign volume leads to a contradiction. If X is finite, this cannot happen. Therefore, it is hard to visualize the problem; see [420] for a construction of the bizarre nonmeasurable sets. Due to this problem, a workaround was developed by defining a collection of subsets that avoids the paradoxical sets. A collection $\mathcal{B}$ of subsets of X is called a *sigma algebra* if the following axioms are satisfied:

1. The empty set is in $\mathcal{B}$.
2. If $B \in \mathcal{B}$, then $X \setminus B \in \mathcal{B}$.
3. For any collection of a countable number of sets in $\mathcal{B}$, their union must also be in $\mathcal{B}$.

Note that the last two conditions together imply that the intersection of a countable number of sets in $\mathcal{B}$ is also in $\mathcal{B}$. The sets in $\mathcal{B}$ are called the *measurable sets*.

A nice sigma algebra, called the *Borel sets*, can be formed from any metric space (X, ρ) as follows. Start with the set of all open balls in X. These are the sets of the form

$$B(x, r) = \{x' \in X \mid \rho(x, x') < r\} \tag{5.13}$$

for any $x \in X$ and any $r \in (0, \infty)$. From the open balls, the *Borel sets* $\mathcal{B}$ are the sets that can be constructed from these open balls by using the sigma algebra axioms. For example, an open square in $\mathbb{R}^2$ is in $\mathcal{B}$ because it can be constructed as the union of a countable number of balls (infinitely many are needed because the curved balls must converge to covering the straight square edges). By using Borel sets, the nastiness of nonmeasurable sets is safely avoided.

Example 5.9 (Borel Sets) A simple example of $\mathcal{B}$ can be constructed for $\mathbb{R}$. The open balls are just the set of all open intervals, $(x_1, x_2) \subset \mathbb{R}$, for any $x_1, x_2 \in \mathbb{R}$ such that $x_1 < x_2$. ■

Using $\mathcal{B}$, a *measure* μ is now defined as a function $\mu : \mathcal{B} \rightarrow [0, \infty]$ such that the *measure axioms* are satisfied:

1. For the empty set, $\mu(\emptyset) = 0$.

2. For any collection, $E_1, E_2, E_3, \ldots$, of a countable (possibly finite) number of pairwise disjoint, measurable sets, let E denote their union. The measure μ must satisfy
$$\mu(E) = \sum_i \mu(E_i), \tag{5.14}$$
in which i counts over the whole collection.

Example 5.10 (Lebesgue Measure) The most common and important measure is the *Lebesgue measure*, which becomes the standard notions of length in $\mathbb{R}$, area in $\mathbb{R}^2$, and volume in $\mathbb{R}^n$ for $n \geq 3$. One important concept with Lebesgue measure is the existence of sets of *measure zero*. For any countable set A, the Lebesgue measure yields $\mu(A) = 0$. For example, what is the total length of the point $\{1\} \subset \mathbb{R}$? The length of any single point must be zero. To satisfy the measure axioms, sets such as $\{1, 3, 4, 5\}$ must also have measure zero. Even infinite subsets such as $\mathbb{Z}$ and $\mathbb{Q}$ have measure zero in $\mathbb{R}$. If the dimension of a set $A \subseteq \mathbb{R}^n$ is m for some integer $m < n$, then $\mu(A) = 0$, according to the Lebesgue measure on $\mathbb{R}^n$. For example, the set $\mathbb{S}^2 \subset \mathbb{R}^3$ has measure zero because the sphere has no volume. However, if the measure space is restricted to $\mathbb{S}^2$ and then the surface area is defined, then nonzero measure is obtained. ■

Example 5.11 (The Counting Measure) If (X, ρ) is finite, then the *counting measure* can be defined. In this case, the measure can be defined over the entire power set of X. For any $A \subset X$, the counting measure yields $\mu(A) = |A|$, the number of elements in A. Verify that this satisfies the measure axioms. ■

Example 5.12 (Probability Measure) Measure theory even unifies discrete and continuous probability theory. The measure μ can be defined to yield probability mass. The probability axioms (see Section 9.1.2) are consistent with the measure axioms, which therefore yield a measure space. The integrals and sums needed to define expectations of random variables for continuous and discrete cases, respectively, unify into a single measure-theoretic integral. ■

Measure theory can be used to define very general notions of integration that are much more powerful than the Riemann integral that is learned in classical calculus. One of the most important concepts is the *Lebesgue integral*. Instead of being limited to partitioning the domain of integration into intervals, virtually any partition into measurable sets can be used. Its definition requires the notion of a *measurable function* to ensure that the function domain is partitioned into measurable sets. For further study, see [178, 287, 420].

5.1.4 Using the Correct Measure

Since many metrics and measures are possible, it may sometimes seem that there is no "correct" choice. This can be frustrating because the performance of sampling-based planning algorithms can depend strongly on these. Conveniently, there is a natural measure, called the Haar measure, for some transformation groups, including $SO(N)$. Good metrics also follow from the Haar measure, but unfortunately, there are still arbitrary alternatives.

The basic requirement is that the measure does not vary when the sets are transformed using the group elements. More formally, let G represent a matrix group with real-valued entries, and let μ denote a measure on G. If for any measurable subset $A \subseteq G$, and any element $g \in G$, $\mu(A) = \mu(gA) = \mu(Ag)$, then μ is called the *Haar measure*[2] for G. The notation gA represents the set of all matrices obtained by the product ga, for any $a \in A$. Similarly, Ag represents all products of the form ag.

Example 5.13 (Haar Measure for $SO(2)$) The Haar measure for $SO(2)$ can be obtained by parameterizing the rotations as $[0, 1]/\sim$ with 0 and 1 identified, and letting μ be the Lebesgue measure on the unit interval. To see the invariance property, consider the interval $[1/4, 1/2]$, which produces a set $A \subset SO(2)$ of rotation matrices. This corresponds to the set of all rotations from $\theta = \pi/2$ to $\theta = \pi$. The measure yields $\mu(A) = 1/4$. Now consider multiplying every matrix $a \in A$ by a rotation matrix, $g \in SO(2)$, to yield Ag. Suppose g is the rotation matrix for $\theta = \pi$. The set Ag is the set of all rotation matrices from $\theta = 3\pi/2$ up to $\theta = 2\pi = 0$. The measure $\mu(Ag) = 1/4$ remains unchanged. Invariance for gA may be checked similarly. The transformation g translates the intervals in $[0, 1]/\sim$. Since the measure is based on interval lengths, it is invariant with respect to translation. Note that μ can be multiplied by a fixed constant (such as 2π) without affecting the invariance property.

An invariant metric can be defined from the Haar measure on $SO(2)$. For any points $x_1, x_2 \in [0, 1]$, let $\rho = \mu([x_1, x_2])$, in which $[x_1, x_2]$ is the shortest length (smallest measure) interval that contains x_1 and x_2 as endpoints. This metric was already given in Example 5.2.

To obtain examples that are not the Haar measure, let μ represent probability mass over $[0, 1]$ and define any nonuniform probability density function (the uniform density yields the Haar measure). Any shifting of intervals will change the probability mass, resulting in a different measure.

Failing to use the Haar measure weights some parts of $SO(2)$ more heavily than others. Sometimes imposing a bias may be desirable, but it is at least as important to know how to eliminate bias. These ideas may appear obvious, but in the case of $SO(3)$ and many other groups it is more challenging to eliminate

[2]Such a measure is unique up to scale and exists for any locally compact topological group [178, 420].

this bias and obtain the Haar measure. ■

Example 5.14 (Haar Measure for $SO(3)$) For $SO(3)$ it turns out once again that quaternions come to the rescue. If unit quaternions are used, recall that $SO(3)$ becomes parameterized in terms of $\mathbb{S}^3$, but opposite points are identified. It can be shown that the surface area on $\mathbb{S}^3$ is the Haar measure. (Since $\mathbb{S}^3$ is a 3D manifold, it may more appropriately be considered as a surface "volume.") It will be seen in Section 5.2.2 that uniform random sampling over $SO(3)$ must be done with a uniform probability density over $\mathbb{S}^3$. This corresponds exactly to the Haar measure. If instead $SO(3)$ is parameterized with Euler angles, the Haar measure will not be obtained. An unintentional bias will be introduced; some rotations in $SO(3)$ will have more weight than others for no particularly good reason. ■

5.2 Sampling Theory

5.2.1 Motivation and Basic Concepts

The state space for motion planning, $\mathcal{C}$, is uncountably infinite, yet a sampling-based planning algorithm can consider at most a countable number of samples. If the algorithm runs forever, this may be countably infinite, but in practice we expect it to terminate early after only considering a finite number of samples. This mismatch between the cardinality of $\mathcal{C}$ and the set that can be probed by an algorithm motivates careful consideration of sampling techniques. Once the sampling component has been defined, discrete planning methods from Chapter 2 may be adapted to the current setting. Their performance, however, hinges on the way the C-space is sampled.

Since sampling-based planning algorithms are often terminated early, the particular order in which samples are chosen becomes critical. Therefore, a distinction is made between a sample *set* and a sample *sequence*. A unique sample set can always be constructed from a sample sequence, but many alternative sequences can be constructed from one sample set.

Denseness Consider constructing an infinite sample sequence over $\mathcal{C}$. What would be some desirable properties for this sequence? It would be nice if the sequence eventually reached every point in $\mathcal{C}$, but this is impossible because $\mathcal{C}$ is uncountably infinite. Strangely, it is still possible for a sequence to get arbitrarily close to every element of $\mathcal{C}$ (assuming $\mathcal{C} \subseteq \mathbb{R}^m$). In topology, this is the notion of denseness. Let U and V be any subsets of a topological space. The set U is said to be *dense* in V if $\mathrm{cl}(U) = V$ (recall the closure of a set from Section 4.1.1). This means adding the boundary points to U produces V. A simple example is that $(0, 1) \subset \mathbb{R}$ is dense in $[0, 1] \subset \mathbb{R}$. A more interesting example is that the set $\mathbb{Q}$ of

rational numbers is both countable and dense in $\mathbb{R}$. Think about why. For any real number, such as $\pi \in \mathbb{R}$, there exists a sequence of fractions that converges to it. This sequence of fractions must be a subset of $\mathbb{Q}$. A sequence (as opposed to a set) is called *dense* if its underlying set is dense. The bare minimum for sampling methods is that they produce a dense sequence. Stronger requirements, such as uniformity and regularity, will be explained shortly.

A random sequence is probably dense Suppose that $\mathcal{C} = [0, 1]$. One of the simplest ways conceptually to obtain a dense sequence is to pick points at random. Suppose $I \subset [0, 1]$ is an interval of length e. If k samples are chosen independently at random,[3] the probability that none of them falls into I is $(1-e)^k$. As k approaches infinity, this probability converges to zero. This means that the probability that any nonzero-length interval in $[0, 1]$ contains no points converges to zero. One small technicality exists. The infinite sequence of independently, randomly chosen points is only dense *with probability one*, which is not the same as being guaranteed. This is one of the strange outcomes of dealing with uncountably infinite sets in probability theory. For example, if a number between $[0, 1]$ is chosen at random, the probably that $\pi/4$ is chosen is zero; however, it is still possible. (The probability is just the Lebesgue measure, which is zero for a set of measure zero.) For motion planning purposes, this technicality has no practical implications; however, if k is not very large, then it might be frustrating to obtain only probabilistic assurances, as opposed to absolute guarantees of coverage. The next sequence is guaranteed to be dense because it is deterministic.

The van der Corput sequence A beautiful yet underutilized sequence was published in 1935 by van der Corput, a Dutch mathematician [467]. It exhibits many ideal qualities for applications. At the same time, it is based on a simple idea. Unfortunately, it is only defined for the unit interval. The quest to extend many of its qualities to higher dimensional spaces motivates the formal quality measures and sampling techniques in the remainder of this section.

To explain the van der Corput sequence, let $\mathcal{C} = [0, 1]/\sim$, in which $0 \sim 1$, which can be interpreted as $SO(2)$. Suppose that we want to place 16 samples in $\mathcal{C}$. An ideal choice is the set $S = \{i/16 \mid 0 \leq i < 16\}$, which evenly spaces the points at intervals of length $1/16$. This means that no point in $\mathcal{C}$ is further than $1/32$ from the nearest sample. What if we want to make S into a sequence? What is the best ordering? What if we are not even sure that 16 points are sufficient? Maybe 16 is too few or even too many.

The first two columns of Figure 5.2 show a naive attempt at making S into a sequence by sorting the points by increasing value. The problem is that after $i = 8$, half of $\mathcal{C}$ has been neglected. It would be preferable to have a nice covering of $\mathcal{C}$ for any i. Van der Corput's clever idea was to reverse the order of the bits, when the sequence is represented with binary decimals. In the original sequence,

[3] See Section 9.1.2 for a review of probability theory.

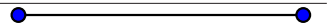
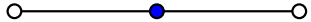
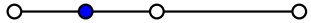
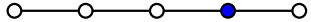
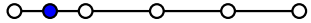
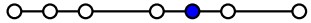
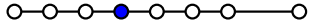
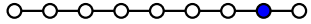
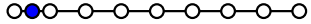
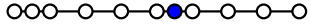
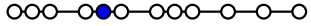
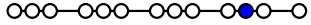
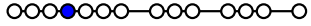
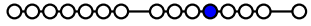
i	Naive Sequence	Binary	Reverse Binary	Van der Corput	Points in $[0,1]/\sim$
1	0	.0000	.0000	0	
2	1/16	.0001	.1000	1/2	
3	1/8	.0010	.0100	1/4	
4	3/16	.0011	.1100	3/4	
5	1/4	.0100	.0010	1/8	
6	5/16	.0101	.1010	5/8	
7	3/8	.0110	.0110	3/8	
8	7/16	.0111	.1110	7/8	
9	1/2	.1000	.0001	1/16	
10	9/16	.1001	.1001	9/16	
11	5/8	.1010	.0101	5/16	
12	11/16	.1011	.1101	13/16	
13	3/4	.1100	.0011	3/16	
14	13/16	.1101	.1011	11/16	
15	7/8	.1110	.0111	7/16	
16	15/16	.1111	.1111	15/16	

Figure 5.2: The van der Corput sequence is obtained by reversing the bits in the binary decimal representation of the naive sequence.

the most significant bit toggles only once, whereas the least significant bit toggles in every step. By reversing the bits, the most significant bit toggles in every step, which means that the sequence alternates between the lower and upper halves of $\mathcal{C}$. The third and fourth columns of Figure 5.2 show the original and reversed-order binary representations. The resulting sequence dances around $[0,1]/\sim$ in a nice way, as shown in the last two columns of Figure 5.2. Let $\nu(i)$ denote the ith point of the van der Corput sequence.

In contrast to the naive sequence, each $\nu(i)$ lies far away from $\nu(i+1)$. Furthermore, the first i points of the sequence, for any i, provide reasonably uniform coverage of $\mathcal{C}$. When i is a power of 2, the points are perfectly spaced. For other i, the coverage is still good in the sense that the number of points that appear in any interval of length l is roughly il. For example, when $i = 10$, every interval of length $1/2$ contains roughly 5 points.

The length, 16, of the naive sequence is actually not important. If instead 8 is used, the same $\nu(1)$, ..., $\nu(8)$ are obtained. Observe in the reverse binary column of Figure 5.2 that this amounts to removing the last zero from each binary decimal representation, which does not alter their values. If 32 is used for the naive sequence, then the same $\nu(1)$, ..., $\nu(16)$ are obtained, and the sequence continues nicely from $\nu(17)$ to $\nu(32)$. To obtain the van der Corput sequence from $\nu(33)$ to $\nu(64)$, six-bit sequences are reversed (corresponding to the case in which the naive sequence has 64 points). The process repeats to produce an infinite sequence that

does not require a fixed number of points to be specified a priori. In addition to the nice uniformity properties for every i, the infinite van der Corput sequence is also dense in $[0,1]/\sim$. This implies that every open subset must contain at least one sample.

You have now seen ways to generate nice samples in a unit interval both randomly and deterministically. Sections 5.2.2–5.2.4 explain how to generate dense samples with nice properties in the complicated spaces that arise in motion planning.

5.2.2 Random Sampling

Now imagine moving beyond $[0,1]$ and generating a dense sample sequence for any bounded C-space, $\mathcal{C} \subseteq \mathbb{R}^m$. In this section the goal is to generate *uniform random* samples. This means that the probability density function $p(q)$ over $\mathcal{C}$ is uniform. Wherever relevant, it also will mean that the probability density is also consistent with the Haar measure. We will not allow any artificial bias to be introduced by selecting a poor parameterization. For example, picking uniform random Euler angles does *not* lead to uniform random samples over $SO(3)$. However, picking uniform random unit quaternions works perfectly because quaternions use the same parameterization as the Haar measure; both choose points on $\mathbb{S}^3$.

Random sampling is the easiest of all sampling methods to apply to C-spaces. One of the main reasons is that C-spaces are formed from Cartesian products, and independent random samples extend easily across these products. If $X = X_1 \times X_2$, and uniform random samples x_1 and x_2 are taken from X_1 and X_2, respectively, then (x_1, x_2) is a uniform random sample for X. This is very convenient in implementations. For example, suppose the motion planning problem involves 15 robots that each translate for any $(x_t, y_t) \in [0,1]^2$; this yields $\mathcal{C} = [0,1]^{30}$. In this case, 30 points can be chosen uniformly at random from $[0,1]$ and combined into a 30-dimensional vector. Samples generated this way are uniformly randomly distributed over $\mathcal{C}$. Combining samples over Cartesian products is much more difficult for nonrandom (deterministic) methods, which are presented in Sections 5.2.3 and 5.2.4.

Generating a random element of $SO(3)$ One has to be very careful about sampling uniformly over the space of rotations. The probability density must correspond to the Haar measure, which means that a random rotation should be obtained by picking a point at random on $\mathbb{S}^3$ and forming the unit quaternion. An extremely clever way to sample $SO(3)$ uniformly at random is given in [439] and is reproduced here. Choose three points $u_1, u_2, u_3 \in [0,1]$ uniformly at random. A uniform, random quaternion is given by the simple expression

$$h = (\sqrt{1-u_1}\sin 2\pi u_2,\ \sqrt{1-u_1}\cos 2\pi u_2,\ \sqrt{u_1}\sin 2\pi u_3,\ \sqrt{u_1}\cos 2\pi u_3). \quad (5.15)$$

A full explanation of the method is given in [439], and a brief intuition is given here. First drop down a dimension and pick $u_1, u_2 \in [0,1]$ to generate points

on $\mathbb{S}^2$. Let u_1 represent the value for the third coordinate, $(0, 0, u_1) \in \mathbb{R}^3$. The slice of points on $\mathbb{S}^2$ for which u_1 is fixed for $0 < u_1 < 1$ yields a circle on $\mathbb{S}^2$ that corresponds to some line of latitude on $\mathbb{S}^2$. The second parameter selects the longitude, $2\pi u_2$. Fortunately, the points are uniformly distributed over $\mathbb{S}^2$. Why? Imagine $\mathbb{S}^2$ as the crust on a spherical loaf of bread that is run through a bread slicer. The slices are cut in a direction parallel to the equator and are of equal thickness. The crusts of each slice have equal area; therefore, the points are uniformly distributed. The method proceeds by using that fact that $\mathbb{S}^3$ can be partitioned into a spherical arrangement of circles (known as the Hopf fibration); there is an $\mathbb{S}^1$ copy for each point in $\mathbb{S}^2$. The method above is used to provide a random point on $\mathbb{S}^2$ using u_2 and u_3, and u_1 produces a random point on $\mathbb{S}^1$; they are carefully combined in (5.15) to yield a random rotation. To respect the antipodal identification for rotations, any quaternion h found in the lower hemisphere (i.e., $a < 0$) can be negated to yield $-h$. This does not distort the uniform random distribution of the samples.

Generating random directions Some sampling-based algorithms require choosing motion directions at random.[4] From a configuration q, the possible directions of motion can be imagined as being distributed around a sphere. In an $(n+1)$-dimensional C-space, this corresponds to sampling on $\mathbb{S}^n$. For example, choosing a direction in $\mathbb{R}^2$ amounts to picking an element of $\mathbb{S}^1$; this can be parameterized as $\theta \in [0, 2\pi]/\sim$. If $n = 4$, then the previously mentioned trick for $SO(3)$ should be used. If $n = 3$ or $n > 4$, then samples can be generated using a slightly more expensive method that exploits spherical symmetries of the multidimensional Gaussian density function [176]. The method is explained for $\mathbb{R}^{n+1}$; boundaries and identifications must be taken into account for other spaces. For each of the $n+1$ coordinates, generate a sample u_i from a zero-mean Gaussian distribution with the same variance for each coordinate. Following from the Central Limit Theorem, u_i can be approximately obtained by generating k samples at random over $[-1, 1]$ and adding them ($k \geq 12$ is usually sufficient in practice). The vector $(u_1, u_2, \ldots, u_{n+1})$ gives a random direction in $\mathbb{R}^{n+1}$ because each u_i was obtained independently, and the level sets of the resulting probability density function are spheres. We did not use uniform random samples for each u_i because this would bias the directions toward the corners of a cube; instead, the Gaussian yields spherical symmetry. The final step is to normalize the vector by taking $u_i/\|u\|$ for each coordinate.

Pseudorandom number generation Although there are advantages to uniform random sampling, there are also several disadvantages. This motivates the consideration of deterministic alternatives. Since there are trade-offs, it is important to understand how to use both kinds of sampling in motion planning. One of the first issues is that computer-generated numbers are not random.[5] A *pseudorandom number generator* is usually employed, which is a deterministic method that simulates the behavior of randomness. Since the samples are not truly random, the advantage of extending the samples over Cartesian products does not necessarily hold. Sometimes problems are caused by unforeseen deterministic dependencies. One of the best pseudorandom number generators for avoiding such troubles is the Mersenne twister [355], for which implementations can be found on the Internet.

To help see the general difficulties, the classical *linear congruential* pseudorandom number generator is briefly explained [317, 381]. The method uses three integer parameters, M, a, and c, which are chosen by the user. The first two, M and a, must be relatively prime, meaning that $\gcd(M, a) = 1$. The third parameter, c, must be chosen to satisfy $0 \leq c < M$. Using modular arithmetic, a sequence can be generated as

$$y_{i+1} = ay_i + c \mod M, \tag{5.16}$$

by starting with some arbitrary seed $1 \leq y_0 \leq M$. Pseudorandom numbers in $[0, 1]$ are generated by the sequence

$$x_i = y_i/M. \tag{5.17}$$

The sequence is periodic; therefore, M is typically very large (e.g., $M = 2^{31} - 1$). Due to periodicity, there are potential problems of regularity appearing in the samples, especially when applied across a Cartesian product to generate points in $\mathbb{R}^n$. Particular values must be chosen for the parameters, and statistical tests are used to evaluate the samples either experimentally or theoretically [381].

Testing for randomness Thus, it is important to realize that even the "random" samples are deterministic. They are designed to optimize performance on statistical tests. Many sophisticated statistical tests of uniform randomness are used. One of the simplest, the *chi-square test*, is described here. This test measures how far computed statistics are from their expected value. As a simple example, suppose $\mathcal{C} = [0, 1]^2$ and is partitioned into a 10 by 10 array of 100 square boxes. If a set P of k samples is chosen at random, then intuitively each box should receive roughly $k/100$ of the samples. An error function can be defined to measure how far from true this intuition is:

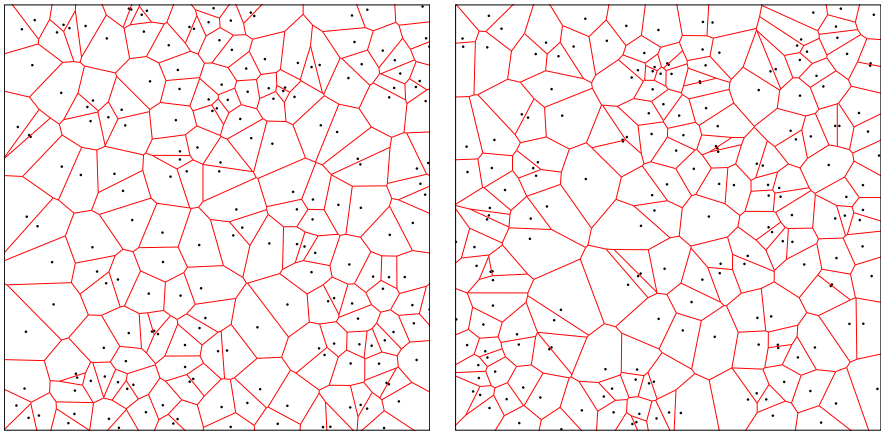
$$e(P) = \sum_{i=1}^{100} (b_i - k/100)^2, \tag{5.18}$$

in which b_i is the number of samples that fall into box i. It is shown [269] that $e(P)$ follows a *chi-squared* distribution. A surprising fact is that the goal is not to

[4] The directions will be formalized in Section 8.3.2 when smooth manifolds are introduced. In that case, the directions correspond to the set of possible velocities that have unit magnitude. Presently, the notion of a direction is only given informally.

[5] There are exceptions, which use physical phenomena as a random source [407].

minimize $e(P)$. If the error is too small, we would declare that the samples are too uniform to be random! Imagine $k = 1,000,000$ and exactly $10,000$ samples appear in each of the 100 boxes. This yields $e(P) = 0$, but how likely is this to ever occur? The error must generally be larger (it appears in many statistical tables) to account for the irregularity due to randomness.



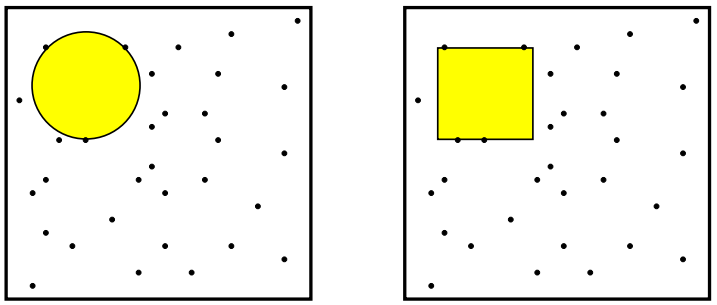
(a) 196 pseudorandom samples (b) 196 pseudorandom samples

Figure 5.3: Irregularity in a collection of (pseudo)random samples can be nicely observed with Voronoi diagrams.

This irregularity can be observed in terms of *Voronoi diagrams*, as shown in Figure 5.3. The Voronoi diagram partitions $\mathbb{R}^2$ into regions based on the samples. Each sample x has an associated *Voronoi region* $\mathrm{Vor}(x)$. For any point $y \in \mathrm{Vor}(x)$, x is the closest sample to y using Euclidean distance. The different sizes and shapes of these regions give some indication of the required irregularity of random sampling. This irregularity may be undesirable for sampling-based motion planning and is somewhat repaired by the deterministic sampling methods of Sections 5.2.3 and 5.2.4 (however, these methods also have drawbacks).

5.2.3 Low-Dispersion Sampling

This section describes an alternative to random sampling. Here, the goal is to optimize a criterion called *dispersion* [381]. Intuitively, the idea is to place samples in a way that makes the largest uncovered area be as small as possible. This generalizes of the idea of *grid resolution*. For a grid, the *resolution* may be selected by defining the step size for each axis. As the step size is decreased, the resolution increases. If a grid-based motion planning algorithm can increase the resolution arbitrarily, it becomes resolution complete. Using the concepts in this section, it may instead reduce its dispersion arbitrarily to obtain a resolution complete



(a) L_2 dispersion (b) L_∞ dispersion

Figure 5.4: Reducing the dispersion means reducing the radius of the largest empty ball.

algorithm. Thus, dispersion can be considered as a powerful generalization of the notion of "resolution."

Dispersion definition The *dispersion*[6] of a finite set P of samples in a metric space (X, ρ) is[7]

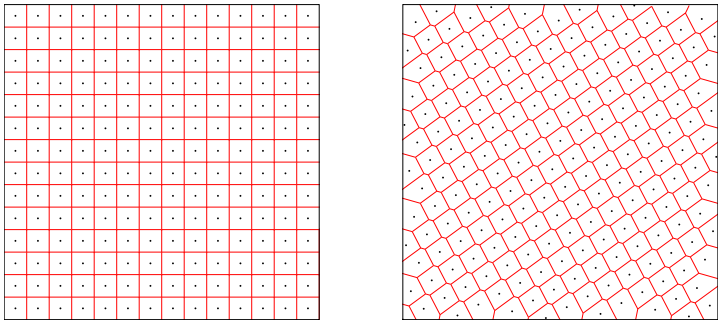
$$\delta(P) = \sup_{x \in X} \Big\{ \min_{p \in P} \big\{ \rho(x, p) \big\} \Big\}. \tag{5.19}$$

Figure 5.4 gives an interpretation of the definition for two different metrics. An alternative way to consider dispersion is as the radius of the largest empty ball (for the L_∞ metric, the balls are actually cubes). Note that at the boundary of X (if it exists), the empty ball becomes truncated because it cannot exceed the boundary. There is also a nice interpretation in terms of Voronoi diagrams. Figure 5.3 can be used to help explain L_2 dispersion in $\mathbb{R}^2$. The *Voronoi vertices* are the points at which three or more Voronoi regions meet. These are points in $\mathcal{C}$ for which the nearest sample is far. An open, empty disc can be placed at any Voronoi vertex, with a radius equal to the distance to the three (or more) closest samples. The radius of the largest disc among those placed at all Voronoi vertices is the dispersion. This interpretation also extends nicely to higher dimensions.

Making good grids Optimizing dispersion forces the points to be distributed more uniformly over $\mathcal{C}$. This causes them to fail statistical tests, but the point distribution is often better for motion planning purposes. Consider the best way to reduce dispersion if ρ is the L_∞ metric and $X = [0, 1]^n$. Suppose that the number of samples, k, is given. Optimal dispersion is obtained by partitioning

[6] The definition is unfortunately backward from intuition. Lower dispersion means that the points are nicely dispersed. Thus, more dispersion is bad, which is counterintuitive.

[7] The sup represents the *supremum*, which is the least upper bound. If X is closed, then the sup becomes a max. See Section 9.1.1 for more details.



(a) 196-point Sukharev grid (b) 196 lattice points

Figure 5.5: The Sukharev grid and a nongrid lattice.

$[0, 1]$ into a grid of cubes and placing a point at the center of each cube, as shown for $n = 2$ and $k = 196$ in Figure 5.5a. The number of cubes per axis must be $\lfloor k^{\frac{1}{n}} \rfloor$, in which $\lfloor \cdot \rfloor$ denotes the *floor*. If $k^{\frac{1}{n}}$ is not an integer, then there are leftover points that may be placed anywhere without affecting the dispersion. Notice that $k^{\frac{1}{n}}$ just gives the number of points per axis for a grid of k points in n dimensions. The resulting grid will be referred to as a *Sukharev grid* [456].

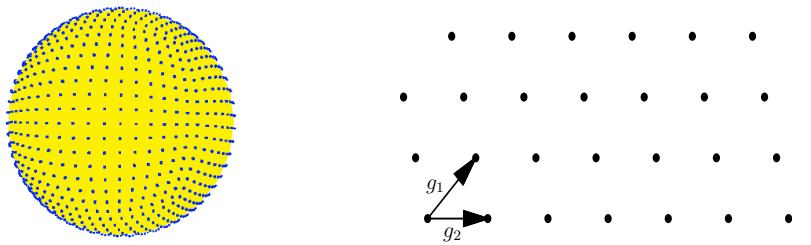
The dispersion obtained by the Sukharev grid is the best possible. Therefore, a useful lower bound can be given for *any* set P of k samples [456]:

$$\delta(P) \geq \frac{1}{2\lfloor k^{\frac{1}{d}} \rfloor}. \tag{5.20}$$

This implies that keeping the dispersion fixed *requires* exponentially many points in the dimension, d.

At this point you might wonder why L_∞ was used instead of L_2, which seems more natural. This is because the L_2 case is extremely difficult to optimize (except in $\mathbb{R}^2$, where a tiling of equilateral triangles can be made, with a point in the center of each one). Even the simple problem of determining the best way to distribute a fixed number of points in $[0, 1]^3$ is unsolved for most values of k. See [131] for extensive treatment of this problem.

Suppose now that other topologies are considered instead of $[0, 1]^n$. Let $X = [0, 1]/\sim$, in which the identification produces a torus. The situation is quite different because X no longer has a boundary. The Sukharev grid still produces optimal dispersion, but it can also be shifted without increasing the dispersion. In this case, a *standard grid* may also be used, which has the same number of points as the Sukharev grid but is translated to the origin. Thus, the first grid point is $(0, 0)$, which is actually the same as $2^n - 1$ other points by identification. If X represents a cylinder and the number of points, k, is given, then it is best to just use the Sukharev grid. It is possible, however, to shift each coordinate that



(a) (b)

Figure 5.6: (a) A distorted grid can even be placed over spheres and $SO(3)$ by putting grids on the faces of an inscribed cube and lifting them to the surface [482]. (b) A lattice can be considered as a grid in which the generators are not necessarily orthogonal.

behaves like $\mathbb{S}^1$. If X is rectangular but not a square, a good grid can still be made by tiling the space with cubes. In some cases this will produce optimal dispersion. For complicated spaces such as $SO(3)$, no grid exists in the sense defined so far. It is possible, however, to generate grids on the faces of an inscribed Platonic solid [139] and lift the samples to $\mathbb{S}^n$ with relatively little distortion [482]. For example, to sample $\mathbb{S}^2$, Sukharev grids can be placed on each face of a cube. These are lifted to obtain the warped grid shown in Figure 5.6a.

Example 5.15 (Sukharev Grid) Suppose that $n = 2$ and $k = 9$. If $X = [0, 1]^2$, then the Sukharev grid yields points for the nine cases in which either coordinate may be $1/6$, $1/2$, or $5/6$. The L_∞ dispersion is $1/6$. The spacing between the points along each axis is $1/3$, which is twice the dispersion. If instead $X = [0, 1]^2/\sim$, which represents a torus, then the nine points may be shifted to yield the standard grid. In this case each coordinate may be 0, $1/3$, or $2/3$. The dispersion and spacing between the points remain unchanged. ■

One nice property of grids is that they have a lattice structure. This means that neighboring points can be obtained very easily be adding or subtracting vectors. Let g_j be an n-dimensional vector called a *generator*. A point on a lattice can be expressed as

$$x = \sum_{j=1}^{n} k_j g_j \tag{5.21}$$

for n independent generators, as depicted in Figure 5.6b. In a 2D grid, the generators represent "up" and "right." If $X = [0, 100]^2$ and a standard grid with integer spacing is used, then the neighbors of the point $(50, 50)$ are obtained by adding $(0, 1)$, $(0, -1)$, $(-1, 0)$, or $(1, 0)$. In a general lattice, the generators need

not be orthogonal. An example is shown in Figure 5.5b. In Section 5.4.2, lattice structure will become important and convenient for defining the search graph.

Infinite grid sequences Now suppose that the number, k, of samples is not given. The task is to define an infinite sequence that has the nice properties of the van der Corput sequence but works for any dimension. This will become the notion of a *multi-resolution grid*. The resolution can be iteratively doubled. For a multi-resolution standard grid in $\mathbb{R}^n$, the sequence will first place one point at the origin. After 2^n points have been placed, there will be a grid with two points per axis. After 4^n points, there will be four points per axis. Thus, after 2^{ni} points for any positive integer i, a grid with 2^i points per axis will be represented. If only complete grids are allowed, then it becomes clear why they appear inappropriate for high-dimensional problems. For example, if $n = 10$, then full grids appear after 1, 2^{10}, 2^{20}, 2^{30}, and so on, samples. Each doubling in resolution multiplies the number of points by 2^n. Thus, to use grids in high dimensions, one must be willing to accept *partial grids* and define an infinite sequence that places points in a nice way.

The van der Corput sequence can be extended in a straightforward way as follows. Suppose $X = \mathbb{T}^2 = [0,1]^2/\sim$. The original van der Corput sequence started by counting in binary. The least significant bit was used to select which half of $[0,1]$ was sampled. In the current setting, the two least significant bits can be used to select the quadrant of $[0,1]^2$. The next two bits can be used to select the quadrant within the quadrant. This procedure continues recursively to obtain a complete grid after $k = 2^{2i}$ points, for any positive integer i. For any k, however, there is only a partial grid. The points are distributed with optimal L_∞ dispersion. This same idea can be applied in dimension n by using n bits at a time from the binary sequence to select the orthant (n-dimensional quadrant). Many other orderings produce L_∞-optimal dispersion. Selecting orderings that additionally optimize other criteria, such as discrepancy or L_2 dispersion, are covered in [330, 335]. Unfortunately, it is more difficult to make a multi-resolution Sukharev grid. The base becomes 3 instead of 2; after every 3^{ni} points a complete grid is obtained. For example, in one dimension, the first point appears at $1/2$. The next two points appear at $1/6$ and $5/6$. The next complete one-dimensional grid appears after there are 9 points.

Dispersion bounds Since the sample sequence is infinite, it is interesting to consider asymptotic bounds on dispersion. It is known that for $X = [0,1]^n$ and any L_p metric, the best possible asymptotic dispersion is $O(k^{-1/n})$ for k points and n dimensions [381]. In this expression, k is the variable in the limit and n is treated as a constant. Therefore, any function of n may appear as a constant (i.e., $O(f(n)k^{-1/n}) = O(k^{-1/n})$ for any positive $f(n)$). An important practical consideration is the size of $f(n)$ in the asymptotic analysis. For example, for the van der Corput sequence from Section 5.2.1, the dispersion is bounded by $1/k$,

which means that $f(n) = 1$. This does not seem good because for values of k that are powers of two, the dispersion is $1/2k$. Using a multi-resolution Sukharev grid, the constant becomes $3/2$ because it takes a longer time before a full grid is obtained. Nongrid, low-dispersion infinite sequences exist that have $f(n) = 1/\ln 4$ [381]; these are not even uniformly distributed, which is rather surprising.

5.2.4 Low-Discrepancy Sampling

In some applications, selecting points that align with the coordinate axis may be undesirable. Therefore, extensive sampling theory has been developed to determine methods that avoid alignments while distributing the points uniformly. In sampling-based motion planning, grids sometimes yield unexpected behavior because a row of points may align nicely with a corridor in $\mathcal{C}_{free}$. In some cases, a solution is obtained with surprisingly few samples, and in others, too many samples are necessary. These alignment problems, when they exist, generally drive the variance higher in computation times because it is difficult to predict when they will help or hurt. This provides motivation for developing sampling techniques that try to reduce this sensitivity.

Discrepancy theory and its corresponding sampling methods were developed to avoid these problems for numerical integration [381]. Let X be a measure space, such as $[0,1]^n$. Let $\mathcal{R}$ be a collection of subsets of X that is called a *range space*. In most cases, $\mathcal{R}$ is chosen as the set of all axis-aligned rectangular subsets; hence, this will be assumed from this point onward. With respect to a particular point set, P, and range space, $\mathcal{R}$, the *discrepancy* [472] for k samples is defined as (see Figure 5.7)

$$D(P, \mathcal{R}) = \sup_{R \in \mathcal{R}} \left\{ \left| \frac{|P \cap R|}{k} - \frac{\mu(R)}{\mu(X)} \right| \right\}, \tag{5.22}$$

in which $|P \cap R|$ denotes the number of points in $P \cap R$. Each term in the supremum considers how well P can be used to estimate the volume of R. For example, if $\mu(R)$ is $1/5$, then we would hope that about $1/5$ of the points in P fall into R. The discrepancy measures the largest volume estimation error that can be obtained over all sets in $\mathcal{R}$.

Asymptotic bounds There are many different asymptotic bounds for discrepancy, depending on the particular range space and measure space [353]. The most widely referenced bounds are based on the standard range space of axis-aligned rectangular boxes in $[0,1]^n$. There are two different bounds, depending on whether the number of points, k, is given. The best possible asymptotic discrepancy for a single sequence is $O(k^{-1} \log^n k)$. This implies that k is not specified. If, however, for every k a new set of points can be chosen, then the best possible discrepancy is $O(k^{-1} \log^{n-1} k)$. This bound is lower because it considers the best that can be achieved by a sequence of points sets, in which every point set may be completely

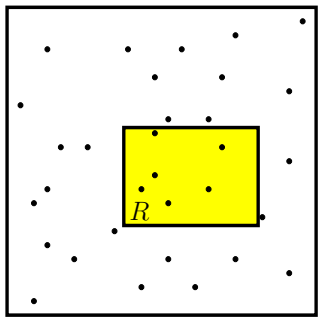


Figure 5.7: Discrepancy measures whether the right number of points fall into boxes. It is related to the chi-square test but optimizes over all possible boxes.

different. In a single sequence, the next set must be extended from the current set by adding a single sample.

Relating dispersion and discrepancy Since balls have positive volume, there is a close relationship between discrepancy, which is measure-based, and dispersion, which is metric-based. For example, for any $P \subset [0,1]^n$,

$$\delta(P, L_\infty) \leq D(P, \mathcal{R})^{1/d}, \tag{5.23}$$

which means low-discrepancy implies low-dispersion. Note that the converse is not true. An axis-aligned grid yields high discrepancy because of alignments with the boundaries of sets in $\mathcal{R}$, but the dispersion is very low. Even though low-discrepancy implies low-dispersion, lower dispersion can usually be obtained by ignoring discrepancy (this is one less constraint to worry about). Thus, a trade-off must be carefully considered in applications.

Low-discrepancy sampling methods Due to the fundamental importance of numerical integration and the intricate link between discrepancy and integration error, most sampling literature has led to low-discrepancy sequences and point sets [381, 443, 458]. Although motion planning is quite different from integration, it is worth evaluating these carefully constructed and well-analyzed samples. Their potential use in motion planning is no less reasonable than using pseudorandom sequences, which were also designed with a different intention in mind (satisfying statistical tests of randomness).

Low-discrepancy sampling methods can be divided into three categories: 1) Halton/Hammersley sampling; 2) (t,s)-sequences and (t,m,s)-nets; and 3) lattices. The first category represents one of the earliest methods, and is based on extending the van der Corput sequence. The *Halton sequence* is an n-dimensional generalization of the van der Corput sequence, but instead of using binary representations,

a different basis is used for each coordinate [220]. The result is a reasonable deterministic replacement for random samples in many applications. The resulting discrepancy (and dispersion) is lower than that for random samples (with high probability). Figure 5.8a shows the first 196 Halton points in $\mathbb{R}^2$.

Choose n relatively prime integers $p_1, p_2, \ldots, p_n$ (usually the first n primes, $p_1 = 2$, $p_2 = 3$, ..., are chosen). To construct the ith sample, consider the base-p representation for i, which takes the form $i = a_0 + pa_1 + p^2a_2 + p^3a_3 + \ldots$. The following point in $[0,1]$ is obtained by reversing the order of the bits and moving the decimal point (as was done in Figure 5.2):

$$r(i,p) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \frac{a_3}{p^4} + \cdots . \tag{5.24}$$

For $p = 2$, this yields the ith point in the van der Corput sequence. Starting from $i = 0$, the ith sample in the Halton sequence is

$$\big(r(i,p_1), r(i,p_2), \ldots, r(i,p_n)\big). \tag{5.25}$$

Suppose instead that k, the required number of points, is known. In this case, a better distribution of samples can be obtained. The *Hammersley point set* [221] is an adaptation of the Halton sequence. Using only $d-1$ distinct primes and starting at $i = 0$, the ith sample in a Hammersley point set with k elements is

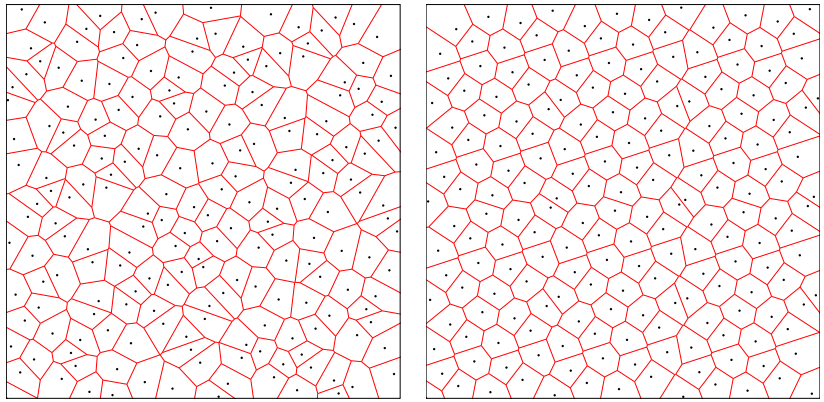
$$\big(i/k, r(i,p_1), \ldots, r(i,p_{n-1})\big). \tag{5.26}$$

Figure 5.8b shows the Hammersley set for $n = 2$ and $k = 196$.

The construction of Halton/Hammersley samples is simple and efficient, which has led to widespread application. They both achieve asymptotically optimal discrepancy; however, the constant in their asymptotic analysis increases more than exponentially with dimension [381]. The constant for the dispersion also increases exponentially, which is much worse than for the methods of Section 5.2.3.

Improved constants are obtained for sequences and finite points by using (t,s)-sequences, and (t,m,s)-nets, respectively [381]. The key idea is to enforce zero discrepancy over particular subsets of $\mathcal{R}$ known as *canonical rectangles*, and all remaining ranges in $\mathcal{R}$ will contribute small amounts to discrepancy. The most famous and widely used (t,s)-sequences are Sobol' and Faure (see [381]). The Niederreiter-Xing (t,s)-sequence has the best-known asymptotic constant, $(a/n)^n$, in which a is a small positive constant [382].

The third category is *lattices*, which can be considered as a generalization of grids that allows nonorthogonal axes [353, 443, 469]. As an example, consider Figure 5.5b, which shows 196 lattice points generated by the following technique. Let α be a positive irrational number. For a fixed k, generate the ith point according to $(i/k, \{i\alpha\})$, in which $\{\cdot\}$ denotes the fractional part of the real value (modulo-one arithmetic). In Figure 5.5b, $\alpha = (\sqrt{5}+1)/2$, the *golden ratio*.



(a) 196 Halton points (b) 196 Hammersley points

Figure 5.8: The Halton and Hammersley points are easy to construct and provide a nice alternative to random sampling that achieves more regularity. Compare the Voronoi regions to those in Figure 5.3. Beware that although these sequences produce asymptotically optimal discrepancy, their performance degrades substantially in higher dimensions (e.g., beyond 10).

This procedure can be generalized to n dimensions by picking $n-1$ distinct irrational numbers. A technique for choosing the α parameters by using the roots of irreducible polynomials is discussed in [353]. The ith sample in the lattice is

$$\left(\frac{i}{k}, \{i\alpha_1\}, \ldots, \{i\alpha_{n-1}\} \right). \tag{5.27}$$

Recent analysis shows that some lattice sets achieve asymptotic discrepancy that is very close to that of the best-known nonlattice sample sets [229, 459]. Thus, restricting the points to lie on a lattice seems to entail little or no loss in performance, but has the added benefit of a regular neighborhood structure that is useful for path planning. Historically, lattices have required the specification of k in advance; however, there has been increasing interest in extensible lattices, which are infinite sequences [230, 459].

5.3 Collision Detection

Once it has been decided where the samples will be placed, the next problem is to determine whether the configuration is in collision. Thus, collision detection is a critical component of sampling-based planning. Even though it is often treated as a black box, it is important to study its inner workings to understand the information it provides and its associated computational cost. In most motion planning applications, the majority of computation time is spent on collision checking.

A variety of collision detection algorithms exist, ranging from theoretical algorithms that have excellent computational complexity to heuristic, practical algorithms whose performance is tailored to a particular application. The techniques from Section 4.3 can be used to develop a collision detection algorithm by defining a logical predicate using the geometric model of $\mathcal{C}_{obs}$. For the case of a 2D world with a convex robot and obstacle, this leads to an linear-time collision detection algorithm. In general, however, it can be determined whether a configuration is in collision more efficiently by avoiding the full construction of $\mathcal{C}_{obs}$.

5.3.1 Basic Concepts

As in Section 3.1.1, collision detection may be viewed as a logical predicate. In the current setting it appears as $\phi : \mathcal{C} \rightarrow \{\text{TRUE}, \text{FALSE}\}$, in which the domain is $\mathcal{C}$ instead of $\mathcal{W}$. If $q \in \mathcal{C}_{obs}$, then $\phi(q) = \text{TRUE}$; otherwise, $\phi(q) = \text{FALSE}$.

Distance between two sets For the Boolean-valued function ϕ, there is no information about how far the robot is from hitting the obstacles. Such information is very important in planning algorithms. A *distance function* provides this information and is defined as $d : \mathcal{C} \rightarrow [0, \infty)$, in which the real value in the range of f indicates the distance in the world, $\mathcal{W}$, between the closest pair of points over all pairs from $\mathcal{A}(q)$ and $\mathcal{O}$. In general, for two closed, bounded subsets, E and F, of $\mathbb{R}^n$, the *distance* is defined as

$$\rho(E, F) = \min_{e \in E} \left\{ \min_{f \in F} \left\{ \| e - f \| \right\} \right\}, \tag{5.28}$$

in which $\| \cdot \|$ is the Euclidean norm. Clearly, if $E \cap F \neq \emptyset$, then $\rho(E, F) = 0$. The methods described in this section may be used to either compute distance or only determine whether $q \in \mathcal{C}_{obs}$. In the latter case, the computation is often much faster because less information is required.

Two-phase collision detection Suppose that the robot is a collection of m attached links, $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$, and that $\mathcal{O}$ has k connected components. For this complicated situation, collision detection can be viewed as a two-phase process.

1. **Broad Phase:** In the *broad phase*, the task is to avoid performing expensive computations for bodies that are far away from each other. Simple bounding boxes can be placed around each of the bodies, and simple tests can be performed to avoid costly collision checking unless the boxes overlap. Hashing schemes can be employed in some cases to greatly reduce the number of pairs of boxes that have to be tested for overlap [368]. For a robot that consists of multiple bodies, the pairs of bodies that should be considered for collision must be specified in advance, as described in Section 4.3.1.

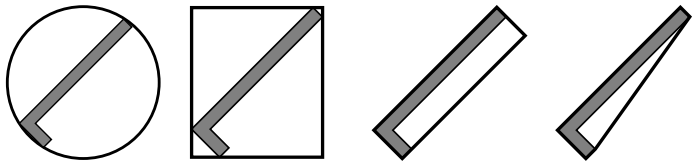


Figure 5.9: Four different kinds of bounding regions: (a) sphere, (b) axis-aligned bounding box (AABB), (c) oriented bounding box (OBB), and (d) convex hull. Each usually provides a tighter approximation than the previous one but is more expensive to test for overlapping pairs.

2. **Narrow Phase:** In the *narrow phase*, individual pairs of bodies are each checked carefully for collision. Approaches to this phase are described in Sections 5.3.2 and 5.3.3.

5.3.2 Hierarchical Methods

In this section, suppose that two complicated, nonconvex bodies, E and F, are to be checked for collision. Each body could be part of either the robot or the obstacle region. They are subsets of $\mathbb{R}^2$ or $\mathbb{R}^3$, defined using any kind of geometric primitives, such as triangles in $\mathbb{R}^3$. *Hierarchical methods* generally decompose each body into a tree. Each vertex in the tree represents a *bounding region* that contains some subset of the body. The bounding region of the root vertex contains the whole body.

There are generally two opposing criteria that guide the selection of the type of bounding region:

1. The region should fit the intended body points as tightly as possible.

2. The intersection test for two regions should be as efficient as possible.

Several popular choices are shown in Figure 5.9 for an L-shaped body.

The tree is constructed for a body, E (or alternatively, F) recursively as follows. For each vertex, consider the set X of all points in E that are contained in the bounding region. Two child vertices are constructed by defining two smaller bounding regions whose union covers X. The split is made so that the portion covered by each child is of similar size. If the geometric model consists of primitives such as triangles, then a split could be made to separate the triangles into two sets of roughly the same number of triangles. A bounding region is then computed for each of the children. Figure 5.10 shows an example of a split for the case of an L-shaped body. Children are generated recursively by making splits until very simple sets are obtained. For example, in the case of triangles in space,

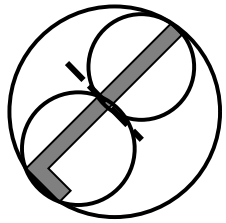


Figure 5.10: The large circle shows the bounding region for a vertex that covers an L-shaped body. After performing a split along the dashed line, two smaller circles are used to cover the two halves of the body. Each circle corresponds to a child vertex.

a split is made unless the vertex represents a single triangle. In this case, it is easy to test for the intersection of two triangles.

Consider the problem of determining whether bodies E and F are in collision. Suppose that the trees T_e and T_f have been constructed for E and F, respectively. If the bounding regions of the root vertices of T_e and T_f do not intersect, then it is known that T_e and T_f are not in collision without performing any additional computation. If the bounding regions do intersect, then the bounding regions of the children of T_e are compared to the bounding region of T_f. If either of these intersect, then the bounding region of T_f is replaced with the bounding regions of its children, and the process continues recursively. As long as the bounding regions overlap, lower levels of the trees are traversed, until eventually the leaves are reached. If triangle primitives are used for the geometric models, then at the leaves the algorithm tests the individual triangles for collision, instead of bounding regions. Note that as the trees are traversed, if a bounding region from the vertex v_1 of T_e does not intersect the bounding region from a vertex, v_2, of T_f, then no children of v_1 have to be compared to children of v_1. Usually, this dramatically reduces the number of comparisons, relative in a naive approach that tests all pairs of triangles for intersection.

It is possible to extend the hierarchical collision detection scheme to also compute distance. The closest pair of points found so far serves as an upper bound that prunes aways some future pairs from consideration. If a pair of bounding regions has a distance greater than the smallest distance computed so far, then their children do not have to be considered [329]. In this case, an additional requirement usually must be imposed. Every bounding region must be a proper subset of its parent bounding region [406]. If distance information is not needed, then this requirement can be dropped.

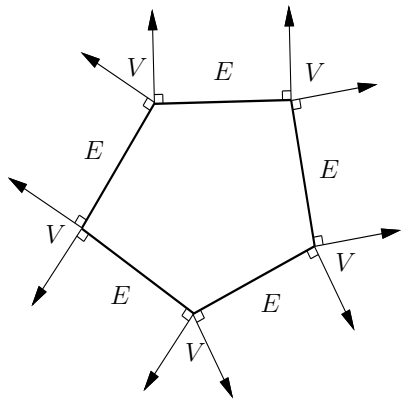


Figure 5.11: The Voronoi regions alternate between being edge-based and vertex-based. The Voronoi regions of vertices are labeled with a "V" and the Voronoi regions of edges are labeled with an "E."

5.3.3 Incremental Methods

This section focuses on a particular approach called *incremental distance computation*, which assumes that between successive calls to the collision detection algorithm, the bodies move only a small amount. Under this assumption the algorithm achieves "almost constant time" performance for the case of convex polyhedral bodies [327, 367]. Nonconvex bodies can be decomposed into convex components.

These collision detection algorithms seem to offer wonderful performance, but this comes at a price. The models must be *coherent*, which means that all of the primitives must fit together nicely. For example, if a 2D model uses line segments, all of the line segments must fit together perfectly to form polygons. There can be no isolated segments or chains of segments. In three dimensions, polyhedral models are required to have all faces come together perfectly to form the boundaries of 3D shapes. The model cannot be an arbitrary collection of 3D triangles.

The method will be explained for the case of 2D convex polygons, which are interpreted as convex subsets of $\mathbb{R}^2$. Voronoi regions for a convex polygon will be defined in terms of features. The *feature set* is the set of all vertices and edges of a convex polygon. Thus, a polygon with n edges has $2n$ features. Any point outside of the polygon has a closest feature in terms of Euclidean distance. For a given feature, F, the set of all points in $\mathbb{R}^2$ from which F is the closest feature is called the Voronoi region of F and is denoted $\mathrm{Vor}(F)$. Figure 5.11 shows all ten Voronoi regions for a pentagon. Each feature is considered as a point set in the discussion below.

For any two convex polygons that do not intersect, the closest point is determined by a pair of points, one on each polygon (the points are unique, except in the case of parallel edges). Consider the feature for each point in the closest pair. There are only three possible combinations:

- **Vertex-Vertex** Each point of the closest pair is a vertex of a polygon.
- **Edge-Vertex** One point of the closest pair lies on an edge, and the other lies on a vertex.
- **Edge-Edge** Each point of the closest pair lies on an edge. In this case, the edges must be parallel.

Let P_1 and P_2 be two convex polygons, and let F_1 and F_2 represent any feature pair, one from each polygon. Let $(x_1, y_1) \in F_1$ and $(x_2, y_2) \in F_2$ denote the closest pair of points, among all pairs of points in F_1 and F_2, respectively. The following condition implies that the distance between (x_1, y_1) and (x_2, y_2) is the distance between P_1 and P_2:

$$(x_1, y_1) \in \mathrm{Vor}(F_2) \text{ and } (x_2, y_2) \in \mathrm{Vor}(F_1). \tag{5.29}$$

If (5.29) is satisfied for a given feature pair, then the distance between P_1 and P_2 equals the distance between F_1 and F_2. This implies that the distance between P_1 and P_2 can be determined in constant time. The assumption that P_1 moves only a small amount relative to P_2 is made to increase the likelihood that the closest feature pair remains the same. This is why the phrase "almost constant time" is used to describe the performance of the algorithm. Of course, it is possible that the closest feature pair will change. In this case, neighboring features are tested using the condition above until the new closest pair of features is found. In this worst case, this search could be costly, but this violates the assumption that the bodies do not move far between successive collision detection calls.

The 2D ideas extend to 3D convex polyhedra [136, 327, 367]. The primary difference is that three kinds of features are considered: faces, edges, and vertices. The cases become more complicated, but the idea is the same. Once again, the condition regarding mutual Voronoi regions holds, and the resulting incremental collision detection algorithm has "almost constant time" performance.

5.3.4 Checking a Path Segment

Collision detection algorithms determine whether a configuration lies in $\mathcal{C}_{free}$, but motion planning algorithms require that an entire path maps into $\mathcal{C}_{free}$. The interface between the planner and collision detection usually involves validation of a path segment (i.e., a path, but often a short one). This cannot be checked point-by-point because it would require an uncountably infinite number of calls to the collision detection algorithm.

Suppose that a path, $\tau : [0, 1] \rightarrow \mathcal{C}$, needs to be checked to determine whether $\tau([0, 1]) \subset \mathcal{C}_{free}$. A common approach is to sample the interval $[0, 1]$ and call the

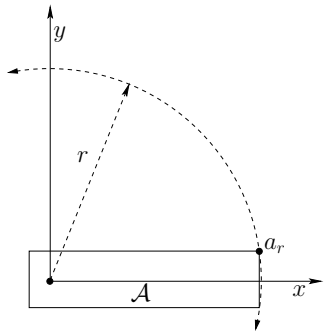


Figure 5.12: The furthest point on $\mathcal{A}$ from the origin travels the fastest when $\mathcal{A}$ is rotated. At most it can be displaced by $2\pi r$, if x_t and y_t are fixed.

collision checker only on the samples. What resolution of sampling is required? How can one ever guarantee that the places where the path is not sampled are collision-free? There are both practical and theoretical answers to these questions. In practice, a fixed $\Delta q > 0$ is often chosen as the C-space step size. Points $t_1, t_2 \in [0, 1]$ are then chosen close enough together to ensure that $\rho(\tau(t_1), \tau(t_2)) \leq \Delta q$, in which ρ is the metric on $\mathcal{C}$. The value of Δq is often determined experimentally. If Δq is too small, then considerable time is wasted on collision checking. If Δq is too large, then there is a chance that the robot could jump through a thin obstacle.

Setting Δq empirically might not seem satisfying. Fortunately, there are sound algorithmic ways to verify that a path is collision-free. In some applications the methods are still not used because they are trickier to implement and they often yield worse performance. Therefore, both methods are presented here, and you can decide which is appropriate, depending on the context and your personal tastes.

Ensuring that $\tau([0, 1]) \subset \mathcal{C}_{free}$ requires the use of both distance information and bounds on the distance that points on $\mathcal{A}$ can travel in $\mathbb{R}$. Such bounds can be obtained by using the robot displacement metric from Example 5.6. Before expressing the general case, first we will explain the concept in terms of a rigid robot that translates and rotates in $\mathcal{W} = \mathbb{R}^2$. Let $x_t, y_t \in \mathbb{R}^2$ and $\theta \in [0, 2\pi]/\sim$. Suppose that a collision detection algorithm indicates that $\mathcal{A}(q)$ is at least d units away from collision with obstacles in $\mathcal{W}$. This information can be used to determine a region in $\mathcal{C}_{free}$ that contains q. Suppose that the next candidate configuration to be checked along τ is q'. If no point on $\mathcal{A}$ travels more than distance d when moving from q to q' along τ, then q' and all configurations between q and q' must be collision-free. This assumes that on the path from q to q', every visited configuration must lie between q_i and q'_i for the ith coordinate and any i from 1 to n. If the robot can instead take any path between q and q', then no such guarantee can be made).

When $\mathcal{A}$ undergoes a translation, all points move the same distance. For rotation, however, the distance traveled depends on how far the point on $\mathcal{A}$ is from the rotation center, $(0, 0)$. Let $a_r = (x_r, y_r)$ denote the point on $\mathcal{A}$ that has the largest magnitude, $r = \sqrt{x_r^2 + y_r^2}$. Figure 5.12 shows an example. A transformed point $a \in \mathcal{A}$ may be denoted by $a(x_t, y_t, \theta)$. The following bound is obtained for any $a \in \mathcal{A}$, if the robot is rotated from orientation θ to θ':

$$\|a(x_t, y_t, \theta) - a(x_t, y_t, \theta')\| \leq \|a_r(x_t, y_t, \theta) - a_r(x_t, y_t, \theta')\| < r|\theta - \theta'|, \quad (5.30)$$

assuming that a path in $\mathcal{C}$ is followed that interpolates between θ and θ' (using the shortest path in $\mathbb{S}^1$ between θ and θ'). Thus, if $\mathcal{A}(q)$ is at least d away from the obstacles, then the orientation may be changed without causing collision as long as $r|\theta - \theta'| < d$. Note that this is a loose upper bound because a_r travels along a circular arc and can be displaced by no more than $2\pi r$.

Similarly, x_t and y_t may individually vary up to d, yielding $|x_t - x'_t| < d$ and $|y_t - y'_t| < d$. If all three parameters vary simultaneously, then a region in $\mathcal{C}_{free}$ can be defined as

$$\{(x'_t, y'_t, \theta') \in \mathcal{C} \mid |x_t - x'_t| + |y_t - y'_t| + r|\theta - \theta'| < d\}. \quad (5.31)$$

Such bounds can generally be used to set a step size, Δq, for collision checking that guarantees the intermediate points lie in $\mathcal{C}_{free}$. The particular value used may vary depending on d and the direction[8] of the path.

For the case of $SO(3)$, once again the displacement of the point on $\mathcal{A}$ that has the largest magnitude can be bounded. It is best in this case to express the bounds in terms of quaternion differences, $\|h - h'\|$. Euler angles may also be used to obtain a straightforward generalization of (5.31) that has six terms, three for translation and three for rotation. For each of the three rotation parts, a point with the largest magnitude in the plane perpendicular to the rotation axis must be chosen.

If there are multiple links, it becomes much more complicated to determine the step size. Each point $a \in \mathcal{A}_i$ is transformed by some nonlinear function based on the kinematic expressions from Sections 3.3 and 3.4. Let $a : \mathcal{C} \to \mathcal{W}$ denote this transformation. In some cases, it might be possible to derive a *Lipschitz condition* of the form

$$\|a(q) - a(q')\| < c\|q - q'\|, \quad (5.32)$$

in which $c \in (0, \infty)$ is a fixed constant, a is any point on $\mathcal{A}_i$, and the expression holds for any $q, q' \in \mathcal{C}$. The goal is to make the *Lipschitz constant*, c, as small as possible; this enables larger variations in q.

A better method is to individually bound the link displacement with respect to each parameter,

$$\|a(q_1, \ldots, q_{i-1}, q_i, q_{i+1}, \ldots, q_n) - a(q_1, \ldots, q_{i-1}, q'_i, q_{i+1}, \ldots, q_n)\| < c_i|q_i - q'_i|, \quad (5.33)$$

[8] To formally talk about directions, it would be better to define a differentiable structure on $\mathcal{C}$. This will be deferred to Section 8.3, where it seems unavoidable.

to obtain the Lipschitz constants $c_1, \ldots, c_n$. The bound on robot displacement becomes

$$\|a(q) - a(q')\| < \sum_{i=1}^{n} c_i |q_i - q'_i|. \tag{5.34}$$

The benefit of using individual parameter bounds can be seen by considering a long chain. Consider a 50-link chain of line segments in $\mathbb{R}^2$, and each link has length 10. The C-space is $\mathbb{T}^{50}$, which can be parameterized as $[0, 2\pi]^{50}/\sim$. Suppose that the chain is in a straight-line configuration ($\theta_i = 0$ for all $1 \leq i \leq 50$), which means that the last point is at $(500, 0) \in \mathcal{W}$. Changes in θ_1, the orientation of the first link, dramatically move $\mathcal{A}_{50}$. However, changes in θ_{50} move $\mathcal{A}_{50}$ a smaller amount. Therefore, it is advantageous to pick a different Δq_i for each $1 \leq i \leq 50$. In this example, a smaller value should be used for $\Delta\theta_1$ in comparison to $\Delta\theta_{50}$.

Unfortunately, there are more complications. Suppose the 50-link chain is in a configuration that folds all of the links on top of each other ($\theta_i = \pi$ for each $1 \leq i \leq n$). In this case, $\mathcal{A}_{50}$ does not move as fast when θ_1 is perturbed, in comparison to the straight-line configuration. A larger step size for θ_1 could be used for this configuration, relative to other parts of $\mathcal{C}$. The implication is that, although Lipschitz constants can be made to hold over all of $\mathcal{C}$, it might be preferable to determine a better bound in a local region around $q \in \mathcal{C}$. A linear method could be obtained by analyzing the Jacobian of the transformations, such as (3.53) and (3.57).

Another important concern when checking a path is the order in which the samples are checked. For simplicity, suppose that Δq is constant and that the path is a constant-speed parameterization. Should the collision checker step along from 0 up to 1? Experimental evidence indicates that it is best to use a recursive binary strategy [191]. This makes no difference if the path is collision-free, but it often saves time if the path is in collision. This is a kind of sampling problem over $[0, 1]$, which is addressed nicely by the van der Corput sequence, ν. The last column in Figure 5.2 indicates precisely where to check along the path in each step. Initially, $\tau(1)$ is checked. Following this, points from the van der Corput sequence are checked in order: $\tau(0)$, $\tau(1/2)$, $\tau(1/4)$, $\tau(3/4)$, $\tau(1/8)$, ... The process terminates if a collision is found or when the dispersion falls below Δq. If Δq is not constant, then it is possible to skip over some points of ν in regions where the allowable variation in q is larger.

5.4 Incremental Sampling and Searching

5.4.1 The General Framework

The algorithms of Sections 5.4 and 5.5 follow the *single-query model*, which means (q_I, q_G) is given only once per robot and obstacle set. This means that there are no advantages to precomputation, and the sampling-based motion planning problem

can be considered as a kind of search. The *multiple-query model*, which favors precomputation, is covered in Section 5.6.

The sampling-based planning algorithms presented in the present section are strikingly similar to the family of search algorithms summarized in Section 2.2.4. The main difference lies in step 3 below, in which applying an action, u, is replaced by generating a path segment, τ_s. Another difference is that the search graph, $\mathcal{G}$, is undirected, with edges that represent paths, as opposed to a directed graph in which edges represent actions. It is possible to make these look similar by defining an action space for motion planning that consists of a collection of paths, but this is avoided here. In the case of motion planning with differential constraints, this will actually be required; see Chapter 14.

Most single-query, sampling-based planning algorithms follow this template:

1. **Initialization:** Let $\mathcal{G}(V, E)$ represent an undirected *search graph*, for which V contains at least one vertex and E contains no edges. Typically, V contains q_I, q_G, or both. In general, other points in $\mathcal{C}_{free}$ may be included.

2. **Vertex Selection Method (VSM):** Choose a vertex $q_{cur} \in V$ for expansion.

3. **Local Planning Method (LPM):** For some $q_{new} \in \mathcal{C}_{free}$ that may or may not be represented by a vertex in V, attempt to construct a path $\tau_s : [0, 1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{cur}$ and $\tau(1) = q_{new}$. Using the methods of Section 5.3.4, τ_s must be checked to ensure that it does not cause a collision. If this step fails to produce a collision-free path segment, then go to step 2.

4. **Insert an Edge in the Graph:** Insert τ_s into E, as an edge from q_{cur} to q_{new}. If q_{new} is not already in V, then it is inserted.

5. **Check for a Solution:** Determine whether $\mathcal{G}$ encodes a solution path. As in the discrete case, if there is a single search tree, then this is trivial; otherwise, it can become complicated and expensive.

6. **Return to Step 2:** Iterate unless a solution has been found or some termination condition is satisfied, in which case the algorithm reports failure.

In the present context, $\mathcal{G}$ is a topological graph, as defined in Example 4.6. Each vertex is a configuration and each edge is a path that connects two configurations. In this chapter, it will be simply referred to as a graph when there is no chance of confusion. Some authors refer to such a graph as a *roadmap*; however, we reserve the term roadmap for a graph that contains enough paths to make any motion planning query easily solvable. This case is covered in Section 5.6 and throughout Chapter 6.

A large family of sampling-based algorithms can be described by varying the implementations of steps 2 and 3. Implementations of the other steps may also

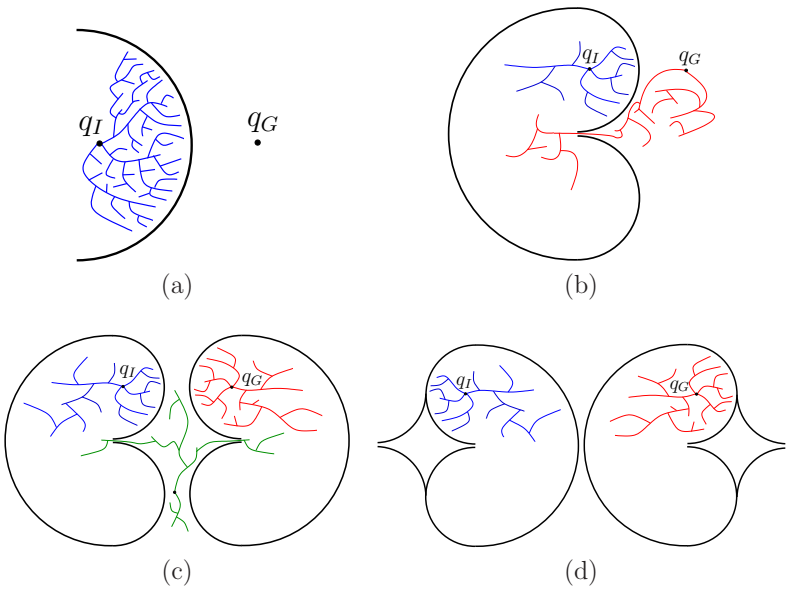


Figure 5.13: All of these depict high-dimensional obstacle regions in C-space. (a) The search must involve some sort of multi-resolution aspect, otherwise, that algorithm may explore too many points within a cavity. (b) Sometimes the problem is like a bug trap, in which case bidirectional search can help. (c) For a double bug trap, multi-directional search may be needed. (d) This example is hard to solve even for multi-directional search.

vary, but this is less important and will be described where appropriate. For convenience, step 2 will be called the vertex selection method (VSM) and step 3 will be called the *local planning method* (LPM). The role of the VSM is similar to that of the priority queue, Q, in Section 2.2.1. The role of the LPM is to compute a collision-free path segment that can be added to the graph. It is called *local* because the path segment is usually simple (e.g., the shortest path) and travels a short distance. It is not *global* in the sense that the LPM does not try to solve the entire planning problem; it is expected that the LPM may often fail to construct path segments.

It will be formalized shortly, but imagine for the time being that any of the search algorithms from Section 2.2 may be applied to motion planning by approximating $\mathcal{C}$ with a high-resolution grid. The resulting problem looks like a multi-dimensional extension of Example 2.1 (the "labyrinth" walls are formed by $\mathcal{C}_{obs}$). For a high-resolution grid in a high-dimensional space, most classical discrete searching algorithms have trouble getting trapped in a local minimum. There could be an astronomical number of configurations that fall within a concavity in $\mathcal{C}_{obs}$ that must be escaped to solve the problem, as shown in Figure 5.13a. Imagine a problem in which the C-space obstacle is a giant "bowl" that can trap the configuration. This figure is drawn in two dimensions, but imagine that the $\mathcal{C}$ has many dimensions, such as six for $SE(3)$ or perhaps dozens for a linkage. If the discrete planning algorithms from Section 2.2 are applied to a high-resolution grid approximation of $\mathcal{C}$, then they will all waste their time filling up the bowl before being able to escape to q_G. The number of grid points in this bowl would typically be on the order of 100^n for an n-dimensional C-space. Therefore, sampling-based motion planning algorithms combine sampling and searching in a way that attempts to overcome this difficulty.

As in the case of discrete search algorithms, there are several classes of algorithms based on the number of search trees.

Unidirectional (single-tree) methods: In this case, the planning appears very similar to discrete forward search, which was given in Figure 2.4. The main difference between algorithms in this category is how they implement the VSM and LPM. Figure 5.13b shows a *bug trap*[9] example for which forward-search algorithms would have great trouble; however, the problem might not be difficult for backward search, if the planner incorporates some kind of greedy, best-first behavior. This example, again in high dimensions, can be considered as a kind of "bug trap." To leave the trap, a path must be found from q_I into the narrow opening. Imagine a fly buzzing around through the high-dimensional trap. The escape opening might not look too difficult in two dimensions, but if it has a small range with respect to each configuration parameter, it is nearly impossible to find the opening. The tip of the "volcano" would be astronomically small compared to the rest of the bug trap. Examples such as this provide some motivation for bidirectional algorithms. It might be easier for a search tree that starts in q_G to arrive in the bug trap.

Bidirectional (two-tree) methods: Since it is not known whether q_I or q_G might lie in a bug trap (or another challenging region), a bidirectional approach is often preferable. This follows from an intuition that two propagating wavefronts, one centered on q_I and the other on q_G, will meet after covering less area in comparison to a single wavefront centered at q_I that must arrive at q_G. A bidirectional search is achieved by defining the VSM to alternate between trees when selecting vertices. The LPM sometimes generates paths that explore new parts of $\mathcal{C}_{free}$, and at other times it tries to generate a path that connects the two trees.

Multi-directional (more than two trees) methods: If the problem is so bad that a double bug trap exists, as shown in Figure 5.13c, then it

[9]This principle is actually used in real life to trap flying bugs. This analogy was suggested by James O'Brien in a discussion with James Kuffner.

might make sense to grow trees from other places in the hopes that there are better chances to enter the traps in the other direction. This complicates the problem of connecting trees, however. Which pairs of trees should be selected in each iteration for possible connection? How often should the same pair be selected? Which vertex pairs should be selected? Many heuristic parameters may arise in practice to answer these questions.

Of course, one can play the devil's advocate and construct the example in Figure 5.13d, for which virtually all sampling-based planning algorithms are doomed. Even harder versions can be made in which a sequence of several narrow corridors must be located and traversed. We must accept the fact that some problems are hopeless to solve using sampling-based planning methods, unless there is some problem-specific structure that can be additionally exploited.

5.4.2 Adapting Discrete Search Algorithms

One of the most convenient and straightforward ways to make sampling-based planning algorithms is to define a grid over $\mathcal{C}$ and conduct a discrete search using the algorithms of Section 2.2. The resulting planning problem actually looks very similar to Example 2.1. Each edge now corresponds to a path in $\mathcal{C}_{free}$. Some edges may not exist because of collisions, but this will have to be revealed incrementally during the search because an explicit representation of $\mathcal{C}_{obs}$ is too expensive to construct (recall Section 4.3).

Assume that an n-dimensional C-space is represented as a unit cube, $\mathcal{C} = [0,1]^n/\sim$, in which $\sim$ indicates that identifications of the sides of the cube are made to reflect the C-space topology. Representing $\mathcal{C}$ as a unit cube usually requires a reparameterization. For example, an angle $\theta \in [0, 2\pi)$ would be replaced with $\theta/2\pi$ to make the range lie within $[0, 1]$. If quaternions are used for $SO(3)$, then the upper half of $\mathbb{S}^3$ must be deformed into $[0,1]^3/\sim$.

Discretization Assume that $\mathcal{C}$ is *discretized* by using the *resolutions* k_1, k_2,..., and k_n, in which each k_i is a positive integer. This allows the resolution to be different for each C-space coordinate. Either a standard grid or a Sukharev grid can be used. Let

$$\Delta q_i = [0 \ \cdots \ 0 \ \ 1/k_i \ \ 0 \ \cdots \ 0], \tag{5.35}$$

in which the first $i-1$ components and the last $n-i$ components are 0. A *grid point* is a configuration $q \in \mathcal{C}$ that can be expressed in the form[10]

$$\sum_{i=1}^{n} j_i \Delta q_i, \tag{5.36}$$

in which each $j_i \in \{0, 1, \ldots, k_i\}$. The integers j_1, ..., j_n can be imagined as array indices for the grid. Let the term *boundary grid point* refer to a grid point for which $j_i = 0$ or $j_i = k_i$ for some i. Due to identifications, boundary grid points might have more than one representation using (5.36).

[10] Alternatively, the general lattice definition in (5.21) could be used.

Neighborhoods For each grid point q we need to define the set of nearby grid points for which an edge may be constructed. Special care must be given to defining the neighborhood of a boundary grid point to ensure that identifications and the C-space boundary (if it exists) are respected. If q is not a boundary grid point, then the *1-neighborhood* is defined as

$$N_1(q) = \{q + \Delta q_1, \ldots, q + \Delta q_n, q - \Delta q_1, \ldots, q - \Delta q_n\}. \tag{5.37}$$

For an n-dimensional C-space there at most $2n$ 1-neighbors. In two dimensions, this yields at most four 1-neighbors, which can be thought of as "up," "down," "left," and "right." There are *at most* four because some directions may be blocked by the obstacle region.

A *2-neighborhood* is defined as

$$N_2(q) = \{q \pm \Delta q_i \pm \Delta q_j \ \mid \ 1 \leq i, j \leq n, \ i \neq j\} \cup N_1(q). \tag{5.38}$$

Similarly, a *k-neighborhood* can be defined for any positive integer $k \leq n$. For an n-neighborhood, there are at most $3^n - 1$ neighbors; there may be fewer due to boundaries or collisions. The definitions can be easily extended to handle the boundary points.

Obtaining a discrete planning problem Once the grid and neighborhoods have been defined, a discrete planning problem is obtained. Figure 5.14 depicts the process for a problem in which there are nine Sukharev grid points in $[0,1]^2$. Using 1-neighborhoods, the potential edges in the search graph, $\mathcal{G}(V, E)$, appear in Figure 5.14a. Note that $\mathcal{G}$ is a topological graph, as defined in Example 4.6, because each vertex is a configuration and each edge is a path. If q_I and q_G do not coincide with grid points, they need to be connected to some nearby grid points, as shown in Figure 5.14b. What grid points should q_I and q_G be connected to? As a general rule, if k-neighbors are used, then one should try connecting q_I and q_G to any grid points that are at least as close as the furthest k-neighbor from a typical grid point.

Usually, all of the vertices and edges shown in Figure 5.14b do not appear in $\mathcal{G}$ because some intersect with $\mathcal{C}_{obs}$. Figure 5.14c shows a more typical situation, in which some of the potential vertices and edges are removed because of collisions. This representation could be computed in advance by checking all potential vertices and edges for collision. This would lead to a roadmap, which is suited for multiple queries and is covered in Section 5.6. In this section, it is assumed that $\mathcal{G}$ is revealed "on the fly" during the search. This is the same situation that occurs for the discrete planning methods from Section 2.2. In the current setting, the potential edges of $\mathcal{G}$ are validated during the search. The candidate edges to

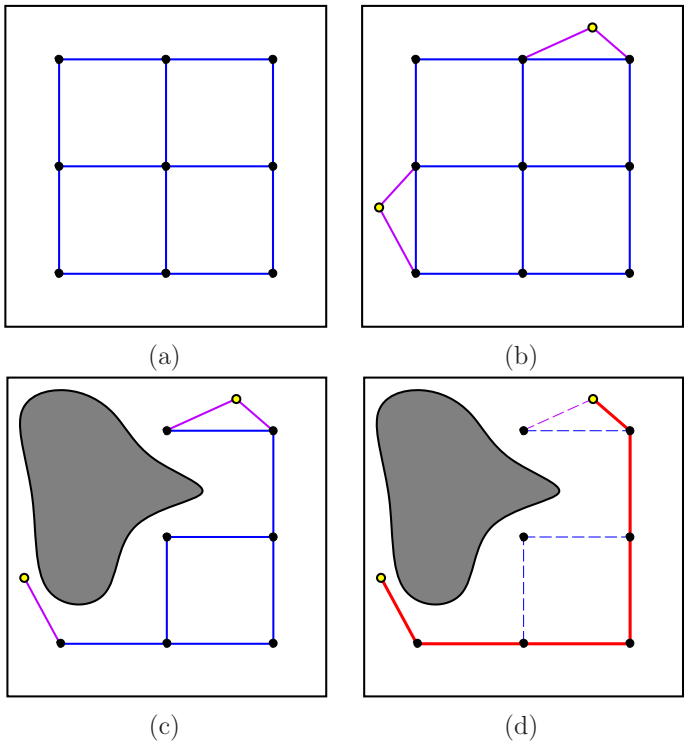


Figure 5.14: A topological graph can be constructed during the search and can successfully solve a motion planning problem using very few samples.

evaluate are given by the definition of the k-neighborhoods. During the search, any edge or vertex that has been checked for collision explicitly appears in a data structure so that it does not need to be checked again. At the end of the search, a path is found, as depicted in Figure 5.14d.

Grid resolution issues The method explained so far will nicely find the solution to many problems when provided with the correct resolution. If the number of points per axis is too high, then the search may be too slow. This motivates selecting fewer points per axis, but then solutions might be missed. This trade-off is fundamental to sampling-based motion planning. In a more general setting, if other forms of sampling and neighborhoods are used, then enough samples have to be generated to yield a sufficiently small dispersion.

There are two general ways to avoid having to select this resolution (or more generally, dispersion):

1. Iteratively refine the resolution until a solution is found. In this case, sampling and searching become interleaved. One important variable is how frequently to alternate between the two processes. This will be presented shortly.

2. An alternative is to abandon the adaptation of discrete search algorithms and develop algorithms directly for the continuous problem. This forms the basis of the methods in Sections 5.4.3, 5.4.4, and 5.5.

The most straightforward approach is to iteratively improve the grid resolution. Suppose that initially a standard grid with 2^n points total and 2 points per axis is searched using one of the discrete search algorithms, such as best-first or A^*. If the search fails, what should be done? One possibility is to double the resolution, which yields a grid with 4^n points. Many of the edges can be reused from the first grid; however, the savings diminish rapidly in higher dimensions. Once the resolution is doubled, the search can be applied again. If it fails again, then the resolution can be doubled again to yield 8^n points. In general, there would be a full grid for 2^{ni} points, for each i. The problem is that if n is large, then the rate of growth is too large. For example, if $n = 10$, then there would initially be 1024 points; however, when this fails, the search is not performed again until there are over one million points! If this also fails, then it might take a very long time to reach the next level of resolution, which has 2^{30} points.

A method similar to iterative deepening from Section 2.2.2 would be preferable. Simply discard the efforts of the previous resolution and make grids that have i^n points per axis for each iteration i. This yields grids of sizes 2^n, 3^n, 4^n, and so on, which is much better. The amount of effort involved in searching a larger grid is insignificant compared to the time wasted on lower resolution grids. Therefore, it seems harmless to discard previous work.

A better solution is not to require that a complete grid exists before it can be searched. For example, the resolution can be increased for one axis at a time before attempting to search again. Even better yet may be to tightly interleave searching and sampling. For example, imagine that the samples appear as an infinite, dense sequence α. The graph can be searched after every 100 points are added, assuming that neighborhoods can be defined or constructed even though the grid is only partially completed. If the search is performed too frequently, then searching would dominate the running time. An easy way make this efficient is to apply the *union-find algorithm* [132, 413] to iteratively keep track of connected components in $\mathcal{G}$ instead of performing explicit searching. If q_I and q_G become part of the same connected component, then a solution path has been found. Every time a new point in the sequence α is added, the "search" is performed in nearly[11] constant time by the union-find algorithm. This is the tightest interleaving of the sampling and searching, and results in a nice sampling-based algorithm that

[11] It is not constant because the running time is proportional to the inverse Ackerman function, which grows very, very slowly. For all practical purposes, the algorithm operates in constant time. See Section 6.5.2.

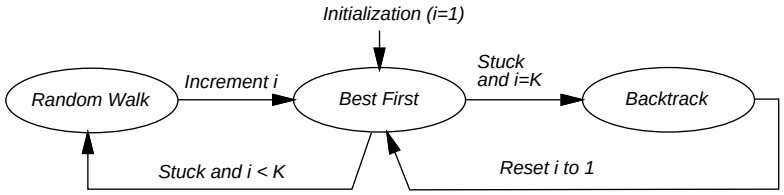


Figure 5.15: The randomized potential field method can be modeled as a three-state machine.

requires no resolution parameter. It is perhaps best to select a sequence α that contains some lattice structure to facilitate the determination of neighborhoods in each iteration.

What if we simply declare the resolution to be outrageously high at the outset? Imagine there are 100^n points in the grid. This places all of the burden on the search algorithm. If the search algorithm itself is good at avoiding local minima and has built-in multi-resolution qualities, then it may perform well without the iterative refinement of the sampling. The method of Section 5.4.3 is based on this idea by performing best-first search on a high-resolution grid, combined with random walks to avoid local minima. The algorithms of Section 5.5 go one step further and search in a multi-resolution way without requiring resolutions and neighborhoods to be explicitly determined. This can be considered as the limiting case as the number of points per axis approaches infinity.

Although this section has focused on grids, it is also possible to use other forms of sampling from Section 5.2. This requires defining the neighborhoods in a suitable way that generalizes the k-neighborhoods of this section. In every case, an infinite, dense sample sequence must be defined to obtain resolution completeness by reducing the dispersion to zero in the limit. Methods for obtaining neighborhoods for irregular sample sets have been developed in the context of sampling-based roadmaps; see Section 5.6. The notion of improving resolution becomes generalized to adding samples that improve dispersion (or even discrepancy).

5.4.3 Randomized Potential Fields

Adapting the discrete algorithms from Section 2.2 works well if the problem can be solved with a small number of points. The number of points per axis must be small or the dimension must be low, to ensure that the number of points, k^n, for k points per axis and n dimensions is small enough so that every vertex in g can be reached in a reasonable amount of time. If, for example, the problem requires 50 points per axis and the dimension is 10, then it is impossible to search all of the 50^{10} samples. Planners that exploit best-first heuristics might find the answer without searching most of them; however, for a simple problem such as

that shown in Figure 5.13a, the planner will take too long exploring the vertices in the bowl.[12]

The *randomized potential field* [48, 50, 304] approach uses random walks to attempt to escape local minima when best-first search becomes stuck. It was one of the first sampling-based planners that developed specialized techniques beyond classical discrete search, in an attempt to better solve challenging motion planning problems. In many cases, remarkable results were obtained. In its time, the approach was able to solve problems up to 31 degrees of freedom, which was well beyond what had been previously possible. The main drawback, however, was that the method involved many heuristic parameters that had to be adjusted for each problem. This frustration eventually led to the development of better approaches, which are covered in Sections 5.4.4, 5.5, and 5.6. Nevertheless, it is worthwhile to study the clever heuristics involved in this earlier method because they illustrate many interesting issues, and the method was very influential in the development of other sampling-based planning algorithms.[13]

The most complicated part of the algorithm is the definition of a *potential function*, which can be considered as a pseudometric that tries to estimate the distance from any configuration to the goal. In most formulations, there is an *attractive* term, which is a metric on $\mathcal{C}$ that yields the distance to the goal, and a *repulsive* term, which penalizes configurations that come too close to obstacles. The construction of potential functions involves many heuristics and is covered in great detail in [304]. One of the most effective methods involves constructing cost-to-go functions over $\mathcal{W}$ and lifting them to $\mathcal{C}$ [49]. In this section, it will be sufficient to assume that some potential function, $g(q)$, is defined, which is the same notation (and notion) as a cost-to-go function in Section 2.2.2. In this case, however, there is no requirement that $g(q)$ is optimal or even an underestimate of the true cost to go.

When the search becomes stuck and a random walk is needed, it is executed for some number of iterations. Using the discretization procedures of Section 5.4.2, a high-resolution grid (e.g., 50 points per axis) is initially defined. In each iteration, the current configuration is modified as follows. Each coordinate, q_i, is increased or decreased by Δq_i (the grid step size) based on the outcome of a fair coin toss. Topological identifications must be respected, of course. After each iteration, the new configuration is checked for collision, or whether it exceeds the boundary of $\mathcal{C}$ (if it has a boundary). If so, then it is discarded, and another attempt is made from the previous configuration. The failures can repeat indefinitely until a new configuration in $\mathcal{C}_{free}$ is obtained.

The resulting planner can be described in terms of a three-state machine, which is shown in Figure 5.15. Each state is called a *mode* to avoid confusion with

[12] Of course, that problem does not appear to need so many points per axis; fewer may be used instead, if the algorithm can adapt the sampling resolution or dispersion.

[13] The exciting results obtained by the method even helped inspire me many years ago to work on motion planning.

earlier state-space concepts. The VSM and LPM are defined in terms of the mode. Initially, the planner is in the BEST FIRST mode and uses q_I to start a gradient descent. While in the BEST FIRST mode, the VSM selects the newest vertex, $v \in V$. In the first iteration, this is q_I. The LPM creates a new vertex, v_n, in a neighborhood of v, in a direction that minimizes g. The direction sampling may be performed using randomly selected or deterministic samples. Using random samples, the sphere sampling method from Section 5.2.2 can be applied. After some number of tries (another parameter), if the LPM is unsuccessful at reducing g, then the mode is changed to RANDOM WALK because the best-first search is stuck in a local minimum of g.

In the RANDOM WALK mode, a random walk is executed from the newest vertex. The random walk terminates if either g is lowered or a specified limit of iterations is reached. The limit is actually sampled from a predetermined random variable (which contains parameters that also must be selected). When the RANDOM WALK mode terminates, the mode is changed back to BEST FIRST. A counter is incremented to keep track of the number of times that the random walk was attempted. A parameter K determines the maximum number of attempted random walks (a reasonable value is $K = 20$ [49]). If BEST FIRST fails after K random walks have been attempted, then the BACKTRACK mode is entered. The BACKTRACK mode selects a vertex at random from among the vertices in V that were obtained during a random walk. Following this, the counter is reset, and the mode is changed back to BEST FIRST.

Due to the random walks, the resulting paths are often too complicated to be useful in applications. Fortunately, it is straightforward to transform a computed path into a simpler one that is still collision-free. A common approach is to iteratively pick pairs of points at random along the domain of the path and attempt to replace the path segment with a straight-line path (in general, the shortest path in $\mathcal{C}$). For example, suppose $t_1, t_2 \in [0, 1]$ are chosen at random, and $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ is the computed solution path. This path is transformed into a new path,

$$\tau'(t) = \begin{cases} \tau(t) & \text{if } 0 \leq t \leq t_1 \\ a\tau(t_1) + (1-a)\tau(t_2) & \text{if } t_1 \leq t \leq t_2 \\ \tau(t) & \text{if } t_2 \leq t \leq 1, \end{cases} \tag{5.39}$$

in which $a \in [0, 1]$ represents the fraction of the way from t_1 to t_2. Explicitly, $a = (t_2 - t)/(t_2 - t_1)$. The new path must be checked for collision. If it passes, then it replaces the old path; otherwise, it is discarded and a new pair t_1, t_2, is chosen.

The randomized potential field approach can escape high-dimensional local minima, which allow interesting solutions to be found for many challenging high-dimensional problems. Unfortunately, the heavy amount of parameter tuning caused most people to abandon the method in recent times, in favor of newer methods.

5.4.4 Other Methods

Several influential sampling-based methods are given here. Each of them appears to offer advantages over the randomized potential field method. All of them use randomization, which was perhaps inspired by the potential field method.

Ariadne's Clew algorithm This approach grows a search tree that is biased to explore as much new territory as possible in each iteration [358, 357]. There are two modes, SEARCH and EXPLORE, which alternate over successive iterations. In the EXPLORE mode, the VSM selects a vertex, v_e, at random, and the LPM finds a new configuration that can be easily connected to v_e and is as far as possible from the other vertices in $\mathcal{G}$. A global optimization function that aggregates the distances to other vertices is optimized using a genetic algorithm. In the SEARCH mode, an attempt is made to extend the vertex added in the EXPLORE mode to the goal configuration. The key idea from this approach, which influenced both the next approach and the methods in Section 5.5, is that some of the time must be spent exploring the space, as opposed to focusing on finding the solution. The greedy behavior of the randomized potential field led to some efficiency but was also its downfall for some problems because it was all based on escaping local minima with respect to the goal instead of investing time on global exploration. One disadvantage of Ariadne's Clew algorithm is that it is very difficult to solve the optimization problem for placing a new vertex in the EXPLORE mode. Genetic algorithms were used in [357], which are generally avoided for motion planning because of the required problem-specific parameter tuning.

Expansive-space planner This method [242, 424] generates samples in a way that attempts to explore new parts of the space. In this sense, it is similar to the explore mode of the Ariadne's Clew algorithm. Furthermore, the planner is made more efficient by borrowing the bidirectional search idea from discrete search algorithms, as covered in Section 2.2.3. The VSM selects a vertex, v_e, from $\mathcal{G}$ with a probability that is inversely proportional to the number of other vertices of $\mathcal{G}$ that lie within a predetermined neighborhood of v_e. Thus, "isolated" vertices are more likely to be chosen. The LPM generates a new vertex v_n at random within a predetermined neighborhood of v_e. It will decide to insert v_n into $\mathcal{G}$ with a probability that is inversely proportional to the number of other vertices of $\mathcal{G}$ that lie within a predetermined neighborhood of v_n. For a fixed number of iterations, the VSM repeatedly chooses the same vertex, until moving on to another vertex. The resulting planner is able to solve many interesting problems by using a surprisingly simple criterion for the placement of points. The main drawbacks are that the planner requires substantial parameter tuning, which is problem-specific (or at least specific to a similar family of problems), and the performance tends to degrade if the query requires systematically searching a long labyrinth. Choosing the radius of the predetermined neighborhoods essentially amounts to determining the appropriate resolution.

Random-walk planner A surprisingly simple and efficient algorithm can be made entirely from random walks [96]. To avoid parameter tuning, the algorithm adjusts its distribution of directions and magnitude in each iteration, based on the success of the past k iterations (perhaps k is the only parameter). In each iteration, the VSM just selects the vertex that was most recently added to $\mathcal{G}$. The LPM generates a direction and magnitude by generating samples from a multivariate Gaussian distribution whose covariance parameters are adaptively tuned. The main drawback of the method is similar to that of the previous method. Both have difficulty traveling through long, winding corridors. It is possible to combine adaptive random walks with other search algorithms, such as the potential field planner [95].

5.5 Rapidly Exploring Dense Trees

This section introduces an incremental sampling and searching approach that yields good performance in practice without any parameter tuning.[14] The idea is to incrementally construct a search tree that gradually improves the resolution but does not need to explicitly set any resolution parameters. In the limit, the tree densely covers the space. Thus, it has properties similar to space filling curves [423], but instead of one long path, there are shorter paths that are organized into a tree. A dense sequence of samples is used as a guide in the incremental construction of the tree. If this sequence is random, the resulting tree is called a *rapidly exploring random tree (RRT)*. In general, this family of trees, whether the sequence is random or deterministic, will be referred to as *rapidly exploring dense trees (RDTs)* to indicate that a dense covering of the space is obtained. This method was originally developed for motion planning under differential constraints [311, 314]; that case is covered in Section 14.4.3.

5.5.1 The Exploration Algorithm

Before explaining how to use these trees to solve a planning query, imagine that the goal is to get as close as possible to every configuration, starting from an initial configuration. The method works for any dense sequence. Once again, let α denote an infinite, dense sequence of samples in $\mathcal{C}$. The ith sample is denoted by $\alpha(i)$. This may possibly include a uniform, random sequence, which is only dense with probability one. Random sequences that induce a nonuniform bias are also acceptable, as long as they are dense with probability one.

An RDT is a topological graph, $\mathcal{G}(V, E)$. Let $S \subset \mathcal{C}_{free}$ indicate the set of all points reached by $\mathcal{G}$. Since each $e \in E$ is a path, this can be expressed as the

[14] The original RRT [306] was introduced with a step size parameter, but this is eliminated in the current presentation. For implementation purposes, one might still want to revert to this older way of formulating the algorithm because the implementation is a little easier. This will be discussed shortly.


```
SIMPLE_RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      G.add_vertex(α(i));
4      q_n ← NEAREST(S(G), α(i));
5      G.add_edge(q_n, α(i));
```

Figure 5.16: The basic algorithm for constructing RDTs (which includes RRTs as a special case) when there are no obstacles. It requires the availability of a dense sequence, α, and iteratively connects from $\alpha(i)$ to the nearest point among all those reached by $\mathcal{G}$.

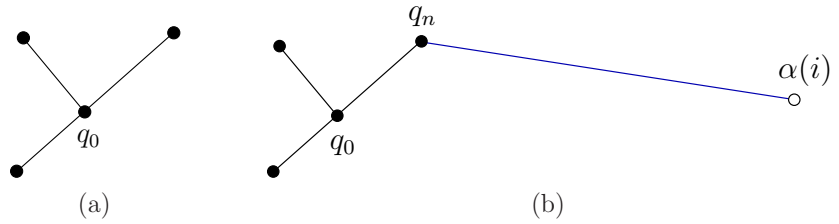


Figure 5.17: (a) Suppose inductively that this tree has been constructed so far using the algorithm in Figure 5.16. (b) A new edge is added that connects from the sample $\alpha(i)$ to the nearest point in S, which is the vertex q_n.

swath, S, of the graph, which is defined as

$$S = \bigcup_{e \in E} e([0,1]). \tag{5.40}$$

In (5.40), $e([0,1]) \subseteq \mathcal{C}_{free}$ is the image of the path e.

The exploration algorithm is first explained in Figure 5.16 without any obstacles or boundary obstructions. It is assumed that $\mathcal{C}$ is a metric space. Initially, a vertex is made at q_0. For k iterations, a tree is iteratively grown by connecting $\alpha(i)$ to its nearest point in the swath, S. The connection is usually made along

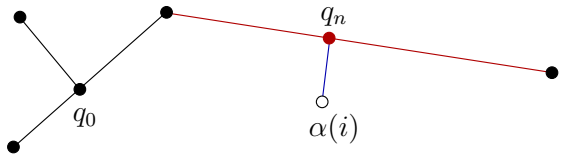


Figure 5.18: If the nearest point in S lies in an edge, then the edge is split into two, and a new vertex is inserted into $\mathcal{G}$.

45 iterations 2345 iterations

Figure 5.19: In the early iterations, the RRT quickly reaches the unexplored parts. However, the RRT is dense in the limit (with probability one), which means that it gets arbitrarily close to any point in the space.

the shortest possible path. In every iteration, $\alpha(i)$ becomes a vertex. Therefore, the resulting tree is dense. Figures 5.17–5.18 illustrate an iteration graphically. Suppose the tree has three edges and four vertices, as shown in Figure 5.17a. If the nearest point, $q_n \in S$, to $\alpha(i)$ is a vertex, as shown in Figure 5.17b, then an edge is made from q_n to $\alpha(i)$. However, if the nearest point lies in the interior of an edge, as shown in Figure 5.18, then the existing edge is split so that q_n appears as a new vertex, and an edge is made from q_n to $\alpha(i)$. The edge splitting, if required, is assumed to be handled in line 4 by the method that adds edges. Note that the total number of edges may increase by one or two in each iteration.

The method as described here does not fit precisely under the general framework from Section 5.4.1; however, with the modifications suggested in Section 5.5.2, it can be adapted to fit. In the RDT formulation, the NEAREST function serves the purpose of the VSM, but in the RDT, a point may be selected from anywhere in the swath of the graph. The VSM can be generalized to a *swath-point selection method*, SSM. This generalization will be used in Section 14.3.4. The LPM tries to connect $\alpha(i)$ to q_n along the shortest path possible in $\mathcal{C}$.

Figure 5.19 shows an execution of the algorithm in Figure 5.16 for the case in which $\mathcal{C} = [0,1]^2$ and $q_0 = (1/2, 1/2)$. It exhibits a kind of fractal behavior.[15] Several main branches are first constructed as it rapidly reaches the far corners of the space. Gradually, more and more area is filled in by smaller branches. From the pictures, it is clear that in the limit, the tree densely fills the space. Thus,

[15]If α is uniform, random, then a *stochastic fractal* [303] is obtained. Deterministic fractals can be constructed using sequences that have appropriate symmetries.

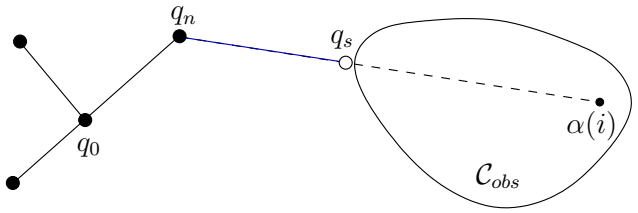


Figure 5.20: If there is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by the collision detection algorithm.

```
RDT(q_0)
1  G.init(q_0);
2  for i = 1 to k do
3      q_n ← NEAREST(S, α(i));
4      q_s ← STOPPING-CONFIGURATION(q_n, α(i));
5      if q_s ≠ q_n then
6          G.add_vertex(q_s);
7          G.add_edge(q_n, q_s);
```

Figure 5.21: The RDT with obstacles.

it can be seen that the tree gradually improves the resolution (or dispersion) as the iterations continue. This behavior turns out to be ideal for sampling-based motion planning.

Recall that in sampling-based motion planning, the obstacle region $\mathcal{C}_{obs}$ is not explicitly represented. Therefore, it must be taken into account in the construction of the tree. Figure 5.20 indicates how to modify the algorithm in Figure 5.16 so that collision checking is taken into account. The modified algorithm appears in Figure 5.21. The procedure STOPPING-CONFIGURATION yields the nearest configuration possible to the boundary of $\mathcal{C}_{free}$, along the direction toward $\alpha(i)$. The nearest point $q_n \in S$ is defined to be same (obstacles are ignored); however, the new edge might not reach to $\alpha(i)$. In this case, an edge is made from q_n to q_s, the last point possible before hitting the obstacle. How close can the edge come to the obstacle boundary? This depends on the method used to check for collision, as explained in Section 5.3.4. It is sometimes possible that q_n is already as close as possible to the boundary of $\mathcal{C}_{free}$ in the direction of $\alpha(i)$. In this case, no new edge or vertex is added that for that iteration.

5.5.2 Efficiently Finding Nearest Points

There are several interesting alternatives for implementing the NEAREST function in line 3 of the algorithm in Figure 5.16. There are generally two families of methods: *exact* or *approximate*. First consider the exact case.

Exact solutions Suppose that all edges in $\mathcal{G}$ are line segments in $\mathbb{R}^m$ for some dimension $m \geq n$. An edge that is generated early in the construction process will be split many times in later iterations. For the purposes of finding the nearest point in S, however, it is best to handle this as a single segment. For example, see the three large branches that extend from the root in Figure 5.19. As the number of points increases, the benefit of agglomerating the segments increases. Let each of these agglomerated segments be referred to as a *supersegment*. To implement NEAREST, a primitive is needed that computes the distance between a point and a line segment. This can be performed in constant time with simple vector calculus. Using this primitive, NEAREST is implemented by iterating over all supersegments and taking the point with minimum distance among all of them. It may be possible to improve performance by building hierarchical data structures that can eliminate large sets of supersegments, but this remains to be seen experimentally.

In some cases, the edges of $\mathcal{G}$ may not be line segments. For example, the shortest paths between two points in $SO(3)$ are actually circular arcs along $\mathbb{S}^3$. One possible solution is to maintain a separate parameterization of $\mathcal{C}$ for the purposes of computing the NEAREST function. For example, $SO(3)$ can be represented as $[0,1]^3/\sim$, by making the appropriate identifications to obtain $\mathbb{RP}^3$. Straight-line segments can then be used. The problem is that the resulting metric is not consistent with the Haar measure, which means that an accidental bias would result. Another option is to tightly enclose $\mathbb{S}^3$ in a 4D cube. Every point on $\mathbb{S}^3$ can be mapped outward onto a cube face. Due to antipodal identification, only four of the eight cube faces need to be used to obtain a bijection between the set of all rotation and the cube surface. Linear interpolation can be used along the cube faces, as long as both points remain on the same face. If the points are on different faces, then two line segments can be used by bending the shortest path around the corner between the two faces. This scheme will result in less distortion than mapping $SO(3)$ to $[0,1]^3/\sim$; however, some distortion will still exist.

Another approach is to avoid distortion altogether and implement primitives that can compute the distance between a point and a curve. In the case of $SO(3)$, a primitive is needed that can find the distance between a circular arc in $\mathbb{R}^m$ and a point in $\mathbb{R}^m$. This might not be too difficult, but if the curves are more complicated, then an exact implementation of the NEAREST function may be too expensive computationally.

Approximate solutions Approximate solutions are much easier to construct, however, a resolution parameter is introduced. Each path segment can be approximated by inserting intermediate vertices along long segments, as shown in Figure 5.22. The intermediate vertices should be added each time a new sample, $\alpha(i)$, is inserted into $\mathcal{G}$. A parameter Δq can be defined, and intermediate samples are inserted to ensure that no two consecutive vertices in $\mathcal{G}$ are ever further than Δq from each other. Using intermediate vertices, the interiors of the edges in $\mathcal{G}$ are ignored when finding the nearest point in S. The approximate computation of

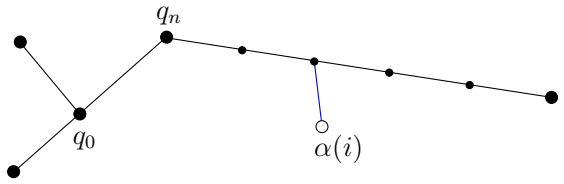


Figure 5.22: For implementation ease, intermediate vertices can be inserted to avoid checking for closest points along line segments. The trade-off is that the number of vertices is increased dramatically.

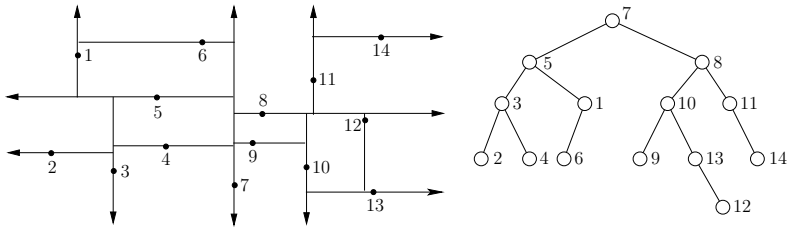


Figure 5.23: A Kd-tree can be used for efficient nearest-neighbor computations.

NEAREST is performed by finding the closest vertex to $\alpha(i)$ in $\mathcal{G}$. This approach is by far the simplest to implement. It also fits precisely under the incremental sampling and searching framework from Section 5.4.1.

When using intermediate vertices, the trade-offs are clear. The computation time for each evaluation of NEAREST is linear in the number of vertices. Increasing the number of vertices improves the quality of the approximation, but it also dramatically increases the running time. One way to recover some of this cost is to insert the vertices into an efficient data structure for nearest-neighbor searching. One of the most practical and widely used data structures is the *Kd-tree* [146, 183, 391]. A depiction is shown in Figure 5.23 for 14 points in $\mathbb{R}^2$. The Kd-tree can be considered as a multi-dimensional generalization of a binary search tree. The Kd-tree is constructed for points, P, in $\mathbb{R}^2$ as follows. Initially, sort the points with respect to the x coordinate. Take the median point, $p \in P$, and divide P into two sets, depending on which side of a vertical line through p the other points fall. For each of the two sides, sort the points by the y coordinate and find their medians. Points are divided at this level based on whether they are above or below horizontal lines. At the next level of recursion, vertical lines are used again, followed by horizontal again, and so on. The same idea can be applied in $\mathbb{R}^n$ by cycling through the n coordinates, instead of alternating between x and y, to form the divisions. In [38], the Kd-tree is extended to topological spaces that arise in motion planning and is shown to yield good performance for RRTs and sampling-based roadmaps. A Kd-tree of k points can be constructed in $O(nk \lg k)$ time. Topological identifications must be carefully considered when traversing

the tree. To find the nearest point in the tree to some given point, the query algorithm descends to a leaf vertex whose associated region contains the query point, finds all distances from the data points in this leaf to the query point, and picks the closest one. Next, it recursively visits those surrounding leaf vertices that are further from the query point than the closest point found so far [36, 38]. The nearest point can be found in time logarithmic in k.

Unfortunately, these bounds hide a constant that increases exponentially with the dimension, n. In practice, the Kd-tree is useful in motion planning for problems of up to about 20 dimensions. After this, the performance usually degrades too much. As an empirical rule, if there are more than 2^n points, then the Kd-tree should be more efficient than naive nearest neighbors. In general, the trade-offs must be carefully considered in a particular application to determine whether exact solutions, approximate solutions with naive nearest-neighbor computations, or approximate solutions with Kd-trees will be more efficient. There is also the issue of implementation complexity, which probably has caused most people to prefer the approximate solution with naive nearest-neighbor computations.

5.5.3 Using the Trees for Planning

So far, the discussion has focused on exploring $\mathcal{C}_{free}$, but this does not solve a planning query by itself. RRTs and RDTs can be used in many ways in planning algorithms. For example, they could be used to escape local minima in the randomized potential field planner of Section 5.4.3.

Single-tree search A reasonably efficient planner can be made by directly using the algorithm in Figure 5.21 to grow a tree from q_I and periodically check whether it is possible to connect the RDT to q_G. An easy way to achieve this is to start with a dense sequence α and periodically insert q_G at regularly spaced intervals. For example, every 100th sample could be q_G. Each time this sample is reached, an attempt is made to reach q_G from the closest vertex in the RDT. If the sample sequence is random, which generates an RRT, then the following modification works well. In each iteration, toss a biased coin that has probability 99/100 of being HEADS and 1/100 of being TAILS. If the result is HEADS, then set $\alpha(i)$, to be the next element of the pseudorandom sequence; otherwise, set $\alpha(i) = q_G$. This forces the RRT to occasionally attempt to make a connection to the goal, q_G. Of course, 1/100 is arbitrary, but it is in a range that works well experimentally. If the bias is too strong, then the RRT becomes too greedy like the randomized potential field. If the bias is not strong enough, then there is no incentive to connect the tree to q_G. An alternative is to consider other dense, but not necessarily nonuniform sequences in $\mathcal{C}$. For example, in the case of random sampling, the probability density function could contain a gentle bias towards the goal. Choosing such a bias is a difficult heuristic problem; therefore, such a technique should be used with caution (or avoided altogether).


```
RDT_BALANCED_BIDIRECTIONAL(q_I, q_G)
 1  T_a.init(q_I); T_b.init(q_G);
 2  for i = 1 to K do
 3      q_n ← NEAREST(S_a, α(i));
 4      q_s ← STOPPING-CONFIGURATION(q_n, α(i));
 5      if q_s ≠ q_n then
 6          T_a.add_vertex(q_s);
 7          T_a.add_edge(q_n, q_s);
 8          q'_n ← NEAREST(S_b, q_s);
 9          q'_s ← STOPPING-CONFIGURATION(q'_n, q_s);
10          if q'_s ≠ q'_n then
11              T_b.add_vertex(q'_s);
12              T_b.add_edge(q'_n, q'_s);
13          if q'_s = q_s then return SOLUTION;
14      if |T_b| > |T_a| then SWAP(T_a, T_b);
15  return FAILURE
```

Figure 5.24: A bidirectional RDT-based planner.

Balanced, bidirectional search Much better performance can usually be obtained by growing two RDTs, one from q_I and the other from q_G. This is particularly valuable for escaping one of the bug traps, as mentioned in Section 5.4.1. For a grid search, it is straightforward to implement a bidirectional search that ensures that the two trees meet. For the RDT, special considerations must be made to ensure that the two trees will connect while retaining their "rapidly exploring" property. One additional idea is to make sure that the bidirectional search is balanced [293], which ensures that both trees are the same size.

Figure 5.24 gives an outline of the algorithm. The graph $\mathcal{G}$ is decomposed into two trees, denoted by T_a and T_b. Initially, these trees start from q_I and q_G, respectively. After some iterations, T_a and T_b are swapped; therefore, keep in mind that T_a is not always the tree that contains q_I. In each iteration, T_a is grown exactly the same way as in one iteration of the algorithm in Figure 5.16. If a new vertex, q_s, is added to T_a, then an attempt is made in lines 10–12 to extend T_b. Rather than using $\alpha(i)$ to extend T_b, the new vertex q_s of T_a is used. This causes T_b to try to grow toward T_a. If the two connect, which is tested in line 13, then a solution has been found.

Line 14 represents an important step that balances the search. This is particularly important for a problem such as the bug trap shown in Figure 5.13b or the puzzle shown in Figure 1.2. If one of the trees is having trouble exploring, then it makes sense to focus more energy on it. Therefore, new exploration is always performed for the smaller tree. How is "smaller" defined? A simple criterion is to use the total number of vertices. Another reasonable criterion is to use the total length of all segments in the tree.

An unbalanced bidirectional search can instead be made by forcing the trees to be swapped in every iteration. Once the trees are swapped, then the roles are reversed. For example, after the first swap, T_b is extended in the same way as an integration in Figure 5.16, and if a new vertex q_s is added then an attempt is made to connect T_a to q_s.

One important concern exists when α is deterministic. It might be possible that even though α is dense, when the samples are divided among the trees, each may not receive a dense set. If each uses its own deterministic sequence, then this problem can be avoided. In the case of making a bidirectional RRT planner, the same (pseudo)random sequence can be used for each tree without encountering such troubles.

More than two trees If a dual-tree approach offers advantages over a single tree, then it is natural to ask whether growing three or more RDTs might be even better. This is particularly helpful for problems like the double bug trap in Figure 5.13c. New trees can be grown from parts of $\mathcal{C}$ that are difficult to reach. Controlling the number of trees and determining when to attempt connections between them is difficult. Some interesting recent work has been done in this direction [54, 453, 454].

These additional trees could be started at arbitrary (possibly random) configurations. As more trees are considered, a complicated decision problem arises. The computation time must be divided between attempting to explore the space and attempting to connect trees to each other. It is also not clear which connections should be attempted. Many research issues remain in the development of this and other RRT-based planners. A limiting case would be to start a new tree from every sample in $\alpha(i)$ and to try to connect nearby trees whenever possible. This approach results in a graph that covers the space in a nice way that is independent of the query. This leads to the main topic of the next section.

5.6 Roadmap Methods for Multiple Queries

Previously, it was assumed that a single initial-goal pair was given to the planning algorithm. Suppose now that numerous initial-goal queries will be given to the algorithm, while keeping the robot model and obstacles fixed. This leads to a *multiple-query* version of the motion planning problem. In this case, it makes sense to invest substantial time to preprocess the models so that future queries can be answered efficiently. The goal is to construct a topological graph called a *roadmap*, which efficiently solves multiple initial-goal queries. Intuitively, the paths on the roadmap should be easy to reach from each of q_I and q_G, and the graph can be quickly searched for a solution. The general framework presented here was mainly introduced in [265] under the name *probabilistic roadmaps (PRMs)*. The probabilistic aspect, however, is not important to the method. Therefore, we call


```
BUILD_ROADMAP
1  G.init(); i ← 0;
2  while i < N
3      if α(i) ∈ C_free then
4          G.add_vertex(α(i)); i ← i + 1;
5          for each q ∈ NEIGHBORHOOD(α(i),G)
6              if ((not G.same_component(α(i), q)) and CONNECT(α(i), q)) then
7                  G.add_edge(α(i), q);
```

Figure 5.25: The basic construction algorithm for sampling-based roadmaps. Note that i is not incremented if $\alpha(i)$ is in collision. This forces i to correctly count the number of vertices in the roadmap.

this family of methods *sampling-based roadmaps*. This distinguishes them from *combinatorial roadmaps*, which will appear in Chapter 6.

5.6.1 The Basic Method

Once again, let $\mathcal{G}(V, E)$ represent a topological graph in which V is a set of vertices and E is the set of paths that map into $\mathcal{C}_{free}$. Under the multiple-query philosophy, motion planning is divided into two phases of computation:

> **Preprocessing Phase:** During the preprocessing phase, substantial effort is invested to build $\mathcal{G}$ in a way that is useful for quickly answering future queries. For this reason, it is called a *roadmap*, which in some sense should be accessible from every part of $\mathcal{C}_{free}$.
>
> **Query Phase:** During the query phase, a pair, q_I and q_G, is given. Each configuration must be connected easily to $\mathcal{G}$ using a local planner. Following this, a discrete search is performed using any of the algorithms in Section 2.2 to obtain a sequence of edges that forms a path from q_I to q_G.

Generic preprocessing phase Figure 5.25 presents an outline of the basic preprocessing phase, and Figure 5.26 illustrates the algorithm. As seen throughout this chapter, the algorithm utilizes a uniform, dense sequence α. In each iteration, the algorithm must check whether $\alpha(i) \in \mathcal{C}_{free}$. If $\alpha(i) \in \mathcal{C}_{obs}$, then it must continue to iterate until a collision-free sample is obtained. Once $\alpha(i) \in \mathcal{C}_{free}$, then in line 4 it is inserted as a vertex of $\mathcal{G}$. The next step is to try to connect $\alpha(i)$ to some nearby vertices, q, of $\mathcal{G}$. Each connection is attempted by the CONNECT function, which is a typical LPM (local planning method) from Section 5.4.1. In most implementations, this simply tests the shortest path between $\alpha(i)$ and q. Experimentally, it seems most efficient to use the multi-resolution, van der Corput–based method described at the end of Section 5.3.4 [191]. Instead of the shortest path, it is possible to use more sophisticated connection methods, such

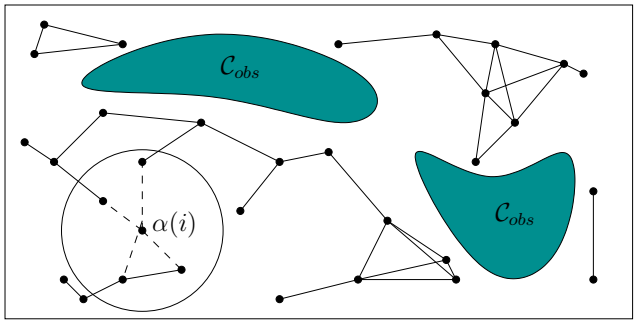


Figure 5.26: The sampling-based roadmap is constructed incrementally by attempting to connect each new sample, $\alpha(i)$, to nearby vertices in the roadmap.

as the bidirectional algorithm in Figure 5.24. If the path is collision-free, then CONNECT returns TRUE.

The same_component condition in line 6 checks to make sure $\alpha(i)$ and q are in different components of $\mathcal{G}$ before wasting time on collision checking. This ensures that every time a connection is made, the number of connected components of $\mathcal{G}$ is decreased. This can be implemented very efficiently (near constant time) using the previously mentioned *union-find algorithm* [132, 413]. In some implementations this step may be ignored, especially if it is important to generate multiple, alternative solutions. For example, it may be desirable to generate solution paths from different homotopy classes. In this case the condition (**not** $\mathcal{G}$.same_component$(\alpha(i), q)$) is replaced with $\mathcal{G}$.vertex_degree$(q) < K$, for some fixed K (e.g., K = 15).

Selecting neighboring samples Several possible implementations of line 5 can be made. In all of these, it seems best to sort the vertices that will be considered for connection in order of increasing distance from $\alpha(i)$. This makes sense because shorter paths are usually less costly to check for collision, and they also have a higher likelihood of being collision-free. If a connection is made, this avoids costly collision checking of longer paths to configurations that would eventually belong to the same connected component.

Several useful implementations of NEIGHBORHOOD are

1. **Nearest K:** The K closest points to $\alpha(i)$ are considered. This requires setting the parameter K (a typical value is 15). If you are unsure which implementation to use, try this one.

2. **Component K:** Try to obtain up to K nearest samples from each connected component of $\mathcal{G}$. A reasonable value is $K = 1$; otherwise, too many connections would be tried.

3. **Radius:** Take all points within a ball of radius r centered at $\alpha(i)$. An upper limit, K, may be set to prevent too many connections from being attempted. Typically, $K = 20$. A radius can be determined adaptively by shrinking the ball as the number of points increases. This reduction can be based on dispersion or discrepancy, if either of these is available for α. Note that if the samples are highly regular (e.g., a grid), then choosing the nearest K and taking points within a ball become essentially equivalent. If the point set is highly irregular, as in the case of random samples, then taking the nearest K seems preferable.

4. **Visibility:** In Section 5.6.2, a variant will be described for which it is worthwhile to try connecting α to all vertices in $\mathcal{G}$.

Note that all of these require $\mathcal{C}$ to be a metric space. One variation that has not yet been given much attention is to ensure that the directions of the NEIGHBORHOOD points relative to $\alpha(i)$ are distributed uniformly. For example, if the 20 closest points are all clumped together in the same direction, then it may be preferable to try connecting to a further point because it is in the opposite direction.

Query phase In the query phase, it is assumed that $\mathcal{G}$ is sufficiently complete to answer many queries, each of which gives an initial configuration, q_I, and a goal configuration, q_G. First, the query phase pretends as if q_I and q_G were chosen from α for connection to $\mathcal{G}$. This requires running two more iterations of the algorithm in Figure 5.25. If q_I and q_G are successfully connected to other vertices in $\mathcal{G}$, then a search is performed for a path that connects the vertex q_I to the vertex q_G. The path in the graph corresponds directly to a path in $\mathcal{C}_{free}$, which is a solution to the query. Unfortunately, if this method fails, it cannot be determined conclusively whether a solution exists. If the dispersion is known for a sample sequence, α, then it is at least possible to conclude that no solution exists for the resolution of the planner. In other words, if a solution does exist, it would require the path to travel through a corridor no wider than the radius of the largest empty ball [307].

Some analysis There have been many works that analyze the performance of sampling-based roadmaps. The basic idea from one of them [47] is briefly presented here. Consider problems such as the one in Figure 5.27, in which the CONNECT method will mostly likely fail in the thin tube, even though a connection exists. The higher dimensional versions of these problems are even more difficult. Many planning problems involve moving a robot through an area with tight clearance. This generally causes narrow channels to form in $\mathcal{C}_{free}$, which leads to a challenging planning problem for the sampling-based roadmap algorithm. Finding the escape of a bug trap is also challenging, but for the roadmap methods, even traveling through a single corridor is hard (unless more sophisticated LPMs are used [249]).

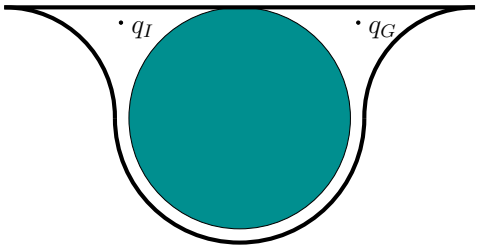
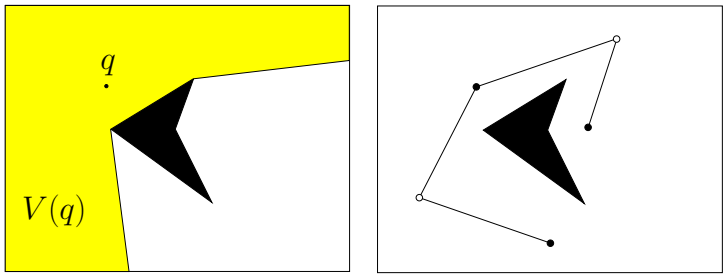


Figure 5.27: An example such as this is difficult for sampling-based roadmaps (in higher dimensional C-spaces) because some samples must fall along many points in the curved tube. Other methods, however, may be able to easily solve it.



(a) Visibility definition (b) Visibility roadmap

Figure 5.28: (a) $V(q)$ is the set of points reachable by the LPM from q. (b) A visibility roadmap has two kinds of vertices: guards, which are shown in black, and connectors, shown in white. Guards are not allowed to see other guards. Connectors must see at least two guards.

Let $V(q)$ denote the set of all configurations that can be connected to q using the CONNECT method. Intuitively, this is considered as the set of all configurations that can be "seen" using line-of-sight visibility, as shown in Figure 5.28a

The ϵ-goodness of $\mathcal{C}_{free}$ is defined as

$$\epsilon(\mathcal{C}_{free}) = \min_{q \in \mathcal{C}_{free}} \left\{ \frac{\mu(V(q))}{\mu(\mathcal{C}_{free})} \right\}, \tag{5.41}$$

in which μ represents the measure. Intuitively, $\epsilon(\mathcal{C}_{free})$ represents the small fraction of $\mathcal{C}_{free}$ that is visible from any point. In terms of ϵ and the number of vertices in $\mathcal{G}$, bounds can be established that yield the probability that a solution will be found [47]. The main difficulties are that the ϵ-goodness concept is very conservative (it uses worst-case analysis over all configurations), and ϵ-goodness is defined in terms of the structure of $\mathcal{C}_{free}$, which cannot be computed efficiently. This result and other related results help to gain a better understanding of sampling-based planning, but such bounds are difficult to apply to particular problems to determine whether an algorithm will perform well.

5.6.2 Visibility Roadmap

One of the most useful variations of sampling-based roadmaps is the *visibility roadmap* [440]. The approach works very hard to ensure that the roadmap representation is small yet covers $\mathcal{C}_{free}$ well. The running time is often greater than the basic algorithm in Figure 5.25, but the extra expense is usually worthwhile if the multiple-query philosophy is followed to its fullest extent.

The idea is to define two different kinds of vertices in $\mathcal{G}$:

Guards: To become a *guard*, a vertex, q must not be able to see other guards. Thus, the visibility region, $V(q)$, must be empty of other guards.

Connectors: To become a *connector*, a vertex, q, must see at least two guards. Thus, there exist guards q_1 and q_2, such that $q \in V(q_1) \cap V(q_2)$.

The roadmap construction phase proceeds similarly to the algorithm in Figure 5.25. The *neighborhood function* returns all vertices in $\mathcal{G}$. Therefore, for each new sample $\alpha(i)$, an attempt is made to connect it to every other vertex in $\mathcal{G}$.

The main novelty of the visibility roadmap is using a strong criterion to determine whether to keep $\alpha(i)$ and its associated edges in $\mathcal{G}$. There are three possible cases for each $\alpha(i)$:

1. The new sample, $\alpha(i)$, is not able to connect to any guards. In this case, $\alpha(i)$ earns the privilege of becoming a guard itself and is inserted into $\mathcal{G}$.

2. The new sample can connect to guards from at least two different connected components of $\mathcal{G}$. In this case, it becomes a connector that is inserted into $\mathcal{G}$ along with its associated edges, which connect it to these guards from different components.

3. Neither of the previous two conditions were satisfied. This means that the sample could only connect to guards in the same connected component. In this case, $\alpha(i)$ is discarded.

The final condition causes a dramatic reduction in the number of roadmap vertices.

One problem with this method is that it does not allow guards to be deleted in favor of better guards that might appear later. The placement of guards depends strongly on the order in which samples appear in α. The method may perform poorly if guards are not positioned well early in the sequence. It would be better to have an adaptive scheme in which guards could be reassigned in later iterations as better positions become available. Accomplishing this efficiently remains an open problem. Note the algorithm is still probabilistically complete using random sampling or resolution complete if α is dense, even though many samples are rejected.

5.6.3 Heuristics for Improving Roadmaps

The quest to design a good roadmap through sampling has spawned many heuristic approaches to sampling and making connections in roadmaps. Most of these exploit properties that are specific to the shape of the C-space and/or the particular geometry and kinematics of the robot and obstacles. The emphasis is usually on finding ways to dramatically reduce the number or required samples. Several of these methods are briefly described here.

Vertex enhancement [265] This heuristic strategy focuses effort on vertices that were difficult to connect to other vertices in the roadmap construction algorithm in Figure 5.25. A probability distribution, $P(v)$, is defined over the vertices $v \in V$. A number of iterations are then performed in which a vertex is sampled from V according to $P(v)$, and then some random motions are performed from v to try to reach new configurations. These new configurations are added as vertices, and attempts are made to connect them to other vertices, as selected by the NEIGHBORHOOD function in an ordinary iteration of the algorithm in Figure 5.25. A recommended heuristic [265] for defining $P(v)$ is to define a statistic for each v as $n_f/(n_t+1)$, in which n_t is the total number of connections attempted for v, and n_f is the number of times these attempts failed. The probability $P(v)$ is assigned as $n_f/(n_t+1)m$, in which m is the sum of the statistics over all $v \in V$ (this normalizes the statistics to obtain a valid probability distribution).

Sampling on the $\mathcal{C}_{free}$ boundary [16, 20] This scheme is based on the intuition that it is sometimes better to sample along the boundary, $\partial\mathcal{C}_{free}$, rather than waste samples on large areas of $\mathcal{C}_{free}$ that might be free of obstacles. Figure 5.29a shows one way in which this can be implemented. For each sample of $\alpha(i)$ that falls into $\mathcal{C}_{obs}$, a number of random directions are chosen in $\mathcal{C}$; these directions can be sampled using the $\mathbb{S}^n$ sampling method from Section 5.2.2. For each direction,

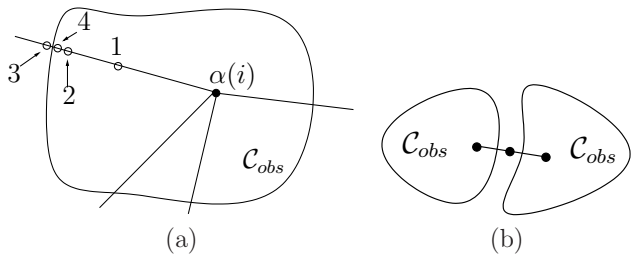


Figure 5.29: (a) To obtain samples along the boundary, binary search is used along random directions from a sample in $\mathcal{C}_{obs}$. (b) The bridge test finds narrow corridors by examining a triple of nearby samples along a line.

a binary search is performed to get a sample in $\mathcal{C}_{free}$ that is as close as possible to $\mathcal{C}_{obs}$. The order of point evaluation in the binary search is shown in Figure 5.29a. Let $\tau : [0,1]$ denote the path for which $\tau(0) \in \mathcal{C}_{obs}$ and $\tau(1) \in \mathcal{C}_{free}$. In the first step, test the midpoint, $\tau(1/2)$. If $\tau(1/2) \in \mathcal{C}_{free}$, this means that $\partial\mathcal{C}_{free}$ lies between $\tau(0)$ and $\tau(1/2)$; otherwise, it lies between $\tau(1/2)$ and $\tau(1)$. The next iteration selects the midpoint of the path segment that contains $\partial\mathcal{C}_{free}$. This will be either $\tau(1/4)$ or $\tau(3/4)$. The process continues recursively until the desired resolution is obtained.

Gaussian sampling [72] The Gaussian sampling strategy follows some of the same motivation for sampling on the boundary. In this case, the goal is to obtain points near $\partial\mathcal{C}_{free}$ by using a Gaussian distribution that biases the samples to be closer to $\partial\mathcal{C}_{free}$, but the bias is gentler, as prescribed by the variance parameter of the Gaussian. The samples are generated as follows. Generate one sample, $q_1 \in \mathcal{C}$, uniformly at random. Following this, generate another sample, $q_2 \in \mathcal{C}$, according to a Gaussian with mean q_1; the distribution must be adapted for any topological identifications and/or boundaries of $\mathcal{C}$. If one of q_1 or q_2 lies in $\mathcal{C}_{free}$ and the other lies in $\mathcal{C}_{obs}$, then the one that lies in $\mathcal{C}_{free}$ is kept as a vertex in the roadmap. For some examples, this dramatically prunes the number of required vertices.

Bridge-test sampling [241] The Gaussian sampling strategy decides to keep a point based in part on testing a pair of samples. This idea can be carried one step further to obtain a *bridge test*, which uses three samples along a line segment. If the samples are arranged as shown in Figure 5.29b, then the middle sample becomes a roadmap vertex. This is based on the intuition that narrow corridors are thin in at least one direction. The bridge test indicates that a point lies in a thin corridor, which is often an important place to locate a vertex.

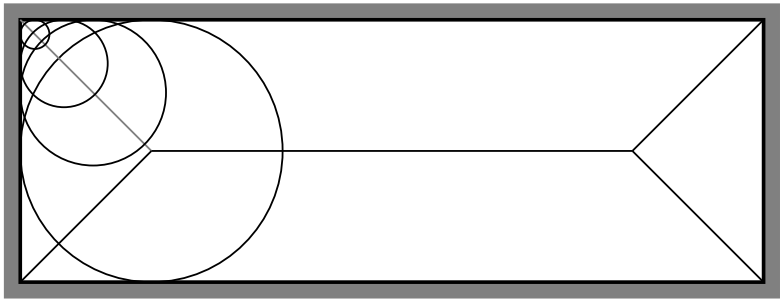


Figure 5.30: The medial axis is traced out by the centers of the largest inscribed balls. The five line segments inside of the rectangle correspond to the medial axis.

Medial-axis sampling [235, 326, 474] Rather than trying to sample close to the boundary, another strategy is to force the samples to be as far from the boundary as possible. Let (X, ρ) be a metric space. Let a *maximal ball* be a ball $B(x, r) \subseteq X$ such that no other ball can be a proper subset. The centers of all maximal balls trace out a one-dimensional set of points referred to as the *medial axis*. A simple example of a medial axis is shown for a rectangular subset of $\mathbb{R}^2$ in Figure 5.30. The medial axis in $\mathcal{C}_{free}$ is based on the largest balls that can be inscribed in $\mathrm{cl}(\mathcal{C}_{free})$. Sampling on the medial axis is generally difficult, especially because the representation of $\mathcal{C}_{free}$ is implicit. Distance information from collision checking can be used to start with a sample, $\alpha(i)$, and iteratively perturb it to increase its distance from $\partial\mathcal{C}_{free}$ [326, 474]. Sampling on the medial axis of $W \setminus \mathcal{O}$ has also been proposed [235]. In this case, the medial axis in $\mathcal{W} \setminus \mathcal{O}$ is easier to compute, and it can be used to heuristically guide the placement of good roadmap vertices in $\mathcal{C}_{free}$.

Further Reading

Unlike the last two chapters, the material of Chapter 5 is a synthesis of very recent research results. Some aspects of sampling-based motion planning are still evolving. Early approaches include [48, 77, 107, 152, 153, 171, 172, 345, 392]. The Gilbert-Johnson-Keerthi algorithm [199] is an early collision detection approach that helped inspire sampling-based motion planning; see [245] and [304] for many early references. In much of the early work, randomization appeared to be the main selling point; however, more recently it has been understood that deterministic sampling can work at least as well while obtaining resolution completeness. For a more recent survey of sampling-based motion planning, see [331].

Section 5.1 is based on material from basic mathematics books. For a summary of basic theorems and numerous examples of metric spaces, see [362]. More material appears in basic point-set topology books (e.g., [232, 256]) and analysis books (e.g., [178]). Metric issues in the context of sampling-based motion planning are discussed in [15, 312]. Measure theory is most often introduced in the context of real analysis

[178, 215, 287, 420, 421]. More material on Haar measure appears in [215].

Section 5.2 is mainly inspired by literature on Monte Carlo and quasi–Monte Carlo methods for numerical integration and optimization. An excellent source of material is [381]. Other important references for further reading include [105, 283, 353, 458, 459]. Sampling issues in the context of motion planning are considered in [192, 292, 307, 330, 482]. Comprehensive introductions to pure Monte Carlo algorithms appear in [176, 259]. The original source for the Monte Carlo method is [361]. For a survey on algorithms that compute Voronoi diagrams, see [40].

For further reading on collision detection (beyond Section 5.3), see the surveys in [252, 328, 329, 368]. Hierarchical collision detection is covered in [207, 329, 367]. The incremental collision detection ideas in Section 5.3.3 were inspired by the algorithm [327] and V-Clip [136, 367]. Distance computation is covered in [87, 161, 198, 207, 212, 367, 406]. A method suited for detecting self-collisions of linkages appears in [341]. A combinatorial approach to collision detection for motion planning appears in [431]. Numerous collision detection packages are available for use in motion planning research. One of the most widely used is PQP because it works well for any mess of 3D triangles [463].

The incremental sampling and searching framework was synthesized by unifying ideas from many planning methods. Some of these include grid-based search [49, 289, 318] and probabilistic roadmaps (PRMs) [265]. Although the PRM was developed for multiple queries, the single-query version developed in [69] helped shed light on the connection to earlier planning methods. This even led to grid-based variants of PRMs [67, 307]. Another single-query variant is presented in [425].

RDTs were developed in the literature mainly as RRTs, and were introduced in [306, 313]. RRTs have been used in several applications, and many variants have been developed [54, 75, 78, 109, 120, 133, 147, 181, 202, 255, 258, 257, 275, 324, 332, 333, 453, 454, 464, 479, 481]. Originally, they were developed for planning under differential constraints, but most of their applications to date have been for ordinary motion planning. For more information on efficient nearest-neighbor searching, see the recent survey [247], and [35, 36, 37, 38, 61, 123, 183, 248, 282, 391, 448, 483].

Section 5.6 is based mainly on the PRM framework [265]. The "probabilistic" part is not critical to the method; thus, it was referred to here as a *sampling-based roadmap*. A related precursor to the PRM was proposed in [200, 201]. The PRM has been widely used in practice, and many variants have been proposed [1, 17, 43, 44, 69, 82, 98, 133, 249, 286, 307, 321, 322, 383, 398, 400, 440, 444, 465, 474, 479, 486]. An experimental comparison of many of these variants appears in [192]. Some analysis of PRMs appears in [47, 242, 299]. In some works, the term PRM has been applied to virtually any sampling-based planning algorithm (e.g., [242]); however, in recent years the term has been used more consistently with its original meaning in [265].

Many other methods and issues fall outside of the scope of this chapter. Several interesting methods based on *approximate cell decomposition* [77, 170, 337, 345] can be considered as a form of sampling-based motion planning. A sampling-based method of developing global potential functions appears in [68]. Other sampling-based planning algorithms appear in [108, 179, 213, 214, 240]. The algorithms of this chapter are generally unable to guarantee that a solution does not exist for a motion planning problem. It is possible, however, to use sampling-based techniques to establish in finite time that

no solution exists [51]. Such a result is called a *disconnection proof*. Parallelization issues have also been investigated in the context of sampling-based motion planning [54, 94, 99, 143, 401].

Exercises

1. Prove that the Cartesian product of a metric space is a metric space by taking a linear combination as in (5.4).

2. Prove or disprove: If ρ is a metric, then ρ^2 is a metric.

3. Determine whether the following function is a metric on any topological space: X: $\rho(x, x') = 1$ is $x \neq x'$; otherwise, $\rho(x, x') = 0$.

4. State and prove whether or not (5.28) yields a metric space on $\mathcal{C} = SE(3)$, assuming that the two sets are rigid bodies.

5. The dispersion definition given in (5.19) is based on the worst case. Consider defining the *average dispersion*:

$$\bar{\delta}(P) = \frac{1}{\mu(X)} \int_X \min_{p \in P} \{\rho(x, p)\} dx. \tag{5.42}$$

   Describe a Monte Carlo (randomized) method to approximately evaluate (5.42).

6. Determine the average dispersion (as a function of i) for the van der Corput sequence (base 2) on $[0, 1]/\sim$.

7. Show that using the Lebesgue measure on $\mathbb{S}^3$ (spreading mass around uniformly on $\mathbb{S}^3$) yields the Haar measure for $SO(3)$.

8. Is the Haar measure useful in selecting an appropriate C-space metric? Explain.

9. Determine an expression for the (worst-case) dispersion of the ith sample in the base-p (Figure 5.2 shows base-2) van der Corput sequence in $[0, 1]/\sim$, in which 0 and 1 are identified.

10. Determine the dispersion of the following sequence on $[0, 1]$. The first point is $\alpha(1) = 1$. For each $i > 1$, let $c_i = \ln(2i - 3)/\ln 4$ and $\alpha(i) = c_i - \lfloor c_i \rfloor$. It turns out that this sequence achieves the best asymptotic dispersion possible, even in terms of the preceding constant. Also, the points are not uniformly distributed. Can you explain why this happens? [It may be helpful to plot the points in the sequence.]

11. Prove that (5.20) holds.

12. Prove that (5.23) holds.

13. Show that for any given set of points in $[0, 1]^n$, a range space $\mathcal{R}$ can be designed so that the discrepancy is as close as desired to 1.

14. Suppose $\mathcal{A}$ is a rigid body in $\mathbb{R}^3$ with a fixed orientation specified by a quaternion, h. Suppose that h is perturbed a small amount to obtain another quaternion, h' (no translation occurs). Construct a good upper bound on distance traveled by points on $\mathcal{A}$, expressed in terms of the change in the quaternion.

15. Design combinations of robots and obstacles in $\mathcal{W}$ that lead to C-space obstacles resembling bug traps.

16. How many k-neighbors can there be at most in an n-dimensional grid with $1 \leq k \leq n$?

17. In a high-dimensional grid, it becomes too costly to consider all $3^n - 1$ n-neighbors. It might not be enough to consider only $2n$ 1-neighbors. Determine a scheme for selecting neighbors that are spatially distributed in a good way, but without requiring too many. For example, what is a good way to select 50 neighbors for a grid in $\mathbb{R}^{10}$?

18. Explain the difference between searching an implicit, high-resolution grid and growing search trees directly on the C-space without a grid.

19. Improve the bound in (5.31) by considering the fact that rotating points trace out a circle, instead of a straight line.

20. (Open problem) Prove there are $n+1$ main branches for an RRT starting from the center of an "infinite" n-dimensional ball in $\mathbb{R}^n$. The directions of the branches align with the vertices of a regular simplex centered at the initial configuration.

Implementations

21. Implement 2D incremental collision checking for convex polygons to obtain "near constant time" performance.

22. Implement the sampling-based roadmap approach. Select an appropriate family of motion planning problems: 2D rigid bodies, 2D chains of bodies, 3D rigid bodies, etc.

    (a) Compare the roadmaps obtained using visibility-based sampling to those obtained for the ordinary sampling method.

    (b) Study the sensitivity of the method with respect to the particular NEIGHBORHOOD method.

    (c) Compare random and deterministic sampling methods.

    (d) Use the bridge test to attempt to produce better samples.

23. Implement the balanced, bidirectional RRT planning algorithm.

    (a) Study the effect of varying the amount of intermediate vertices created along edges.

(b) Try connecting to the random sample using more powerful descent functions.

(c) Explore the performance gains from using Kd-trees to select nearest neighbors.

24. Make an RRT-based planning algorithm that uses more than two trees. Carefully resolve issues such as the maximum number of allowable trees, when to start a tree, and when to attempt connections between trees.

25. Implement both the expansive-space planner and the RRT, and conduct comparative experiments on planning problems. For the full set of problems, keep the algorithm parameters fixed.

26. Implement a sampling-based algorithm that computes collision-free paths for a rigid robot that can translate or rotate on any of the flat 2D manifolds shown in Figure 4.5.

Chapter 6

Combinatorial Motion Planning

Combinatorial approaches to motion planning find paths through the continuous configuration space without resorting to approximations. Due to this property, they are alternatively referred to as *exact* algorithms. This is in contrast to the sampling-based motion planning algorithms from Chapter 5.

6.1 Introduction

All of the algorithms presented in this chapter are *complete*, which means that for any problem instance (over the space of problems for which the algorithm is designed), the algorithm will either find a solution or will correctly report that no solution exists. By contrast, in the case of sampling-based planning algorithms, weaker notions of completeness were tolerated: resolution completeness and probabilistic completeness.

Representation is important When studying combinatorial motion planning algorithms, it is important to carefully consider the definition of the input. What is the representation used for the robot and obstacles? What set of transformations may be applied to the robot? What is the dimension of the world? Are the robot and obstacles convex? Are they piecewise linear? The specification of possible inputs defines a set of problem instances on which the algorithm will operate. If the instances have certain convenient properties (e.g., low dimensionality, convex models), then a combinatorial algorithm may provide an elegant, practical solution. If the set of instances is too broad, then a requirement of both completeness and practical solutions may be unreasonable. Many general formulations of general motion planning problems are PSPACE-hard[1]; therefore, such a hope appears unattainable. Nevertheless, there exist general, complete motion planning algorithms. Note that focusing on the representation is the opposite philosophy from sampling-based planning, which hides these issues in the collision detection module.

[1]This implies NP-hard. An overview of such complexity statements appears in Section 6.5.1.

Reasons to study combinatorial methods There are generally two good reasons to study combinatorial approaches to motion planning:

1. In many applications, one may only be interested in a special class of planning problems. For example, the world might be 2D, and the robot might only be capable of translation. For many special classes, elegant and efficient algorithms can be developed. These algorithms are complete, do not depend on approximation, and can offer much better performance than sampling-based planning methods, such as those in Chapter 5.

2. It is both interesting and satisfying to know that there are complete algorithms for an extremely broad class of motion planning problems. Thus, even if the class of interest does not have some special limiting assumptions, there still exist general-purpose tools and algorithms that can solve it. These algorithms also provide theoretical upper bounds on the time needed to solve motion planning problems.

Warning: Some methods are impractical Be careful not to make the wrong assumptions when studying the algorithms of this chapter. A few of them are efficient and easy to implement, but many might be neither. Even if an algorithm has an amazing asymptotic running time, it might be close to impossible to implement. For example, one of the most famous algorithms from computational geometry can split a simple[2] polygon into triangles in $O(n)$ time for a polygon with n edges [104]. This is so amazing that it was covered in the *New York Times*, but the algorithm is so complicated that it is doubtful that anyone will ever implement it. Sometimes it is preferable to use an algorithm that has worse theoretical running time but is much easier to understand and implement. In general, though, it is valuable to understand both kinds of methods and decide on the trade-offs for yourself. It is also an interesting intellectual pursuit to try to determine how efficiently a problem can be solved, even if the result is mainly of theoretical interest. This might motivate others to look for simpler algorithms that have the same or similar asymptotic running times.

Roadmaps Virtually all combinatorial motion planning approaches construct a *roadmap* along the way to solving queries. This notion was introduced in Section 5.6, but in this chapter stricter requirements are imposed in the roadmap definition because any algorithm that constructs one needs to be complete. Some of the algorithms in this chapter first construct a cell decomposition of $\mathcal{C}_{free}$ from which the roadmap is consequently derived. Other methods directly construct a roadmap without the consideration of cells.

Let $\mathcal{G}$ be a topological graph (defined in Example 4.6) that maps into $\mathcal{C}_{free}$. Furthermore, let $S \subset \mathcal{C}_{free}$ be the swath, which is set of all points reached by $\mathcal{G}$,

[2]A polygonal region that has no holes.

as defined in (5.40). The graph $\mathcal{G}$ is called a *roadmap* if it satisfies two important conditions:

1. **Accessibility:** From any $q \in \mathcal{C}_{free}$, it is simple and efficient to compute a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q$ and $\tau(1) = s$, in which s may be any point in S. Usually, s is the closest point to q, assuming $\mathcal{C}$ is a metric space.

2. **Connectivity-preserving:** Using the first condition, it is always possible to connect some q_I and q_G to some s_1 and s_2, respectively, in S. The second condition requires that if there exists a path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$, then there also exists a path $\tau' : [0,1] \rightarrow S$, such that $\tau'(0) = s_1$ and $\tau'(1) = s_2$. Thus, solutions are not missed because $\mathcal{G}$ fails to capture the connectivity of $\mathcal{C}_{free}$. This ensures that complete algorithms are developed.

By satisfying these properties, a roadmap provides a discrete representation of the continuous motion planning problem without losing any of the original connectivity information needed to solve it. A query, (q_I, q_G), is solved by connecting each query point to the roadmap and then performing a discrete graph search on $\mathcal{G}$. To maintain completeness, the first condition ensures that any query can be connected to $\mathcal{G}$, and the second condition ensures that the search always succeeds if a solution exists.

6.2 Polygonal Obstacle Regions

Rather than diving into the most general forms of combinatorial motion planning, it is helpful to first see several methods explained for a case that is easy to visualize. Several elegant, straightforward algorithms exist for the case in which $\mathcal{C} = \mathbb{R}^2$ and $\mathcal{C}_{obs}$ is polygonal. Most of these cannot be directly extended to higher dimensions; however, some of the general principles remain the same. Therefore, it is very instructive to see how combinatorial motion planning approaches work in two dimensions. There are also applications where these algorithms may directly apply. One example is planning for a small mobile robot that may be modeled as a point moving in a building that can be modeled with a 2D polygonal floor plan.

After covering representations in Section 6.2.1, Sections 6.2.2–6.2.4 present three different algorithms to solve the same problem. The one in Section 6.2.2 first performs *cell decomposition* on the way to building the roadmap, and the ones in Sections 6.2.3 and 6.2.4 directly produce a roadmap. The algorithm in Section 6.2.3 computes maximum clearance paths, and the one in Section 6.2.4 computes shortest paths (which consequently have no clearance).

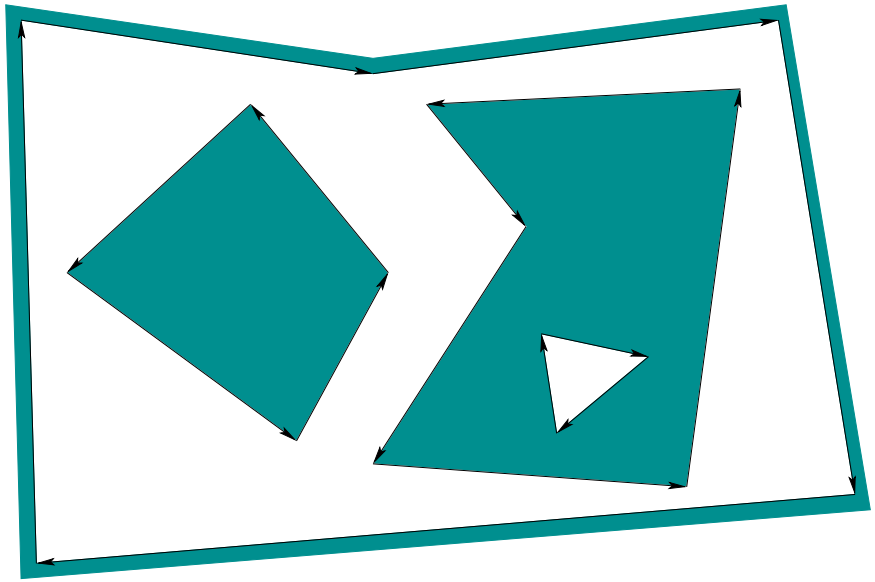


Figure 6.1: A polygonal model specified by four oriented simple polygons.

6.2.1 Representation

Assume that $\mathcal{W} = \mathbb{R}^2$; the obstacles, $\mathcal{O}$, are polygonal; and the robot, $\mathcal{A}$, is a polygonal body that is only capable of translation. Under these assumptions, $\mathcal{C}_{obs}$ will be polygonal. For the special case in which $\mathcal{A}$ is a point in $\mathcal{W}$, $\mathcal{O}$ maps directly to $\mathcal{C}_{obs}$ without any distortion. Thus, the problems considered in this section may also be considered as planning for a *point robot*. If $\mathcal{A}$ is not a point robot, then the Minkowski difference, (4.37), of $\mathcal{O}$ and $\mathcal{A}$ must be computed. For the case in which both $\mathcal{A}$ and each component of $\mathcal{O}$ are convex, the algorithm in Section 4.3.2 can be applied to compute each component of $\mathcal{C}_{obs}$. In general, both $\mathcal{A}$ and $\mathcal{O}$ may be nonconvex. They may even contain holes, which results in a $\mathcal{C}_{obs}$ model such as that shown in Figure 6.1. In this case, $\mathcal{A}$ and $\mathcal{O}$ may be decomposed into convex components, and the Minkowski difference can be computed for each pair of components. The decompositions into convex components can actually be performed by adapting the cell decomposition algorithm that will be presented in Section 6.2.2. Once the Minkowski differences have been computed, they need to be merged to obtain a representation that can be specified in terms of simple polygons, such as those in Figure 6.1. An efficient algorithm to perform this merging is given in Section 2.4 of [146]. It can also be based on many of the same principles as the planning algorithm in Section 6.2.2.

To implement the algorithms described in this section, it will be helpful to have a data structure that allows convenient access to the information contained

in a model such as Figure 6.1. How is the outer boundary represented? How are holes inside of obstacles represented? How do we know which holes are inside of which obstacles? These questions can be efficiently answered by using the doubly connected edge list data structure, which was described in Section 3.1.3 for consistent labeling of polyhedral faces. We will need to represent models, such as the one in Figure 6.1, and any other information that planning algorithms need to maintain during execution. There are three different records:

Vertices: Every vertex v contains a pointer to a point $(x, y) \in \mathcal{C} = \mathbb{R}^2$ and a pointer to some half-edge that has v as its origin.

Faces: Every face has one pointer to a half-edge on the boundary that surrounds the face; the pointer value is NIL if the face is the outermost boundary. The face also contains a list of pointers for each connected component (i.e., hole) that is contained inside of that face. Each pointer in the list points to a half-edge of the component's boundary.

Half-edges: Each half-edge is directed so that the obstacle portion is always to its left. It contains five different pointers. There is a pointer to its *origin vertex*. There is a *twin* half-edge pointer, which may point to a half-edge that runs in the opposite direction (see Section 3.1.3). If the half-edge borders an obstacle, then this pointer is NIL. Half-edges are always arranged in circular chains to form the boundary of a face. Such chains are oriented so that the obstacle portion (or a twin half-edge) is always to its left. Each half-edge stores a pointer to its internal face. It also contains pointers to the next and previous half-edges in the circular chain of half-edges.

For the example in Figure 6.1, there are four circular chains of half-edges that each bound a different face. The face record of the small triangular hole points to the obstacle face that contains the hole. Each obstacle contains a pointer to the face represented by the outermost boundary. By consistently assigning orientations to the half-edges, circular chains that bound an obstacle always run counterclockwise, and chains that bound holes run clockwise. There are no twin half-edges because all half-edges bound part of $\mathcal{C}_{obs}$. The doubly connected edge list data structure is general enough to allow extra edges to be inserted that slice through $\mathcal{C}_{free}$. These edges will not be on the border of $\mathcal{C}_{obs}$, but they can be managed using twin half-edge pointers. This will be useful for the algorithm in Section 6.2.2.

6.2.2 Vertical Cell Decomposition

Cell decompositions will be defined formally in Section 6.3, but here we use the notion informally. Combinatorial methods must construct a finite data structure that exactly encodes the planning problem. Cell decomposition algorithms achieve

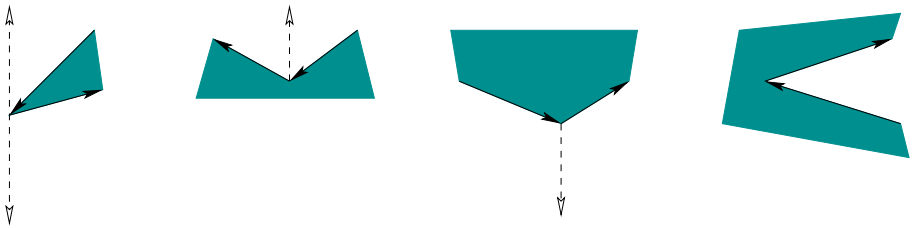


Figure 6.2: There are four general cases: 1) extending upward and downward, 2) upward only, 3) downward only, and 4) no possible extension.

this partitioning of $\mathcal{C}_{free}$ into a finite set of regions called *cells*. The term *k-cell* refers to a k-dimensional cell. The cell decomposition should satisfy three properties:

1. Computing a path from one point to another inside of a cell must be trivially easy. For example, if every cell is convex, then any pair of points in a cell can be connected by a line segment.

2. Adjacency information for the cells can be easily extracted to build the roadmap.

3. For a given q_I and q_G, it should be efficient to determine which cells contain them.

If a cell decomposition satisfies these properties, then the motion planning problem is reduced to a graph search problem. Once again the algorithms of Section 2.2 may be applied; however, in the current setting, the entire graph, $\mathcal{G}$, is usually known in advance.[3] This was not assumed for discrete planning problems.

Defining the vertical decomposition We next present an algorithm that constructs a *vertical cell decomposition* [103], which partitions $\mathcal{C}_{free}$ into a finite collection of 2-cells and 1-cells. Each 2-cell is either a trapezoid that has vertical sides or a triangle (which is a degenerate trapezoid). For this reason, the method is sometimes called *trapezoidal decomposition*. The decomposition is defined as follows. Let P denote the set of vertices used to define $\mathcal{C}_{obs}$. At every $p \in P$, try to extend rays upward and downward through $\mathcal{C}_{free}$, until $\mathcal{C}_{obs}$ is hit. There are four possible cases, as shown in Figure 6.2, depending on whether or not it is possible to extend in each of the two directions. If $\mathcal{C}_{free}$ is partitioned according to these rays, then a vertical decomposition results. Extending these rays for the example in Figure 6.3a leads to the decomposition of $\mathcal{C}_{free}$ shown in Figure 6.3b. Note that only trapezoids and triangles are obtained for the 2-cells in $\mathcal{C}_{free}$.

[3] Exceptions to this are some algorithms mentioned in Section 6.5.3, which obtain greater efficiency by only maintaining one connected component of $\mathcal{C}_{obs}$.

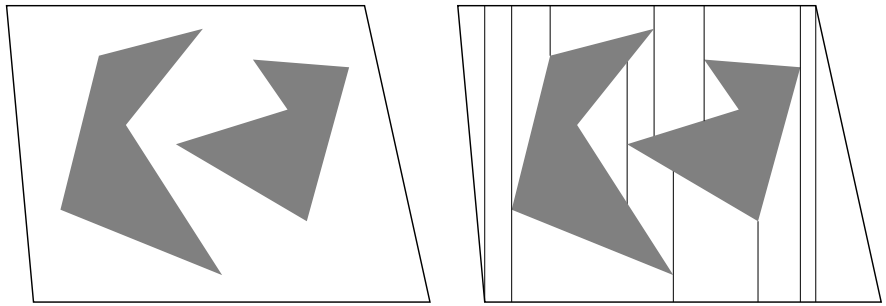


Figure 6.3: The vertical cell decomposition method uses the cells to construct a roadmap, which is searched to yield a solution to a query.

Every 1-cell is a vertical segment that serves as the border between two 2-cells. We must ensure that the topology of $\mathcal{C}_{free}$ is correctly represented. Recall that $\mathcal{C}_{free}$ was defined to be an open set. Every 2-cell is actually defined to be an open set in $\mathbb{R}^2$; thus, it is the interior of a trapezoid or triangle. The 1-cells are the interiors of segments. It is tempting to make 0-cells, which correspond to the endpoints of segments, but these are not allowed because they lie in $\mathcal{C}_{obs}$.

General position issues What if two points along $\mathcal{C}_{obs}$ lie on a vertical line that slices through $\mathcal{C}_{free}$? What happens when one of the edges of $\mathcal{C}_{obs}$ is vertical? These are special cases that have been ignored so far. Throughout much of combinatorial motion planning it is common to ignore such special cases and assume $\mathcal{C}_{obs}$ is in *general position*. This usually means that if all of the data points are perturbed by a small amount in some random direction, the probability that the special case remains is zero. Since a vertical edge is no longer vertical after being slightly perturbed, it is not in general position. The general position assumption is usually made because it greatly simplifies the presentation of an algorithm (and, in some cases, its asymptotic running time is even lower). In practice, however, this assumption can be very frustrating. Most of the implementation time is often devoted to correctly handling such special cases. Performing random perturbations may avoid this problem, but it tends to unnecessarily complicate the solutions. For the vertical decomposition, the problems are not too difficult to handle without resorting to perturbations; however, in general, it is important to be aware of this difficulty, which is not as easy to fix in most other settings.

Defining the roadmap To handle motion planning queries, a roadmap is constructed from the vertical cell decomposition. For each cell C_i, let q_i denote a designated *sample point* such that $q_i \in C_i$. The sample points can be selected as the cell centroids, but the particular choice is not too important. Let $\mathcal{G}(V, E)$ be

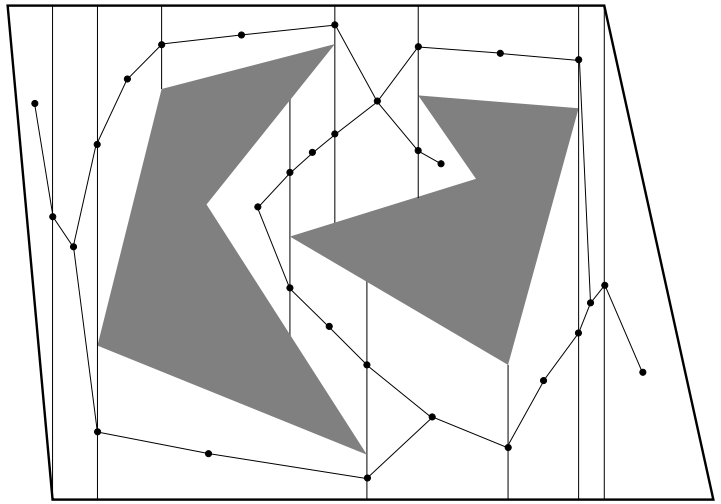


Figure 6.4: The roadmap derived from the vertical cell decomposition.

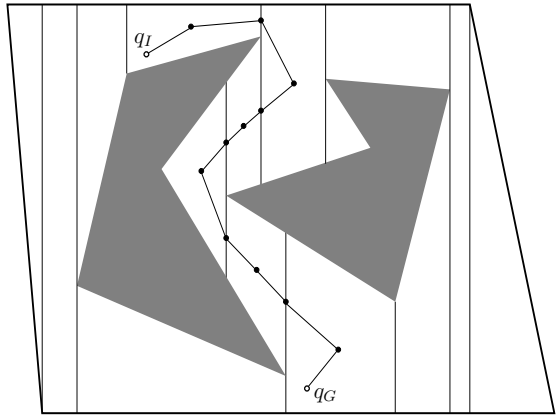


Figure 6.5: An example solution path.

a topological graph defined as follows. For every cell, C_i, define a vertex $q_i \in V$. There is a vertex for every 1-cell and every 2-cell. For each 2-cell, define an edge from its sample point to the sample point of every 1-cell that lies along its boundary. Each edge is a line-segment path between the sample points of the cells. The resulting graph is a roadmap, as depicted in Figure 6.4. The accessibility condition is satisfied because every sample point can be reached by a straight-line path thanks to the convexity of every cell. The connectivity condition is also satisfied because $\mathcal{G}$ is derived directly from the cell decomposition, which also preserves the connectivity of $\mathcal{C}_{free}$. Once the roadmap is constructed, the cell information is no longer needed for answering planning queries.

Solving a query Once the roadmap is obtained, it is straightforward to solve a motion planning query, (q_I, q_G). Let C_0 and C_k denote the cells that contain q_I and q_G, respectively. In the graph $\mathcal{G}$, search for a path that connects the sample point of C_0 to the sample point of C_k. If no such path exists, then the planning algorithm correctly declares that no solution exists. If one does exist, then let C_1, C_2, …, C_{k-1} denote the sequence of 1-cells and 2-cells visited along the computed path in $\mathcal{G}$ from C_0 to C_k.

A solution path can be formed by simply "connecting the dots." Let q_0, q_1, q_2, …, q_{k-1}, q_k, denote the sample points along the path in $\mathcal{G}$. There is one sample point for every cell that is crossed. The solution path, $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, is formed by setting $\tau(0) = q_I$, $\tau(1) = q_G$, and visiting each of the points in the sequence from q_0 to q_k by traveling along the shortest path. For the example, this leads to the solution shown in Figure 6.5. In selecting the sample points, it was important to ensure that each path segment from the sample point of one cell to the sample point of its neighboring cell is collision-free.[4]

Computing the decomposition The problem of efficiently computing the decomposition has not yet been considered. Without concern for efficiency, the problem appears simple enough that all of the required steps can be computed by brute-force computations. If $\mathcal{C}_{obs}$ has n vertices, then this approach would take at least $O(n^2)$ time because intersection tests have to be made between each vertical ray and each segment. This even ignores the data structure issues involved in finding the cells that contain the query points and in building the roadmap that holds the connectivity information. By careful organization of the computation, it turns out that all of this can be nicely handled, and the resulting running time is only $O(n \lg n)$.

Plane-sweep principle The algorithm is based on the *plane-sweep* (or *line-sweep*) principle from computational geometry [71, 146, 158], which forms the basis

[4]This is the reason why the approach is defined differently from Chapter 1 of [304]. In that case, sample points were not placed in the interiors of the 2-cells, and collision could result for some queries.

of many combinatorial motion planning algorithms and many other algorithms in general. Much of computational geometry can be considered as the development of data structures and algorithms that generalize the sorting problem to multiple dimensions. In other words, the algorithms carefully "sort" geometric information.

The word "sweep" is used to refer to these algorithms because it can be imagined that a line (or plane, etc.) sweeps across the space, only to stop where some critical change occurs in the information. This gives the intuition, but the sweeping line is not explicitly represented by the algorithm. To construct the vertical decomposition, imagine that a vertical line sweeps from $x = -\infty$ to $x = \infty$, using (x, y) to denote a point in $\mathcal{C} = \mathbb{R}^2$.

From Section 6.2.1, note that the set P of $\mathcal{C}_{obs}$ vertices are the only data in $\mathbb{R}^2$ that appear in the problem input. It therefore seems reasonable that interesting things can only occur at these points. Sort the points in P in increasing order by their X coordinate. Assuming general position, no two points have the same X coordinate. The points in P will now be visited in order of increasing x value. Each visit to a point will be referred to as an *event*. Before, after, and in between every event, a list, L, of some $\mathcal{C}_{obs}$ edges will be maintained. This list must be maintained at all times in the order that the edges appear when stabbed by the vertical sweep line. The ordering is maintained from lower to higher.

Algorithm execution Figures 6.6 and 6.7 show how the algorithm proceeds. Initially, L is empty, and a doubly connected edge list is used to represent $\mathcal{C}_{free}$. Each connected component of $\mathcal{C}_{free}$ yields a single face in the data structure. Suppose inductively that after several events occur, L is correctly maintained. For each event, one of the four cases in Figure 6.2 occurs. By maintaining L in a balanced binary search tree [132], the edges above and below p can be determined in $O(\lg n)$ time. This is much better than $O(n)$ time, which would arise from checking every segment. Depending on which of the four cases from Figure 6.2 occurs, different updates to L are made. If the first case occurs, then two different edges are inserted, and the face of which p is on the border is split two times by vertical line segments. For each of the two vertical line segments, two half-edges are added, and all faces and half-edges must be updated correctly (this operation is local in that only records adjacent to where the change occurs need to be updated). The next two cases in Figure 6.2 are simpler; only a single face split is made. For the final case, no splitting occurs.

Once the face splitting operations have been performed, L needs to be updated. When the sweep line crosses p, two edges are always affected. For example, in the first and last cases of Figure 6.2, two edges must be inserted into L (the mirror images of these cases cause two edges to be deleted from L). If the middle two cases occur, then one edge is replaced by another in L. These insertion and deletion operations can be performed in $O(\lg n)$ time. Since there are n events, the running time for the construction algorithm is $O(n \lg n)$.

The roadmap $\mathcal{G}$ can be computed from the face pointers of the doubly con-

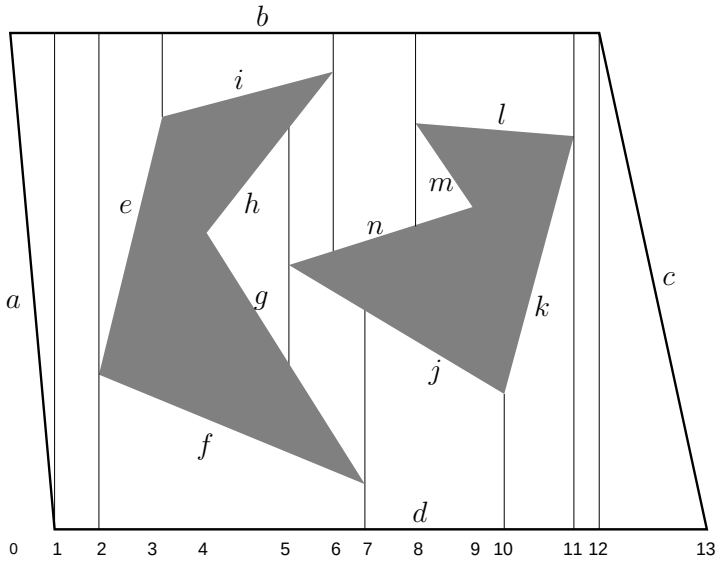


Figure 6.6: There are 14 events in this example.

Event	Sorted Edges in L	Event	Sorted Edges in L
0	$\{a, b\}$	7	$\{d, j, n, b\}$
1	$\{d, b\}$	8	$\{d, j, n, m, l, b\}$
2	$\{d, f, e, b\}$	9	$\{d, j, l, b\}$
3	$\{d, f, i, b\}$	10	$\{d, k, l, b\}$
4	$\{d, f, g, h, i, b\}$	11	$\{d, b\}$
5	$\{d, f, g, j, n, h, i, b\}$	12	$\{d, c\}$
6	$\{d, f, g, j, n, b\}$	13	$\{\}$

Figure 6.7: The status of L is shown after each of 14 events occurs. Before the first event, L is empty.

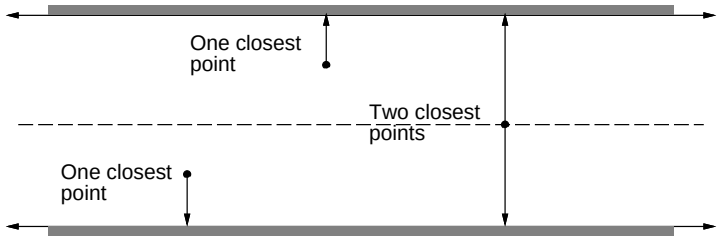


Figure 6.8: The maximum clearance roadmap keeps as far away from the $\mathcal{C}_{obs}$ as possible. This involves traveling along points that are equidistant from two or more points on the boundary of $\mathcal{C}_{obs}$.

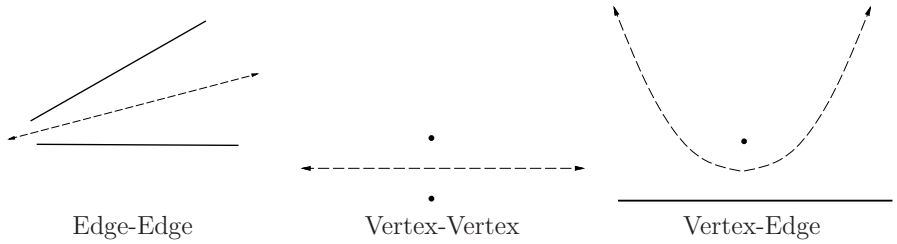


Figure 6.9: Voronoi roadmap pieces are generated in one of three possible cases. The third case leads to a quadratic curve.

nected edge list. A more elegant approach is to incrementally build $\mathcal{G}$ at each event. In fact, all of the pointer maintenance required to obtain a consistent doubly connected edge list can be ignored if desired, as long as $\mathcal{G}$ is correctly built and the sample point is obtained for each cell along the way. We can even go one step further, by forgetting about the cell decomposition and directly building a topological graph of line-segment paths between all sample points of adjacent cells.

6.2.3 Maximum-Clearance Roadmaps

A *maximum-clearance roadmap* tries to keep as far as possible from $\mathcal{C}_{obs}$, as shown for the corridor in Figure 6.8. The resulting solution paths are sometimes preferred in mobile robotics applications because it is difficult to measure and control the precise position of a mobile robot. Traveling along the maximum-clearance roadmap reduces the chances of collisions due to these uncertainties. Other names for this roadmap are *generalized Voronoi diagram* and *retraction method* [387]. It is considered as a generalization of the Voronoi diagram (recall from Section 5.2.2) from the case of points to the case of polygons. Each point along a roadmap edge is equidistant from two points on the boundary of $\mathcal{C}_{obs}$. Each roadmap vertex

corresponds to the intersection of two or more roadmap edges and is therefore equidistant from three or more points along the boundary of $\mathcal{C}_{obs}$.

The retraction term comes from topology and provides a nice intuition about the method. A subspace S is a *deformation retract* of a topological space X if the following continuous homotopy, $h : X \times [0,1] \rightarrow X$, can be defined as follows [232]:

1. $h(x,0) = x$ for all $x \in X$.
2. $h(x,1)$ is a continuous function that maps every element of X to some element of S.
3. For all $t \in [0,1]$, $h(s,t) = s$ for any $s \in S$.

The intuition is that $\mathcal{C}_{free}$ is gradually thinned through the homotopy process, until a skeleton, S, is obtained. An approximation to this shrinking process can be imagined by shaving off a thin layer around the whole boundary of $\mathcal{C}_{free}$. If this is repeated iteratively, the maximum-clearance roadmap is the only part that remains (assuming that the shaving always stops when thin "slivers" are obtained).

To construct the maximum-clearance roadmap, the concept of *features* from Section 5.3.3 is used again. Let the *feature set* refer to the set of all edges and vertices of $\mathcal{C}_{obs}$. Candidate paths for the roadmap are produced by every pair of features. This leads to a naive $O(n^4)$ time algorithm as follows. For every edge-edge feature pair, generate a line as shown in Figure 6.9a. For every vertex-vertex pair, generate a line as shown in Figure 6.9b. The maximum-clearance path between a point and a line is a parabola. Thus, for every edge-point pair, generate a parabolic curve as shown in Figure 6.9c. The portions of the paths that actually lie on the maximum-clearance roadmap are determined by intersecting the curves. Several algorithms exist that provide better asymptotic running times [316, 320], but they are considerably more difficult to implement. The best-known algorithm runs in $O(n \lg n)$ time in which n is the number of roadmap curves [435].

6.2.4 Shortest-Path Roadmaps

Instead of generating paths that maximize clearance, suppose that the goal is to find shortest paths. This leads to the *shortest-path roadmap*, which is also called the *reduced visibility graph* in [304]. The idea was first introduced in [384] and may perhaps be the first example of a motion planning algorithm. The shortest-path roadmap is in direct conflict with maximum clearance because shortest paths tend to graze the corners of $\mathcal{C}_{obs}$. In fact, the problem is ill posed because $\mathcal{C}_{free}$ is an open set. For any path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$, it is always possible to find a shorter one. For this reason, we must consider the problem of determining shortest paths in $\mathrm{cl}(\mathcal{C}_{free})$, the closure of $\mathcal{C}_{free}$. This means that the robot is allowed to "touch" or "graze" the obstacles, but it is not allowed to penetrate them. To actually use the computed paths as solutions to a motion planning problem, they need

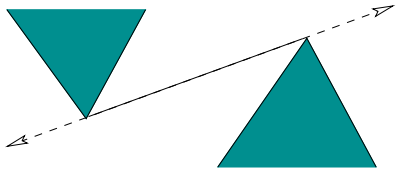


Figure 6.10: A bitangent edge must touch two reflex vertices that are mutually visible from each other, and the line must extend outward past each of them without poking into $\mathcal{C}_{obs}$.

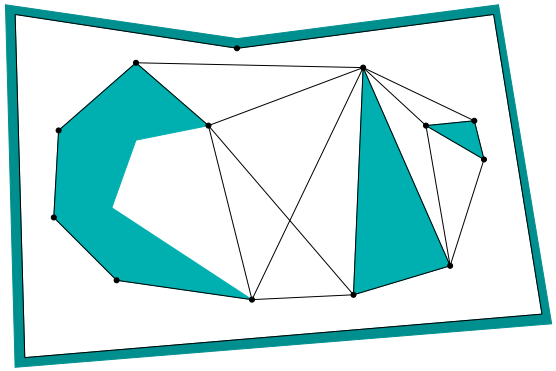


Figure 6.11: The shortest-path roadmap includes edges between consecutive reflex vertices on $\mathcal{C}_{obs}$ and also bitangent edges.

to be slightly adjusted so that they come very close to $\mathcal{C}_{obs}$ but do not make contact. This slightly increases the path length, but the additional cost can be made arbitrarily small as the path gets arbitrarily close to $\mathcal{C}_{obs}$.

The *shortest-path roadmap*, $\mathcal{G}$, is constructed as follows. Let a *reflex vertex* be a polygon vertex for which the interior angle (in $\mathcal{C}_{free}$) is greater than π. All vertices of a convex polygon (assuming that no three consecutive vertices are collinear) are reflex vertices. The vertices of $\mathcal{G}$ are the reflex vertices. Edges of $\mathcal{G}$ are formed from two different sources:

Consecutive reflex vertices: If two reflex vertices are the endpoints of an edge of $\mathcal{C}_{obs}$, then an edge between them is made in $\mathcal{G}$.

Bitangent edges: If a *bitangent line* can be drawn through a pair of reflex vertices, then a corresponding edge is made in $\mathcal{G}$. A bitangent line, depicted in Figure 6.10, is a line that is incident to two reflex vertices and does not poke into the interior of $\mathcal{C}_{obs}$ at any of these vertices. Furthermore, these vertices must be mutually visible from each other.

An example of the resulting roadmap is shown in Figure 6.11. Note that the roadmap may have isolated vertices, such as the one at the top of the figure. To

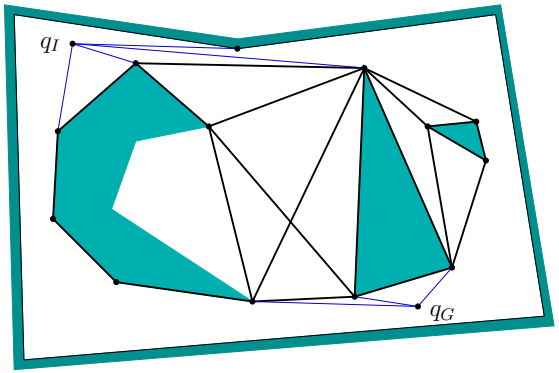


Figure 6.12: To solve a query, q_I and q_G are connected to all visible roadmap vertices, and graph search is performed.

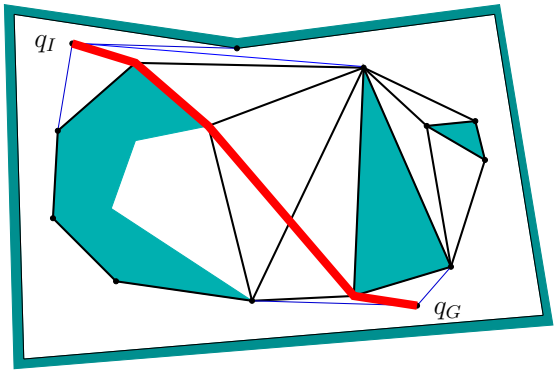


Figure 6.13: The shortest path in the extended roadmap is the shortest path between q_I and q_G.

solve a query, q_I and q_G are connected to all roadmap vertices that are visible; this is shown in Figure 6.12. This makes an extended roadmap that is searched for a solution. If Dijkstra's algorithm is used, and if each edge is given a cost that corresponds to its path length, then the resulting solution path is the shortest path between q_I and q_G. The shortest path for the example in Figure 6.12 is shown in Figure 6.13.

If the bitangent tests are performed naively, then the resulting algorithm requires $O(n^3)$ time, in which n is the number of vertices of $\mathcal{C}_{obs}$. There are $O(n^2)$ pairs of reflex vertices that need to be checked, and each check requires $O(n)$ time to make certain that no other edges prevent their mutual visibility. The plane-sweep principle from Section 6.2.2 can be adapted to obtain a better algorithm, which takes only $O(n^2 \lg n)$ time. The idea is to perform a *radial sweep* from each

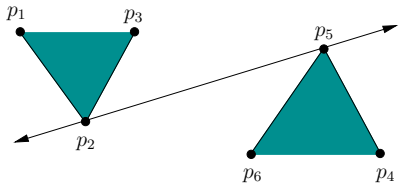


Figure 6.14: Potential bitangents can be identified by checking for left turns, which avoids the use of trigonometric functions and their associated numerical problems.

reflex vertex, v. A ray is started at $\theta = 0$, and events occur when the ray touches vertices. A set of bitangents through v can be computed in this way in $O(n \lg n)$ time. Since there are $O(n)$ reflex vertices, the total running time is $O(n^2 \lg n)$. See Chapter 15 of [146] for more details. There exists an algorithm that can compute the shortest-path roadmap in time $O(n \lg n + m)$, in which m is the total number of edges in the roadmap [195]. If the obstacle region is described by a simple polygon, the time complexity can be reduced to $O(n)$; see [372] for many shortest-path variations and references.

To improve numerical robustness, the shortest-path roadmap can be implemented without the use of trigonometric functions. For a sequence of three points, p_1, p_2, p_3, define the *left-turn predicate*, $f_l : \mathbb{R}^2 \times \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \{\text{TRUE, FALSE}\}$, as $f_l(p_1, p_2, p_3) = \text{TRUE}$ if and only if p_3 is to the left of the ray that starts at p_1 and pierces p_2. A point p_2 is a reflex vertex if and only if $f_l(p_1, p_2, p_3) = \text{TRUE}$, in which p_1 and p_3 are the points before and after, respectively, along the boundary of $\mathcal{C}_{obs}$. The bitangent test can be performed by assigning points as shown in Figure 6.14. Assume that no three points are collinear and the segment that connects p_2 and p_5 is not in collision. The pair, p_2, p_5, of vertices should receive a bitangent edge if the following sentence is FALSE:

$$\big(f_l(p_1, p_2, p_5) \oplus f_l(p_3, p_2, p_5)\big) \vee \big(f_l(p_4, p_5, p_2) \oplus f_l(p_6, p_5, p_2)\big), \tag{6.1}$$

in which $\oplus$ denotes logical "exclusive or." The f_l predicate can be implemented without trigonometric functions by defining

$$M(p_1, p_2, p_3) = \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{pmatrix}, \tag{6.2}$$

in which $p_i = (x_i, y_i)$. If $\det(M) > 0$, then $f_l(p_1, p_2, p_3) = \text{TRUE}$; otherwise, $f_l(p_1, p_2, p_3) = \text{FALSE}$.

6.3 Cell Decompositions

Section 6.2.2 introduced the vertical cell decomposition to solve the motion planning problem when $\mathcal{C}_{obs}$ is polygonal. It is important to understand, however, that

this is just one choice among many for the decomposition. Some of these choices may not be preferable in 2D; however, they might generalize better to higher dimensions. Therefore, other cell decompositions are covered in this section, to provide a smoother transition from vertical cell decomposition to cylindrical algebraic decomposition in Section 6.4, which solves the motion planning problem in any dimension for any semi-algebraic model. Along the way, a cylindrical decomposition will appear in Section 6.3.4 for the special case of a line-segment robot in $\mathcal{W} = \mathbb{R}^2$.

6.3.1 General Definitions

In this section, the term *complex* refers to a collection of cells together with their boundaries. A partition into cells can be derived from a complex, but the complex contains additional information that describes how the cells must fit together. The term *cell decomposition* still refers to the partition of the space into cells, which is derived from a *complex*.

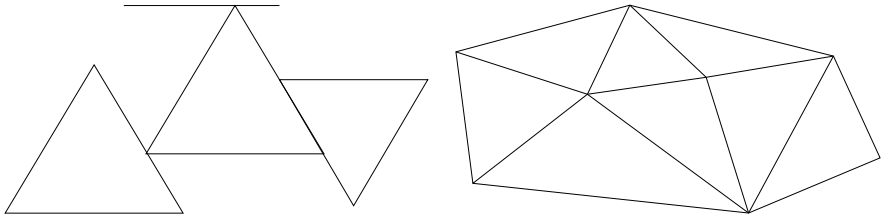
It is tempting to define complexes and cell decompositions in a very general manner. Imagine that any partition of $\mathcal{C}_{free}$ could be called a cell decomposition. A cell could be so complicated that the notion would be useless. Even $\mathcal{C}_{free}$ itself could be declared as one big cell. It is more useful to build decompositions out of simpler cells, such as ones that contain no holes. Formally, this requires that every k-dimensional cell is homeomorphic to $B^k \subset \mathbb{R}^k$, an open k-dimensional unit ball. From a motion planning perspective, this still yields cells that are quite complicated, and it will be up to the particular cell decomposition method to enforce further constraints to yield a complete planning algorithm.

Two different complexes will be introduced. The *simplicial complex* is explained because it is one of the easiest to understand. Although it is useful in many applications, it is not powerful enough to represent all of the complexes that arise in motion planning. Therefore, the *singular complex* is also introduced. Although it is more complicated to define, it encompasses all of the cell complexes that are of interest in this book. It also provides an elegant way to represent topological spaces. Another important cell complex, which is not covered here, is the *CW-complex* [226].

Simplicial Complex For this definition, it is assumed that $X = \mathbb{R}^n$. Let p_1, p_2, …, p_{k+1}, be $k+1$ linearly independent[5] points in $\mathbb{R}^n$. A k-simplex, $[p_1, \ldots, p_{k+1}]$, is formed from these points as

$$[p_1, \ldots, p_{k+1}] = \left\{ \sum_{i=1}^{k+1} \alpha_i p_i \in \mathbb{R}^n \;\middle|\; \alpha_i \geq 0 \text{ for all } i \text{ and } \sum_{i=1}^{k+1} \alpha_i = 1 \right\}, \qquad (6.3)$$

[5]Form k vectors by subtracting p_1 from the other k points for some positive integer k such that $k \leq n$. Arrange the vectors into a $k \times n$ matrix. For linear independence, there must be at least one $k \times k$ cofactor with a nonzero determinant. For example, if $k = 2$, then the three points cannot be collinear.



Not a simplicial complex A simplicial complex

Figure 6.15: To become a simplicial complex, the simplex faces must fit together nicely.

in which $\alpha_i p_i$ is the scalar multiplication of α_i by each of the point coordinates. Another way to view (6.3) is as the convex hull of the $k+1$ points (i.e., all ways to linearly interpolate between them). If $k = 2$, a triangular region is obtained. For $k = 3$, a tetrahedron is produced.

For any k-simplex and any i such that $1 \leq i \leq k+1$, let $\alpha_i = 0$. This yields a $(k-1)$-dimensional simplex that is called a *face* of the original simplex. A 2-simplex has three faces, each of which is a 1-simplex that may be called an edge. Each 1-simplex (or edge) has two faces, which are 0-simplexes called *vertices*.

To form a complex, the simplexes must fit together in a nice way. This yields a high-dimensional notion of a *triangulation*, which in $\mathbb{R}^2$ is a tiling composed of triangular regions. A *simplicial complex*, $\mathcal{K}$, is a finite set of simplexes that satisfies the following:

1. Any face of a simplex in $\mathcal{K}$ is also in $\mathcal{K}$.

2. The intersection of any two simplexes in $\mathcal{K}$ is either a common face of both of them or the intersection is empty.

Figure 6.15 illustrates these requirements. For $k > 0$, a k-cell of $\mathcal{K}$ is defined to be interior, $\text{int}([p_1, \ldots, p_{k+1}])$, of any k-simplex. For $k = 0$, every 0-simplex is a 0-cell. The union of all of the cells forms a partition of the point set covered by $\mathcal{K}$. This therefore provides a *cell decomposition* in a sense that is consistent with Section 6.2.2.

Singular complex Simplicial complexes are useful in applications such as geometric modeling and computer graphics for computing the topology of models. Due to the complicated topological spaces, implicit, nonlinear models, and decomposition algorithms that arise in motion planning, they are insufficient for the most general problems. A *singular complex* is a generalization of the *simplicial complex*. Instead of being limited to $\mathbb{R}^n$, a singular complex can be defined on any manifold, X (it can even be defined on any Hausdorff topological space). The main difference is that, for a simplicial complex, each simplex is a subset of $\mathbb{R}^n$;

however, for a singular complex, each *singular simplex* is actually a homeomorphism from a (simplicial) simplex in $\mathbb{R}^n$ to a subset of X.

To help understand the idea, first consider a 1D singular complex, which happens to be a topological graph (as introduced in Example 4.6). The interval $[0,1]$ is a 1-simplex, and a continuous path $\tau : [0,1] \rightarrow X$ is a *singular* 1*-simplex* because it is a homeomorphism of $[0,1]$ to the image of τ in X. Suppose $\mathcal{G}(V,E)$ is a topological graph. The cells are subsets of X that are defined as follows. Each point $v \in V$ is a 0-cell in X. To follow the formalism, each is considered as the image of a function $f : \{0\} \rightarrow X$, which makes it a *singular* 0*-simplex*, because $\{0\}$ is a 0-simplex. For each path $\tau \in E$, the corresponding 1-cell is

$$\{x \in X \mid \tau(s) = x \text{ for some } s \in (0,1)\}. \tag{6.4}$$

Expressed differently, it is $\tau((0,1))$, the image of the path τ, except that the endpoints are removed because they are already covered by the 0-cells (the cells must form a partition).

These principles will now be generalized to higher dimensions. Since all balls and simplexes of the same dimension are homeomorphic, balls can be used instead of a simplex in the definition of a singular simplex. Let $B^k \subset \mathbb{R}^k$ denote a closed, k-dimensional unit ball,

$$D^k = \{x \in \mathbb{R}^n \mid \|x\| \leq 1\}, \tag{6.5}$$

in which $\|\cdot\|$ is the Euclidean norm. A *singular* k*-simplex* is a continuous mapping $\sigma : D^k \rightarrow X$. Let $\text{int}(D^k)$ refer to the interior of D^k. For $k \geq 1$, the k*-cell*, C, corresponding to a singular k-simplex, σ, is the image $C = \sigma(\text{int}(D^k)) \subseteq X$. The 0-cells are obtained directly as the images of the 0 singular simplexes. Each singular 0-simplex maps to the 0-cell in X. If σ is restricted to $\text{int}(D^k)$, then it actually defines a homeomorphism between D^k and C. Note that both of these are open sets if $k > 0$.

A simplicial complex requires that the simplexes fit together nicely. The same concept is applied here, but topological concepts are used instead because they are more general. Let $\mathcal{K}$ be a set of singular simplexes of varying dimensions. Let S_k denote the union of the images of all singular i-simplexes for all $i \leq k$.

A collection of singular simplexes that map into a topological space X is called a *singular complex* if:

1. For each dimension k, the set $S_k \subseteq X$ must be closed. This means that the cells must all fit together nicely.

2. Each k-cell is an open set in the topological subspace S_k. Note that 0-cells are open in S_0, even though they are usually closed in X.

Example 6.1 (Vertical Decomposition) The vertical decomposition of Section 6.2.2 is a nice example of a singular complex that is not a simplicial complex because it contains trapezoids. The interior of each trapezoid and triangle forms a 2-cell, which is an open set. For every pair of adjacent 2-cells, there is a 1-cell on their common boundary. There are no 0-cells because the vertices lie in $\mathcal{C}_{obs}$, not in $\mathcal{C}_{free}$. The subspace S_2 is formed by taking the union of all 2-cells and 1-cells to yield $S_2 = \mathcal{C}_{free}$. This satisfies the closure requirement because the complex is built in $\mathcal{C}_{free}$ only; hence, the topological space is $\mathcal{C}_{free}$. The set $S_2 = \mathcal{C}_{free}$ is both open and closed. The set S_1 is the union of all 1-cells. This is also closed because the 1-cell endpoints all lie in $\mathcal{C}_{obs}$. Each 1-cell is also an open set.

One way to avoid some of these strange conclusions from the topology restricted to $\mathcal{C}_{free}$ is to build the vertical decomposition in $\text{cl}(\mathcal{C}_{free})$, the closure of $\mathcal{C}_{free}$. This can be obtained by starting with the previously defined vertical decomposition and adding a new 1-cell for every edge of $\mathcal{C}_{obs}$ and a 0-cell for every vertex of $\mathcal{C}_{obs}$. Now $S_3 = \text{cl}(\mathcal{C}_{free})$, which is closed in $\mathbb{R}^2$. Likewise, S_2, S_1, and S_0, are closed in the usual way. Each of the individual k-dimensional cells, however, is open in the topological space S_k. The only strange case is that the 0-cells are considered open, but this is true in the discrete topological space S_0. ■

6.3.2 2D Decompositions

The vertical decomposition method of Section 6.2.2 is just one choice of many cell decomposition methods for solving the problem when $\mathcal{C}_{obs}$ is polygonal. It provides a nice balance between the number of cells, computational efficiency, and implementation ease. It is usually possible to decompose $\mathcal{C}_{obs}$ into far fewer convex cells. This would be preferable for multiple-query applications because the roadmap would be smaller. It is unfortunately quite difficult to optimize the number of cells. Determining the decomposition of a polygonal $\mathcal{C}_{obs}$ with holes that uses the smallest number of convex cells is NP-hard [268, 336]. Therefore, we are willing to tolerate nonoptimal decompositions.

Triangulation One alternative to the vertical decomposition is to perform a *triangulation*, which yields a simplicial complex over $\mathcal{C}_{free}$. Figure 6.16 shows an example. Since $\mathcal{C}_{free}$ is an open set, there are no 0-cells. Each 2-simplex (triangle) has either one, two, or three faces, depending on how much of its boundary is shared with $\mathcal{C}_{obs}$. A roadmap can be made by connecting the samples for 1-cells and 2-cells as shown in Figure 6.17. Note that there are many ways to triangulate $\mathcal{C}_{free}$ for a given problem. Finding good triangulations, which for example means trying to avoid thin triangles, is given considerable attention in computational geometry [71, 146, 158].

How can the triangulation be computed? It might seem tempting to run the vertical decomposition algorithm of Section 6.2.2 and split each trapezoid into two triangles. Even though this leads to triangular cells, it does not produce a simplicial complex (two triangles could abut the same side of a triangle edge).

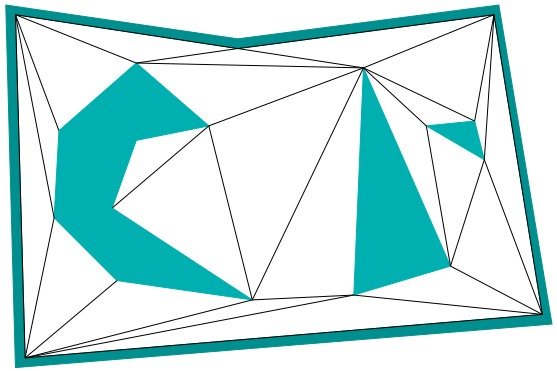


Figure 6.16: A triangulation of $\mathcal{C}_{free}$.

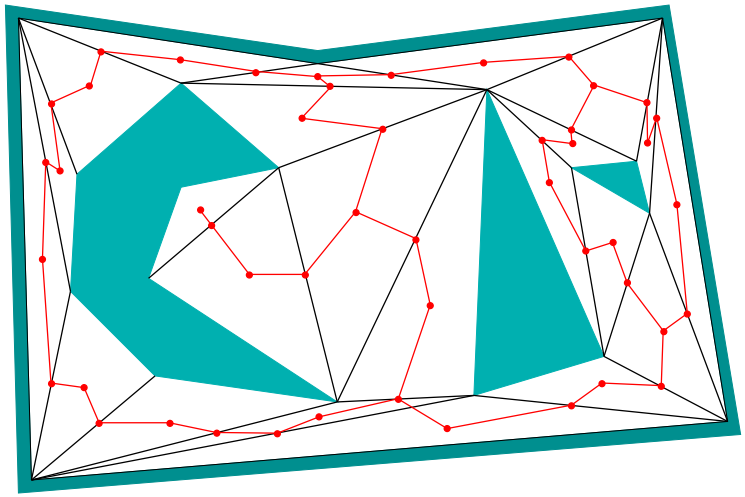


Figure 6.17: A roadmap obtained from the triangulation.

A naive approach is to incrementally split faces by attempting to connect two vertices of a face by a line segment. If this segment does not intersect other segments, then the split can be made. This process can be iteratively performed over all vertices of faces that have more than three vertices, until a triangulation is eventually obtained. Unfortunately, this results in an $O(n^3)$ time algorithm because $O(n^2)$ pairs must be checked in the worst case, and each check requires $O(n)$ time to determine whether an intersection occurs with other segments. This can be easily reduced to $O(n^2 \lg n)$ by performing radial sweeping. Chapter 3 of [146] presents an algorithm that runs in $O(n \lg n)$ time by first partitioning $\mathcal{C}_{free}$ into *monotone polygons*, and then efficiently triangulating each monotone polygon. If $\mathcal{C}_{free}$ is simply connected, then, surprisingly, a triangulation can be computed in linear time [104]. Unfortunately, this algorithm is too complicated to use in practice (there are, however, simpler algorithms for which the complexity is close to $O(n)$; see [57] and the end of Chapter 3 of [146] for surveys).

Cylindrical decomposition The *cylindrical decomposition* is very similar to the vertical decomposition, except that when any of the cases in Figure 6.2 occurs, then a vertical line slices through all faces, all the way from $y = -\infty$ to $y = \infty$. The result is shown in Figure 6.18, which may be considered as a singular complex. This may appear very inefficient in comparison to the vertical decomposition; however, it is presented here because it generalizes nicely to any dimension, any C-space topology, and any semi-algebraic model. Therefore, it is presented here to ease the transition to more general decompositions. The most important property of the cylindrical decomposition is shown in Figure 6.19. Consider each vertical strip between two events. When traversing a strip from $y = -\infty$ to $y = \infty$, the points alternate between being $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$. For example, between events 4 and 5, the points below edge f are in $\mathcal{C}_{free}$. Points between f and g lie in $\mathcal{C}_{obs}$. Points between g and h lie in $\mathcal{C}_{free}$, and so forth. The cell decomposition can be defined so that 2D cells are also created in $\mathcal{C}_{obs}$. Let $S(x, y)$ denote the logical predicate (3.6) from Section 3.1.1. When traversing a strip, the value of $S(x, y)$ also alternates. This behavior is the main reason to construct a cylindrical decomposition, which will become clearer in Section 6.4.2. Each vertical strip is actually considered to be a *cylinder*, hence, the name cylindrical decomposition (i.e., there are not necessarily any cylinders in the 3D geometric sense).

6.3.3 3D Vertical Decomposition

It turns out that the vertical decomposition method of Section 6.2.2 can be extended to any dimension by recursively applying the sweeping idea. The method requires, however, that $\mathcal{C}_{obs}$ is piecewise linear. In other words, $\mathcal{C}_{obs}$ is represented as a semi-algebraic model for which all primitives are linear. Unfortunately, most of the general motion planning problems involve nonlinear algebraic primitives because of the nonlinear transformations that arise from rotations. Recall the

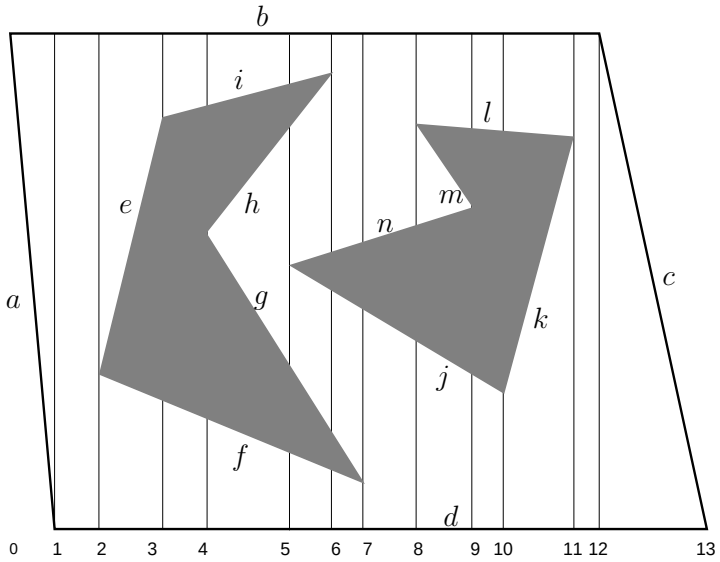


Figure 6.18: The cylindrical decomposition differs from the vertical decomposition in that the rays continue forever instead of stopping at the nearest edge. Compare this figure to Figure 6.6.

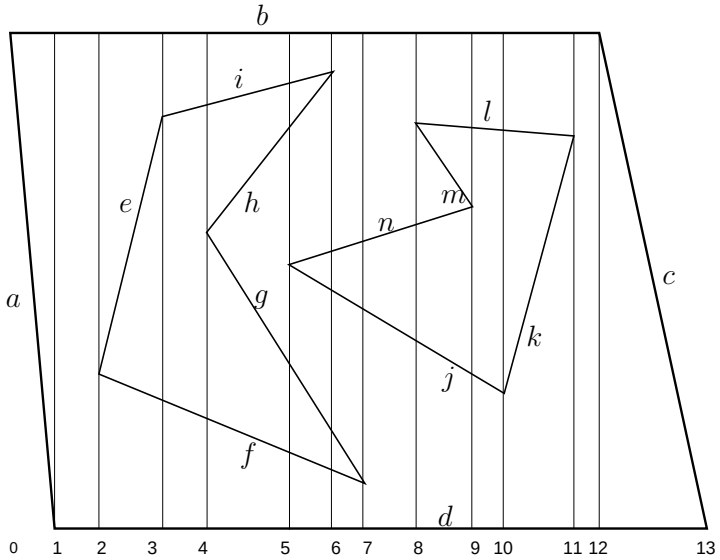


Figure 6.19: The cylindrical decomposition produces vertical strips. Inside of a strip, there is a stack of collision-free cells, separated by $\mathcal{C}_{obs}$.

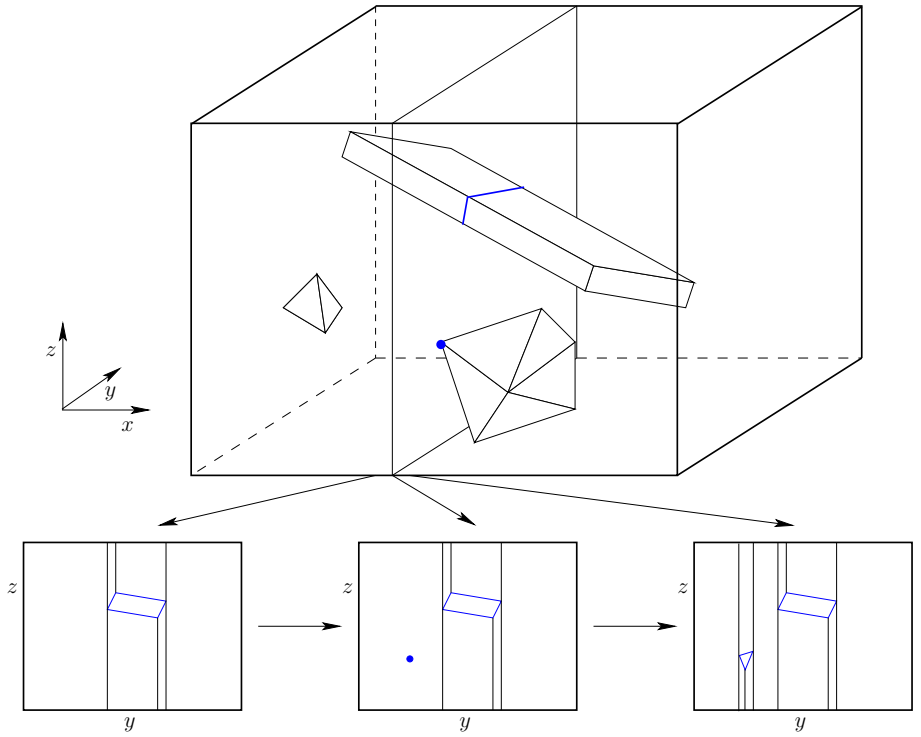


Figure 6.20: In higher dimensions, the sweeping idea can be applied recursively.

complicated algebraic $\mathcal{C}_{obs}$ model constructed in Section 4.3.3. To handle generic algebraic models, powerful techniques from computational algebraic geometry are needed. This will be covered in Section 6.4.

One problem for which $\mathcal{C}_{obs}$ is piecewise linear is a polyhedral robot that can translate in $\mathbb{R}^3$, and the obstacles in $\mathcal{W}$ are polyhedra. Since the transformation equations are linear in this case, $\mathcal{C}_{obs} \subset \mathbb{R}^3$ is polyhedral. The polygonal faces of $\mathcal{C}_{obs}$ are obtained by forming geometric primitives for each of the Type FV, Type VF, and Type EE cases of contact between $\mathcal{A}$ and $\mathcal{O}$, as mentioned in Section 4.3.2.

Figure 6.20 illustrates the algorithm that constructs the 3D vertical decomposition. Compare this to the algorithm in Section 6.2.2. Let (x, y, z) denote a point in $\mathcal{C} = \mathbb{R}^3$. The vertical decomposition yields convex 3-cells, 2-cells, and 1-cells. Neglecting degeneracies, a generic 3-cell is bounded by six planes. The cross section of a 3-cell for some fixed x value yields a trapezoid or triangle, exactly as in the 2D case, but in a plane parallel to the yz plane. Two sides of a generic 3-cell are parallel to the yz plane, and two other sides are parallel to the xz plane. The 3-cell is bounded above and below by two polygonal faces of $\mathcal{C}_{obs}$.

Initially, sort the $\mathcal{C}_{obs}$ vertices by their x coordinate to obtain the events. Now consider sweeping a plane perpendicular to the x-axis. The plane for a fixed value of x produces a 2D polygonal slice of $\mathcal{C}_{obs}$. Three such slices are shown at the bottom of Figure 6.20. Each slice is parallel to the yz plane and appears to look exactly like a problem that can be solved by the 2D vertical decomposition method. The 2-cells in a slice are actually slices of 3-cells in the 3D decomposition. The only places in which these 3-cells can critically change is when the sweeping plane stops at some x value. The center slice in Figure 6.20 corresponds to the case in which a vertex of a convex polyhedron is encountered, and all of the polyhedron lies to right of the sweep plane (i.e., the rest of the polyhedron has not been encountered yet). This corresponds to a place where a critical change must occur in the slices. These are 3D versions of the cases in Figure 6.2, which indicate how the vertical decomposition needs to be updated. The algorithm proceeds by first building the 2D vertical decomposition at the first x event. At each event, the 2D vertical decomposition must be updated to take into account the critical changes. During this process, the 3D cell decomposition and roadmap can be incrementally constructed, as in the 2D case.

The roadmap is constructed by placing a sample point in the center of each 3-cell and 2-cell. The vertices are the sample points, and edges are added to the roadmap by connecting the sample points for each case in which a 3-cell is adjacent to a 2-cell.

This same principle can be extended to any dimension, but the applications to motion planning are limited because the method requires linear models (or at least it is very challenging to adapt to nonlinear models; in some special cases, this can be done). See [216] for a summary of the complexity of vertical decompositions for various geometric primitives and dimensions.

6.3.4 A Decomposition for a Line-Segment Robot

This section presents one of the simplest cell decompositions that involves nonlinear models, yet it is already fairly complicated. This will help to give an appreciation of the difficulty of combinatorial planning in general. Consider the planning problem shown in Figure 6.21. The robot, $\mathcal{A}$, is a single line segment that can translate or rotate in $\mathcal{W} = \mathbb{R}^2$. The dot on one end of $\mathcal{A}$ is used to illustrate its origin and is not part of the model. The C-space, $\mathcal{C}$, is homeomorphic to $\mathbb{R}^2 \times \mathbb{S}^1$. Assume that the parameterization $\mathbb{R}^2 \times [0, 2\pi]/\sim$ is used in which the identification equates $\theta = 0$ and $\theta = 2\pi$. A point in $\mathcal{C}$ is represented as (x, y, θ).

An approximate solution First consider making a cell decomposition for the case in which the segment can only translate. The method from Section 4.3.2 can be used to compute $\mathcal{C}_{obs}$ by treating the robot-obstacle interaction with Type EV and Type VE contacts. When the interior of $\mathcal{A}$ touches an obstacle vertex, then Type EV is obtained. An endpoint of $\mathcal{A}$ touching an object interior yields Type

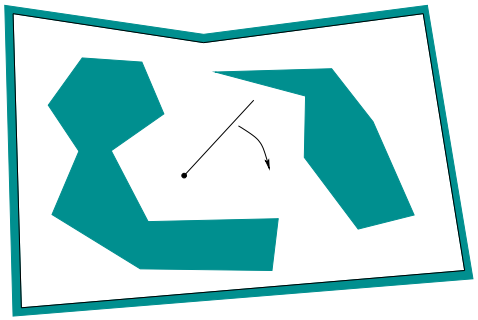


Figure 6.21: Motion planning for a line segment that can translate and rotate in a 2D world.

VE. Each case produces an edge of $\mathcal{C}_{obs}$, which is polygonal. Once this is represented, the vertical decomposition can be used to solve the problem. This inspires a reasonable numerical approach to the rotational case, which is to discretize θ into K values, $i\Delta\theta$, for $0 \leq i \leq K$, and $\Delta\theta = 2\pi/K$ [14]. The obstacle region, $\mathcal{C}_{obs}$, is polygonal for each case, and we can imagine having a stack of K polygonal regions. A roadmap can be formed by connecting sampling points inside of a slice in the usual way, and also by connecting samples between corresponding cells in neighboring slices. If K is large enough, this strategy works well, but the method is not complete because a sufficient value for K cannot be determined in advance. The method is actually an interesting hybrid between combinatorial and sampling-based motion planning. A resolution-complete version can be imagined.

In the limiting case, as K tends to infinity, the surfaces of $\mathcal{C}_{obs}$ become curved along the θ direction. The conditions in Section 4.3.3 must be applied to generate the actual obstacle regions. This is possible, but it yields a semi-algebraic representation of $\mathcal{C}_{obs}$ in terms of implicit polynomial primitives. It is no easy task to determine an explicit representation in terms of simple cells that can be used for motion planning. The method of Section 6.3.3 cannot be used because $\mathcal{C}_{obs}$ is not polyhedral. Therefore, special analysis is warranted to produce a cell decomposition.

The general idea is to construct a cell decomposition in $\mathbb{R}^2$ by considering only the translation part, (x, y). Each cell in $\mathbb{R}^2$ is then lifted into $\mathcal{C}$ by considering θ as a third axis that is "above" the xy plane. A cylindrical decomposition results in which each cell in the xy plane produces a cylindrical stack of cells for different θ values. Recall the cylinders in Figures 6.18 and 6.19. The vertical axis corresponds to θ in the current setting, and the horizontal axis is replaced by two axes, x and y.

To construct the decomposition in $\mathbb{R}^2$, consider the various robot-obstacle contacts shown in Figure 6.22. In Figure 6.22a, the segment swings around from a fixed (x, y). Two different kinds of contacts arise. For some orientation (value of

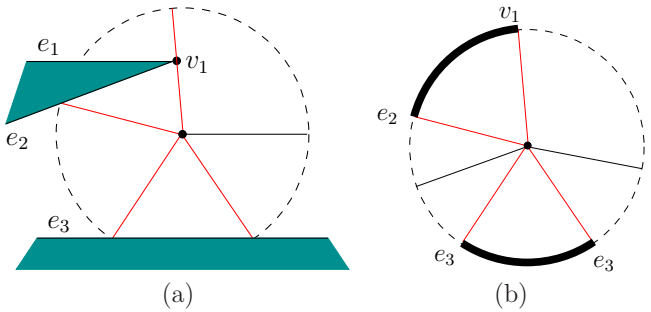


Figure 6.22: Fix (x, y) and swing the segment around for all values of $\theta \in [0, 2\pi]/\sim$. (a) Note the vertex and edge features that are hit by the segment. (b) Record orientation intervals over which the robot is not in collision.

θ), the segment contacts v_1, forming a Type EV contact. For three other orientations, the segment contacts an edge, forming Type VE contacts. Once again using the *feature* concept, there are four orientations at which the segment contacts a feature. Each feature may be either a vertex or an edge. Between the two contacts with e_2 and e_3, the robot is not in collision. These configurations lie in $\mathcal{C}_{free}$. Also, configurations for which the robot is between contacts e_3 (the rightmost contact) and v_1 are also in $\mathcal{C}_{free}$. All other orientations produce configurations in $\mathcal{C}_{obs}$. Note that the line segment cannot get from being between e_2 and e_3 to being between e_3 and v_1, unless the (x, y) position is changed. It therefore seems sensible that these must correspond to different cells in whatever decomposition is made.

Radar maps Figure 6.22b illustrates which values of θ produce collision. We will refer to this representation as a *radar map*. The four contact orientations are indicated by the contact feature. The notation $[e_3, v_1]$ and $[e_2, e_3]$ identifies the two intervals for which $(x, y, \theta) \in \mathcal{C}_{free}$. Now imagine changing (x, y) by a small amount, to obtain (x', y'). How would the radar map change? The precise angles at which the contacts occur would change, but the notation $[e_3, v_1]$ and $[e_2, e_3]$, for configurations that lie in $\mathcal{C}_{free}$, remains unchanged. Even though the angles change, there is no interesting change in terms of the contacts; therefore, it makes sense to declare (x, y, θ) and (x, y, θ') to lie in the same cell in $\mathcal{C}_{free}$ because θ and θ' both place the segment between the same contacts. Imagine a column of two 3-cells above a small area around (x, y). One 3-cell is for orientations in $[e_3, v_1]$, and the other is for orientations in $[e_2, e_3]$. These appear to be 3D regions in $\mathcal{C}_{free}$ because each of x, y, and θ can be perturbed a small amount without leaving the cell.

Of course, if (x, y) is changed enough, then eventually we expect a dramatic change to occur in the radar map. For example, imagine e_3 is infinitely long, and the x value is gradually increased in Figure 6.22a. The black band between v_1 and

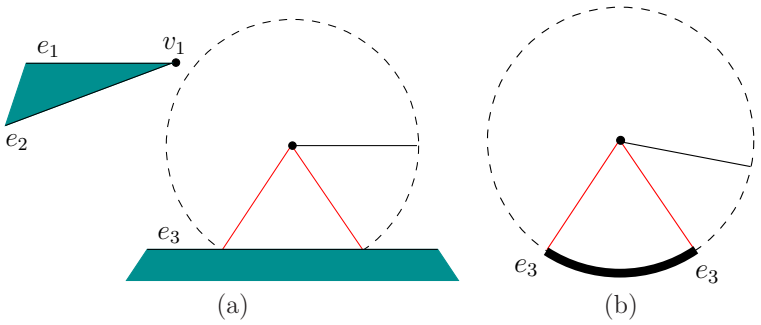


Figure 6.23: If x is increased enough, a critical change occurs in the radar map because v_1 can no longer be reached by the robot.

e_2 in Figure 6.22b shrinks in length. Eventually, when the distance from (x', y') to v_1 is greater than the length of $\mathcal{A}$, the black band disappears. This situation is shown in Figure 6.23. The change is very important to notice because after that region vanishes, any orientation θ' between e_3 and e_3, traveling the long way around the circle, produces a configuration $(x', y', \theta') \in \mathcal{C}_{free}$. This seems very important because it tells us that we can travel between the original two cells by moving the robot further way from v_1, rotating the robot, and then moving back. Now move from the position shown in Figure 6.23 into the positive y direction. The remaining black band begins to shrink and finally disappears when the distance to e_3 is further than the robot length. This represents another critical change.

The radar map can be characterized by specifying a circular ordering

$$([f_1, f_2], [f_3, f_4], [f_5, f_6], \ldots, [f_{2k-1}, f_{2k}]), \tag{6.6}$$

when there are k orientation intervals over which the configurations lie in $\mathcal{C}_{free}$. For the radar map in Figure 6.22b, this representation yields $([e_3, v_1], [e_2, e_3])$. Each f_i is a feature, which may be an edge or a vertex. Some of the f_i may be identical; the representation for Figure 6.23b is $([e_3, e_3])$. The intervals are specified in counterclockwise order around the radar map. Since the ordering is circular, it does not matter which interval is specified first. There are two degenerate cases. If $(x, y, \theta) \in \mathcal{C}_{free}$ for all $\theta \in [0, 2\pi)$, then we write () for the ordering. On the other hand, if $(x, y, \theta) \in \mathcal{C}_{obs}$ for all $\theta \in [0, 2\pi)$, then we write $\emptyset$.

Critical changes in cells Now we are prepared to explain the cell decomposition in more detail. Imagine traveling along a path in $\mathbb{R}^2$ and producing an animated version of the radar map in Figure 6.22b. We say that a *critical change* occurs each time the circular ordering representation of (6.6) changes. Changes occur when intervals: 1) appear, 2) disappear, 3) split apart, 4) merge into one, or 5) when the feature of an interval changes. The first task is to partition $\mathbb{R}^2$ into

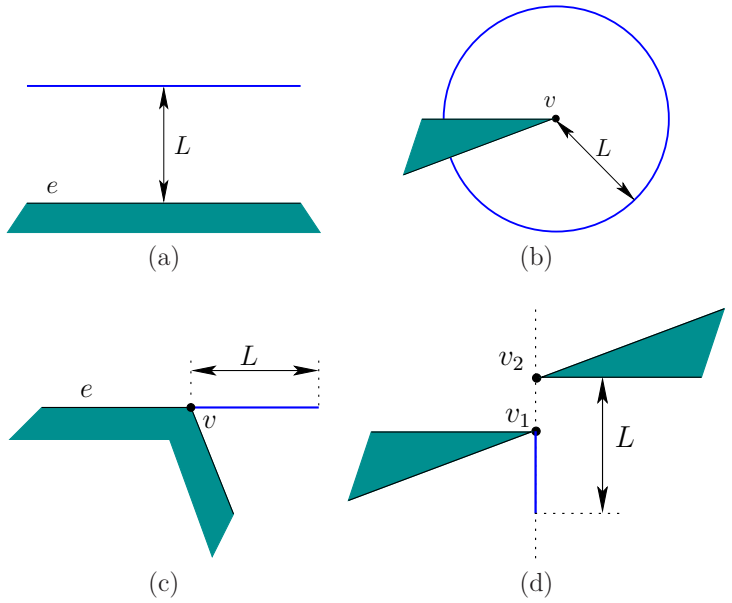


Figure 6.24: Four of the five cases that produce critical curves in $\mathbb{R}^2$.

maximal 2-cells over which no critical changes occur. Each one of these 2-cells, R, represents the projection of a strip of 3-cells in $\mathcal{C}_{free}$. Each 3-cell is defined as follows. Let $\{R, [f_i, f_{i+1}]\}$ denote the 3D region in $\mathcal{C}_{free}$ for which $(x, y) \in R$ and θ places the segment between contacts f_i and f_{i+1}. The *cylinder* of cells above R is given by $\{R, [f_i, f_{i+1}]\}$ for each interval in the circular ordering representation, (6.6). If any orientation is possible because $\mathcal{A}$ never contacts an obstacle while in R, then we write $\{R\}$.

What are the positions in $\mathbb{R}^2$ that cause critical changes to occur? It turns out that there are five different cases to consider, each of which produces a set of *critical curves* in $\mathbb{R}^2$. When one of these curves is crossed, a critical change occurs. If none of these curves is crossed, then no critical change can occur. Therefore, these curves precisely define the boundaries of the desired 2-cells in $\mathbb{R}^2$. Let L denote the length of the robot (which is the line segment).

Consider how the five cases mentioned above may occur. Two of the five cases have already been observed in Figures 6.22 and 6.23. These appear in Figures 6.24a and Figures 6.24b, and occur if (x, y) is within L of an edge or a vertex. The third and fourth cases are shown in Figures 6.24c and 6.24d, respectively. The third case occurs because crossing the curve causes $\mathcal{A}$ to change between being able to touch e and being able to touch v. This must be extended from any edge at an endpoint that is a reflex vertex (interior angle is greater than π). The fourth case is actually a return of the bitangent case from Figure 6.10, which arose for

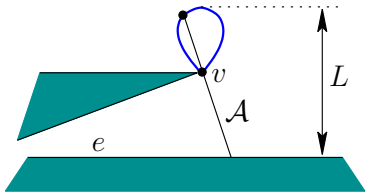


Figure 6.25: The fifth case is the most complicated. It results in a fourth-degree algebraic curve called the Conchoid of Nicomedes.

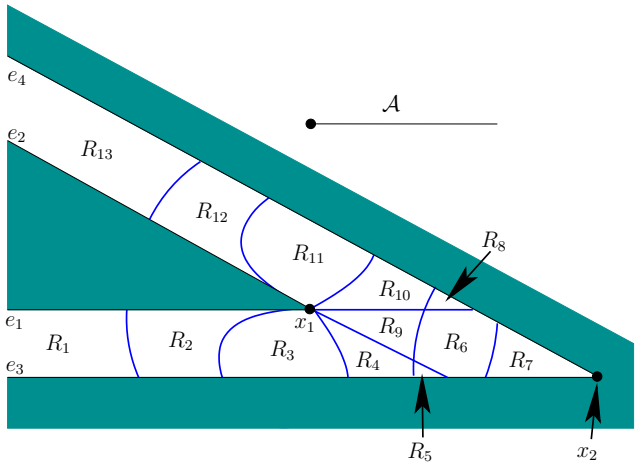


Figure 6.26: The critical curves form the boundaries of the noncritical regions in $\mathbb{R}^2$.

the shortest path graph. If the vertices are within L of each other, then a linear critical curve is generated because $\mathcal{A}$ is no longer able to touch v_2 when crossing it from right to left. Bitangents always produce curves in pairs; the curve above v_2 is not shown. The final case, shown in Figure 6.25, is the most complicated. It is a fourth-degree algebraic curve called the Conchoid of Nicomedes, which arises from $\mathcal{A}$ being in simultaneous contact between v and e. Inside of the teardrop-shaped curve, $\mathcal{A}$ can contact e but not v. Just outside of the curve, it can touch v. If the xy coordinate frame is placed so that v is at $(0, 0)$, then the equation of the curve is

$$(x^2 - y^2)(y + d)^2 - y^2 L^2 = 0, \tag{6.7}$$

in which d is the distance from v to e.

Putting all of the curves together generates a cell decomposition of $\mathbb{R}^2$. There are *noncritical regions*, over which there is no change in (6.6); these form the 2-cells. The boundaries between adjacent 2-cells are sections of the critical curves and form 1-cells. There are also 0-cells at places where critical curves intersect.

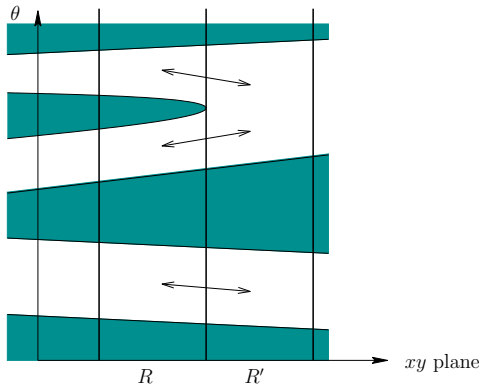


Figure 6.27: Connections are made between neighboring 3-cells that lie above neighboring noncritical regions.

Figure 6.26 shows an example adapted from [304]. Note that critical curves are not drawn if their corresponding configurations are all in $\mathcal{C}_{obs}$. The method still works correctly if they are included, but unnecessary cell boundaries are made. Just for fun, they could be used to form a nice cell decomposition of $\mathcal{C}_{obs}$, in addition to $\mathcal{C}_{free}$. Since $\mathcal{C}_{obs}$ is avoided, is seems best to avoid wasting time on decomposing it. These unnecessary cases can be detected by imagining that $\mathcal{A}$ is a laser with range L. As the laser sweeps around, only features that are contacted by the laser are relevant. Any features that are hidden from view of the laser correspond to unnecessary boundaries.

After the cell decomposition has been constructed in $\mathbb{R}^2$, it needs to be lifted into $\mathbb{R}^2 \times [0, 2\pi] / \sim$. This generates a cylinder of 3-cells above each 2D noncritical region, R. The roadmap could easily be defined to have a vertex for every 3-cell and 2-cell, which would be consistent with previous cell decompositions; however, vertices at 2-cells are not generated here to make the coming example easier to understand. Each 3-cell, $\{R, [f_i, f_{i+1}]\}$, corresponds to the vertex in a roadmap. The roadmap edges connect neighboring 3-cells that have a 2-cell as part of their common boundary. This means that in $\mathbb{R}^2$ they share a one-dimensional portion of a critical curve.

Constructing the roadmap The problem is to determine which 3-cells are actually adjacent. Figure 6.27 depicts the cases in which connections need to be made. The xy plane is represented as one axis (imagine looking in a direction parallel to it). Consider two neighboring 2-cells (noncritical regions), R and R', in the plane. It is assumed that a 1-cell (critical curve) in $\mathbb{R}^2$ separates them. The task is to connect together 3-cells in the cylinders above R and R'. If neighboring cells share the same feature pair, then they are connected. This means that $\{R, [f_i, f_{i+1}]\}$ and $\{R', [f_i, f_{i+1}]\}$ must be connected. In some cases, one feature

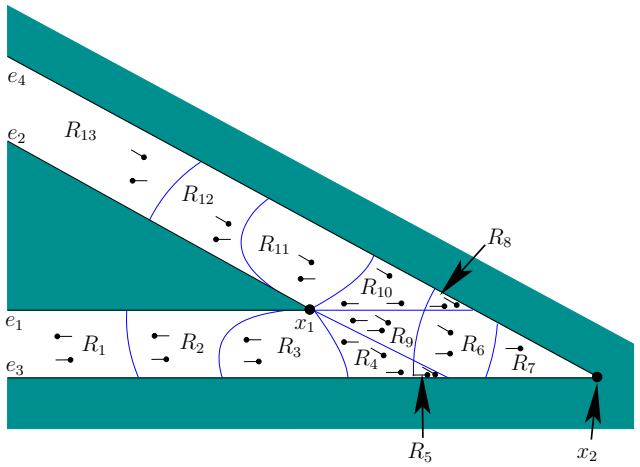


Figure 6.28: A depiction of the 3-cells above the noncritical regions. Sample rod orientations are shown for each cell (however, the rod length is shortened for clarity). Edges between cells are shown in Figure 6.29.

may change, while the interval of orientations remains unchanged. This may happen, for example, when the robot changes from contacting an edge to contacting a vertex of the edge. In these cases, a connection must also be made. One case illustrated in Figure 6.27 is when a splitting or merging of orientation intervals occurs. Traveling from R to R', the figure shows two regions merging into one. In this case, connections must be made from each of the original two 3-cells to the merged 3-cell. When constructing the roadmap edges, sample points of both the 3-cells and 2-cells should be used to ensure collision-free paths are obtained, as in the case of the vertical decomposition in Section 6.2.2. Figure 6.28 depicts the cells for the example in Figure 6.26. Each noncritical region has between one and three cells above it. Each of the various cells is indicated by a shortened robot that points in the general direction of the cell. The connections between the cells are also shown. Using the noncritical region and feature names from Figure 6.26, the resulting roadmap is depicted abstractly in Figure 6.29. Each vertex represents a 3-cell in $\mathcal{C}_{free}$, and each edge represents the crossing of a 2-cell between adjacent 3-cells. To make the roadmap consistent with previous roadmaps, we could insert a vertex into every edge and force the path to travel through the sample point of the corresponding 2-cell.

Once the roadmap has been constructed, it can be used in the same way as other roadmaps in this chapter to solve a query. Many implementation details have been neglected here. Due to the fifth case, some of the region boundaries in $\mathbb{R}^2$ are fourth-degree algebraic curves. Ways to prevent the explicit characterization of every noncritical region boundary, and other implementation details, are covered in [42]. Some of these details are also summarized in [304].

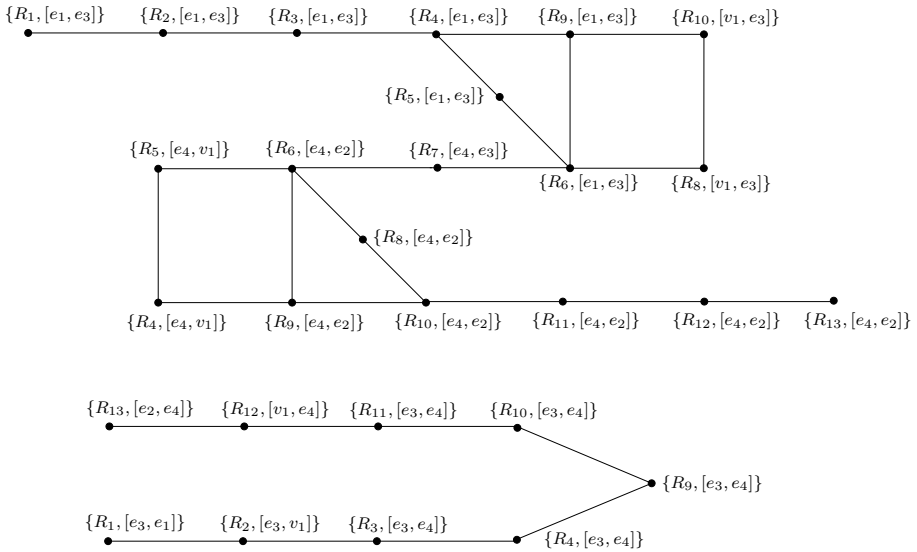


Figure 6.29: The roadmap corresponding to the example in Figure 6.26.

Complexity How many cells can there possibly be in the worst case? First count the number of noncritical regions in $\mathbb{R}^2$. There are $O(n)$ different ways to generate critical curves of the first three types because each corresponds to a single feature. Unfortunately, there are $O(n^2)$ different ways to generate bitangents and the Conchoid of Nicomedes because these are based on pairs of features. Assuming no self-intersections, a collection of $O(n^2)$ curves in $\mathbb{R}^2$, may intersect to generate at most $O(n^4)$ regions. Above each noncritical region in $\mathbb{R}^2$, there could be a cylinder of $O(n)$ 3-cells. Therefore, the size of the cell decomposition is $O(n^5)$ in the worst case. In practice, however, it is highly unlikely that all of these intersections will occur, and the number of cells is expected to be reasonable. In [427], an $O(n^5)$-time algorithm is given to construct the cell decomposition. Algorithms that have much better running time are mentioned in Section 6.5.3, but they are more complicated to understand and implement.

6.4 Computational Algebraic Geometry

This section presents algorithms that are so general that they solve any problem of Formulation 4.1 and even the closed-chain problems of Section 4.4. It is amazing that such algorithms exist; however, it is also unfortunate that they are both extremely challenging to implement and not efficient enough for most applications. The concepts and tools of this section were mostly developed in the context of computational real algebraic geometry [53, 138]. They are powerful

enough to conquer numerous problems in robotics, computer vision, geometric modeling, computer-aided design, and geometric theorem proving. One of these problems happens to be motion planning, for which the connection to computational algebraic geometry was first recognized in [428].

6.4.1 Basic Definitions and Concepts

This section builds on the semi-algebraic model definitions from Section 3.1 and the polynomial definitions from Section 4.4.1. It will be assumed that $\mathcal{C} \subseteq \mathbb{R}^n$, which could for example arise by representing each copy of $SO(2)$ or $SO(3)$ in its 2×2 or 3×3 matrix form. For example, in the case of a 3D rigid body, we know that $\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3$, which is a six-dimensional manifold, but it can be embedded in $\mathbb{R}^{12}$, which is obtained from the Cartesian product of $\mathbb{R}^3$ and the set of all 3×3 matrices. The constraints that force the matrices to lie in $SO(2)$ or $SO(3)$ are polynomials, and they can therefore be added to the semi-algebraic models of $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$. If the dimension of $\mathcal{C}$ is less than n, then the algorithm presented below is sufficient, but there are some representation and complexity issues that motivate using a special parameterization of $\mathcal{C}$ to make both dimensions the same while altering the topology of $\mathcal{C}$ to become homeomorphic to $\mathbb{R}^n$. This is discussed briefly in Section 6.4.2.

Suppose that the models in $\mathbb{R}^n$ are all expressed using polynomials from $\mathbb{Q}[x_1, \ldots, x_n]$, the set of polynomials[6] over the field of rational numbers $\mathbb{Q}$. Let $f \in \mathbb{Q}[x_1, \ldots, x_n]$ denote a polynomial.

Tarski sentences Recall the logical predicates that were formed in Section 3.1. They will be used again here, but now they are defined with a little more flexibility. For any $f \in \mathbb{Q}[x_1, \ldots, x_n]$, an *atom* is an expression of the form $f \bowtie 0$, in which $\bowtie$ may be any relation in the set $\{=, \neq, <, >, \leq, \geq\}$. In Section 3.1, such expressions were used to define logical predicates. Here, we assume that relations other than $\leq$ can be used and that the vector of polynomial variables lies in $\mathbb{R}^n$.

A *quantifier-free formula*, $\phi(x_1, \ldots, x_n)$, is a logical predicate composed of atoms and logical connectives, "and," "or," and "not," which are denoted by $\wedge$, $\vee$, and $\neg$, respectively. Each atom itself is considered as a logical predicate that yields TRUE if and only if the relation is satisfied when the polynomial is evaluated at the point $(x_1, \ldots, x_n) \in \mathbb{R}^n$.

Example 6.2 (An Example Predicate) Let ϕ be a predicate over $\mathbb{R}^3$, defined as

$$\phi(x_1, x_2, x_3) = (x_1^2 x_3 - x_2^4 < 0) \vee \big(\neg(3x^2x^3 \neq 0) \wedge (2x_3^2 - x_1 x_2 x_3 + 2 \geq 0)\big). \quad (6.8)$$

The precedence order of the connectives follows the laws of Boolean algebra. ■

[6] It will be explained shortly why $\mathbb{Q}[x_1, \ldots, x_n]$ is preferred over $\mathbb{R}[x_1, \ldots, x_n]$.

Let a *quantifier* $\mathcal{Q}$ be either of the symbols, $\forall$, which means "for all," or $\exists$, which means "there exists." A *Tarski sentence* Φ is a logical predicate that may additionally involve quantifiers on some or all of the variables. In general, a Tarski sentence takes the form

$$\Phi(x_1, \ldots, x_{n-k}) = (\mathcal{Q}z_1)(\mathcal{Q}z_2) \ldots (\mathcal{Q}z_k)\ \phi(z_1, \ldots, z_k, x_1, \ldots, x_{n-k}), \tag{6.9}$$

in which the z_i are the *quantified variables*, the x_i are the *free variables*, and ϕ is a quantifier-free formula. The quantifiers do not necessarily have to appear at the left to be a valid Tarski sentence; however, any expression can be manipulated into an equivalent expression that has all quantifiers in front, as shown in (6.9). The procedure for moving quantifiers to the front is as follows [370]: 1) Eliminate any redundant quantifiers; 2) rename some of the variables to ensure that the same variable does not appear both free and bound; 3) move negation symbols as far inward as possible; and 4) push the quantifiers to the left.

Example 6.3 (Several Tarski Sentences) Tarski sentences that have no free variables are either TRUE or FALSE in general because there are no arguments on which the results depend. The sentence

$$\Phi = \forall x \exists y\ (x^2 - y < 0), \tag{6.10}$$

is TRUE because for any $x \in \mathbb{R}$, some $y \in \mathbb{R}$ can always be chosen so that $y > x^2$. In the general notation of (6.9), this example becomes $\mathcal{Q}z_1 = \forall x$, $\mathcal{Q}z_2 = \exists y$, and $\phi(z_1, z_2) = (x^2 - y < 0)$.

Swapping the order of the quantifiers yields the Tarski sentence

$$\Phi = \exists y \forall x\ (x^2 - y < 0), \tag{6.11}$$

which is FALSE because for any y, there is always an x such that $x^2 > y$.

Now consider a Tarski sentence that has a free variable:

$$\Phi(z) = \exists y \forall x\ (x^2 - zx^2 - y < 0). \tag{6.12}$$

This yields a function $\Phi : \mathbb{R} \rightarrow \{\text{TRUE}, \text{FALSE}\}$, in which

$$\Phi(z) = \begin{cases} \text{TRUE} & \text{if } z \geq 1 \\ \text{FALSE} & \text{if } z < 1. \end{cases} \tag{6.13}$$

An equivalent quantifier-free formula ϕ can be defined as $\phi(z) = (z > 1)$, which takes on the same truth values as the Tarski sentence in (6.12). This might make you wonder whether it is always possible to make a simplification that eliminates the quantifiers. This is called the *quantifier-elimination problem*, which will be explained shortly. ■

The decision problem The sentences in (6.10) and (6.11) lead to an interesting problem. Consider the set of all Tarski sentences that have no free variables. The subset of these that are TRUE comprise the *first-order theory of the reals*. Can an algorithm be developed to determine whether such a sentence is true? This is called the *decision problem* for the first-order theory of the reals. At first it may appear hopeless because $\mathbb{R}^n$ is uncountably infinite, and an algorithm must work with a finite set. This is a familiar issue faced throughout motion planning. The sampling-based approaches in Chapter 5 provided one kind of solution. This idea could be applied to the decision problem, but the resulting lack of completeness would be similar. It is not possible to check all possible points in $\mathbb{R}^n$ by sampling. Instead, the decision problem can be solved by constructing a combinatorial representation that exactly represents the decision problem by partitioning $\mathbb{R}^n$ into a finite collection of regions. Inside of each region, only one point needs to be checked. This should already seem related to cell decompositions in motion planning; it turns out that methods developed to solve the decision problem can also conquer motion planning.

The quantifier-elimination problem Another important problem was exemplified in (6.12). Consider the set of all Tarski sentences of the form (6.9), which may or may not have free variables. Can an algorithm be developed that takes a Tarski sentence Φ and produces an equivalent quantifier-free formula ϕ? Let $x_1, \ldots, x_n$ denote the free variables. To be equivalent, both must take on the same true values over $\mathbb{R}^n$, which is the set of all assignments $(x_1, \ldots, x_n)$ for the free variables.

Given a Tarski sentence, (6.9), the *quantifier-elimination problem* is to find a quantifier-free formula ϕ such that

$$\Phi(x_1, \ldots, x_n) = \phi(x_1, \ldots, x_n) \tag{6.14}$$

for all $(x_1, \ldots, x_n) \in \mathbb{R}^n$. This is equivalent to constructing a semi-algebraic model because ϕ can always be expressed in the form

$$\phi(x_1, \ldots, x_n) = \bigvee_{i=1}^{k} \bigwedge_{j=1}^{m_i} \left(f_{i,j}(x_1, \ldots, x_n) \bowtie 0 \right), \tag{6.15}$$

in which $\bowtie$ may be either $<$, $=$, or $>$. This appears to be the same (3.6), except that (6.15) uses the relations $<$, $=$, and $>$ to allow open and closed semi-algebraic sets, whereas (3.6) only used $\leq$ to construct closed semi-algebraic sets for $\mathcal{O}$ and $\mathcal{A}$.

Once again, the problem is defined on $\mathbb{R}^n$, which is uncountably infinite, but an algorithm must work with a finite representation. This will be achieved by the cell decomposition technique presented in Section 6.4.2.

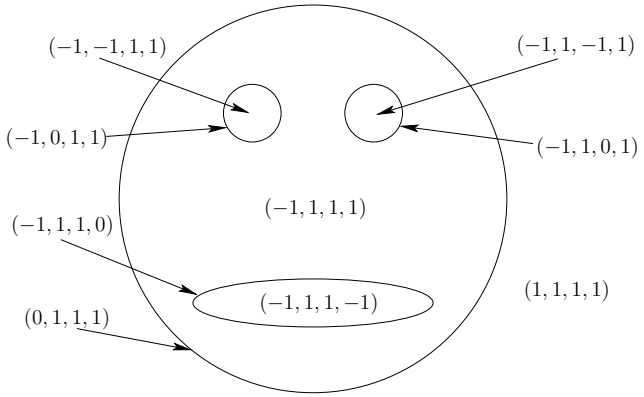


Figure 6.30: A semi-algebraic decomposition of the gingerbread face yields 9 sign-invariant regions.

Semi-algebraic decomposition As stated in Section 6.3.1, motion planning inside of each cell in a complex should be trivial. To solve the decision and quantifier-elimination problems, a cell decomposition was developed for which these problems become trivial in each cell. The decomposition is designed so that only a single point in each cell needs to be checked to solve the decision problem.

The semi-algebraic set $Y \subseteq \mathbb{R}^n$ that is expressed with (6.15) is

$$Y = \bigcup_{i=1}^{k} \bigcap_{j=1}^{m_i} \{(x_1, \ldots, x_n) \in \mathbb{R}^n \mid \mathrm{sgn}(f_{i,j}(x_1, \ldots, x_n)) = s_{i,j}\}, \tag{6.16}$$

in which sgn is the sign function, and each $s_{i,j} \in \{-1, 0, 1\}$, which is the range of sgn. Once again the nice relationship between set-theory and logic, which was described in Section 3.1, appears here. We convert from a set-theoretic description to a logical predicate by changing $\cup$ and $\cap$ to $\vee$ and $\wedge$, respectively.

Let $\mathcal{F}$ denote the set of $m = \sum_{i=1}^{k} m_i$ polynomials that appear in (6.16). A *sign assignment* with respect to $\mathcal{F}$ is a vector-valued function, $\mathrm{sgn}_{\mathcal{F}} : \mathbb{R}^n \rightarrow \{-1, 0, 1\}^m$. Each $f \in \mathcal{F}$ has a corresponding position in the sign assignment vector. At this position, the sign, $\mathrm{sgn}(f(x_1, \ldots, x_n)) \in \{-1, 0, 1\}$, appears. A *semi-algebraic decomposition* is a partition of $\mathbb{R}^n$ into a finite set of connected regions that are each *sign invariant*. This means that inside of each region, $\mathrm{sgn}_{\mathcal{F}}$ must remain constant. The regions will not be called *cells* because a semi-algebraic decomposition is not necessarily a singular complex as defined in Section 6.3.1; the regions here may contain holes.

Example 6.4 (Sign assignment) Recall Example 3.1 and Figure 3.4 from Section 3.1.2. Figure 3.4a shows a sign assignment for a case in which there is only

one polynomial, $\mathcal{F} = \{x^2 + y^2 - 4\}$. The sign assignment is defined as

$$\mathrm{sgn}_{\mathcal{F}}(x, y) = \begin{cases} -1 & \text{if } x^2 + y^2 - 4 < 0 \\ 0 & \text{if } x^2 + y^2 - 4 = 0 \\ 1 & \text{if } x^2 + y^2 - 4 > 0. \end{cases} \tag{6.17}$$

Now consider the sign assignment $\mathrm{sgn}_{\mathcal{F}}$, shown in Figure 6.30 for the gingerbread face of Figure 3.4b. The polynomials of the semi-algebraic model are $\mathcal{F} = \{f_1, f_2, f_3, f_4\}$, as defined in Example 3.1. In order, these are the "head," "left eye," "right eye," and "mouth." The sign assignment produces a four-dimensional vector of signs. Note that if (x, y) lies on one of the zeros of a polynomial in $\mathcal{F}$, then a 0 appears in the sign assignment. If the curves of two or more of the polynomials had intersected, then the sign assignment would produce more than one 0 at the intersection points.

For the semi-algebraic decomposition for the gingerbread face in Figure 6.30, there are nine regions. Five 2D regions correspond to: 1) being outside of the face, 2)inside of the left eye, 3) inside of the right eye, 4) inside of the mouth, and 5) inside of the face but outside of the mouth and eyes. There are four 1D regions, each of which corresponds to points that lie on one of the zero sets of a polynomial. The resulting decomposition is not a singular complex because the $(-1, 1, 1, 1)$ region contains three holes. ■

A decomposition such as the one in Figure 6.30 would not be very useful for motion planning because of the holes in the regions. Further refinement is needed for motion planning, which is fortunately produced by cylindrical algebraic decomposition. On the other hand, any semi-algebraic decomposition is quite useful for solving the decision problem. Only one point needs to be checked inside of each region to determine whether some Tarski sentence that has no free variables is true. Why? If the polynomial signs cannot change over some region, then the TRUE/FALSE value of the corresponding logical predicate, Φ, cannot change. Therefore, it sufficient only to check one point per sign-invariant region.

6.4.2 Cylindrical Algebraic Decomposition

Cylindrical algebraic decomposition is a general method that produces a cylindrical decomposition in the same sense considered in Section 6.3.2 for polygons in $\mathbb{R}^2$ and also the decomposition in Section 6.3.4 for the line-segment robot. It is also referred to as *Collins decomposition* after its original developer [31, 124, 125]. The decomposition in Figure 6.19 can even be considered as a cylindrical algebraic decomposition for a semi-algebraic set in which every geometric primitive is a linear polynomial. In this section, such a decomposition is generalized to any semi-algebraic set in $\mathbb{R}^n$.

The idea is to develop a sequence of projections that drops the dimension of the semi-algebraic set by one each time. Initially, the set is defined over $\mathbb{R}^n$,

and after one projection, a semi-algebraic set is obtained in $\mathbb{R}^{n-1}$. Eventually, the projection reaches $\mathbb{R}$, and a univariate polynomial is obtained for which the zeros are at the critical places where cell boundaries need to be formed. A cell decomposition of 1-cells (intervals) and 0-cells is formed by partitioning $\mathbb{R}$. The sequence is then reversed, and decompositions are formed from $\mathbb{R}^2$ up to $\mathbb{R}^n$. Each iteration starts with a cell decomposition in $\mathbb{R}^i$ and lifts it to obtain a cylinder of cells in $\mathbb{R}^{i+1}$. Figure 6.35 shows how the decomposition looks for the gingerbread example; since $n = 2$, it only involves one projection and one lifting.

Semi-algebraic projections are semi-algebraic The following is implied by the *Tarski-Seidenberg Theorem* [53]:

> *A projection of a semi-algebraic set from dimension n to dimension $n-1$ is a semi-algebraic set.*

This gives a kind of closure of semi-algebraic sets under projection, which is required to ensure that every projection of a semi-algebraic set in $\mathbb{R}^i$ leads to a semi-algebraic set in $\mathbb{R}^{i-1}$. This property is actually not true for (real) algebraic varieties, which were introduced in Section 4.4.1. Varieties are defined using only the $=$ relation and are not closed under the projection operation. Therefore, it is a good thing (not just a coincidence!) that we are using semi-algebraic sets.

Real algebraic numbers As stated previously, the sequence of projections ends with a univariate polynomial over $\mathbb{R}$. The sides of the cells will be defined based on the precise location of the roots of this polynomial. Furthermore, representing a sample point for a cell of dimension k in a complex in $\mathbb{R}^n$ for $k < n$ requires perfect precision. If the coordinates are slightly off, the point will lie in a different cell. This raises the complicated issue of how these roots are represented and manipulated in a computer.

For univariate polynomials of degree 4 or less, formulas exist to compute all of the roots in terms of functions of square roots and higher order roots. From Galois theory [244, 394], it is known that such formulas and nice expressions for roots do not exist for most higher degree polynomials, which can certainly arise in the complicated semi-algebraic models that are derived in motion planning. The roots in $\mathbb{R}$ could be any real number, and many real numbers require infinite representations.

One way of avoiding this mess is to assume that only polynomials in $\mathbb{Q}[x_1, \ldots, x_n]$ are used, instead of the more general $\mathbb{R}[x_1, \ldots, x_n]$. The field $\mathbb{Q}$ is not *algebraically closed* because zeros of the polynomials lie outside of $\mathbb{Q}^n$. For example, if $f(x_1) = x_1^2 - 2$, then $f = 0$ for $x_1 = \pm\sqrt{2}$, and $\sqrt{2} \notin \mathbb{Q}$. However, some elements of $\mathbb{R}$ can never be roots of a polynomial in $\mathbb{Q}[x_1, \ldots, x_n]$.

The set $\mathbb{A}$ of all real roots to all polynomials in $\mathbb{Q}[x]$ is called the set of *real algebraic numbers*. The set $\mathbb{A} \subset \mathbb{R}$ actually represents a field (recall from Section 4.4.1). Several nice algorithmic properties of the numbers in $\mathbb{A}$ are 1) they all have

finite representations, 2) addition and multiplication operations on elements of $\mathbb{A}$ can be computed in polynomial time, and 3) conversions between different representations of real algebraic numbers can be performed in polynomial time. This means that all operations can be computed efficiently without resorting to some kind of numerical approximation. In some applications, such approximations are fine; however, for algebraic decompositions, they destroy critical information by potentially confusing roots (e.g., how can we know for sure whether a polynomial has a double root or just two roots that are very close together?).

The details are not presented here, but there are several methods for representing real algebraic numbers and the corresponding algorithms for manipulating them efficiently. The running time of cylindrical algebraic decomposition ultimately depends on this representation. In practice, a numerical root-finding method that has a precision parameter, ϵ, can be used by choosing ϵ small enough to ensure that roots will not be confused. A sufficiently small value can be determined by applying *gap theorems*, which give lower bounds on the amount of real root separation, expressed in terms of the polynomial coefficients [92]. Some methods avoid requiring a precision parameter. One well-known example is the derivation of a Sturm sequence of polynomials based on the given polynomial. The polynomials in the Sturm sequence are then used to find isolating intervals for each of the roots [53]. The polynomial, together with its isolating interval, can be considered as an exact root representation. Algebraic operations can even be performed using this representation in time $O(d \lg^2 d)$, in which d is the degree of the polynomial [428]. See [53, 92, 428] for detailed presentations on the exact representation and calculation with real algebraic numbers.

One-dimensional decomposition To explain the cylindrical algebraic decomposition method, we first perform a semi-algebraic decomposition of $\mathbb{R}$, which is the final step in the projection sequence. Once this is explained, then the multi-dimensional case follows more easily.

Let $\mathcal{F}$ be a set of m univariate polynomials,

$$\mathcal{F} = \{f_i \in \mathbb{Q}[x] \mid i = 1, \ldots, m\}, \tag{6.18}$$

which are used to define some semi-algebraic set in $\mathbb{R}$. The polynomials in $\mathcal{F}$ could come directly from a quantifier-free formula ϕ (which could even appear inside of a Tarski sentence, as in (6.9)).

Define a single polynomial as $f = \prod_{i=1}^{m} f_i$. Suppose that f has k distinct, real roots, which are sorted in increasing order:

$$-\infty < \beta_1 < \beta_2 < \cdots < \beta_{i-1} < \beta_i < \beta_{i+1} < \cdots < \beta_k < \infty. \tag{6.19}$$

The one-dimensional semi-algebraic decomposition is given by the following sequence of alternating 1-cells and 0-cells:

$$\begin{array}{l}(-\infty, \beta_1),\ [\beta_1, \beta_1],\ (\beta_1, \beta_2),\ \ldots,\ (\beta_{i-1}, \beta_i),\ [\beta_i, \beta_i], \\ (\beta_i, \beta_{i+1}),\ \ldots,\ [\beta_k, \beta_k],\ (\beta_k, \infty).\end{array} \tag{6.20}$$

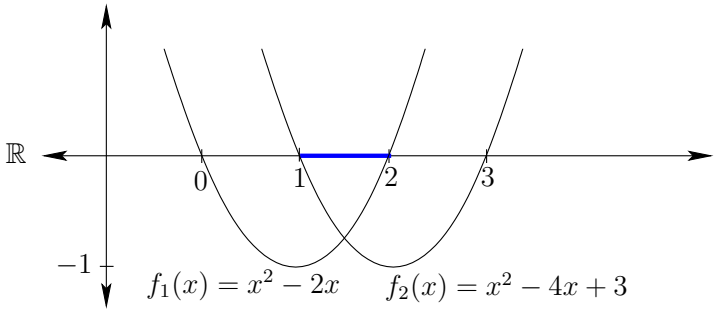


Figure 6.31: Two parabolas are used to define the semi-algebraic set $[1, 2]$.

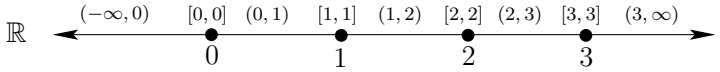


Figure 6.32: A semi-algebraic decomposition for the polynomials in Figure 6.31.

Any semi-algebraic set that can be expressed using the polynomials in $\mathcal{F}$ can also be expressed as the union of some of the 0-cells and 1-cells given in (6.20). This can also be considered as a singular complex (it can even be considered as a simplicial complex, but this does not extend to higher dimensions).

Sample points can be generated for each of the cells as follows. For the unbounded cells $[-\infty, \beta_1)$ and $(\beta_k, \infty]$, valid samples are $\beta_1 - 1$ and $\beta_k + 1$, respectively. For each finite 1-cell, (β_i, β_{i+1}), the midpoint $(\beta_i + \beta_{i+1})/2$ produces a sample point. For each 0-cell, $[\beta_i, \beta_i]$, the only choice is to use β_i as the sample point.

Example 6.5 (One-Dimensional Decomposition) Figure 6.31 shows a semi-algebraic subset of $\mathbb{R}$ that is defined by two polynomials, $f_1(x) = x^2 - 2x$ and $f_2(x) = x^2 - 4x + 3$. Here, $\mathcal{F} = \{f_1, f_2\}$. Consider the quantifier-free formula

$$\phi(x) = (x^2 - 2x \geq 0) \wedge (x^2 - 4x + 3 \geq 0). \tag{6.21}$$

The semi-algebraic decomposition into five 1-cells and four 0-cells is shown in Figure 6.32. Each cell is sign invariant. The sample points for the 1-cells are -1, $1/2$, $3/2$, $5/2$, and 4, respectively. The sample points for the 0-cells are 0, 1, 2, and 3, respectively.

A decision problem can be nicely solved using the decomposition. Suppose a Tarski sentence that uses the polynomials in $\mathcal{F}$ has been given. Here is one possibility:

$$\Phi = \exists x[(x^2 - 2x \geq 0) \wedge (x^2 - 4x + 3 \geq 0)] \tag{6.22}$$

The sample points alone are sufficient to determine whether Φ is TRUE or FALSE. Once $x = 1$ is attempted, it is discovered that Φ is TRUE. The quantifier-elimination problem cannot yet be considered because more dimensions are needed.

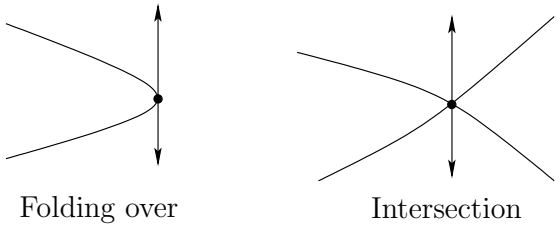


Figure 6.33: Critical points occur either when the surface folds over in the vertical direction or when surfaces intersect.

■

The inductive step to higher dimensions Now consider constructing a cylindrical algebraic decomposition for $\mathbb{R}^n$ (note the decomposition is actually semi-algebraic). Figure 6.35 shows an example for $\mathbb{R}^2$. First consider how to iteratively project the polynomials down to $\mathbb{R}$ to ensure that when the decomposition of $\mathbb{R}^n$ is constructed, the sign-invariant property is maintained. The resulting decomposition corresponds to a singular complex.

There are two cases that cause cell boundaries to be formed, as shown in Figure 6.33. Let $\mathcal{F}_n$ denote the original set of polynomials in $\mathbb{Q}[x_1, \ldots, x_n]$ that are used to define the semi-algebraic set (or Tarski sentence) in $\mathbb{R}^n$. Form a single polynomial $f = \prod_{i=1}^{m} f_i$. Let $f' = \partial f / \partial x_n$, which is also a polynomial. Let $g = GCD(f, f')$, which is the greatest common divisor of f and f'. The set of zeros of g is the set of all points that are zeros of both f and f'. Being a zero of f' means that the surface given by $f = 0$ does not vary locally when perturbing x_n. These are places where a cell boundary needs to be formed because the surface may fold over itself in the x_n direction, which is not permitted for a cylindrical decomposition. Another place where a cell boundary needs to be formed is at the intersection of two or more polynomials in $\mathcal{F}_n$. The projection technique from $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$ generates a set, $\mathcal{F}_{n-1}$, of polynomials in $\mathbb{Q}[x_1, \ldots, x_{n-1}]$ that satisfies these requirements. The polynomials $\mathcal{F}_{n-1}$ have the property that at least one contains a zero point below every point in $x \in \mathbb{R}^n$ for which $f(x) = 0$ and $f'(x) = 0$, or polynomials in $\mathcal{F}_n$ intersect. The projection method that constructs $\mathcal{F}_{n-1}$ involves computing *principle subresultant coefficients*, which are covered in [53, 429]. Resultants, of which the subresultants are an extension, are covered in [138].

The polynomials in $\mathcal{F}_{n-1}$ are then projected to $\mathbb{R}^{n-2}$ to obtain $\mathcal{F}_{n-2}$. This process continues until $\mathcal{F}_1$ is obtained, which is a set of polynomials in $\mathbb{Q}[x_1]$. A one-dimensional decomposition is formed, as defined earlier. From $\mathcal{F}_1$, a single polynomial is formed by taking the product, and $\mathbb{R}$ is partitioned into 0-cells and 1-cells. We next describe the process of lifting a decomposition over $\mathbb{R}^{i-1}$ up to $\mathbb{R}^i$. This technique is applied iteratively until $\mathbb{R}^n$ is reached.

Assume inductively that a cylindrical algebraic decomposition has been computed for a set of polynomials $\mathcal{F}_{i-1}$ in $\mathbb{Q}[x_1, \ldots, x_{i-1}]$. The decomposition consists of k-cells for which $0 \leq k \leq i$. Let $p = (x_1, \ldots, x_{i-1}) \in \mathbb{R}^{i-1}$. For each one of the k-cells C_{i-1}, a *cylinder* over C_{i-1} is defined as the $(k+1)$-dimensional set

$$\{(p, x_i) \in \mathbb{R}^i \mid p \in C_{i-1}\}. \tag{6.23}$$

The cylinder is sliced into a strip of k-dimensional and $k+1$-dimensional cells by using polynomials in $\mathcal{F}_i$. Let f_j denote one of the ℓ slicing polynomials in the cylinder, sorted in increasing x_i order as f_1, f_2, …, f_j, f_{j+1}, …, f_ℓ. The following kinds of cells are produced (see Figure 6.34):

1. **Lower unbounded sector:**

$$\{(p, x_i) \in \mathbb{R}^i \mid p \in C_{i-1} \text{ and } x_i < f_1(p)\}. \tag{6.24}$$

2. **Section:**

$$\{(p, x_i) \in \mathbb{R}^i \mid p \in C_{i-1} \text{ and } x_i = f_j(p)\}. \tag{6.25}$$

3. **Bounded sector:**

$$\{(p, x_i) \in \mathbb{R}^i \mid p \in C_{i-1} \text{ and } f_j(p) < x_i < f_{j+1}(p)\}. \tag{6.26}$$

4. **Upper unbounded sector:**

$$\{(p, x_i) \in \mathbb{R}^i \mid p \in C_{i-1} \text{ and } f_\ell(p) < x_i\}. \tag{6.27}$$

There is one degenerate possibility in which there are no slicing polynomials and the cylinder over C_{i-1} can be extended into one unbounded cell. In general, the sample points are computed by picking a point in $p \in C_{i-1}$ and making a vertical column of samples of the form (p, x_i). A polynomial in $\mathbb{Q}[x_i]$ can be generated, and the samples are placed using the same assignment technique that was used for the one-dimensional decomposition.

Example 6.6 (Mutilating the Gingerbread Face) Figure 6.35 shows a cylindrical algebraic decomposition of the gingerbread face. Observe that the resulting complex is very similar to that obtained in Figure 6.19. ■

Note that the cells do not necessarily project onto a rectangular set, as in the case of a higher dimensional vertical decomposition. For example, a generic n-cell C_n for a decomposition of $\mathbb{R}^n$ is described as the open set of $(x_1, \ldots, x_n) \in \mathbb{R}^n$ such that

- $C_0 < x_n < C_0'$ for some 0-cells $C_0, C_0' \in \mathbb{R}$, which are roots of some $f, f' \in \mathcal{F}_1$.

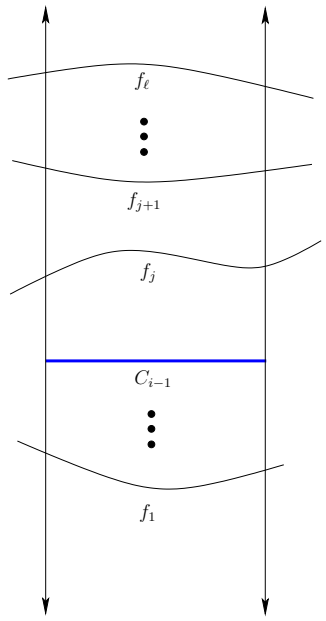


Figure 6.34: A cylinder over every k-cell C_{i-1} is formed. A sequence of polynomials, f_1, …, f_ℓ, slices the cylinder into k-dimensional sections and $(k+1)$-dimensional sectors.

- (x_{n-1}, x_n) lies between C_1 and C_1' for some 1-cells C_1, C_1', which are zeros of some $f, f' \in \mathcal{F}_2$.

  ⋮

- $(x_{n-i+1}, \ldots, x_n)$ lies between C_{i-1} and C_{i-1}' for some i-cells C_{i-1}, C_{i-1}', which are zeros of some $f, f' \in \mathcal{F}_i$.

  ⋮

- $(x_1, \ldots, x_n)$ lies between C_{n-1} and C_{n-1}' for some $(n-1)$-cells C_{n-1}, C_{n-1}', which are zeros of some $f, f' \in \mathcal{F}_n$.

The resulting decomposition is sign invariant, which allows the decision and quantifier-elimination problems to be solved in finite time. To solve a decision problem, the polynomials in $\mathcal{F}_n$ are evaluated at every sample point to determine whether one of them satisfies the Tarski sentence. To solve the quantifier-elimination problem, note that any semi-algebraic sets that can be constructed from $\mathcal{F}_n$ can be defined as a union of some cells in the decomposition. For the given Tarski sentence, $\mathcal{F}_n$ is formed from all polynomials that are mentioned in

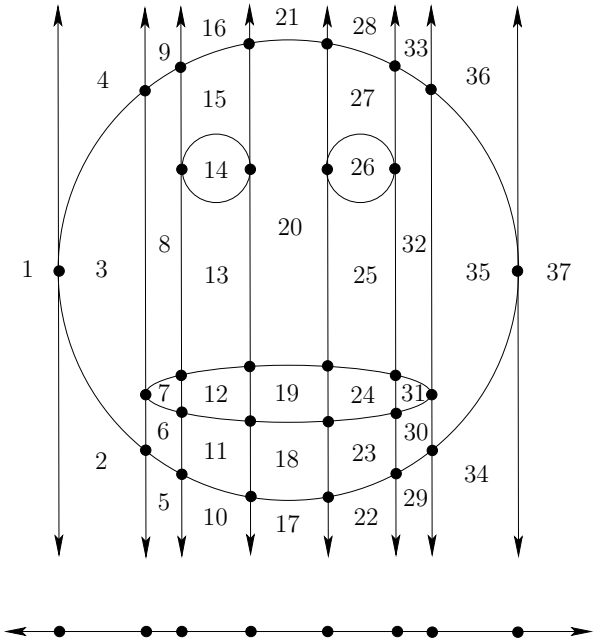


Figure 6.35: A cylindrical algebraic decomposition of the gingerbread face. There are 37 2-cells, 64 1-cells, and 28 0-cells. The straight 1-cells are intervals of the vertical lines, and the curved ones are portions of the zero set of a polynomial in $\mathcal{F}$. The decomposition of $\mathbb{R}$ is also shown.

the sentence, and the cell decomposition is performed. Once obtained, the sign information is used to determine which cells need to be included in the union. The resulting union of cells is designed to include only the points in $\mathbb{R}^n$ at which the Tarski sentence is TRUE.

Solving a motion planning problem Cylindrical algebraic decomposition is also capable of solving any of the motion planning problems formulated in Chapter 4. First assume that $\mathcal{C} = \mathbb{R}^n$. As for other decompositions, a roadmap is formed in which every vertex is an n-cell and edges connect every pair of adjacent n-cells by traveling through an $(n-1)$-cell. It is straightforward to determine adjacencies inside of a cylinder, but there are several technical details associated with determining adjacencies of cells from different cylinders (pages 152–154 of [53] present an example that illustrates the problem). The cells of dimension less than $n-1$ are not needed for motion planning purposes (just as vertices were not needed for the vertical decomposition in Section 6.2.2). The query points q_I and q_G are connected to the roadmap depending on the cell in which they lie, and a discrete search is performed.

If $\mathcal{C} \subset \mathbb{R}^n$ and its dimension is k for $k < n$, then all of the interesting cells are of lower dimension. This occurs, for example, due to the constraints on the matrices to force them to lie in $SO(2)$ or $SO(3)$. This may also occur for problems from Section 4.4, in which closed chains reduce the degrees of freedom. The cylindrical algebraic decomposition method can still solve such problems; however, the exact root representation problem becomes more complicated when determining the cell adjacencies. A discussion of these issues appears in [428]. For the case of $SO(2)$ and $SO(3)$, this complication can be avoided by using *stereographic projection* to map $\mathbb{S}^1$ or $\mathbb{S}^3$ to $\mathbb{R}$ or $\mathbb{R}^3$, respectively. This mapping removes a single point from each, but the connectivity of $\mathcal{C}_{free}$ remains unharmed. The antipodal identification problem for unit quaternions represented by $\mathbb{S}^3$ also does not present a problem; there is a redundant copy of $\mathcal{C}$, which does not affect the connectivity.

The running time for cylindrical algebraic decomposition depends on many factors, but in general it is polynomial in the number of polynomials in $\mathcal{F}_n$, polynomial in the maximum algebraic degree of the polynomials, and doubly exponential in the dimension. Complexity issues are covered in more detail in Section 6.5.3.

6.4.3 Canny's Roadmap Algorithm

The doubly exponential running time of cylindrical algebraic decomposition inspired researchers to do better. It has been shown that quantifier elimination requires doubly exponential time [144]; however, motion planning is a different problem. Canny introduced a method that produces a roadmap directly from the semi-algebraic set, rather than constructing a cell decomposition along the way. Since there are doubly exponentially many cells in the cylindrical algebraic decomposition, avoiding this construction pays off. The resulting roadmap method of Canny solves the motion planning problem in time that is again polynomial in the number of polynomials and polynomial in the algebraic degree, but it is only singly exponential in dimension [90, 92]; see also [53].

Much like the other combinatorial motion planning approaches, it is based on finding critical curves and critical points. The main idea is to construct linear mappings from $\mathbb{R}^n$ to $\mathbb{R}^2$ that produce *silhouette curves* of the semi-algebraic sets. Performing one such mapping on the original semi-algebraic set yields a roadmap, but it might not preserve the original connectivity. Therefore, linear mappings from $\mathbb{R}^{n-1}$ to $\mathbb{R}^2$ are performed on some $(n-1)$-dimensional slices of the original semi-algebraic set to yield more roadmap curves. This process is applied recursively until the slices are already one-dimensional. The resulting roadmap is formed from the union of all of the pieces obtained in the recursive calls. The resulting roadmap has the same connectivity as the original semi-algebraic set [92].

Suppose that $\mathcal{C} = \mathbb{R}^n$. Let $\mathcal{F} = \{f_1, \ldots, f_m\}$ denote the set of polynomials that define the semi-algebraic set, which is assumed to be a disjoint union of

manifolds. Assume that each $f_i \in \mathbb{Q}[x_1, \ldots, x_n]$. First, a small perturbation to the input polynomials $\mathcal{F}$ is performed to ensure that every sign-invariant set of $\mathbb{R}^n$ is a manifold. This forces the polynomials into a kind of general position, which can be achieved with probability one using random perturbations; there are also deterministic methods to solve this problem. The general position requirements on the input polynomials and the 2D projection directions are fairly strong, which has stimulated more recent work that eliminates many of the problems [53]. From this point onward, it will be assumed that the polynomials are in general position.

Recall the sign-assignment function from Section 6.4.1. Each sign-invariant set is a manifold because of the general position assumption. Canny's method computes a roadmap for any k-dimensional manifold for $k < n$. Such a manifold has precisely $n - k$ signs that are 0 (which means that points lie precisely on the zero sets of $n - k$ polynomials in $\mathcal{F}$). At least one of the signs must be 0, which means that Canny's roadmap actually lies in $\partial \mathcal{C}_{free}$ (this technically is not permitted, but the algorithm nevertheless correctly decides whether a solution path exists through $\mathcal{C}_{free}$).

Recall that each f_i is a function, $f_i : \mathbb{R}^n \to \mathbb{R}$. Let x denote $(x_1, \ldots, x_n) \in \mathbb{R}^n$. The k polynomials that have zero signs can be put together sequentially to produce a mapping $\psi : \mathbb{R}^n \to \mathbb{R}^k$. The ith component of the vector $\psi(x)$ is $\psi_i(x) = f_i(x)$. This is closely related to the sign assignment function of Section 6.4.1, except that now the real value from each polynomial is directly used, rather than taking its sign.

Now introduce a function $g : \mathbb{R}^n \to \mathbb{R}^j$, in which either $j = 1$ or $j = 2$ (the general concepts presented below work for other values of j, but 1 and 2 are the only values needed for Canny's method). The function g serves the same purpose as a projection in cylindrical algebraic decomposition, but note that g immediately drops from dimension n to dimension 2 or 1, instead of dropping to $n - 1$ as in the case of cylindrical projections.

Let $h : \mathbb{R}^n \to \mathbb{R}^{k+j}$ denote a mapping constructed directly from ψ and g as follows. For the ith component, if $i \leq k$, then $h_i(x) = \psi_i(x) = f_i(x)$. Assume that $k + j \leq n$. If $i > k$, then $h_i(x) = g_{i-k}(x)$. Let $J_x(h)$ denote the *Jacobian* of h and be defined at x as

$$J_x(h) = \begin{pmatrix} \frac{\partial h_1(x)}{\partial x_1} & \cdots & \frac{\partial h_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial h_{m+k}(x)}{\partial x_1} & \cdots & \frac{\partial h_{m+k}(x)}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_k(x)}{\partial x_1} & \cdots & \frac{\partial f_k(x)}{\partial x_n} \\ \\ \frac{\partial g_1(x)}{\partial x_1} & \cdots & \frac{\partial g_1(x)}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial g_j(x)}{\partial x_1} & \cdots & \frac{\partial g_j(x)}{\partial x_n} \end{pmatrix}. \quad (6.28)$$

A point $x \in \mathbb{R}^n$ at which $J_x(h)$ is singular is called a *critical point*. The matrix is defined to be *singular* if every $(m+k) \times (m+k)$ subdeterminant is zero. Each of the first k rows of $J_x(h)$ calculates the surface normal to $f_i(x) = 0$. If these normals are not linearly independent of the directions given by the last j rows, then the matrix becomes singular. The following example from [89] nicely illustrates this principle.

Example 6.7 (Canny's Roadmap Algorithm) Let $n = 3$, $k = 1$, and $j = 1$. The zeros of a single polynomial f_1 define a 2D subset of $\mathbb{R}^3$. Let f_1 be the unit sphere, $\mathbb{S}^2$, defined as the zeros of the polynomial

$$f_1(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - 1. \quad (6.29)$$

Suppose that $g : \mathbb{R}^3 \to \mathbb{R}$ is defined as $g(x_1, x_2, x_3) = x_1$. The Jacobian, (6.28), becomes

$$\begin{pmatrix} 2x_1 & 2x_2 & 2x_3 \\ 1 & 0 & 0 \end{pmatrix} \quad (6.30)$$

and is singular when all three of the possible 2×2 subdeterminants are zero. This occurs if and only if $x_2 = x_3 = 0$. This yields the critical points $(-1, 0, 0)$ and $(1, 0, 0)$ on $\mathbb{S}^2$. Note that this is precisely when the surface normals of $\mathbb{S}^2$ are parallel to the vector [1 0 0].

Now suppose that $j = 2$ to obtain $g : \mathbb{R}^3 \to \mathbb{R}^2$, and suppose $g(x_1, x_2, x_3) = (x_1, x_2)$. In this case, (6.28) becomes

$$\begin{pmatrix} 2x_1 & 2x_2 & 2x_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \quad (6.31)$$

which is singular if and only if $x_3 = 0$. The critical points are therefore the x_1x_2 plane intersected with $\mathbb{S}^3$, which yields the equator points (all $(x_1, x_2) \in \mathbb{R}^2$ such that $x_1^2 + x_2^2 = 1$). In this case, more points are generated because the matrix becomes degenerate for any surface normal of $\mathbb{S}^2$ that is parallel to [1 0 0], [0 1 0] or any linear combination of these. ■

The first mapping in Example 6.7 yielded two isolated critical points, and the second mapping yielded a one-dimensional set of critical points, which is referred to as a *silhouette*. The union of the silhouette and the isolated critical points yields a roadmap for $\mathbb{S}^2$. Now consider generalizing this example to obtain the full algorithm for general n and k. A linear mapping $g : \mathbb{R}^n \to \mathbb{R}^2$ is constructed that might not be axis-aligned as in Example 6.7 because it must be chosen in general position (otherwise degeneracies might arise in the roadmap). Define ψ to be the set of polynomials that become zero on the desired manifold on which to construct a roadmap. Form the matrix (6.28) and determine the silhouette. This is accomplished in general using subresultant techniques that were also needed

for cylindrical algebraic decomposition; see [53, 92] for details. Let g_1 denote the first component of g, which yields a mapping $g_1 : \mathbb{R}^n \rightarrow \mathbb{R}$. Forming (6.28) using g_1 yields a finite set of critical points. Taking the union of the critical points and the silhouette produces part of the roadmap.

So far, however, there are no guarantees that the connectivity is preserved. To handle this problem, Canny's algorithm proceeds recursively. For each of the critical points $x \in \mathbb{R}^n$, an $n-1$-dimensional hyperplane through x is chosen for which the g_1 row of (6.28) is the normal (hence it is perpendicular in some sense to the flow of g_1). Inside of this hyperplane, a new g mapping is formed. This time a new direction is chosen, and the mapping takes the form $g : \mathbb{R}^{n-1} \rightarrow \mathbb{R}^2$. Once again, the silhouettes and critical points are found and added to the roadmap. This process is repeated recursively until the base case in which the silhouettes and critical points are directly obtained without forming g.

It is helpful to consider an example. Since the method involves a *sequence* of 2D projections, it is difficult to visualize. Problems in $\mathbb{R}^4$ and higher involve two or more 2D projections and would therefore be more interesting. An example over $\mathbb{R}^3$ is presented here, even though it unfortunately has only one projection; see [92] for another example over $\mathbb{R}^3$.

Example 6.8 (Canny's Algorithm on a Torus) Consider the 3D algebraic set shown in Figure 6.36. After defining the mapping $g(x_1, x_2, x_3) = (x_1, x_2)$, the roadmap shown in Figure 6.37 is obtained. The silhouettes are obtained from g, and the critical points are obtained from g_1 (this is the first component of g). Note that the original connectivity of the solid torus is not preserved because the inner ring does not connect to the outer ring. This illustrates the need to also compute the roadmap for lower dimensional slices. For each of the four critical points, the critical curves are computed for a plane that is parallel to the x_2x_3 plane and for which the x_1 position is determined by the critical point. The slice for one of the inner critical points is shown in Figure 6.38. In this case, the slice already has two dimensions. New silhouette curves are added to the roadmap to obtain the final result shown in Figure 6.39. ■

To solve a planning problem, the query points q_I and q_G are artificially declared to be critical points in the top level of recursion. This forces the algorithm to generate curves that connect them to the rest of the roadmap.

The completeness of the method requires very careful analysis, which is thoroughly covered in [53, 92]. The main elements of the analysis are showing that: 1) the polynomials can be perturbed and g can be chosen to ensure general position, 2) the singularity conditions on (6.28) lead to algebraic sets (varieties), and 3) the resulting roadmap has the required properties mentioned in Section 6.1 of being accessible and connectivity-preserving for $\mathcal{C}_{free}$ (actually it is shown for $\partial\mathcal{C}_{free}$). The method explained above computes the roadmap for each sign-invariant set, but to obtain a roadmap for the planning problem, the roadmaps

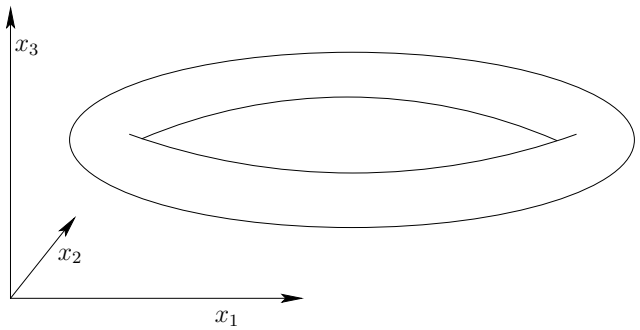


Figure 6.36: Suppose that the semi-algebraic set is a solid torus in $\mathbb{R}^3$.

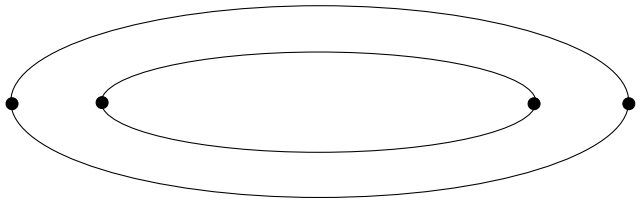


Figure 6.37: The projection into the x_1x_2 plane yields silhouettes for the inner and outer rings and also four critical points.

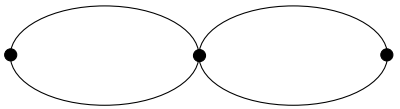


Figure 6.38: A slice taken for the inner critical points is parallel to the x_2x_3 plane. The roadmap for the slice connects to the silhouettes from Figure 6.37, thereby preserving the connectivity of the original set in Figure 6.36.

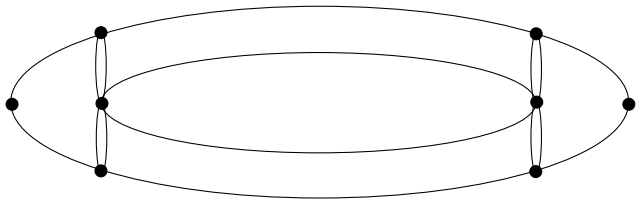


Figure 6.39: All of the silhouettes and critical points are merged to obtain the roadmap.

from each sign-invariant set must be connected together correctly; fortunately, this has been solved via the Linking Lemma of [89]. A major problem, however, is that even after knowing the connectivity of the roadmap, it is a considerable challenge to obtain a parameterization of each curve on the roadmap. For this and many other technical reasons, no general implementation of Canny's algorithm appears to exist at present. Another problem is the requirement of a Whitney stratification (which can be fixed by perturbation of the input). The *Basu-Pollack-Roy roadmap algorithm* overcomes this problem [53].

6.5 Complexity of Motion Planning

This section summarizes theoretical work that characterizes the complexity of motion planning problems. Note that this is not equivalent to characterizing the running time of particular algorithms. The existence of an algorithm serves as an *upper bound* on the problem's difficulty because it is a proof by example that solving the problem requires no more time than what is needed by the algorithm. On the other hand, *lower bounds* are also very useful because they give an indication of the difficulty of the problem itself. Suppose, for example, you are given an algorithm that solves a problem in time $O(n^2)$. Does it make sense to try to find a more efficient algorithm? Does it make sense to try to find a general-purpose motion planning algorithm that runs in time that is polynomial in the dimension? Lower bounds provide answers to questions such as this. Usually lower bounds are obtained by concocting bizarre, complicated examples that are allowed by the problem definition but were usually not considered by the person who first formulated the problem. In this line of research, progress is made by either raising the lower bound (unless it is already tight) or by showing that a narrower version of the problem still allows such bizarre examples. The latter case occurs often in motion planning.

6.5.1 Lower Bounds

Lower bounds have been established for a variety of motion planning problems and also a wide variety of planning problems in general. To interpret these bounds a basic understanding of the *theory of computation* is required [239, 442]. This fascinating subject will be unjustly summarized in a few paragraphs. A *problem* is a set of *instances* that are each carefully encoded as a binary string. An *algorithm* is formally considered as a *Turing machine*, which is a finite-state machine that can read and write bits to an unbounded piece of tape. Other models of computation also exist, such as integer RAM and real RAM (see [65]); there are debates as to which model is most appropriate, especially when performing geometric computations with real numbers. The standard Turing machine model will be assumed from here onward. Algorithms are usually formulated to make a binary output, which involves *accepting* or *rejecting* a problem instance that is

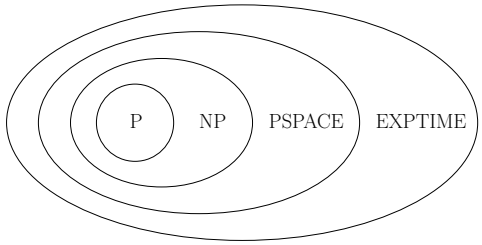


Figure 6.40: It is known that P $\subset$ EXPTIME is a strict subset; however, it is not known precisely how large NP and PSPACE are.

initially written to the tape and given to the algorithm. In motion planning, this amounts to deciding whether a solution path exists for a given problem instance.

Languages A *language* is a set of binary strings associated with a problem. It represents the complete set of instances of a problem. An algorithm is said to *decide* a language if in finite time it correctly accepts all strings that belong to it and rejects all others. The interesting question is: How much time or space is required to decide a language? This question is asked of the problem, under the assumption that the best possible algorithm would be used to decide it. (We can easily think of inefficient algorithms that waste resources.)

A *complexity class* is a set of languages that can all be decided within some specified resource bound. The class P is the set of all languages (and hence problems) for which a polynomial-time algorithm exists (i.e., the algorithm runs in time $O(n^k)$ for some integer k). By definition, an algorithm is called *efficient* if it decides its associated language in polynomial time.[7] If no efficient algorithm exists, then the problem is called *intractable*. The relationship between several other classes that often emerge in theoretical motion planning is shown in Figure 6.40. The class NP is the set of languages that can be solved in polynomial time by a *nondeterministic Turing machine*. Some discussion of nondeterministic machines appears in Section 11.3.2. Intuitively, it means that solutions can be verified in polynomial time because the machine magically knows which choices to make while trying to make the decision. The class PSPACE is the set of languages that can be decided with no more than a polynomial amount of storage space during the execution of the algorithm (NPSPACE=PSPACE, so there is no nondeterministic version). The class EXPTIME is the set of languages that can be decided in time $O(2^{n^k})$ for some integer k. It is known that EXPTIME is larger than P, but it is not known precisely where the boundaries of NP and PSPACE lie. It might be the case that P = NP = PSPACE (although hardly anyone believes this), or it could be that NP = PSPACE = EXPTIME.

[7] Note that this definition may be absurd in practice; an algorithm that runs in time $O(n^{90125})$ would probably not be too efficient for most purposes.

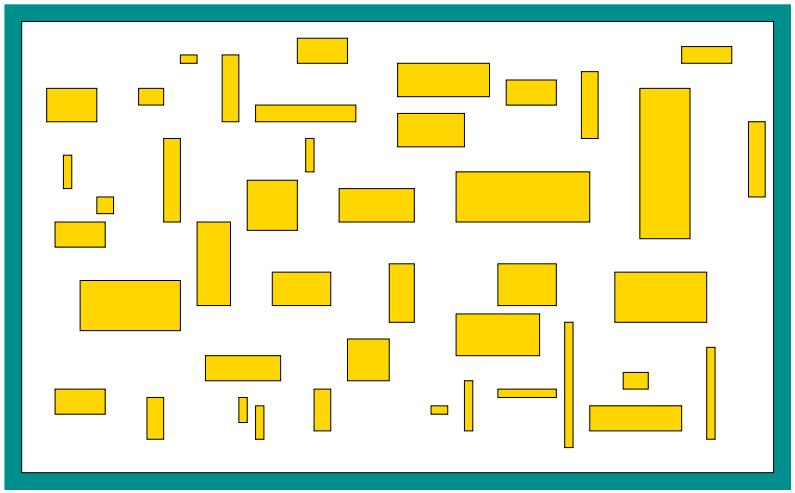


Figure 6.41: Even motion planning for a bunch of translating rectangles inside of a rectangular box in $\mathbb{R}^2$ is PSPACE-hard (and hence, NP-hard).

Hardness and completeness Since an easier class is included as a subset of a harder one, it is helpful to have a notion of a language (i.e., problem) being among the hardest possible within a class. Let X refer to either P, NP, PSPACE, or EXPTIME. A language A is called *X-hard* if every language B in class X is *polynomial-time reducible* to A. In short, this means that in polynomial time, any language in B can be translated into instances for language A, and then the decisions for A can be correctly translated back in polynomial time to correctly decide B. Thus, if A can be decided, then within a polynomial-time factor, every language in X can be decided. The hardness concept can even be applied to a language (problem) that does not belong to the class. For example, we can declare that a language A is NP-hard even if $A \notin$NP (it could be harder and lie in EXPTIME, for example). If it is known that the language is both hard for some class X and is also a member of X, then it is called *X-complete* (i.e., NP-complete, PSPACE-complete, etc.).[8] Note that because of this uncertainty regarding P, NP, and PSPACE, one cannot say that a problem is intractable if it is NP-hard or PSPACE-hard, but one can, however, if the problem is EXPTIME-hard. One additional remark: it is useful to remember that PSPACE-hard implies NP-hard.

Lower bounds for motion planning The general motion planning problem, Formulation 4.1, was shown in 1979 to be PSPACE-hard by Reif [409]. In fact, the problem was restricted to polyhedral obstacles and a finite number of polyhedral robot bodies attached by spherical joints. The coordinates of all polyhedra are assumed to be in $\mathbb{Q}$ (this enables a finite-length string encoding of the problem instance). The proof introduces a fascinating motion planning instance that involves many attached, dangling robot parts that must work their way through a complicated system of tunnels, which together simulates the operation of a *symmetric Turing machine*. Canny later established that the problem in Formulation 4.1 (expressed using polynomials that have rational coefficients) lies in PSPACE [92]. Therefore, the general motion planning problem is PSPACE-complete.

[8] If you remember hearing that a planning problem is NP-something, but cannot remember whether it was NP-hard or NP-complete, then it is safe to say NP-hard because NP-complete implies NP-hard. This can similarly be said for other classes, such as PSPACE-complete vs. PSPACE-hard.

Many other lower bounds have been shown for a variety of planning problems. One famous example is the Warehouseman's problem shown in Figure 6.41. This problem involves a finite number of translating, axis-aligned rectangles in a rectangular world. It was shown in [238] to be PSPACE-hard. This example is a beautiful illustration of how such a deceptively simple problem formulation can lead to such a high lower bound. More recently, it was even shown that planning for Sokoban, which is a warehouseman's problem on a discrete 2D grid, is also PSPACE-hard [141]. Other general motion planning problems that were shown to be PSPACE-hard include motion planning for a chain of bodies in the plane [237, 254] and motion planning for a chain of bodies among polyhedral obstacles in $\mathbb{R}^3$. Many lower bounds have been established for a variety of extensions and variations of the general motion planning problem. For example, in [91] it was established that a certain form of planning under uncertainty for a robot in a 3D polyhedral environment is NEXPTIME-hard, which is harder than any of the classes shown in Figure 6.40; the hardest problems in this NEXPTIME are believed to require doubly exponential time to solve.

The lower bound or hardness results depend significantly on the precise representation of the problem. For example, it is possible to make problems look easier by making instance encodings that are exponentially longer than they should be. The running time or space required is expressed in terms of n, the input size. If the motion planning problem instances are encoded with exponentially more bits than necessary, then a language that belongs to P is obtained. As long as the instance encoding is within a polynomial factor of the optimal encoding (this can be made precise using Kolmogorov complexity [323]), then this bizarre behavior is avoided. Another important part of the representation is to pay attention to how parameters in the problem formulation can vary. We can redefine motion planning to be all instances for which the dimension of $\mathcal{C}$ is never greater than 2^{1000}. The number of dimensions is sufficiently large for virtually any application. The resulting language for this problem belongs to P because cylindrical algebraic decomposition and Canny's algorithm can solve any motion planning problem in polynomial time. Why? This is because now the dimension parameter in the time-complexity expressions can be replaced by 2^{1000}, which is a constant. This formally implies that an *efficient* algorithm is already known for any motion planning problem that we would ever care about. This implication has no practical

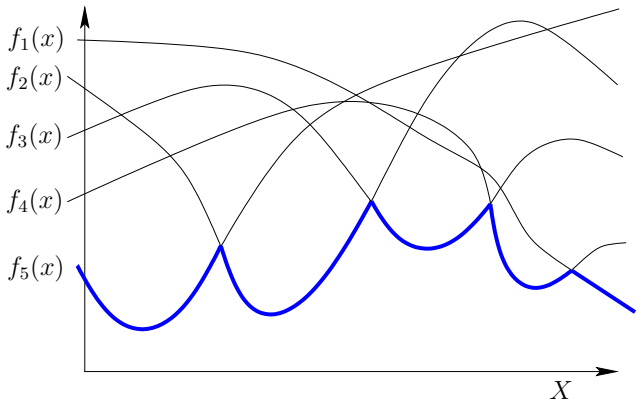


Figure 6.42: The lower envelope of a collection of functions.

value, however. Thus, be very careful when interpreting theoretical bounds.

The lower bounds may appear discouraging. There are two general directions to go from here. One is to weaken the requirements and tolerate algorithms that yield some kind of resolution completeness or probabilistic completeness. This approach was taken in Chapter 5 and leads to many efficient algorithms. Another direction is to define narrower problems that do not include the bizarre constructions that lead to bad lower bounds. For the narrower problems, it may be possible to design interesting, efficient algorithms. This approach was taken for the methods in Sections 6.2 and 6.3. In Section 6.5.3, upper bounds for some algorithms that address these narrower problems will be presented, along with bounds for the general motion planning algorithms. Several of the upper bounds involve Davenport-Schinzel sequences, which are therefore covered next.

6.5.2 Davenport-Schinzel Sequences

Davenport-Schinzel sequences provide a powerful characterization of the structure that arises from the lower or upper envelope of a collection of functions. The lower envelope of five functions is depicted in Figure 6.42. Such envelopes arise in many problems throughout computational geometry, including many motion planning problems. They are an important part of the design and analysis of many modern algorithms, and the resulting algorithm's time complexity usually involves terms that follow directly from the sequences. Therefore, it is worthwhile to understand some of the basics before interpreting some of the results of Section 6.5.3. Much more information on Davenport-Schinzel sequences and their applications appears in [436]. The brief introduction presented here is based on [435].

For positive integers n and s, an (n, s) *Davenport-Schinzel sequence* is a sequence $(u_1, \ldots, u_m)$ composed from a set of n *symbols* such that:

1. The same symbol may not appear consecutively in the sequence. In other words, $u_i \neq u_{i+1}$ for any i such that $1 \leq i < m$.

2. The sequence does not contain any alternating subsequence that uses two symbols and has length $s+2$. A subsequence can be formed by deleting any elements in the original sequence. The condition can be expressed as: There do not exist $s+2$ indices $i_1 < i_2 < \cdots < i_{s+2}$ for which $u_{i_1} = u_{i_3} = u_{i_5} = a$ and $u_{i_2} = u_{i_4} = u_{i_6} = b$, for some symbols a and b.

As an example, an $(n, 3)$ sequence cannot appear as $(a \cdots b \cdots a \cdots b \cdots a)$, in which each $\cdots$ is filled in with any sequence of symbols. Let $\lambda_s(n)$ denote the maximum possible length of an (n, s) Davenport-Schinzel sequence.

The connection between Figure 6.42 and these sequences can now be explained. Consider the sequence of function indices that visit the lower envelope. In the example, this sequence is $(5, 2, 3, 4, 1)$. Suppose it is known that each pair of functions intersects in at most s places. If there are n real-valued continuous functions, then the sequence of function indices must be an (n, s) Davenport-Schinzel sequence. It is amazing that such sequences cannot be very long. For a fixed s, they are close to being linear.

The standard bounds for Davenport-Schinzel sequences are [435][9]

$$\lambda_1(n) = n \tag{6.32}$$

$$\lambda_2(n) = 2n - 1 \tag{6.33}$$

$$\lambda_3(n) = \Theta(n\alpha(n)) \tag{6.34}$$

$$\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)}) \tag{6.35}$$

$$\lambda_{2s}(n) \leq n \cdot 2^{\alpha(n)^{s-1} + C_{2s}(n)} \tag{6.36}$$

$$\lambda_{2s+1}(n) \leq n \cdot 2^{\alpha(n)^{s-1} \lg \alpha(n) + C'_{2s+1}(n)} \tag{6.37}$$

$$\lambda_{2s}(n) = \Omega(n \cdot 2^{\frac{1}{(s-1)!}\alpha(n)^{s-1} + C'_{2s}(n)}). \tag{6.38}$$

In the expressions above $C_r(n)$ and $C'_r(n)$ are terms that are smaller than their leading exponents. The $\alpha(n)$ term is the inverse Ackerman function, which is an extremely slow-growing function that appears frequently in algorithms. The *Ackerman function* is defined as follows. Let $A_1(m) = 2m$ and $A_{n+1}(m)$ represent m applications of A_n. Thus, $A_1(m)$ performs doubling, $A_2(m)$ performs exponentiation, and $A_3(m)$ performs *tower exponentiation*, which makes a stack of 2's,

$$2^{2^{\cdot^{\cdot^{\cdot^{2^2}}}}}, \tag{6.39}$$

that has height m. The *Ackerman function* is defined as $A(n) = A_n(n)$. This function grows so fast that $A(4)$ is already an exponential tower of 2's that has

[9] The following asymptotic notion is used: $O(f(n))$ denotes an upper bound, $\Omega(f(n))$ denotes a lower bound, and $\Theta(f(n))$ means that the bound is tight (both upper and lower). This notation is used in most books on algorithms [132].

height 65536. Thus, the *inverse Ackerman function*, α, grows very slowly. If n is less than or equal to an exponential tower of 65536 2's, then $\alpha(n) \leq 4$. Even when it appears in exponents of the Davenport-Schinzel bounds, it does not represent a significant growth rate.

Example 6.9 (Lower Envelope of Line Segments) One interesting application of Davenport-Schinzel applications is to the lower envelope of a set of line segments in $\mathbb{R}^2$. Since segments in general position may appear multiple times along the lower envelope, the total number of edges is $\Theta(\lambda_3(n)) = \Theta(n\alpha(n))$, which is higher than one would obtain from infinite lines. There are actually arrangements of segments in $\mathbb{R}^2$ that reach this bound; see [436]. ■

6.5.3 Upper Bounds

The upper bounds for motion planning problems arise from the existence of complete algorithms that solve them. This section proceeds by starting with the most general bounds, which are based on the methods of Section 6.4, and concludes with bounds for simpler motion planning problems.

General algorithms The first upper bound for the general motion planning problem of Formulation 4.1 came from the application of cylindrical algebraic decomposition [428]. Let n be the dimension of $\mathcal{C}$. Let m be the number of polynomials in $\mathcal{F}$, which are used to define $\mathcal{C}_{obs}$. Recall from Section 4.3.3 how quickly this grows for simple examples. Let d be the maximum degree among the polynomials in $\mathcal{F}$. The maximum degree of the resulting polynomials is bounded by $O(d^{2^{n-1}})$, and the total number of polynomials is bounded by $O((md)^{3^{n-1}})$. The total running time required to use cylindrical algebraic decomposition for motion planning is bounded by $(md)^{O(1)^n}$.[10] Note that the algorithm is doubly exponential in dimension n but polynomial in m and d. It can theoretically be declared to be *efficient* on a space of motion planning problems of bounded dimension (although, it certainly is not efficient for motion planning in any practical sense).

Since the general problem is PSPACE-complete, it appears unavoidable that a complete, general motion planning algorithm will require a running time that is exponential in dimension. Since cylindrical algebraic decomposition is doubly exponential, it led many in the 1980s to wonder whether this upper bound could be lowered. This was achieved by Canny's roadmap algorithm, for which the running time is bounded by $m^n(\lg m)d^{O(n^4)}$. Hence, it is singly exponential, which appears very close to optimal because it is up against the lower bound that seems to be implied by PSPACE-hardness (and the fact that problems exist

[10] It may seem odd for $O(\cdot)$ to appear in the middle of an expression. In this context, it means that there exists some $c \in [0, \infty)$ such that the running time is bounded by $(md)^{c^n}$. Note that another O is not necessary in front of the whole formula.

that require a roadmap with $(md)^n$ connected components [53]). Much of the algorithm's complexity is due to finding a suitable deterministic perturbation to put the input polynomials into general position. A randomized algorithm can alternatively be used, for which the randomized expected running time is bounded by $m^n(\lg m)d^{O(n^2)}$. For a *randomized algorithm* [375], the *randomized expected* running time is still a worst-case upper bound, but averaged over random "coin tosses" that are introduced internally in the algorithm; it does *not* reflect any kind of average over the expected input distribution. Thus, these two bounds represent the best-known upper bounds for the general motion planning problem. Canny's algorithm may also be applied to solve the kinematic closure problems of Section 4.4, but the complexity does not reflect the fact that the dimension, k, of the algebraic variety is less than n, the dimension of $\mathcal{C}$. A roadmap algorithm that is particularly suited for this problem is introduced in [52], and its running time is bounded by $m^{k+1}d^{O(n^2)}$. This serves as the best-known upper bound for the problems of Section 4.4.

Specialized algorithms Now upper bounds are summarized for some narrower problems, which can be solved more efficiently than the general problem. All of the problems involve either two or three degrees of freedom. Therefore, we expect that the bounds are much lower than those for the general problem. In many cases, the Davenport-Schinzel sequences of Section 6.5.2 arise. Most of the bounds presented here are based on algorithms that are not practical to implement; they mainly serve to indicate the best asymptotic performance that can be obtained for a problem. Most of the bounds mentioned here are included in [435].

Consider the problem from Section 6.2, in which the robot translates in $\mathcal{W} = \mathbb{R}^2$ and $\mathcal{C}_{obs}$ is polygonal. Suppose that $\mathcal{A}$ is a convex polygon that has k edges and $\mathcal{O}$ is the union of m disjoint, convex polygons with disjoint interiors, and their total number of edges is n. In this case, the boundary of $\mathcal{C}_{free}$ (computed by Minkowski difference; see Section 4.3.2) has at most $6m-12$ nonreflex vertices (interior angles less than π) and $n + km$ reflex vertices (interior angles greater than π). The free space, $\mathcal{C}_{free}$, can be decomposed and searched in time $O((n + km)\lg^2 n)$ [267, 435]. Using randomized algorithms, the bound reduces to $O((n + km) \cdot 2^{\alpha(n)} \lg n)$ randomized expected time. Now suppose that $\mathcal{A}$ is a single nonconvex polygonal region described by k edges and that $\mathcal{O}$ is a similar polygonal region described by n edges. The Minkowski difference could yield as many as $\Omega(k^2n^2)$ edges for $\mathcal{C}_{obs}$. This can be avoided if the search is performed within a single connected component of $\mathcal{C}_{free}$. Based on analysis that uses Davenport-Schinzel sequences, it can be shown that the worst connected component may have complexity $\Theta(kn\alpha(k))$, and the planning problem can be solved in time $O(kn\lg^2 n)$ deterministically or for a randomized algorithm, $O(kn \cdot 2^{\alpha(n)} \lg n)$ randomized expected time is needed. More generally, if $\mathcal{C}_{obs}$ consists of n algebraic curves in $\mathbb{R}^2$, each with degree no more than d, then the motion planning problem for translation only can be solved deterministically in time $O(\lambda_{s+2}(n)\lg^2 n)$, or with a randomized algorithm

in $O(\lambda_{s+2}(n) \lg n)$ randomized expected time. In these expressions, $\lambda_{s+2}(n)$ is the bound (6.37) obtained from the $(n, s+2)$ Davenport-Schinzel sequence, and $s \leq d^2$.

For the case of the line-segment robot of Section 6.3.4 in an obstacle region described with n edges, an $O(n^5)$-time algorithm was given. This is not the best possible running time for solving the line-segment problem, but the method is easier to understand than others that are more efficient. In [386], a roadmap algorithm based on retraction is given that solves the problem in $O(n^2 \lg n \lg^* n)$ time, in which $\lg^* n$ is the number of times that lg has to be iterated on n to yield a result less than or equal to 1 (i.e., it is a very small, insignificant term; for practical purposes, you can imagine that the running time is $O(n^2 \lg n)$). The tightest known upper bound is $O(n^2 \lg n)$ [319]. It is established in [266] that there exist examples for which the solution path requires $\Omega(n^2)$ length to encode. For the case of a line segment moving in $\mathbb{R}^3$ among polyhedral obstacles with a total of n vertices, a complete algorithm that runs in time $O(n^4 + \epsilon)$ for any $\epsilon > 0$ was given in [288]. In [266] it was established that solution paths of complexity $\Omega(n^4)$ exist.

Now consider the case for which $\mathcal{C} = SE(2)$, $\mathcal{A}$ is a convex polygon with k edges, and $\mathcal{O}$ is a polygonal region described by n edges. The boundary of $\mathcal{C}_{free}$ has no more than $O(kn\lambda_6(kn))$ edges and can be computed to solve the motion planning problem in time $O(kn\lambda_6(kn) \lg kn)$ [8, 9]. An algorithm that runs in time $O(k^4 n \lambda_3(n) \lg n)$ and provides better clearance between the robot and obstacles is given in [110]. In [41] (some details also appear in [304]), an algorithm is presented, and even implemented, that solves the more general case in which $\mathcal{A}$ is nonconvex in time $O(k^3 n^3 \lg(kn))$. The number of faces of $\mathcal{C}_{obs}$ could be as high as $\Omega(k^3 n^3)$ for this problem. By explicitly representing and searching only one connected component, the best-known upper bound for the problem is $O((kn)^{2+\epsilon})$, in which $\epsilon > 0$ may be chosen arbitrarily small [218].

In the final case, suppose that $\mathcal{A}$ translates in $\mathcal{W} = \mathbb{R}^3$ to yield $\mathcal{C} = \mathbb{R}^3$. For a polyhedron or polyhedral region, let its *complexity* be the total number of faces, edges, and vertices. If $\mathcal{A}$ is a polyhedron with complexity k, and $\mathcal{O}$ is a polyhedral region with complexity n, then the boundary of $\mathcal{C}_{free}$ is a polyhedral surface of complexity $\Theta(k^3 n^3)$. As for other problems, if the search is restricted to a single component, then the complexity is reduced. The motion planning problem in this case can be solved in time $O((kn)^{2+\epsilon})$ [33]. If $\mathcal{A}$ is convex and there are m convex obstacles, then the best-known bound is $O(kmn \lg^2 m)$ time. More generally, if $\mathcal{C}_{obs}$ is bounded by n algebraic patches of constant maximum degree, then a vertical decomposition method solves the motion planning problem within a single connected component of $\mathcal{C}_{free}$ in time $O(n^{2+\epsilon})$.

Further Reading

Most of the literature on combinatorial planning is considerably older than the sampling-based planning literature. A nice collection of early papers appears in [430]; this includes

[237, 386, 387, 409, 427, 428, 429]. The classic motion planning textbook of Latombe [304] covers most of the methods presented in this chapter. The coverage here does not follow [304], which makes separate categories for cell decomposition methods and roadmap methods. A cell decomposition is constructed to produce a roadmap; hence, they are unified in this chapter. An excellent reference for material in combinatorial algorithms, computational geometry, and complete algorithms for motion planning is the collection of survey papers in [206].

Section 6.2 follows the spirit of basic algorithms from computational geometry. For a gentle introduction to computational geometry, including a nice explanation of vertical composition, see [146]. Other sources for computational geometry include [71, 158, 405]. To understand the difficulties in computing optimal decompositions of polygons, see [390]. See [338, 372, 418] for further reading on computing shortest paths.

Cell decompositions and cell complexes are very important in computational geometry and algebraic topology. Section 6.3 provided a brief perspective that was tailored to motion planning. For simplicial complexes in algebraic topology, see [256, 281, 419]; for singular complexes, see [419]. In computational geometry, various kinds of cell decompositions arise. Some of the most widely studied decompositions are *triangulations* [57] and *arrangements* [216], which are regions generated by a collection of primitives, such as lines or circles in the plane. For early cell decomposition methods in motion planning, see [430]. A survey of computational topology appears in [468].

The most modern and complete reference for the material in Section 6.4 is [53]. A gentle introduction to computational algebraic geometry is given in [138]. For details regarding algebraic computations with polynomials, see [369]. A survey of computational algebraic geometry appears in [370]. In addition to [53], other general references to cylindrical algebraic decomposition are [31, 124]. For its use in motion planning, see [304, 428]. The main reference for Canny's roadmap algorithm is [92]. Alternative high-level overviews to the one presented in Section 6.4.3 appear in [118, 304]. Variations and improvements to the algorithm are covered in [53]. A potential function-based extension of Canny's roadmap algorithm is developed in [93].

For further reading on the complexity of motion planning, consult the numerous references given in Section 6.5.

Exercises

1. Extend the vertical decomposition algorithm to correctly handle the case in which $\mathcal{C}_{obs}$ has two or more points that lie on the same vertical line. This includes the case of vertical segments. Random perturbations are not allowed.

2. Fully describe and prove the correctness of the bitangent computation method shown in Figure 6.14, which avoids trigonometric functions. Make certain that all types of bitangents (in general position) are considered.

3. Develop an algorithm that uses the plane-sweep principle to efficiently compute a representation of the union of two nonconvex polygons.

4. Extend the vertical cell decomposition algorithm of Section 6.2.2 to work for obstacle boundaries that are described as chains of circular arcs and line segments.

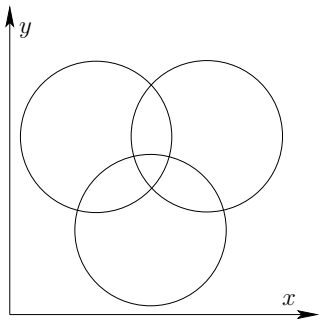


Figure 6.43: Determine the cylindrical algebraic decomposition obtained by projecting onto the x-axis.

5. Extend the shortest-path roadmap algorithm of Section 6.2.4 to work for obstacle boundaries that are described as chains of circular arcs and line segments.

6. Derive the equation for the Conchoid of Nicomedes, shown in Figure 6.24, for the case of a line-segment robot contacting an obstacle vertex and edge simultaneously.

7. Propose a resolution-complete algorithm for motion planning of the line-segment robot in a polygonal obstacle region. The algorithm should compute exact C-space obstacle slices for any fixed orientation, θ; however, the algorithm should use van der Corput sampling over the set $[0, 2\pi)$ of orientations.

8. Determine the result of cylindrical algebraic decomposition for unit spheres $\mathbb{S}^1$, $\mathbb{S}^2$, $\mathbb{S}^3$, $\mathbb{S}^4$, ... Each $\mathbb{S}^n$ is expressed as a unit sphere in $\mathbb{R}^{n+1}$. Graphically depict the cases of $\mathbb{S}^1$ and $\mathbb{S}^2$. Also, attempt to develop an expression for the number of cells as a function of n.

9. Determine the cylindrical algebraic decomposition for the three intersecting circles shown in Figure 6.43. How many cells are obtained?

10. Using the matrix in (6.28), show that the result of Canny's roadmap for the torus, shown in Figure 6.39, is correct. Use the torus equation

$$(x_1^2 + x_2^2 + x_3^2 - (r_1^2 + r_2^2))^2 - 4r_1^2(r_2^2 - x_3^2) = 0, \tag{6.40}$$

in which r_1 is the major circle, r_2 is the minor circle, and $r_1 > r_2$.

11. Propose a vertical decomposition algorithm for a polygonal robot that can translate in the plane and even continuously vary its scale. How would the algorithm be modified to instead work for a robot that can translate or be sheared?

12. Develop a shortest-path roadmap algorithm for a flat torus, defined by identifying opposite edges of a square. Use Euclidean distance but respect the identifications when determining the shortest path. Assume the robot is a point and the obstacles are polygonal.

Implementations

13. Implement the vertical cell decomposition planning algorithm of Section 6.2.2.

14. Implement the maximum-clearance roadmap planning algorithm of Section 6.2.3.

15. Implement a planning algorithm for a point robot that moves in $\mathcal{W} = \mathbb{R}^3$ among polyhedral obstacles. Use vertical decomposition.

16. Implement an algorithm that performs a cylindrical decomposition of a polygonal obstacle region.

17. Implement an algorithm that computes the cell decomposition of Section 6.3.4 for the line-segment robot.

18. Experiment with cylindrical algebraic decomposition. The project can be greatly facilitated by utilizing existing packages for performing basic operations in computational algebraic geometry.

19. Implement the algorithm proposed in Exercise 7.

Chapter 7

Extensions of Basic Motion Planning

This chapter presents many extensions and variations of the motion planning problem considered in Chapters 3 to 6. Each one of these can be considered as a "spin-off" that is fairly straightforward to describe using the mathematical concepts and algorithms introduced so far. Unlike the previous chapters, there is not much continuity in Chapter 7. Each problem is treated independently; therefore, it is safe to jump to whatever sections in the chapter you find interesting without fear of missing important details.

In many places throughout the chapter, a state space X will arise. This is consistent with the general planning notation used throughout the book. In Chapter 4, the C-space, $\mathcal{C}$, was introduced, which can be considered as a special state space: It encodes the set of transformations that can be applied to a collection of bodies. Hence, Chapters 5 and 6 addressed planning in $X = \mathcal{C}$. The C-space alone is insufficient for many of the problems in this chapter; therefore, X will be used because it appears to be more general. For most cases in this chapter, however, X is derived from one or more C-spaces. Thus, C-space and state space terminology will be used in combination.

7.1 Time-Varying Problems

This section brings time into the motion planning formulation. Although the robot has been allowed to move, it has been assumed so far that the obstacle region $\mathcal{O}$ and the goal configuration, $q_G \in \mathcal{C}_{free}$, are stationary for all time. It is now assumed that these entities may vary over time, although their motions are predictable. If the motions are not predictable, then some form of feedback is needed to respond to observations that are made during execution. Such problems are much more difficult and will be handled in Chapters 8 and throughout Part IV.

7.1.1 Problem Formulation

The formulation is designed to allow the tools and concepts learned so far to be applied directly. Let $T \subset \mathbb{R}$ denote the *time interval*, which may be *bounded* or *unbounded*. If T is bounded, then $T = [0, t_f]$, in which 0 is the *initial time* and t_f is the *final time*. If T is unbounded, then $T = [0, \infty)$. An initial time other than 0 could alternatively be defined without difficulty, but this will not be done here.

Let the state space X be defined as $X = \mathcal{C} \times T$, in which $\mathcal{C}$ is the usual C-space of the robot, as defined in Chapter 4. A state x is represented as $x = (q, t)$, to indicate the configuration q and time t components of the state vector. The planning will occur directly in X, and in many ways it can be treated as any C-space seen to far, but there is one critical difference: *Time marches forward.* Imagine a path that travels through X. If it first reaches a state $(q_1, 5)$, and then later some state $(q_2, 3)$, some traveling backward through time is required! There is no mathematical problem with allowing such time travel, but it is not realistic for most applications. Therefore, paths in X are forced to follow a constraint that they must move forward in time.

Now consider making the following time-varying versions of the items used in Formulation 4.1 for motion planning.

Formulation 7.1 (The Time-Varying Motion Planning Problem)

1. A *world* $\mathcal{W}$ in which either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. This is the same as in Formulation 4.1.

2. A *time interval* $T \subset \mathbb{R}$ that is either *bounded* to yield $T = [0, t_f]$ for some *final time*, $t_f > 0$, or *unbounded* to yield $T = [0, \infty)$.

3. A semi-algebraic, time-varying *obstacle region* $\mathcal{O}(t) \subset \mathcal{W}$ for every $t \in T$. It is assumed that the obstacle region is a finite collection of rigid bodies that undergoes continuous, time-dependent rigid-body transformations.

4. The *robot* $\mathcal{A}$ (or $\mathcal{A}_1, \ldots, \mathcal{A}_m$ for a linkage) and *configuration space* $\mathcal{C}$ definitions are the same as in Formulation 4.1.

5. The *state space* X is the Cartesian product $X = \mathcal{C} \times T$ and a state $x \in X$ is denoted as $x = (q, t)$ to denote the configuration q and time t components. See Figure 7.1. The obstacle region, X_{obs}, in the state space is defined as

$$X_{obs} = \{(q, t) \in X \mid \mathcal{A}(q) \cap \mathcal{O}(t) \neq \emptyset\}, \tag{7.1}$$

and $X_{free} = X \setminus X_{obs}$. For a given $t \in T$, slices of X_{obs} and X_{free} are obtained. These are denoted as $\mathcal{C}_{obs}(t)$ and $\mathcal{C}_{free}(t)$, respectively, in which (assuming $\mathcal{A}$ is one body)

$$\mathcal{C}_{obs}(t) = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O}(t) \neq \emptyset\} \tag{7.2}$$

and $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.

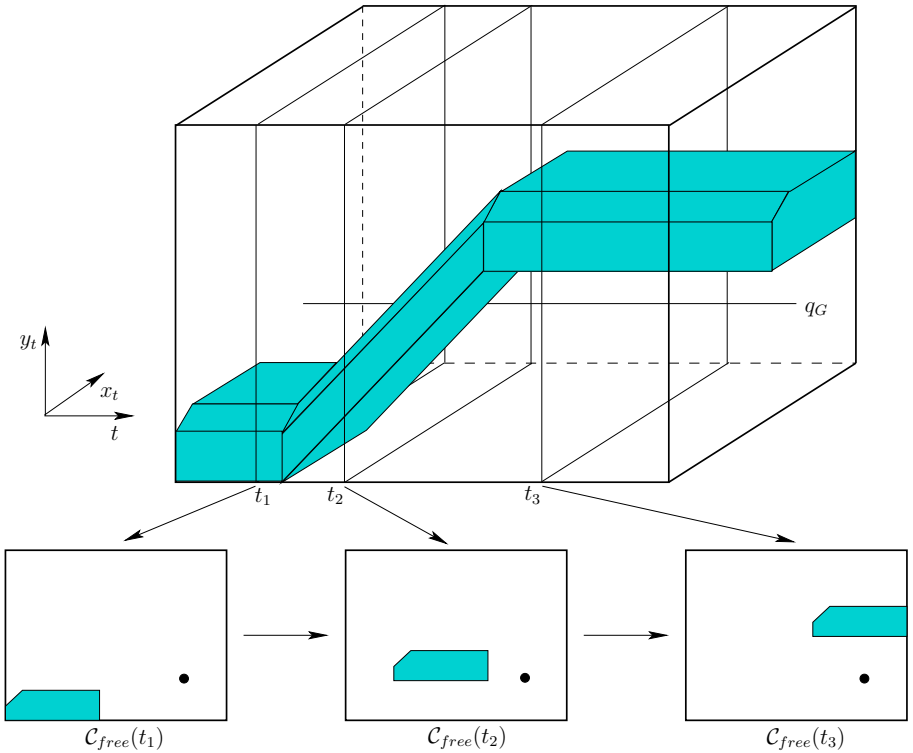


Figure 7.1: A time-varying example with piecewise-linear obstacle motion.

6. A state $x_I \in X_{free}$ is designated as the *initial state*, with the constraint that $x_I = (q_I, 0)$ for some $q_I \in \mathcal{C}_{free}(0)$. In other words, at the initial time the robot cannot be in collision.

7. A subset $X_G \subset X_{free}$ is designated as the *goal region*. A typical definition is to pick some $q_G \in \mathcal{C}$ and let $X_G = \{(q_G, t) \in X_{free} \mid t \in T\}$, which means that the goal is *stationary* for all time.

8. A complete algorithm must compute a continuous, time-monotonic *path*, $\tau[0,1] \rightarrow X_{free}$, such that $\tau(0) = x_I$ and $\tau(1) \in X_G$, or correctly report that such a path does not exist. To be *time-monotonic* implies that for any $s_1, s_2 \in [0,1]$ such that $s_1 < s_2$, we have $t_1 < t_2$, in which $(q_1, t_1) = \tau(s_1)$ and $(q_2, t_2) = \tau(s_2)$.

Example 7.1 (Piecewise-Linear Obstacle Motion) Figure 7.1 shows an example of a convex, polygonal robot $\mathcal{A}$ that translates in $\mathcal{W} = \mathbb{R}^2$. There is a single, convex, polygonal obstacle $\mathcal{O}$. The two of these together yield a convex, polygonal C-space obstacle, $\mathcal{C}_{obs}(t)$, which is shown for times t_1, t_2, and t_3. The obstacle moves with a *piecewise-linear motion model*, which means that transformations applied to $\mathcal{O}$ are a piecewise-linear function of time. For example, let (x, y) be a fixed point on the obstacle. To be a linear motion model, this point must transform as $(x + c_1 t, y + c_2 t)$ for some constants $c_1, c_2 \in \mathbb{R}$. To be piecewise-linear, it may change to a different linear motion at a finite number of critical times. Between these critical times, the motion must remain linear. There are two critical times in the example. If $\mathcal{C}_{obs}(t)$ is polygonal, and a piecewise-linear motion model is used, then X_{obs} is polyhedral, as depicted in Figure 7.1. A stationary goal is also shown, which appears as a line that is parallel to the T-axis. ■

In the general formulation, there are no additional constraints on the path, τ, which means that the robot motion model allows infinite acceleration and unbounded speed. The robot velocity may change instantaneously, but the path through $\mathcal{C}$ must always be continuous. These issues did not arise in Chapter 4 because there was no need to mention time. Now it becomes necessary.[1]

7.1.2 Direct Solutions

Sampling-based methods Many sampling-based methods can be adapted from $\mathcal{C}$ to X without much difficulty. The time dependency of obstacle models must be taken into account when verifying that path segments are collision-free; the techniques from Section 5.3.4 can be extended to handle this. One important concern is the metric for X. For some algorithms, it may be important to permit the use of a pseudometric because symmetry is broken by time (going backward in time is not as easy as going forward).

For example, suppose that the C-space $\mathcal{C}$ is a metric space, $(\mathcal{C}, \rho)$. The metric can be extended across time to obtain a pseudometric, ρ_X, as follows. For a pair of states, $x = (q, t)$ and $x' = (q', t')$, let

$$\rho_X(x, x') = \begin{cases} 0 & \text{if } q = q' \\ \infty & \text{if } q \neq q' \text{ and } t' \leq t \\ \rho(q, q') & \text{otherwise.} \end{cases} \tag{7.3}$$

Using ρ_X, several sampling-based methods naturally work. For example, RDTs from Section 5.5 can be adapted to X. Using ρ_X for a single-tree approach ensures that all path segments travel forward in time. Using bidirectional approaches

[1] The infinite acceleration and unbounded speed assumptions may annoy those with mechanics and control backgrounds. In this case, assume that the present models approximate the case in which every body moves slowly, and the dynamics can be consequently neglected. If this is still not satisfying, then jump ahead to Part IV, where general nonlinear systems are considered. It is still helpful to consider the implications derived from the concepts in this chapter because the issues remain for more complicated problems that involve dynamics.

is more difficult for time-varying problems because X_G is usually not a single point. It is not clear which (q, t) should be the starting vertex for the tree from the goal; one possibility is to initialize the goal tree to an entire time-invariant segment. The sampling-based roadmap methods of Section 5.6 are perhaps the most straightforward to adapt. The notion of a *directed roadmap* is needed, in which every edge must be directed to yield a time-monotonic path. For each pair of states, (q, t) and (q', t'), such that $t \neq t'$, exactly one valid direction exists for making a potential edge. If $t = t'$, then no edge can be attempted because it would require the robot to instantaneously "teleport" from one part of $\mathcal{W}$ to another. Since forward time progress is already taken into account by the directed edges, a symmetric metric may be preferable instead of (7.3) for the sampling-based roadmap approach.

Combinatorial methods In some cases, combinatorial methods can be used to solve time-varying problems. If the motion model is *algebraic* (i.e., expressed with polynomials), then X_{obs} is semi-algebraic. This enables the application of general planners from Section 6.4, which are based on computational real algebraic geometry. The key issue once again is that the resulting roadmap must be directed with all edges being time-monotonic. For Canny's roadmap algorithm, this requirement seems difficult to ensure. Cylindrical algebraic decomposition is straightforward to adapt, provided that time is chosen as the last variable to be considered in the sequence of projections. This yields polynomials in $\mathbb{Q}[t]$, and $\mathbb{R}$ is nicely partitioned into time intervals and time instances. Connections can then be made for a cell of one cylinder to an adjacent cell of a cylinder that occurs later in time.

If X_{obs} is polyhedral as depicted in Figure 7.1, then vertical decomposition can be used. It is best to first sweep the plane along the time axis, stopping at the critical times when the linear motion changes. This yields nice sections, which are further decomposed recursively, as explained in Section 6.3.3, and also facilitates the connection of adjacent cells to obtain time-monotonic path segments. It is not too difficult to imagine the approach working for a four-dimensional state space, X, for which $\mathcal{C}_{obs}(t)$ is polyhedral as in Section 6.3.3, and time adds the fourth dimension. Again, performing the first sweep with respect to the time axis is preferable.

If X is not decomposed into cylindrical slices over each noncritical time interval, then cell decompositions may still be used, but be careful to correctly connect the cells. Figure 7.2 illustrates the problem, for which transitivity among adjacent cells is broken. This complicates sample point selection for the cells.

Bounded speed There has been no consideration so far of the speed at which the robot must move to avoid obstacles. It is obviously impractical in many applications if the solution requires the robot to move arbitrarily fast. One step toward making a realistic model is to enforce a bound on the speed of the robot.

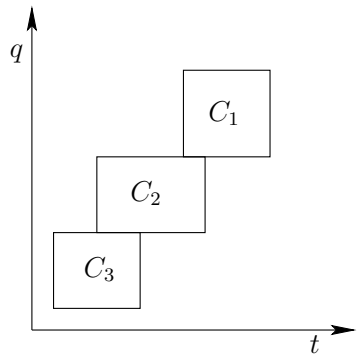


Figure 7.2: Transitivity is broken if the cells are not formed in cylinders over T. A time-monotonic path exists from C_1 to C_2, and from C_2 to C_3, but this does not imply that one exists from C_1 to C_3.

(More steps towards realism are taken in Chapter 13.) For simplicity, suppose $\mathcal{C} = \mathbb{R}^2$, which corresponds to a translating rigid robot, $\mathcal{A}$, that moves in $\mathcal{W} = \mathbb{R}^2$. A configuration, $q \in \mathcal{C}$, is represented as $q = (y, z)$ (since x already refers to the whole state vector). The *robot velocity* is expressed as $v = (\dot{y}, \dot{z}) \in \mathbb{R}^2$, in which $\dot{y} = dy/dt$ and $\dot{z} = dz/dt$. The *robot speed* is $\|v\| = \sqrt{\dot{y}^2 + \dot{z}^2}$. A *speed bound*, b, is a positive constant, $b \in (0, \infty)$, for which $\|v\| \leq b$.

In terms of Figure 7.1, this means that the slope of a solution path τ is bounded. Suppose that the domain of τ is $T = [0, t_f]$ instead of $[0, 1]$. This yields $\tau : T \rightarrow X$ and $\tau(t) = (y, z, t)$. Using this representation, $d\tau_1/dt = \dot{y}$ and $d\tau_2/dt = \dot{z}$, in which τ_i denotes the ith component of τ (because it is a vector-valued function). Thus, it can seen that b constrains the slope of $\tau(t)$ in X. To visualize this, imagine that only motion in the y direction occurs, and suppose $b = 1$. If τ holds the robot fixed, then the speed is zero, which satisfies any bound. If the robot moves at speed 1, then $d\tau_1/dt = 1$ and $d\tau_2/dt = 0$, which satisfies the speed bound. In Figure 7.1 this generates a path that has slope 1 in the yt plane and is horizontal in the zt plane. If $d\tau_1/dt = d\tau_2/dt = 1$, then the bound is exceeded because the speed is $\sqrt{2}$. In general, the velocity vector at any state (y, z, t) points into a cone that starts at (y, z) and is aligned in the positive t direction; this is depicted in Figure 7.3. At time $t + \Delta t$, the state must stay within the cone, which means that

$$\big(y(t + \Delta t) - y(t)\big)^2 + \big(z(t + \Delta t) - z(t)\big)^2 \leq b^2 (\Delta t)^2. \tag{7.4}$$

This constraint makes it considerably more difficult to adapt the algorithms of Chapters 5 and 6. Even for piecewise-linear motions of the obstacles, the problem has been established to be PSPACE-hard [410, 411, 457]. A complete algorithm is presented in [411] that is similar to the shortest-path roadmap algorithm of Section 6.2.4. The sampling-based roadmap of Section 5.6 is perhaps one of the

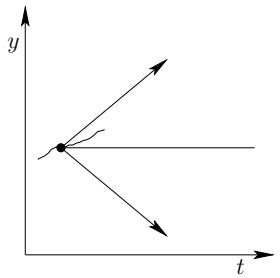


Figure 7.3: A projection of the cone constraint for the bounded-speed problem.

easiest of the sampling-based algorithms to adapt for this problem. The neighbors of point q, which are determined for attempted connections, must lie within the cone that represents the speed bound. If this constraint is enforced, a resolution complete or probabilistically complete planning algorithm results.

7.1.3 The Velocity-Tuning Method

An alternative to defining the problem in $\mathcal{C} \times T$ is to decouple it into a *path planning* part and a *motion timing* part [260]. Algorithms based on this method are not complete, but velocity tuning is an important idea that can be applied elsewhere. Suppose there are both *stationary obstacles* and *moving obstacles*. For the stationary obstacles, suppose that some path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ has been computed using any of the techniques described in Chapters 5 and 6.

The timing part is then handled in a second phase. Design a *timing function* (or *time scaling*), $\sigma : T \rightarrow [0,1]$, that indicates for time, t, the location of the robot along the path, τ. This is achieved by defining the composition $\phi = \tau \circ \sigma$, which maps from T to $\mathcal{C}_{free}$ via $[0,1]$. Thus, $\phi : T \rightarrow \mathcal{C}_{free}$. The configuration at time $t \in T$ is expressed as $\phi(t) = \tau(\sigma(t))$.

A 2D state space can be defined as shown in Figure 7.4. The purpose is to convert the design of σ (and consequently ϕ) into a familiar planning problem. The robot must move along its path from $\tau(0)$ to $\tau(1)$ while an obstacle, $\mathcal{O}(t)$, moves along its path over the time interval T. Let $S = [0,1]$ denote the domain of τ. A state space, $X = T \times S$, is shown in Figure 7.4b, in which each point (t, s) indicates the time $t \in T$ and the position along the path, $s \in [0,1]$. The obstacle region in X is defined as

$$X_{obs} = \{(t,s) \in X \mid \mathcal{A}(\tau(s)) \cap \mathcal{O}(t) \neq \emptyset\}. \tag{7.5}$$

Once again, X_{free} is defined as $X_{free} = X \setminus X_{obs}$. The task is to find a continuous path $g : [0,1] \rightarrow X_{free}$. If g is time-monotonic, then a position $s \in S$ is assigned for every time, $t \in T$. These assignments can be nicely organized into the timing function, $\sigma : T \rightarrow S$, from which ϕ is obtained by $\phi = \tau \circ \sigma$ to determine where

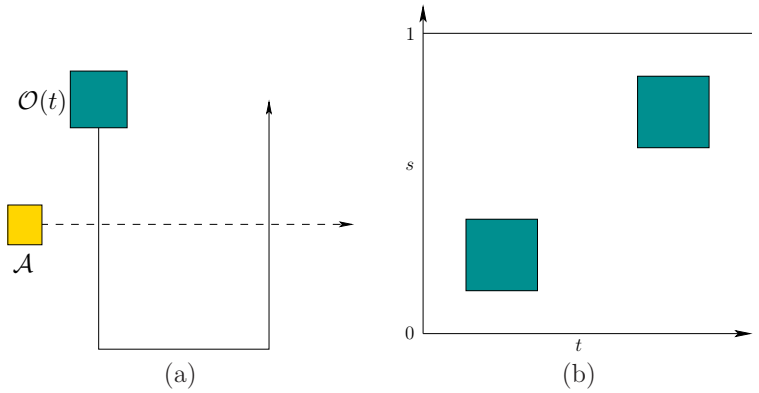


Figure 7.4: An illustration of path tuning. (a) If the robot follows its computed path, it may collide with the moving obstacle. (b) The resulting state space.

the robot will be at each time. Being time-monotonic in this context means that the path must always progress from left to right in Figure 7.4b. It can, however, be nonmonotonic in the positive s direction. This corresponds to moving back and forth along τ, causing some configurations to be revisited.

Any of the methods described in Formulation 7.1 can be applied here. The dimension of X in this case is always 2. Note that X_{obs} is polygonal if $\mathcal{A}$ and $\mathcal{O}$ are both polygonal regions and their paths are piecewise-linear. In this case, the vertical decomposition method of Section 6.2.2 can be applied by sweeping along the time axis to yield a complete algorithm (it is complete after having committed to τ, but it is not complete for Formulation 7.1). The result is shown in Figure 7.5. The cells are connected only if it is possible to reach one from the other by traveling in the forward time direction. As an example of a sampling-based approach that may be preferable when X_{obs} is not polygonal, place a grid over X and apply one of the classical search algorithms described in Section 5.4.2. Once again, only path segments in X that move forward in time are allowed.

7.2 Multiple Robots

Suppose that multiple robots share the same world, $\mathcal{W}$. A path must be computed for each robot that avoids collisions with obstacles and with other robots. In Chapter 4, each robot could be a rigid body, $\mathcal{A}$, or it could be made of k attached bodies, $\mathcal{A}_1, \ldots, \mathcal{A}_k$. To avoid confusion, superscripts will be used in this section to denote different robots. The ith robot will be denoted by $\mathcal{A}^i$. Suppose there are m robots, $\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^m$. Each robot, $\mathcal{A}^i$, has its associated C-space, $\mathcal{C}^i$, and its initial and goal configurations, q^i_{init} and q^i_{goal}, respectively.

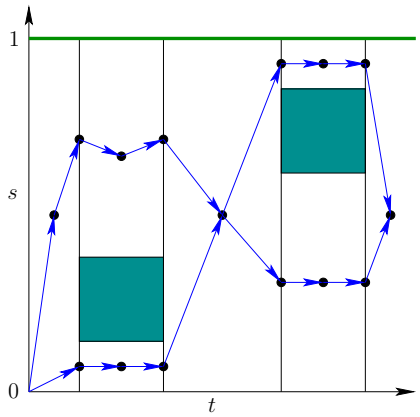


Figure 7.5: Vertical cell decomposition can solve the path tuning problem. Note that this example is not in general position because vertical edges exist. The goal is to reach the horizontal line at the top, which can be accomplished from any adjacent 2-cell. For this example, it may even be accomplished from the first 2-cell if the robot is able to move quickly enough.

7.2.1 Problem Formulation

A state space is defined that considers the configurations of all robots simultaneously,

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m. \tag{7.6}$$

A state $x \in X$ specifies all robot configurations and may be expressed as $x = (q^1, q^2, \ldots, q^m)$. The dimension of X is N, which is $N = \sum_{i=1}^{m} \dim(\mathcal{C}^i)$.

There are two sources of obstacle regions in the state space: 1) *robot-obstacle* collisions, and 2) *robot-robot* collisions. For each i such that $1 \leq i \leq m$, the subset of X that corresponds to robot $\mathcal{A}^i$ in collision with the obstacle region, $\mathcal{O}$, is

$$X^i_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{O} \neq \emptyset\}. \tag{7.7}$$

This only models the robot-obstacle collisions.

For each pair, $\mathcal{A}^i$ and $\mathcal{A}^j$, of robots, the subset of X that corresponds to $\mathcal{A}^i$ in collision with $\mathcal{A}^j$ is

$$X^{ij}_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{A}^j(q^j) \neq \emptyset\}. \tag{7.8}$$

Both (7.7) and (7.8) will be combined in (7.10) later to yield X_{obs}.

Formulation 7.2 (Multiple-Robot Motion Planning)

1. The *world* $\mathcal{W}$ and *obstacle region* $\mathcal{O}$ are the same as in Formulation 4.1.

2. There are m *robots*, $\mathcal{A}^1$, …, $\mathcal{A}^m$, each of which may consist of one or more bodies.

3. Each robot $\mathcal{A}^i$, for i from 1 to m, has an associated *configuration space*, $\mathcal{C}^i$.

4. The *state space* X is defined as the Cartesian product

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m. \tag{7.9}$$

   The obstacle region in X is

$$X_{obs} = \left(\bigcup_{i=1}^{m} X^i_{obs} \right) \bigcup \left(\bigcup_{ij,\ i \neq j} X^{ij}_{obs} \right), \tag{7.10}$$

   in which X^i_{obs} and X^{ij}_{obs} are the robot-obstacle and robot-robot collision states from (7.7) and (7.8), respectively.

5. A state $x_I \in X_{free}$ is designated as the *initial state*, in which $x_I = (q^1_I, \ldots, q^m_I)$. For each i such that $1 \leq i \leq m$, q^i_I specifies the initial configuration of $\mathcal{A}^i$.

6. A state $x_G \in X_{free}$ is designated as the *goal state*, in which $x_G = (q^1_G, \ldots, q^m_G)$.

7. The task is to compute a continuous path $\tau : [0, 1] \rightarrow X_{free}$ such that $\tau(0) = x_{init}$ and $\tau(1) \in x_{goal}$.

An ordinary motion planning problem? On the surface it may appear that there is nothing unusual about the multiple-robot problem because the formulations used in Chapter 4 already cover the case in which the robot consists of multiple bodies. They do not have to be attached; therefore, X can be considered as an ordinary C-space. The planning algorithms of Chapters 5 and 6 may be applied without adaptation. The main concern, however, is that the dimension of X grows linearly with respect to the number of robots. For example, if there are 12 rigid bodies for which each has $\mathcal{C}^i = SE(3)$, then the dimension of X is $6 \cdot 12 = 72$. Complete algorithms require time that is at least exponential in dimension, which makes them unlikely candidates for such problems. Sampling-based algorithms are more likely to scale well in practice when there many robots, but the dimension of X might still be too high.

Reasons to study multi-robot motion planning Even though multiple-robot motion planning can be handled like any other motion planning problem, there are several reasons to study it separately:

1. The motions of the robots can be decoupled in many interesting ways. This leads to several interesting methods that first develop some kind of partial plan for the robots independently, and then consider the plan interactions to produce a solution. This idea is referred to as *decoupled planning*.

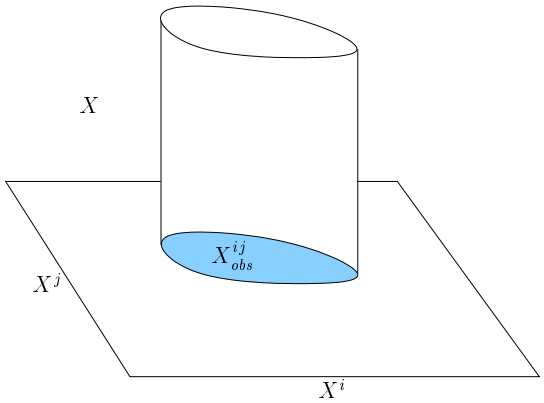


Figure 7.6: The set X^{ij}_{obs} and its cylindrical structure on X.

2. The part of X_{obs} due to robot-robot collisions has a cylindrical structure, depicted in Figure 7.6, which can be exploited to make more efficient planning algorithms. Each X^{ij}_{obs} defined by (7.8) depends only on two robots. A point, $x = (q^1, \ldots, q^m)$, is in X_{obs} if there exists i, j such that $1 \leq i, j \leq m$ and $\mathcal{A}^i(q^i) \cap \mathcal{A}^j(q^j) \neq \emptyset$, regardless of the configurations of the other $m - 2$ robots. For some decoupled methods, this even implies that X_{obs} can be completely characterized by 2D projections, as depicted in Figure 7.9.

3. If optimality is important, then a unique set of issues arises for the case of multiple robots. It is not a standard optimization problem because the performance of each robot has to be optimized. There is no clear way to combine these objectives into a single optimization problem without losing some critical information. It will be explained in Section 7.7.2 that Pareto optimality naturally arises as the appropriate notion of optimality for multiple-robot motion planning.

Assembly planning One important variant of multiple-robot motion planning is called *assembly planning*; recall from Section 1.2 its importance in applications. In automated manufacturing, many complicated objects are assembled step-by-step from individual parts. It is convenient for robots to manipulate the parts one-by-one to insert them into the proper locations (see Section 7.3.2). Imagine a collection of parts, each of which is interpreted as a robot, as shown in Figure 7.7a. The goal is to assemble the parts into one coherent object, such as that shown in Figure 7.7b. The problem is generally approached by starting with the goal configuration, which is tightly constrained, and working outward. The problem formulation may allow that the parts touch, but their interiors cannot overlap. In general, the assembly planning problem with arbitrarily many parts

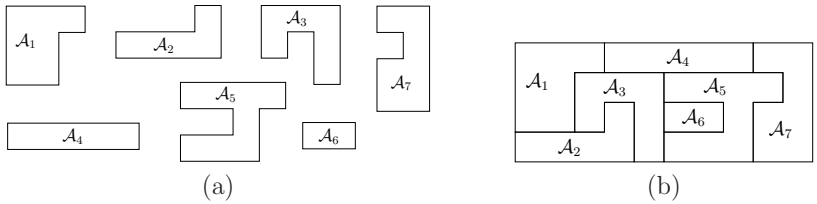


Figure 7.7: (a) A collection of pieces used to define an assembly planning problem; (b) assembly planning involves determining a sequence of motions that assembles the parts. The object shown here is assembled from the parts.

is NP-hard. Polynomial-time algorithms have been developed in several special cases. For the case in which parts can be removed by a sequence of straight-line paths, a polynomial-time algorithm is given in [476, 477].

7.2.2 Decoupled planning

Decoupled approaches first design motions for the robots while ignoring robot-robot interactions. Once these interactions are considered, the choices available to each robot are already constrained by the designed motions. If a problem arises, these approaches are typically unable to reverse their commitments. Therefore, completeness is lost. Nevertheless, decoupled approaches are quite practical, and in some cases completeness can be recovered.

Prioritized planning A straightforward approach to decoupled planning is to sort the robots by priority and plan for higher priority robots first [166, 466]. Lower priority robots plan by viewing the higher priority robots as moving obstacles. Suppose the robots are sorted as $\mathcal{A}^1, \ldots, \mathcal{A}^m$, in which $\mathcal{A}^1$ has the highest priority.

Assume that collision-free paths, $\tau_i : [0, 1] \rightarrow \mathcal{C}^i_{free}$, have been computed for i from 1 to n. The prioritized planning approach proceeds inductively as follows:

Base case: Use any motion planning algorithm from Chapters 5 and 6 to compute a collision-free path, $\tau_1 : [0, 1] \rightarrow \mathcal{C}^1_{free}$ for $\mathcal{A}^1$. Compute a timing function, σ_1, for τ_1, to yield $\phi_1 = \tau_1 \circ \sigma_1 : T \rightarrow \mathcal{C}^1_{free}$.

Inductive step: Suppose that $\phi_1, \ldots, \phi_{i-1}$ have been designed for $\mathcal{A}^1, \ldots, \mathcal{A}^{i-1}$, and that these functions avoid robot-robot collisions between any of the first $i - 1$ robots. Formulate the first $i - 1$ robots as moving obstacles in $\mathcal{W}$. For each $t \in T$ and $j \in \{1, \ldots, i - 1\}$, the configuration q^j of each $\mathcal{A}^j$ is $\phi_j(t)$. This yields $\mathcal{A}^j(\phi_j(t)) \subset \mathcal{W}$, which can be considered as a subset of the obstacle $\mathcal{O}(t)$. Design a path, τ_i, and timing function, σ_i, using any of the time-varying motion planning methods from Section 7.1 and form $\phi_i = \tau_i \circ \sigma_i$.

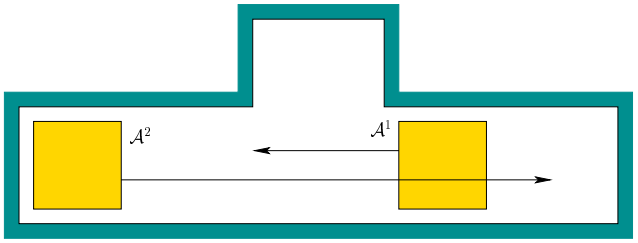


Figure 7.8: If $\mathcal{A}^1$ neglects the query for $\mathcal{A}^2$, then completeness is lost when using the prioritized planning approach. This example has a solution in general, but prioritized planning fails to find it.

Although practical in many circumstances, Figure 7.8 illustrates how completeness is lost.

A special case of prioritized planning is to design all of the paths, τ_1, τ_2, ..., τ_m, in the first phase and then formulate each inductive step as a velocity tuning problem. This yields a sequence of 2D planning problems that can be solved easily. This comes at a greater expense, however, because the choices are even more constrained. The idea of preplanned paths, and even roadmaps, for all robots independently can lead to a powerful method if the coordination of the robots is approached more carefully. This is the next topic.

Fixed-path coordination Suppose that each robot $\mathcal{A}^i$ is constrained to follow a path $\tau_i : [0,1] \rightarrow \mathcal{C}^i_{free}$, which can be computed using any ordinary motion planning technique. For m robots, an m-dimensional state space called a *coordination space* is defined that schedules the motions of the robots along their paths so that they will not collide [385]. One important feature is that time will only be *implicitly* represented in the coordination space. An algorithm must compute a path in the coordination space, from which explicit timings can be easily extracted.

For m robots, the *coordination space* X is defined as the m-dimensional unit cube $X = [0,1]^m$. Figure 7.9 depicts an example for which $m = 3$. The ith coordinate of X represents the domain, $S_i = [0,1]$, of the path τ_i. Let s_i denote a point in S_i (it is also the ith component of x). A state, $x \in X$, indicates the configuration of every robot. For each i, the configuration $q^i \in \mathcal{C}^i$ is given by $q^i = \tau_i(s_i)$. At state $(0, \ldots, 0) \in X$, every robot is in its initial configuration, $q^i_I = \tau_i(0)$, and at state $(1, \ldots, 1) \in X$, every robot is in its goal configuration, $q^i_G = \tau_i(1)$. Any continuous path, $h : [0,1] \rightarrow X$, for which $h(0) = (0, \ldots, 0)$ and $h(1) = (1, \ldots, 1)$, moves the robots to their goal configurations. The path h does not even need to be monotonic, in contrast to prioritized planning.

One important concern has been neglected so far. What prevents us from designing h as a straight-line path between the opposite corners of $[0,1]^m$? We have not yet taken into account the collisions between the robots. This forms an obstacle region X_{obs} that must be avoided when designing a path through X.

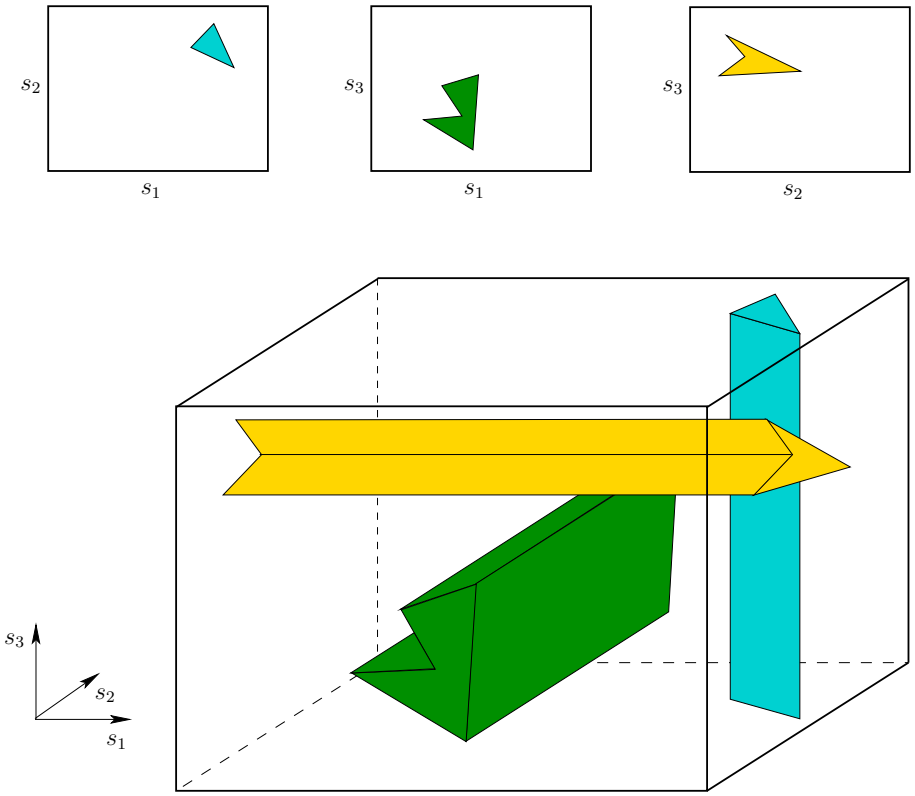


Figure 7.9: The obstacles that arise from coordinating m robots are always cylindrical. The set of all $\frac{1}{2}m(m-1)$ axis-aligned 2D projections completely characterizes X_{obs}.

Thus, the task is to design $h : [0,1] \rightarrow X_{free}$, in which $X_{free} = X \setminus X_{obs}$.

The definition of X_{obs} is very similar to (7.8) and (7.10), except that here the state-space dimension is much smaller. Each q^i is replaced by a single parameter. The cylindrical structure, however, is still retained, as shown in Figure 7.9. Each cylinder of X_{obs} is

$$X^{ij}_{obs} = \{(s_1, \ldots, s_m) \in X \mid \mathcal{A}^i(\tau_i(s_i)) \cap \mathcal{A}^j(\tau_j(s_j)) \neq \emptyset\}, \tag{7.11}$$

which are combined to yield

$$X_{obs} = \bigcup_{ij,\ i \neq j} X^{ij}_{obs}. \tag{7.12}$$

Standard motion planning algorithms can be applied to the coordination space because there is no monotonicity requirement on h. If 1) $\mathcal{W} = \mathbb{R}^2$, 2) $m = 2$

(two robots), 3) the obstacles and robots are polygonal, and 4) the paths, τ_i, are piecewise-linear, then X_{obs} is a polygonal region in X. This enables the methods of Section 6.2, for a polygonal $\mathcal{C}_{obs}$, to directly apply after the representation of X_{obs} is explicitly constructed. For $m > 2$, the multi-dimensional version of vertical cell decomposition given for $m = 3$ in Section 6.3.3 can be applied. For general coordination problems, cylindrical algebraic decomposition or Canny's roadmap algorithm can be applied. For the problem of robots in $\mathcal{W} = \mathbb{R}^2$ that either translate or move along circular paths, a resolution complete planning method based on the exact determination of X_{obs} using special collision detection methods is given in [441].

For very challenging coordination problems, sampling-based solutions may yield practical solutions. Perhaps one of the simplest solutions is to place a grid over X and adapt the classical search algorithms, as described in Section 5.4.2 [309, 385]. Other possibilities include using the RDTs of Section 5.5 or, if the multiple-query framework is appropriate, then the sampling-based roadmap methods of 5.6 are suitable. Methods for validating the path segments, which were covered in Section 5.3.4, can be adapted without trouble to the case of coordination spaces.

Thus far, the particular speeds of the robots have been neglected. For explanation purposes, consider the case of $m = 2$. Moving vertically or horizontally in X holds one robot fixed while the other moves at some maximum speed. Moving diagonally in X moves both robots, and their relative speeds depend on the slope of the path. To carefully regulate these speeds, it may be necessary to reparameterize the paths by distance. In this case each axis of X represents the distance traveled, instead of $[0, 1]$.

Fixed-roadmap coordination The fixed-path coordination approach still may not solve the problem in Figure 7.8 if the paths are designed independently. Fortunately, fixed-path coordination can be extended to enable each robot to move over a roadmap or topological graph. This still yields a coordination space that has only one dimension per robot, and the resulting planning methods are much closer to being complete, assuming each robot utilizes a roadmap that has many alternative paths. There is also motivation to study this problem by itself because of automated guided vehicles (AGVs), which often move in factories on a network of predetermined paths. In this case, coordinating the robots *is* the planning problem, as opposed to being a simplification of Formulation 7.2.

One way to obtain completeness for Formulation 7.2 is to design the independent roadmaps so that each robot has its own *garage* configuration. The conditions for a configuration, q^i, to be a *garage* for $\mathcal{A}^i$ are 1) while at configuration q^i, it is impossible for any other robots to collide with it (i.e., in all coordination states for which the ith coordinate is q^i, no collision occurs); and 2) q^i is always reachable by $\mathcal{A}^i$ from x_I. If each robot has a roadmap and a garage, and if the planning method for X is complete, then the overall planning algorithm is complete. If the

planning method in X uses some weaker notion of completeness, then this is also maintained. For example, a resolution complete planner for X yields a resolution complete approach to the problem in Formulation 7.2.

Cube complex How is the coordination space represented when there are multiple paths for each robot? It turns out that a *cube complex* is obtained, which is a special kind of singular complex (recall from Section 6.3.1). The coordination space for m fixed paths can be considered as a singular m-simplex. For example, the problem in Figure 7.9 can be considered as a singular 3-simplex, $[0, 1]^3 \rightarrow X$. In Section 6.3.1, the domain of a k-simplex was defined using B^k, a k-dimensional ball; however, a cube, $[0, 1]^k$, also works because B^k and $[0, 1]^k$ are homeomorphic.

For a topological space, X, let a *k-cube* (which is also a singular k-simplex), $\square_k$, be a continuous mapping $\sigma : [0, 1]^k \rightarrow X$. A cube complex is obtained by connecting together k-cubes of different dimensions. Every k-cube for $k \geq 1$ has $2k$ *faces*, which are $(k - 1)$-cubes that are obtained as follows. Let $(s_1, \ldots, s_k)$ denote a point in $[0, 1]^k$. For each $i \in \{1, \ldots, k\}$, one face is obtained by setting $s_i = 0$ and another is obtained by setting $s_i = 1$.

The cubes must fit together nicely, much in the same way that the simplexes of a simplicial complex were required to fit together. To be a *cube complex*, $\mathcal{K}$ must be a collection of simplexes that satisfy the following requirements:

1. Any face, $\square_{k-1}$, of a cube $\square_k \in \mathcal{K}$ is also in $\mathcal{K}$.

2. The intersection of the images of any two k-cubes $\square_k, \square'_k \in \mathcal{K}$, is either empty or there exists some cube, $\square_i \in \mathcal{K}$ for $i < k$, which is a common face of both $\square_k$ and $\square'_k$.

Let $\mathcal{G}_i$ denote a topological graph (which may also be a roadmap) for robot $\mathcal{A}^i$. The graph edges are paths of the form $\tau : [0, 1] \rightarrow \mathcal{C}^i_{free}$. Before covering formal definitions of the resulting complex, consider Figure 7.10a, in which $\mathcal{A}^1$ moves along three paths connected in a "T" junction and $\mathcal{A}^2$ moves along one path. In this case, three 2D fixed-path coordination spaces are attached together along one common edge, as shown in Figure 7.10b. The resulting cube complex is defined by three 2-cubes (i.e., squares), one 1-cube (i.e., line segment), and eight 0-cubes (i.e., corner points).

Now suppose more generally that there are two robots, $\mathcal{A}^1$ and $\mathcal{A}^2$, with associated topological graphs, $\mathcal{G}_1(V_1, E_1)$ and $\mathcal{G}_2(V_2, E_2)$, respectively. Suppose that $\mathcal{G}$ and $\mathcal{G}_2$ have n_1 and n_2 edges, respectively. A 2D cube complex, $\mathcal{K}$, is obtained as follows. Let τ_i denote the ith path of $\mathcal{G}_1$, and let σ_j denote the jth path of $\mathcal{G}_2$. A 2-cube (square) exists in $\mathcal{K}$ for every way to select an edge from each graph. Thus, there are $n_1 n_2$ 2-cubes, one for each pair (τ_i, σ_j), such that $\tau_i \in E_1$ and $\sigma_j \in E_2$. The 1-cubes are generated for pairs of the form (v_i, σ_j) for $v_i \in V_1$ and $\sigma_j \in E_2$, or (τ_i, v_j) for $\tau_i \in E_1$ and $v_j \in V_2$. The 0-cubes (corner points) are reached for each pair (v_i, v_j) such that $v_i \in V_1$ and $v_j \in V_2$.

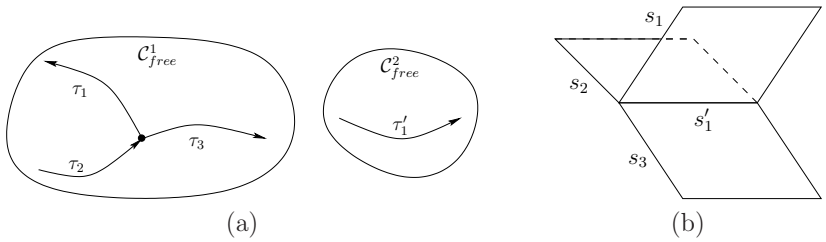


Figure 7.10: (a) An example in which $\mathcal{A}^1$ moves along three paths, and $\mathcal{A}^2$ moves along one. (b) The corresponding coordination space.

If there are m robots, then an m-dimensional cube complex arises. Every m-cube corresponds to a unique combination of paths, one for each robot. The $(m-1)$-cubes are the faces of the m-cubes. This continues iteratively until the 0-cubes are reached.

Planning on the cube complex Once again, any of the planning methods described in Chapters 5 and 6 can be adapted here, but the methods are slightly complicated by the fact that X is a complex. To use sampling-based methods, a dense sequence should be generated over X. For example, if random sampling is used, then an m-cube can be chosen at random, followed by a random point in the cube. The local planning method (LPM) must take into account the connectivity of the cube complex, which requires recognizing when branches occur in the topological graph. Combinatorial methods must also take into account this connectivity. For example, a sweeping technique can be applied to produce a vertical cell decomposition, but the sweep-line (or sweep-plane) must sweep across the various m-cells of the complex.

7.3 Mixing Discrete and Continuous Spaces

Many important applications involve a mixture of discrete and continuous variables. This results in a state space that is a Cartesian product of the C-space and a finite set called the *mode space*. The resulting space can be visualized as having layers of C-spaces that are indexed by the modes, as depicted in Figure 7.11. The main application given in this section is manipulation planning; many others exist, especially when other complications such as dynamics and uncertainties are added to the problem. The framework of this section is inspired mainly from *hybrid systems* in the control theory community [210], which usually model mode-dependent dynamics. The main concern in this section is that the allowable robot configurations and/or the obstacles depend on the mode.

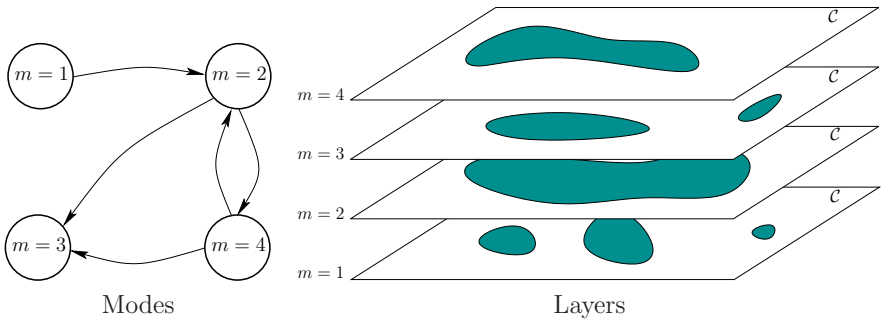


Figure 7.11: A hybrid state space can be imagined as having layers of C-spaces that are indexed by modes.

7.3.1 Hybrid Systems Framework

As illustrated in Figure 7.11, a hybrid system involves interaction between discrete and continuous spaces. The formal model will first be given, followed by some explanation. This formulation can be considered as a combination of the components from discrete feasible planning, Formulation 2.1, and basic motion planning, Formulation 4.1.

Formulation 7.3 (Hybrid-System Motion Planning)

1. The $\mathcal{W}$ and $\mathcal{C}$ components from Formulation 4.1 are included.

2. A nonempty *mode space*, M that is a finite or countably infinite set of *modes*.

3. A semi-algebraic *obstacle region* $\mathcal{O}(m)$ for each $m \in M$.

4. A semi-algebraic *robot* $\mathcal{A}(m)$, for each $m \in M$. It may be a rigid robot or a collection of links. It is assumed that the C-space is not mode-dependent; only the geometry of the robot can depend on the mode. The robot, transformed to configuration q, is denoted as $\mathcal{A}(q, m)$.

5. A *state space* X is defined as the Cartesian product $X = \mathcal{C} \times M$. A state is represented as $x = (q, m)$, in which $q \in \mathcal{C}$ and $m \in M$. Let

$$X_{obs} = \{(q, m) \in X \mid \mathcal{A}(q, m) \cap \mathcal{O}(m) \neq \emptyset\}, \tag{7.13}$$

   and $X_{free} = X \setminus X_{obs}$.

6. For each state, $x \in X$, there is a finite *action space*, $U(x)$. Let U denote the set of all possible actions (the union of $U(x)$ over all $x \in X$).

7. There is a *mode transition function* f_m that produces a mode, $f_m(x, u) \in M$, for every $x \in X$ and $u \in U(x)$. It is assumed that f_m is defined in a way that does not produce race conditions (oscillations of modes within an instant of time). This means that if q is fixed, the mode can change at most once. It then remains constant and can change only if q is changed.

8. There is a *state transition function*, f, that is derived from f_m by changing the mode and holding the configuration fixed. Thus, $f(x, u) = (q, f_m(x, u))$.

9. A configuration $x_I \in X_{free}$ is designated as the *initial state*.

10. A set $X_G \in X_{free}$ is designated as the *goal region*. A region is defined instead of a point to facilitate the specification of a goal configuration that does not depend on the final mode.

11. An algorithm must compute a (continuous) *path* $\tau : [0, 1] \rightarrow X_{free}$ and an *action trajectory* $\sigma : [0, 1] \rightarrow U$ such that $\tau(0) = x_I$ and $\tau(1) \in X_G$, or the algorithm correctly reports that such a combination of a path and an action trajectory does not exist.

The obstacle region and robot may or may not be mode-dependent, depending on the problem. Examples of each will be given shortly. Changes in the mode depend on the action taken by the robot. From most states, it is usually assumed that a "do nothing" action exists, which leaves the mode unchanged. From certain states, the robot may select an action that changes the mode as desired. An interesting degenerate case exists in which there is only a single action available. This means that the robot has no control over the mode from that state. If the robot arrives in such a state, a mode change could unavoidably occur.

The solution requirement is somewhat more complicated because both a path and an action trajectory need to be specified. It is insufficient to specify a path because it is important to know what action was applied to induce the correct mode transitions. Therefore, σ indicates when these occur. Note that τ and σ are closely coupled; one cannot simply associate any σ with a path τ; it must correspond to the actions required to generate τ.

Example 7.2 (The Power of the Portiernia) In this example, a robot, $\mathcal{A}$, is modeled as a square that translates in $\mathcal{W} = \mathbb{R}^2$. Therefore, $\mathcal{C} = \mathbb{R}^2$. The obstacle region in $\mathcal{W}$ is mode-dependent because of two doors, which are numbered "1" and "2" in Figure 7.12a. In the upper left sits the *portiernia*,[2] which is able to give a key to the robot, if the robot is in a configuration as shown in Figure 7.12b. The portiernia only trusts the robot with one key at a time, which may be either for Door 1 or Door 2. The robot can return a key by revisiting the portiernia. As shown in Figures 7.12c and 7.12d, the robot can open a door by making contact with it, as long as it holds the correct key.

[2]This is a place where people guard the keys at some public facilities in Poland.

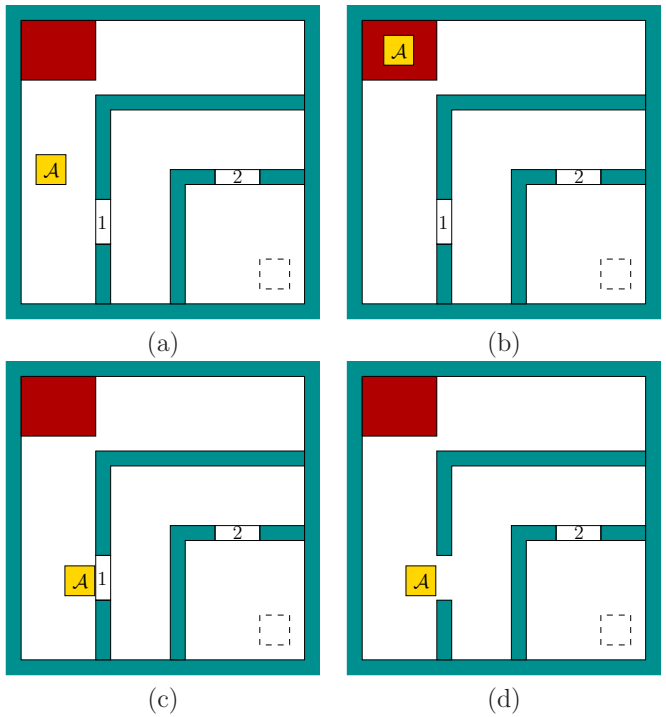


Figure 7.12: In the upper left (at the portiernia), the robot can pick up and drop off keys that open one of two doors. If the robot contacts a door while holding the correct key, then it opens.

The set, M, of modes needs to encode which key, if any, the robot holds, and it must also encode the status of the doors. The robot may have: 1) the key to Door 1; 2) the key to Door 2; or 3) no keys. The doors may have the status: 1) both open; 2) Door 1 open, Door 2 closed; 3) Door 1 closed, Door 2 open; or 4) both closed. Considering keys and doors in combination yields 12 possible modes.

If the robot is at a portiernia configuration as shown in Figure 7.12b, then its available actions correspond to different ways to pick up and drop off keys. For example, if the robot is holding the key to Door 1, it can drop it off and pick up the key to Door 2. This changes the mode, but the door status and robot configuration must remain unchanged when f is applied. The other locations in which the robot may change the mode are when it comes in contact with Door 1 or Door 2. The mode changes only if the robot is holding the proper key. In all other configurations, the robot only has a single action (i.e., no choice), which keeps the mode fixed.

The task is to reach the configuration shown in the lower right with dashed

lines. The problem is solved by: 1) picking up the key for Door 1 at the portiernia; 2) opening Door 1; 3) swapping the key at the portiernia to obtain the key for Door 2; or 4) entering the innermost room to reach the goal configuration. As a final condition, we might want to require that the robot returns the key to the portiernia. ■

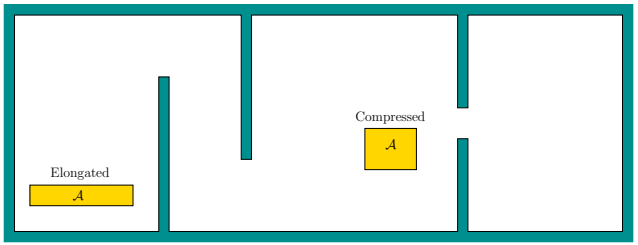
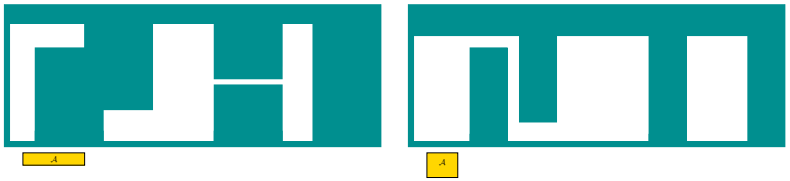


Figure 7.13: An example in which the robot must reconfigure itself to solve the problem. There are two modes: *elongated* and *compressed*.



Elongated mode Compressed mode

Figure 7.14: When the robot reconfigures itself, $\mathcal{C}_{free}(m)$ changes, enabling the problem to be solved.

Example 7.2 allows the robot to change the obstacles in $\mathcal{O}$. The next example involves a robot that can change its shape. This is an illustrative example of a *reconfigurable robot*. The study of such robots has become a popular topic of research [112, 196, 290, 484]; the reconfiguration possibilities in that research area are much more complicated than the simple example considered here.

Example 7.3 (Reconfigurable Robot) To solve the problem shown in Figure 7.13, the robot must change its shape. There are two possible shapes, which correspond directly to the modes: *elongated* and *compressed*. Examples of each are shown in the figure. Figure 7.14 shows how $\mathcal{C}_{free}(m)$ appears for each of the two modes. Suppose the robot starts initially from the left while in the elongated mode and must travel to the last room on the right. This problem must be solved by 1) reconfiguring the robot into the compressed mode; 2) passing through the corridor into the center; 3) reconfiguring the robot into the elongated mode; and 4) passing through the corridor to the rightmost room. The robot has actions that directly change the mode by reconfiguring itself. To make the problem more interesting, we could require the robot to reconfigure itself in specific locations (e.g., where there is enough clearance, or possibly at a location where another robot can assist it).

The examples presented so far barely scratch the surface on the possible hybrid motion planning problems that can be defined. Many such problems can arise, for example, in the context making automated video game characters or digital actors. To solve these problems, standard motion planning algorithms can be adapted if they are given information about how to change the modes. Locations in X from which the mode can be changed may be expressed as subgoals. Much of the planning effort should then be focused on attempting to change modes, in addition to trying to directly reach the goal. Applying sampling-based methods requires the definition of a metric on X that accounts for both changes in the mode and the configuration. A wide variety of hybrid problems can be formulated, ranging from those that are impossible to solve in practice to those that are straightforward extensions of standard motion planning. In general, the hybrid motion planning model is useful for formulating a hierarchical approach, as described in Section 1.4. One particularly interesting class of problems that fit this model, for which successful algorithms have been developed, will be covered next.

7.3.2 Manipulation Planning

This section presents an overview of manipulation planning; the concepts explained here are mainly due to [11, 12]. Returning to Example 7.2, imagine that the robot must carry a key that is so large that it changes the connectivity of $\mathcal{C}_{free}$. For the manipulation planning problem, the robot is called a *manipulator*, which interacts with a *part*. In some configurations it is able to *grasp* the part and move it to other locations in the environment. The *manipulation task* usually requires moving the part to a specified location in $\mathcal{W}$, without particular regard as to how the *manipulator* can accomplish the task. The model considered here greatly simplifies the problems of grasping, stability, friction, mechanics, and uncertainties and instead focuses on the geometric aspects (some of these issues will be addressed in Section 12.5). For a thorough introduction to these other important aspects of manipulation planning, see [352]; see also Sections 13.1.3 and 12.5.

Admissible configurations Assume that $\mathcal{W}$, $\mathcal{O}$, and $\mathcal{A}$ from Formulation 4.1 are used. For manipulation planning, $\mathcal{A}$ is called the *manipulator*, and let $\mathcal{C}^a$ refer to the *manipulator configuration space*. Let $\mathcal{P}$ denote a *part*, which is a rigid body modeled in terms of geometric primitives, as described in Section 3.1. It is assumed that $\mathcal{P}$ is allowed to undergo rigid-body transformations and will

therefore have its own *part configuration space*, $\mathcal{C}^p = SE(2)$ or $\mathcal{C}^p = SE(3)$. Let $q^p \in \mathcal{C}^p$ denote a *part configuration*. The transformed part model is denoted as $\mathcal{P}(q^p)$.

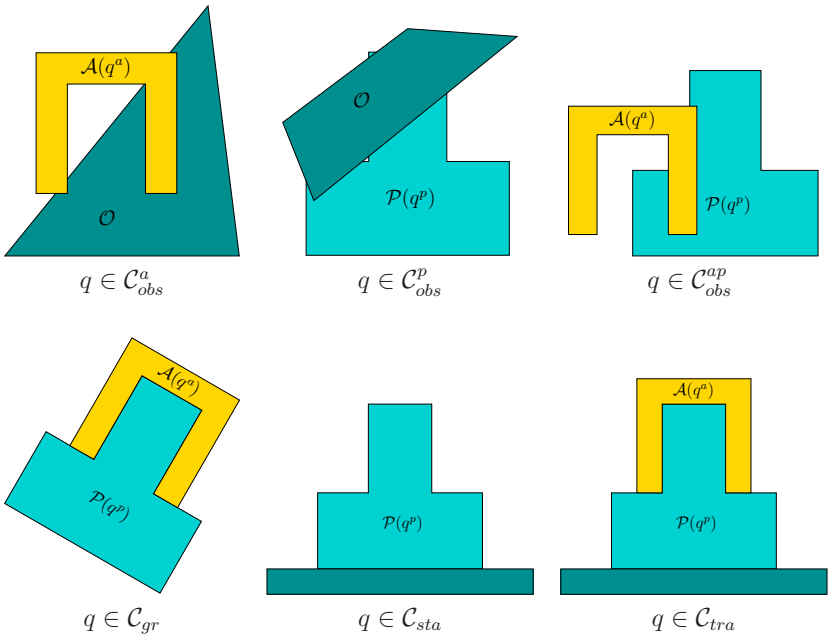


Figure 7.15: Examples of several important subsets of $\mathcal{C}$ for manipulation planning.

The combined *configuration space*, $\mathcal{C}$, is defined as the Cartesian product

$$\mathcal{C} = \mathcal{C}^a \times \mathcal{C}^p, \tag{7.14}$$

in which each configuration $q \in \mathcal{C}$ is of the form $q = (q^a, q^p)$. The first step is to remove all configurations that must be avoided. Parts of Figure 7.15 show examples of these sets. Configurations for which the manipulator collides with obstacles are

$$\mathcal{C}^a_{obs} = \{(q^a, q^p) \in \mathcal{C} \mid \mathcal{A}(q^a) \cap \mathcal{O} \neq \emptyset\}. \tag{7.15}$$

The next logical step is to remove configurations for which the part collides with obstacles. It will make sense to allow the part to "touch" the obstacles. For example, this could model a part sitting on a table. Therefore, let

$$\mathcal{C}^p_{obs} = \{(q^a, q^p) \in \mathcal{C} \mid \text{int}(\mathcal{P}(q^p)) \cap \mathcal{O} \neq \emptyset\} \tag{7.16}$$

denote the open set for which the interior of the part intersects $\mathcal{O}$. Certainly, if the part penetrates $\mathcal{O}$, then the configuration should be avoided.

Consider $\mathcal{C} \setminus (\mathcal{C}^a_{obs} \cup \mathcal{C}^p_{obs})$. The configurations that remain ensure that the robot and part do not inappropriately collide with $\mathcal{O}$. Next consider the interaction between $\mathcal{A}$ and $\mathcal{P}$. The manipulator must be allowed to touch the part, but penetration is once again not allowed. Therefore, let

$$\mathcal{C}^{ap}_{obs} = \{(q^a, q^p) \in \mathcal{C} \mid \mathcal{A}(q^a) \cap \text{int}(\mathcal{P}(q^p)) \neq \emptyset\}. \tag{7.17}$$

Removing all of these bad configurations yields

$$\mathcal{C}_{adm} = \mathcal{C} \setminus (\mathcal{C}^a_{obs} \cup \mathcal{C}^p_{obs} \cup \mathcal{C}^{ap}_{obs}), \tag{7.18}$$

which is called the set of *admissible configurations*.

Stable and grasped configurations Two important subsets of $\mathcal{C}_{adm}$ are used in the manipulation planning problem. See Figure 7.15. Let $\mathcal{C}^p_{sta}$ denote the set of *stable part configurations*, which are configurations at which the part can safely rest without any forces being applied by the manipulator. This means that a part cannot, for example, float in the air. It also cannot be in a configuration from which it might fall. The particular stable configurations depend on properties such as the part geometry, friction, mass distribution, and so on. These issues are not considered here. From this, let $\mathcal{C}_{sta} \subseteq \mathcal{C}_{adm}$ be the corresponding *stable configurations*, defined as

$$\mathcal{C}_{sta} = \{(q^a, q^p) \in \mathcal{C}_{adm} \mid q^p \in \mathcal{C}^p_{sta}\}. \tag{7.19}$$

The other important subset of $\mathcal{C}_{adm}$ is the set of all configurations in which the robot is grasping the part (and is capable of carrying it, if necessary). Let this denote the *grasped configurations*, denoted by $\mathcal{C}_{gr} \subseteq \mathcal{C}_{adm}$. For every configuration, $(q^a, q^p) \in \mathcal{C}_{gr}$, the manipulator touches the part. This means that $\mathcal{A}(q^a) \cap \mathcal{P}(q^p) \neq \emptyset$ (penetration is still not allowed because $\mathcal{C}_{gr} \subseteq \mathcal{C}_{adm}$). In general, many configurations at which $\mathcal{A}(q^a)$ contacts $\mathcal{P}(q^p)$ will not necessarily be in $\mathcal{C}_{gr}$. The conditions for a point to lie in $\mathcal{C}_{gr}$ depend on the particular characteristics of the manipulator, the part, and the contact surface between them. For example, a typical manipulator would not be able to pick up a block by making contact with only one corner of it. This level of detail is not defined here; see [352] for more information about grasping.

We must always ensure that either $x \in \mathcal{C}_{sta}$ or $x \in \mathcal{C}_{gr}$. Therefore, let $\mathcal{C}_{free} = \mathcal{C}_{sta} \cup \mathcal{C}_{gr}$, to reflect the subset of $\mathcal{C}_{adm}$ that is permissible for manipulation planning.

The mode space, M, contains two modes, which are named the *transit mode* and the *transfer mode*. In the transit mode, the manipulator is not carrying the part, which requires that $q \in \mathcal{C}_{sta}$. In the transfer mode, the manipulator carries the part, which requires that $q \in \mathcal{C}_{gr}$. Based on these simple conditions, the only way the mode can change is if $q \in \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$. Therefore, the manipulator has two available actions only when it is in these configurations. In all other configurations

the mode remains unchanged. For convenience, let $\mathcal{C}_{tra} = \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$ denote the set of *transition configurations*, which are the places in which the mode may change.

Using the framework of Section 7.3.1, the mode space, M, and C-space, $\mathcal{C}$, are combined to yield the *state space*, $X = \mathcal{C} \times M$. Since there are only two modes, there are only two copies of $\mathcal{C}$, one for each mode. State-based sets, X_{free}, X_{tra}, X_{sta}, and X_{gr}, are directly obtained from $\mathcal{C}_{free}$, $\mathcal{C}_{tra}$, $\mathcal{C}_{sta}$, and $\mathcal{C}_{gr}$ by ignoring the mode. For example,

$$X_{tra} = \{(q, m) \in X \mid q \in \mathcal{C}_{tra}\}. \tag{7.20}$$

The sets X_{free}, X_{sta}, and X_{gr} are similarly defined.

The task can now be defined. An *initial part configuration*, $q^p_{init} \in \mathcal{C}_{sta}$, and a *goal part configuration*, $q^p_{goal} \in \mathcal{C}_{sta}$, are specified. Compute a path $\tau : [0, 1] \rightarrow X_{free}$ such that $\tau(0) = q^p_{init}$ and $\tau(1) = q^p_{goal}$. Furthermore, the *action trajectory* $\sigma : [0, 1] \rightarrow U$ must be specified to indicate the appropriate mode changes whenever $\tau(s) \in X_{tra}$. A solution can be considered as an alternating sequence of *transit paths* and *transfer paths*, whose names follow from the mode. This is depicted in Figure 7.16.

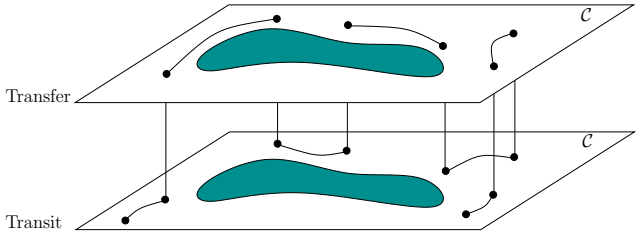


Figure 7.16: The solution to a manipulation planning problem alternates between the two layers of X. The transitions can only occur when $x \in X_{tra}$.

Manipulation graph The manipulation planning problem generally can be solved by forming a manipulation graph, $\mathcal{G}_m$ [11, 12]. Let a *connected component* of X_{tra} refer to any connected component of $\mathcal{C}_{tra}$ that is lifted into the state space by ignoring the mode. There are two copies of the connected component of $\mathcal{C}_{tra}$, one for each mode. For each connected component of X_{tra}, a vertex exists in $\mathcal{G}_m$. An edge is defined for each transfer path or transit path that connects two connected components of X_{tra}. The general approach to manipulation planning then is as follows:

1. Compute the connected components of X_{tra} to yield the vertices of $\mathcal{G}_m$.

2. Compute the edges of $\mathcal{G}_m$ by applying ordinary motion planning methods to each pair of vertices of $\mathcal{G}_m$.

3. Apply motion planning methods to connect the initial and goal states to every possible vertex of X_{tra} that can be reached without a mode transition.

4. Search $\mathcal{G}_m$ for a path that connects the initial and goal states. If one exists, then extract the corresponding solution as a sequence of transit and transfer paths (this yields σ, the actions that cause the required mode changes).

This can be considered as an example of hierarchical planning, as described in Section 1.4.

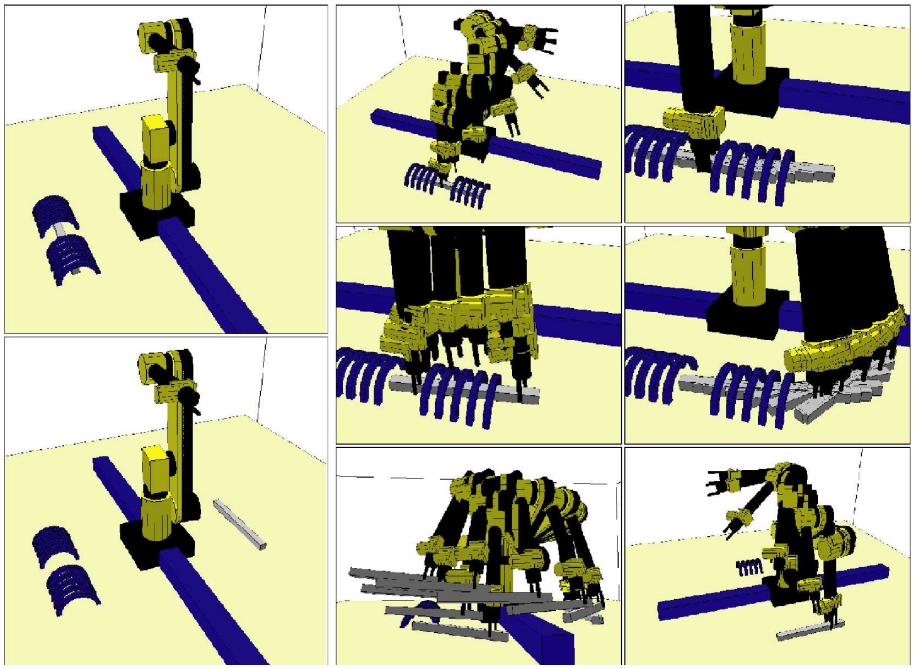


Figure 7.17: This example was solved in [133] using the manipulation planning framework and the visibility-based roadmap planner. It is very challenging because the same part must be regrasped in many places.

Multiple parts The manipulation planning framework nicely generalizes to multiple parts, $\mathcal{P}_1$, ..., $\mathcal{P}_k$. Each part has its own C-space, and $\mathcal{C}$ is formed by taking the Cartesian product of all part C-spaces with the manipulator C-space. The set $\mathcal{C}_{adm}$ is defined in a similar way, but now part-part collisions also have to be removed, in addition to part-manipulator, manipulator-obstacle, and part-obstacle collisions. The definition of $\mathcal{C}_{sta}$ requires that all parts be in stable configurations; the parts may even be allowed to stack on top of each other. The definition of $\mathcal{C}_{gr}$ requires that one part is grasped and all other parts are stable. There are still two modes, depending on whether the manipulator is grasping a part. Once again, transitions occur only when the robot is in $\mathcal{C}_{tra} = \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$.

Figure 7.18: This manipulation planning example was solved in [452] and involves 90 movable pieces of furniture. Some of them must be dragged out of the way to solve the problem. Paths for two different queries are shown.

The task involves moving each part from one configuration to another. This is achieved once again by defining a manipulation graph and obtaining a sequence of transit paths (in which no parts move) and transfer paths (in which one part is carried and all other parts are fixed). Challenging manipulation problems solved by motion planning algorithms are shown in Figures 7.17 and 7.18.

Other generalizations are possible. A generalization to k robots would lead to 2^k modes, in which each mode indicates whether each robot is grasping the part. Multiple robots could even grasp the same object. Another generalization could allow a single robot to grasp more than one object.

7.4 Planning for Closed Kinematic Chains

This section continues where Section 4.4 left off. The subspace of $\mathcal{C}$ that results from maintaining kinematic closure was defined and illustrated through some ex-

amples. Planning in this context requires that paths remain on a lower dimensional variety for which a parameterization is not available. Many important applications require motion planning while maintaining these constraints. For example, consider a manipulation problem that involves multiple manipulators grasping the same object, which forms a closed loop as shown in Figure 7.19. A loop exists because both manipulators are attached to the ground, which may itself be considered as a link. The development of virtual actors for movies and video games also involves related manipulation problems. Loops also arise in this context when more than one human limb is touching a fixed surface (e.g., two feet on the ground). A class of robots called *parallel manipulators* are intentionally designed with internal closed loops [360]. For example, consider the Stewart-Gough platform [208, 451] illustrated in Figure 7.20. The lengths of each of the six arms, $\mathcal{A}_1, \ldots, \mathcal{A}_6$, can be independently varied while they remain attached via spherical joints to the ground and to the *platform*, which is $\mathcal{A}_7$. Each arm can actually be imagined as two links that are connected by a prismatic joint. Due to the total of 6 degrees of freedom introduced by the variable lengths, the platform actually achieves the full 6 degrees of freedom (hence, some six-dimensional region in $SE(3)$ is obtained for $\mathcal{A}_7$). Planning the motion of the Stewart-Gough platform, or robots that are based on the platform (the robot shown in Figure 7.27 uses a stack of several of these mechanisms), requires handling many closure constraints that must be maintained simultaneously. Another application is computational biology, in which the C-space of molecules is searched, many of which are derived from molecules that have closed, flexible chains of bonds [134].

7.4.1 Adaptation of Motion Planning Algorithms

All of the components from the general motion planning problem of Formulation 4.1 are included: $\mathcal{W}$, $\mathcal{O}$, $\mathcal{A}_1, \ldots, \mathcal{A}_m$, $\mathcal{C}$, q_I, and q_G. It is assumed that the robot is a collection of r links that are possibly attached in loops.

It is assumed in this section that $\mathcal{C} = \mathbb{R}^n$. If this is not satisfactory, there are two ways to overcome the assumption. The first is to represent $SO(2)$ and $SO(3)$ as $\mathbb{S}^1$ and $\mathbb{S}^3$, respectively, and include the circle or sphere equation as part of the constraints considered here. This avoids the topology problems. The other option is to abandon the restriction of using $\mathbb{R}^n$ and instead use a parameterization of $\mathcal{C}$ that is of the appropriate dimension. To perform calculus on such manifolds, a *smooth structure* is required, which is introduced in Section 8.3.2. In the presentation here, however, vector calculus on $\mathbb{R}^n$ is sufficient, which intentionally avoids these extra technicalities.

Closure constraints The closure constraints introduced in Section 4.4 can be summarized as follows. There is a set, $\mathcal{P}$, of polynomials $f_1, \ldots, f_k$ that belong to $\mathbb{Q}[q_1, \ldots, q_n]$ and express the constraints for particular points on the links of the robot. The determination of these is detailed in Section 4.4.3. As mentioned

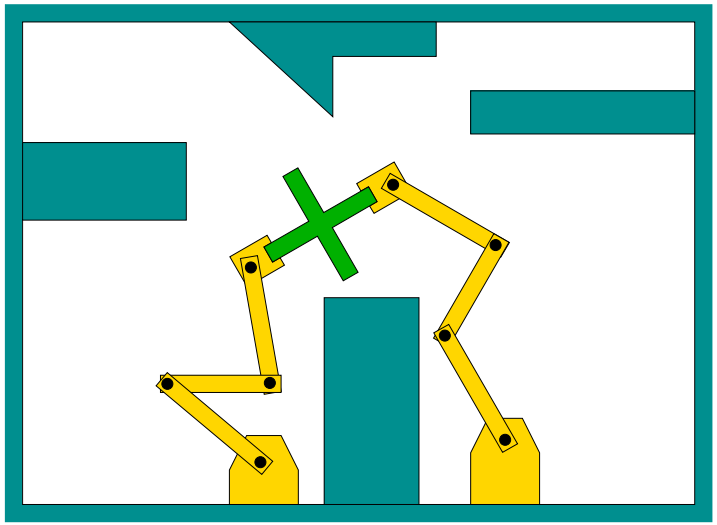


Figure 7.19: Two or more manipulators manipulating the same object causes closed kinematic chains. Each black disc corresponds to a revolute joint.

previously, polynomials that force points to lie on a circle or sphere in the case of rotations may also be included in $\mathcal{P}$.

Let n denote the dimension of $\mathcal{C}$. The *closure space* is defined as

$$\mathcal{C}_{clo} = \{q \in \mathcal{C} \mid \forall f_i \in \mathcal{P}, f_i(q_1, \ldots, q_n) = 0\}, \tag{7.21}$$

which is an m-dimensional subspace of $\mathcal{C}$ that corresponds to all configurations that satisfy the closure constants. The obstacle set must also be taken into account. Once again, $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are defined using (4.34). The *feasible space* is defined as $\mathcal{C}_{fea} = \mathcal{C}_{clo} \cap \mathcal{C}_{free}$, which are the configurations that satisfy closure constraints and avoid collisions.

The motion planning problem is to find a path $\tau : [0, 1] \rightarrow \mathcal{C}_{fea}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$. The new challenge is that there is no explicit parameterization of $\mathcal{C}_{fea}$, which is further complicated by the fact that $m < n$ (recall that m is the dimension of $\mathcal{C}_{clo}$).

Combinatorial methods Since the constraints are expressed with polynomials, it may not be surprising that the computational algebraic geometry methods of Section 6.4 can solve the general motion planning problem with closed kinematic chains. Either cylindrical algebraic decomposition or Canny's roadmap algorithm may be applied. As mentioned in Section 6.5.3, an adaptation of the roadmap algorithm that is optimized for problems in which $m < n$ is given in [52].

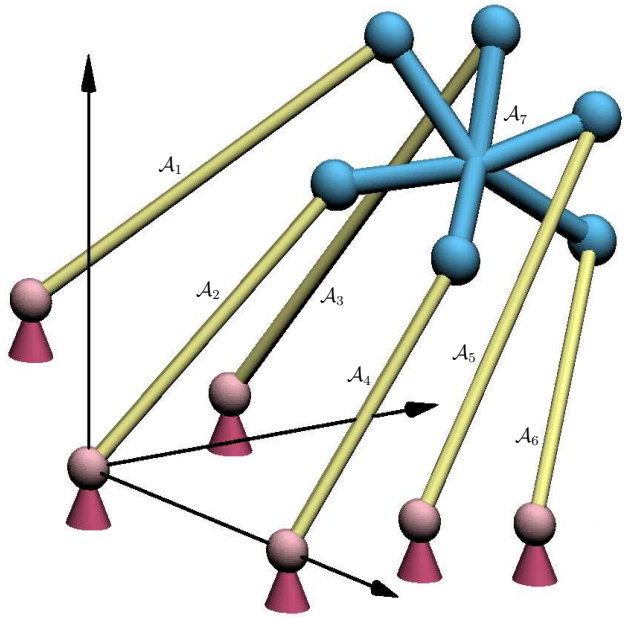


Figure 7.20: An illustration of the Stewart-Gough platform (adapted from a figure made by Frank Sottile).

Sampling-based methods Most of the methods of Chapter 5 are not easy to adapt because they require sampling in $\mathcal{C}_{fea}$, for which a parameterization does not exist. If points in a bounded region of $\mathbb{R}^n$ are chosen at random, the probability is zero that a point on $\mathcal{C}_{fea}$ will be obtained. Some incremental sampling and searching methods can, however, be adapted by the construction of a local planning method (LPM) that is suited for problems with closure constraints. The sampling-based roadmap methods require many samples to be generated directly on $\mathcal{C}_{fea}$. Section 7.4.2 presents some techniques that can be used to generate such samples for certain classes of problems, enabling the development of efficient sampling-based planners and also improving the efficiency of incremental search planners. Before covering these techniques, we first present a method that leads to a more general sampling-based planner and is easier to implement. However, if designed well, planners based on the techniques of Section 7.4.2 are more efficient.

Now consider adapting the RDT of Section 5.5 to work for problems with closure constraints. Similar adaptations may be possible for other incremental sampling and searching methods covered in Section 5.4, such as the randomized potential field planner. A dense sampling sequence, α, is generated over a bounded n-dimensional subset of $\mathbb{R}^n$, such as a rectangle or sphere, as shown in Figure 7.21.

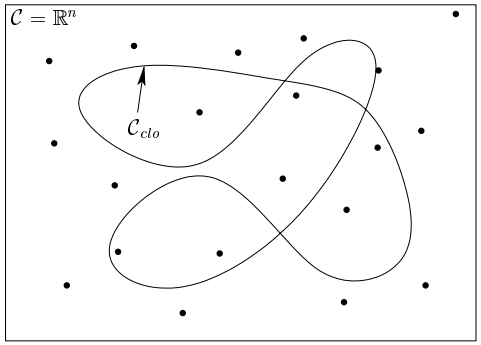


Figure 7.21: For the RDT, the samples can be drawn from a region in $\mathbb{R}^n$, the space in which $\mathcal{C}_{clo}$ is embedded.

The samples are not actually required to lie on $\mathcal{C}_{clo}$ because they do not necessarily become part of the topological graph, $\mathcal{G}$. They mainly serve to pull the search tree in different directions. One concern in choosing the bounding region is that it must include $\mathcal{C}_{clo}$ (at least the connected component that includes q_I) but it must not be unnecessarily large. Such bounds are obtained by analyzing the motion limits for a particular linkage.

Stepping along $\mathcal{C}_{clo}$ The RDT algorithm given Figure 5.21 can be applied directly; however, the STOPPING-CONFIGURATION function in line 4 must be changed to account for both obstacles and the constraints that define $\mathcal{C}_{clo}$. Figure 7.22 shows one general approach that is based on *numerical continuation* [13]. An alternative is to use inverse kinematics, which is part of the approach described in Section 7.4.2. The nearest RDT vertex, $q \in \mathcal{C}$, to the sample $\alpha(i)$ is first computed. Let $v = \alpha(i) - q$, which indicates the direction in which an edge would be made from q if there were no constraints. A local motion is then computed by projecting v into the tangent plane[3] of $\mathcal{C}_{clo}$ at the point q. Since $\mathcal{C}_{clo}$ is generally nonlinear, the local motion produces a point that is not precisely on $\mathcal{C}_{clo}$. Some numerical tolerance is generally accepted, and a small enough step is taken to ensure that the tolerance is maintained. The process iterates by computing v with respect to the new point and moving in the direction of v projected into the new tangent plane. If the error threshold is surpassed, then motions must be executed in the normal direction to return to $\mathcal{C}_{clo}$. This process of executing tangent and normal motions terminates when progress can no longer be made, due either to the alignment of the tangent plane (nearly perpendicular to v) or to an obstacle. This finally yields q_s, the stopping configuration. The new path followed in $\mathcal{C}_{fea}$ is no longer a "straight line" as was possible for some problems in Section 5.5; therefore, the approximate methods in Section 5.5.2 should be used

[3]Tangent planes are defined rigorously in Section 8.3.

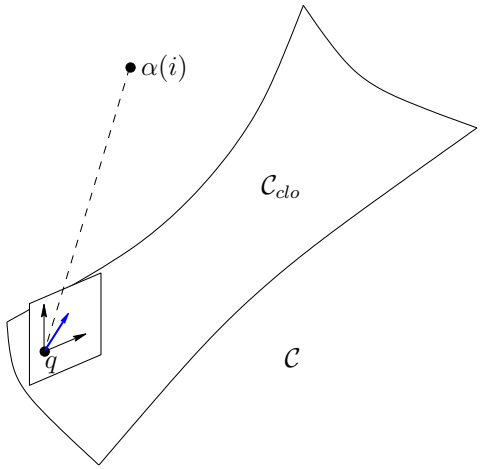


Figure 7.22: For each sample $\alpha(i)$ the nearest point, $q_n \in \mathcal{C}$, is found, and then the local planner generates a motion that lies in the local tangent plane. The motion is the project of the vector from q_n to $\alpha(i)$ onto the tangent plane.

to create intermediate vertices along the path.

In each iteration, the tangent plane computation is computed at some $q \in \mathcal{C}_{clo}$ as follows. The differential configuration vector dq lies in the tangent space of a constraint $f_i(q) = 0$ if

$$\frac{\partial f_i(q)}{\partial q_1} dq_1 + \frac{\partial f_i(q)}{\partial q_2} dq_2 + \cdots + \frac{\partial f_i(q)}{\partial q_n} dq_n = 0. \tag{7.22}$$

This leads to the following homogeneous system for all of the k polynomials in $\mathcal{P}$ that define the closure constraints

$$\begin{pmatrix} \frac{\partial f_1(q)}{\partial q_1} & \frac{\partial f_1(q)}{\partial q_2} & \cdots & \frac{\partial f_1(q)}{\partial q_n} \\ \frac{\partial f_2(q)}{\partial q_1} & \frac{\partial f_2(q)}{\partial q_2} & \cdots & \frac{\partial f_2(q)}{\partial q_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_k(q)}{\partial q_1} & \frac{\partial f_k(q)}{\partial q_2} & \cdots & \frac{\partial f_k(q)}{\partial q_n} \end{pmatrix} \begin{pmatrix} dq_1 \\ dq_2 \\ \vdots \\ dq_n \end{pmatrix} = \mathbf{0}. \tag{7.23}$$

If the rank of the matrix is $m \leq n$, then m configuration displacements can be chosen independently, and the remaining $n - m$ parameters must satisfy (7.23). This can be solved using linear algebra techniques, such as singular value decomposition (SVD) [204, 471], to compute an orthonormal basis for the tangent space at q. Let $e_1, \ldots, e_m$, denote these n-dimensional basis vectors. The components

of the motion direction are obtained from $v = \alpha(i) - q_n$. First, construct the inner products, $a_1 = v \cdot e_1$, $a_2 = v \cdot e_2$, ..., $a_m = v \cdot e_m$. Using these, the projection of v in the tangent plane is the n-dimensional vector w given by

$$w = \sum_{i}^{m} a_i e_i, \tag{7.24}$$

which is used as the direction of motion. The magnitude must be appropriately scaled to take sufficiently small steps. Since $\mathcal{C}_{clo}$ is generally curved, a linear motion leaves $\mathcal{C}_{clo}$. A motion in the inward normal direction is then required to move back onto $\mathcal{C}_{clo}$.

Since the dimension m of $\mathcal{C}_{clo}$ is less than n, the procedure just described can only produce numerical approximations to paths in $\mathcal{C}_{clo}$. This problem also arises in implicit curve tracing in graphics and geometric modeling [234]. Therefore, each constraint $f_i(q_1, \ldots, q_n) = 0$ is actually slightly weakened to $|f_i(q_1, \ldots, q_n)| < \epsilon$ for some fixed tolerance $\epsilon > 0$. This essentially "thickens" $\mathcal{C}_{clo}$ so that its dimension is n. As an alternative to computing the tangent plane, motion directions can be sampled directly inside of this thickened region without computing tangent planes. This results in an easier implementation, but it is less efficient [479].

7.4.2 Active-Passive Link Decompositions

An alternative sampling-based approach is to perform an *active-passive decomposition*, which is used to generate samples in $\mathcal{C}_{clo}$ by directly sampling *active* variables, and computing the closure values for *passive* variables using inverse kinematics methods. This method was introduced in [222] and subsequently improved through the development of the *random loop generator* in [133, 135]. The method serves as a general framework that can adapt virtually any of the methods of Chapter 5 to handle closed kinematic chains, and experimental evidence suggests that the performance is better than the method of Section 7.4.1. One drawback is that the method requires some careful analysis of the linkage to determine the best decomposition and also bounds on its mobility. Such analysis exists for very general classes of linkages [133].

Active and passive variables In this section, let $\mathcal{C}$ denote the C-space obtained from all joint variables, instead of requiring $\mathcal{C} = \mathbb{R}^n$, as in Section 7.4.1. This means that $\mathcal{P}$ includes only polynomials that encode closure constraints, as opposed to allowing constraints that represent rotations. Using the tree representation from Section 4.4.3, this means that $\mathcal{C}$ is of dimension n, arising from assigning one variable for each revolute joint of the linkage in the absence of any constraints. Let $q \in \mathcal{C}$ denote this vector of configuration variables. The *active-passive decomposition* partitions the variables of q to form two vectors, q^a, called the *active variables* and q^p, called the *passive variables*. The values of passive variables are always determined from the active variables by enforcing the closure

constraints and using inverse kinematics techniques. If m is the dimension of $\mathcal{C}_{clo}$, then there are always m active variables and $n - m$ passive variables.

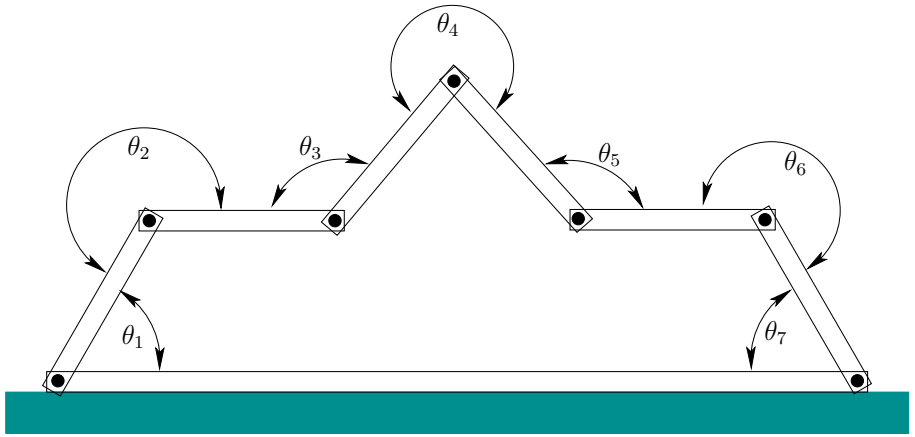


Figure 7.23: A chain of links in the plane. There are seven links and seven joints, which are constrained to form a loop. The dimension of $\mathcal{C}$ is seven, but the dimension of $\mathcal{C}_{clo}$ is four.

Temporarily, suppose that the linkage forms a single loop as shown in Figure 7.23. One possible decomposition into active q^a and passive q^p variables is given in Figure 7.24. If constrained to form a loop, the linkage has four degrees of freedom, assuming the bottom link is rigidly attached to the ground. This means that values can be chosen for four active joint angles q^a and the remaining three q^p can be derived from solving the inverse kinematics. To determine q^p, there are three equations and three unknowns. Unfortunately, these equations are nonlinear and fairly complicated. Nevertheless, efficient solutions exist for this case, and the 3D generalization [350]. For a 3D loop formed of revolute joints, there are six passive variables. The number, 3, of passive links in $\mathbb{R}^2$ and the number 6 for $\mathbb{R}^3$ arise from the dimensions of $SE(2)$ and $SE(3)$, respectively. This is the freedom that is stripped away from the system by enforcing the closure constraints. Methods for efficiently computing inverse kinematics in two and three dimensions are given in [24]. These can also be applied to the RDT stepping method in Section 7.4.1, instead of using continuation.

If the maximal number of passive variables is used, there is at most a finite number of solutions to the inverse kinematics problem; this implies that there are often several choices for the passive variable values. It could be the case that for some assignments of active variables, there are no solutions to the inverse kinematics. An example is depicted in Figure 7.25. Suppose that we want to generate samples in $\mathcal{C}_{clo}$ by selecting random values for q^a and then using inverse kinematics for q^p. What is the probability that a solution to the inverse kinematics

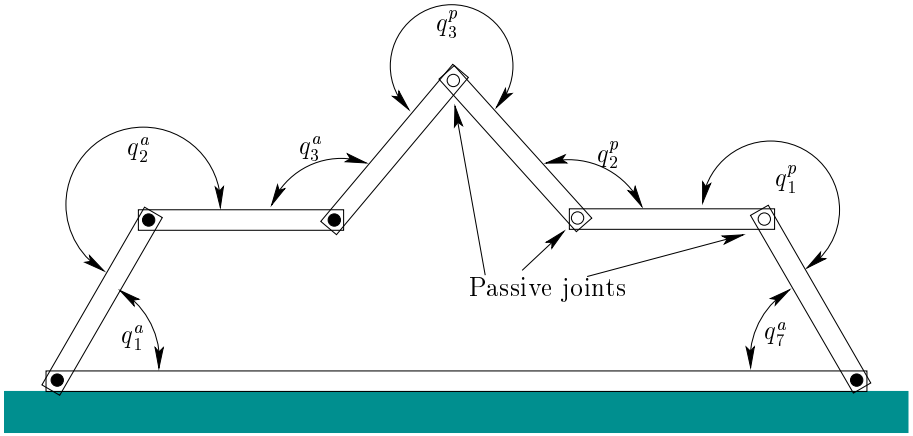


Figure 7.24: Three of the joint variables can be determined automatically by inverse kinematics. Therefore, four of the joints be designated as *active*, and the remaining three will be passive.

exists? For the example shown, it appears that solutions would not exist in most trials.

Loop generator The *random loop generator* greatly improves the chance of obtaining closure by iteratively restricting the range on each of the active variables. The method requires that the active variables appear sequentially along the chain (i.e., there is no interleaving of active and passive variables). The m coordinates of q^a are obtained sequentially as follows. First, compute an interval, I_1, of allowable values for q_1^a. The interval serves as a loose bound in the sense that, for any value $q_1^a \notin I_1$, it is known for certain that closure cannot be obtained. This is ensured by performing a careful geometric analysis of the linkage, which will be explained shortly. The next step is to generate a sample in $q_1^a \in I_1$, which is accomplished in [133] by picking a random point in I_1. Using the value q_1^a, a bounding interval I_2 is computed for allowable values of q_2^a. The value q_2^a is obtained by sampling in I_2. This process continues iteratively until I_m and q_m^a are obtained, unless it terminates early because some $I_i = \emptyset$ for $i < m$. If successful termination occurs, then the active variables q^a are used to find values q^p for the passive variables. This step still might fail, but the probability of success is now much higher. The method can also be applied to linkages in which there are multiple, common loops, as in the Stewart-Gough platform, by breaking the linkage into a tree and closing loops one at a time using the loop generator. The performance depends on how the linkage is decomposed [133].

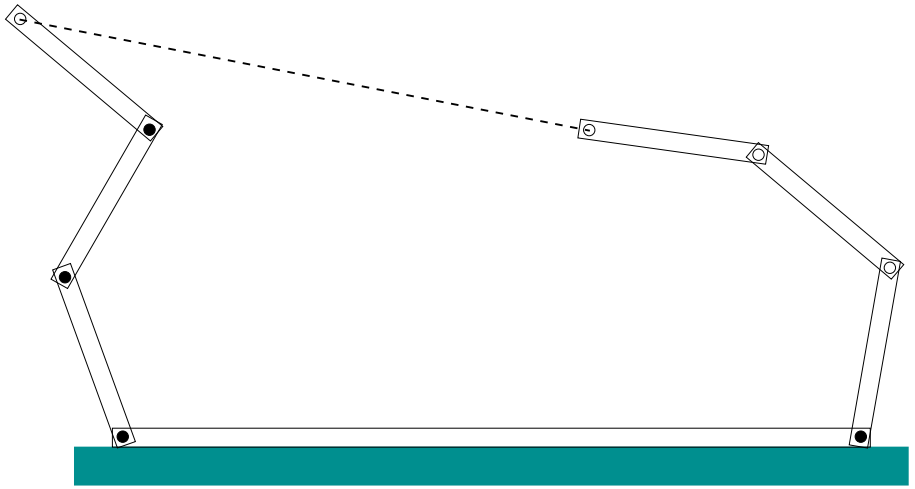


Figure 7.25: In this case, the active variables are chosen in a way that makes it impossible to assign passive variables that close the loop.

Computing bounds on joint angles The main requirement for successful application of the method is the ability to compute bounds on how far a chain of links can travel in $\mathcal{W}$ over some range of joint variables. For example, for a planar chain that has revolute joints with no limits, the chain can sweep out a circle as shown in Figure 7.26a. Suppose it is known that the angle between links must remain between $-\pi/6$ and $\pi/6$. A tighter bounding region can be obtained, as shown in Figure 7.26b. Three-dimensional versions of these bounds, along with many necessary details, are included in [133]. These bounds are then used to compute I_i in each iteration of the sampling algorithm.

Now that there is an efficient method that generates samples directly in $\mathcal{C}_{clo}$, it is straightforward to adapt any of the sampling-based planning methods of Chapter 5. In [133] many impressive results are obtained for challenging problems that have the dimension of $\mathcal{C}$ up to 97 and the dimension of $\mathcal{C}_{clo}$ up to 25; see Figure 7.27. These methods are based on applying the new sampling techniques to the RDTs of Section 5.5 and the visibility sampling-based roadmap of Section 5.6.2. For these algorithms, the local planning method is applied to the active variables, and inverse kinematics algorithms are used for the passive variables in the path validation step. This means that inverse kinematics and collision checking are performed together, instead of only collision checking, as described in Section 5.3.4.

One important issue that has been neglected in this section is the existence of *kinematic singularities*, which cause the dimension of $\mathcal{C}_{clo}$ to drop in the vicinity of certain points. The methods presented here have assumed that solving the motion planning problem does not require passing through a singularity. This assump-

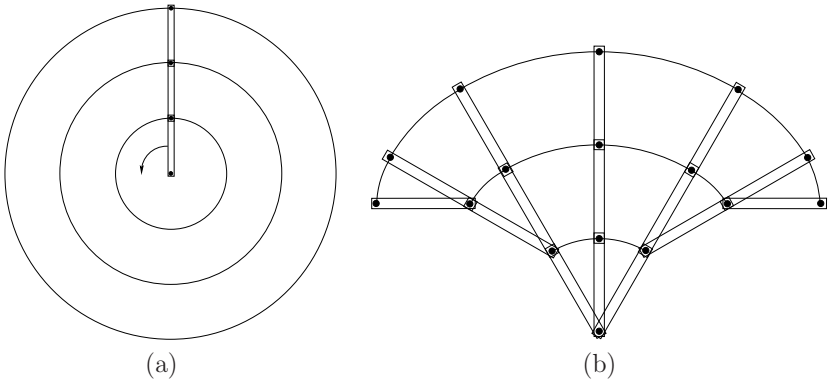


Figure 7.26: (a) If any joint angle is possible, then the links sweep out a circle in the limit. (b) If there are limits on the joint angles, then a tighter bound can be obtained for the reachability of the linkage.

tion is reasonable for robot systems that have many extra degrees of freedom, but it is important to understand that completeness is lost in general because the sampling-based methods do not explicitly handle these degeneracies. In a sense, they occur below the level of sampling resolution. For more information on kinematic singularities and related issues, see [360].

7.5 Folding Problems in Robotics and Biology

A growing number of motion planning applications involve some form of folding. Examples include automated carton folding, computer-aided drug design, protein folding, modular reconfigurable robots, and even robotic origami. These problems are generally modeled as a linkage in which all bodies are connected by revolute joints. In robotics, self-collision between pairs of bodies usually must be avoided. In biological applications, energy functions replace obstacles. Instead of crisp obstacle boundaries, energy functions can be imagined as "soft" obstacles, in which a real value is defined for every $q \in \mathcal{C}$, instead of defining a set $\mathcal{C}_{obs} \subset \mathcal{C}$. For a given threshold value, such energy functions can be converted into an obstacle region by defining $\mathcal{C}_{obs}$ to be the configurations that have energy above the threshold. However, the energy function contains more information because such thresholds are arbitrary. This section briefly shows some examples of folding problems and techniques from the recent motion planning literature.

Carton folding An interesting application of motion planning to the automated folding of boxes is presented in [348]. Figure 7.28 shows a carton in its original flat form and in its folded form. As shown in Figure 7.29, the problem can be

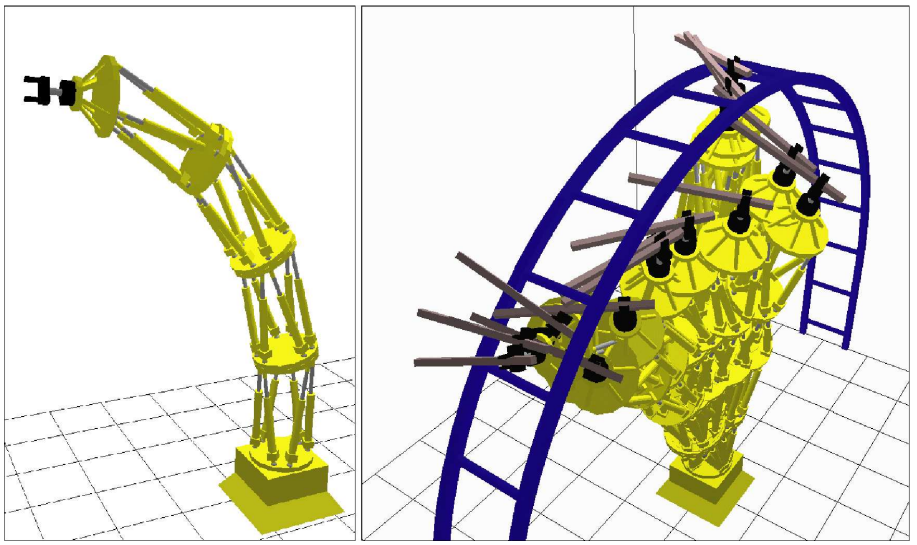


Figure 7.27: Planning for the Logabex LX4 robot [101]. This solution was computed in less than a minute by applying active-passive decomposition to an RDT-based planner [133]. In this example, the dimension of $\mathcal{C}$ is 97 and the dimension of $\mathcal{C}_{clo}$ is 25.

modeled as a tree of bodies connected by revolute joints. Once this model has been formulated, many methods from Chapters 5 and 6 can be adapted for this problem. In [348], a planning algorithm optimized particularly for box folding is presented. It is an adaptation of an approximate cell decomposition algorithm developed for kinematic chains in [345]. Its complexity is exponential in the degrees of freedom of the carton, but it gives good performance on practical examples. One such solution that was found by motion planning is shown in Figure 7.30. To use these solutions in a factory, the manipulation problem has to be additionally considered. For example, as demonstrated in [348], a manipulator arm robot can be used in combination with a well-designed set of fixtures. The fixtures help hold the carton

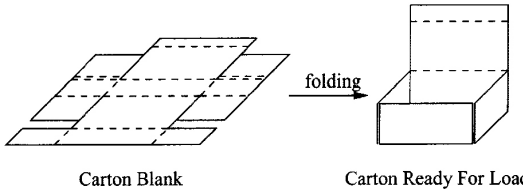


Figure 7.28: An important packaging problem is to automate the folding of a perforated sheet of cardboard into a carton.

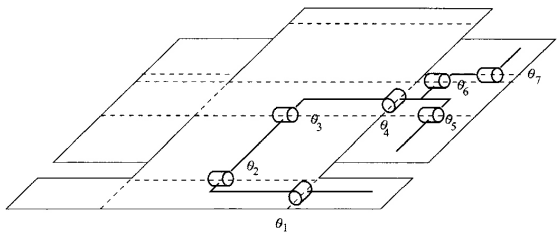


Figure 7.29: The carton can be cleverly modeled as a tree of bodies that are attached by revolute joints.

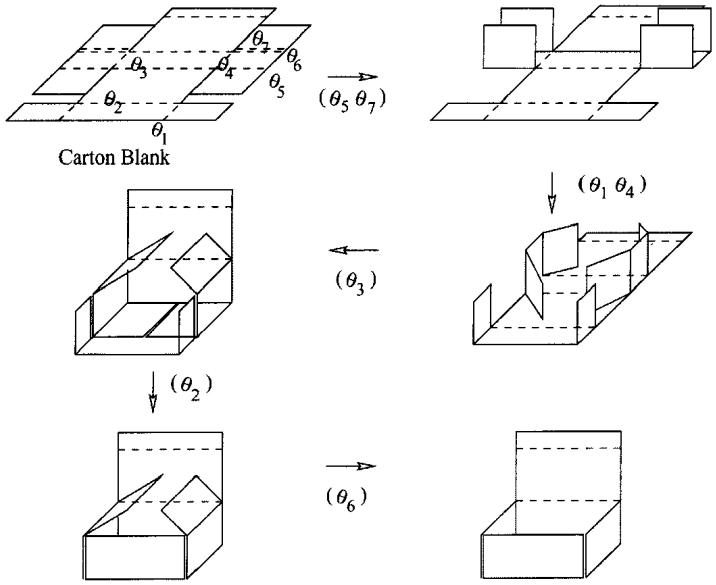


Figure 7.30: A folding sequence that was computed using the algorithm in [348].

in place while the manipulator applies pressure in the right places, which yields the required folds. Since the feasibility with fixtures depends on the particular folding path, the planning algorithm generates all possible distinct paths from the initial configuration (at which the box is completely unfolded).

Simplifying knots A *knot* is a closed curve that does not intersect itself, is embedded in $\mathbb{R}^3$, and cannot be untangled to produce a simple loop (such as a circular path). If the knot is allowed to intersect itself, then any knot can be untangled; therefore, a careful definition of what it means to untangle a knot is needed. For a closed curve, $\tau : [0,1] \rightarrow \mathbb{R}^3$, embedded in $\mathbb{R}^3$ (it cannot intersect itself), let the set $\mathbb{R}^3 \setminus \tau([0,1])$ of points not reached by the curve be called the *ambient space* of τ. In knot theory, an *ambient isotopy* between two closed curves, τ_1 and τ_2, embedded in $\mathbb{R}^3$ is a homeomorphism between their ambient spaces. Intuitively, this means that τ_1 can be warped into τ_2 without allowing any self-intersections. Therefore, determining whether two loops are equivalent seems closely related to motion planning. Such equivalence gives rise to groups that characterize the space of knots and are closely related to the fundamental group described in Section 4.1.3. For more information on knot theory, see [6, 232, 261].

A motion planning approach was developed in [297] to determine whether a closed curve is equivalent to the *unknot*, which is completely untangled. This can be expressed as a curve that maps onto $\mathbb{S}^1$, embedded in $\mathbb{R}^3$. The algorithm takes as input a knot expressed as a circular chain of line segments embedded in $\mathbb{R}^3$. In this case, the unknot can be expressed as a triangle in $\mathbb{R}^3$. One of the most challenging examples solved by the planner is shown in Figure 7.31. The planner is sampling-based and shares many similarities with the RDT algorithm of Section 5.5 and the Ariadne's clew and expansive space planners described in Section 5.4.4. Since the task is not to produce a collision-free path, there are several unique aspects in comparison to motion planning. An energy function is defined over the collection of segments to try to guide the search toward simpler configurations. There are two kinds of local operations that are made by the planner: 1) Try to move a vertex toward a selected subgoal in the ambient space. This is obtained by using random sampling to grow a search tree. 2) Try to delete a vertex, and connect the neighboring vertices by a straight line. If no collision occurs along the intermediate configurations, then the knot has been simplified. The algorithm terminates when it is unable to further simplify the knot.

Drug design A sampling-based motion planning approach to pharmaceutical drug design is taken in [308]. The development of a drug is a long, incremental process, typically requiring years of research and experimentation. The goal is to find a relatively small molecule called a *ligand*, typically comprising a few dozen atoms, that docks with a receptor cavity in a specific protein [315]; Figure 1.14 (Section 1.2) illustrated this. Examples of drug molecules were also given in Figure 1.14. Protein-ligand docking can stimulate or inhibit some biological

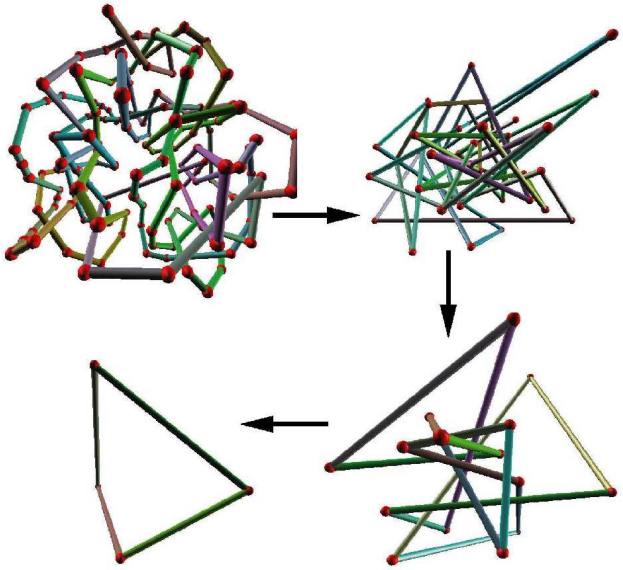


Figure 7.31: The planner in [297] untangles the famous Ochiai unknot benchmark in a few minutes on a standard PC.

activity, ultimately leading to the desired pharmacological effect. The problem of finding suitable ligands is complicated due to both energy considerations and the flexibility of the ligand. In addition to satisfying structural considerations, factors such as synthetic accessibility, drug pharmacology and toxicology greatly complicate and lengthen the search for the most effective drug molecules.

One popular model used by chemists in the context of drug design is a *pharmacophore*, which serves as a template for the desired ligand [122, 175, 194, 433]. The pharmacophore is expressed as a set of *features* that an effective ligand should possess and a set of *spatial constraints* among the features. Examples of features are specific atoms, centers of benzene rings, positive or negative charges, hydrophobic or hydrophilic centers, and hydrogen bond donors or acceptors. Features generally require that parts of the molecule must remain fixed in $\mathbb{R}^3$, which induces kinematic closure constraints. These features are developed by chemists to encapsulate the assumption that ligand binding is due primarily to the interaction of some features of the ligand to "complementary" features of the receptor. The interacting features are included in the pharmacophore, which is a template for screening candidate drugs, and the rest of the ligand atoms merely provide a scaffold for holding the pharmacophore features in their spatial positions. Figure 7.32 illustrates the pharmacophore concept.

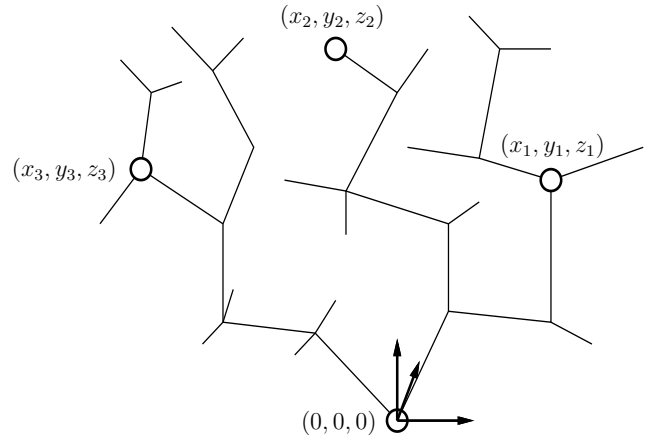


Figure 7.32: A pharmacophore is a model used by chemists to simplify the interaction process between a ligand (candidate drug molecule) and a protein. It often amounts to holding certain features of the molecule fixed in $\mathbb{R}^3$. In this example, the positions of three atoms must be fixed relative to the body frame of an arbitrarily designated root atom. It is assumed that these features interact with some complementary features in the cavity of the protein.

Candidate drug molecules (ligands), such as the ones shown in Figure 1.14, can be modeled as a tree of bodies as shown in Figure 7.33. Some bonds can rotate, yielding revolute joints in the model; other bonds must remain fixed. The drug design problem amounts to searching the space of configurations (called *conformations*) to try to find a low-energy configuration that also places certain atoms in specified locations in $\mathbb{R}^3$. This additional constraint arises from the pharmacophore and causes the planning to occur on $\mathcal{C}_{clo}$ from Section 7.4 because the pharmacophores can be expressed as closure constraints.

An energy function serves a purpose similar to that of a collision detector. The evaluation of a ligand for drug design requires determining whether it can achieve low-energy conformations that satisfy the pharmacophore constraints. Thus, the task is different from standard motion planning in that there is no predetermined goal configuration. One of the greatest difficulties is that the energy functions are extremely complicated, nonlinear, and empirical. Here is typical example (used in [308]):

$$\begin{aligned} e(q) = \quad & \textstyle\sum_{bonds} \tfrac{1}{2} K_b (R - R')^2 + \sum_{ang} \tfrac{1}{2} K_a (\alpha - \alpha')^2 + \\ & \textstyle\sum_{torsions} K_d [1 + \cos(p\theta - \theta')] + \\ & \textstyle\sum_{i,j} \left\{ 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \frac{c_i c_j}{\epsilon r_{ij}} \right\}. \end{aligned} \tag{7.25}$$

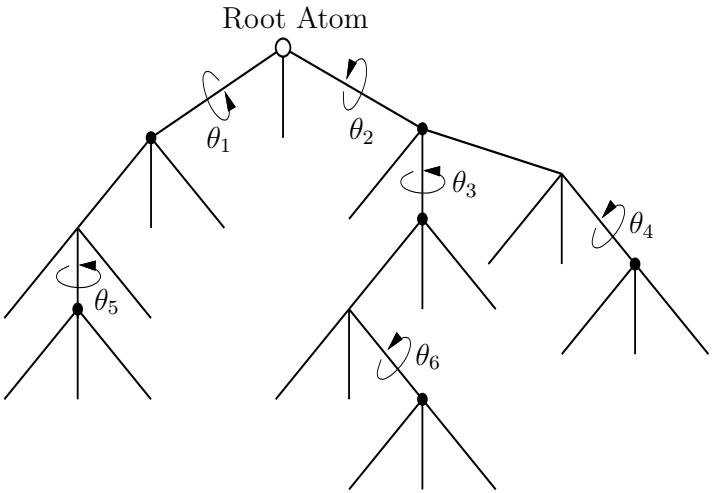


Figure 7.33: The modeling of a flexible molecule is similar to that of a robot. One atom is designated as the root, and the remaining bodies are arranged in a tree. If there are cyclic chains in the molecules, then constraints as described in Section 4.4 must be enforced. Typically, only some bonds are capable of rotation, whereas others must remain rigid.

The energy accounts for torsion-angle deformations, van der Waals potential, and Coulomb potentials. In (7.25), the first sum is taken over all bonds, the second over all bond angles, the third over all rotatable bonds, and the last is taken over all pairs of atoms. The variables are the force constants, K_b, K_a, and K_d; the dielectric constant, ϵ; a periodicity constant, p; the Lennard-Jones radii, σ_{ij}; well depth, ϵ_{ij}; partial charge, c_i; measured bond length, R; equilibrium bond length, R'; measured bond angle, α; equilibrium bond angle, α'; measured torsional angle, θ; equilibrium torsional angle, θ'; and distance between atom centers, r_{ij}. Although the energy expression is very complicated, it only depends on the configuration variables; all others are constants that are estimated in advance.

Protein folding In computational biology, the problem of protein folding shares many similarities with drug design in that the molecules have rotatable bonds and energy functions are used to express good configurations. The problems are much more complicated, however, because the protein molecules are generally much larger than drug molecules. Instead of a dozen degrees of freedom, which is typical for a drug molecule, proteins have hundreds or thousands of degrees of freedom. When proteins appear in nature, they are usually in a folded, low-energy configuration. The *structure problem* involves determining precisely how the protein is folded so that its biological activity can be completely understood. In some studies, biologists are even interested in the pathway that a protein takes to arrive in its folded state [18, 19]. This leads directly to an extension of motion planning that involves arriving at a goal state in which the molecule is folded. In [18, 19], sampling-based planning algorithms were applied to compute folding pathways for proteins. The protein starts in an unfolded configuration and must arrive in a specified folded configuration without violating energy constraints along the way. Figure 7.34 shows an example from [19]. That work also draws interesting connections between protein folding and box folding, which was covered previously.

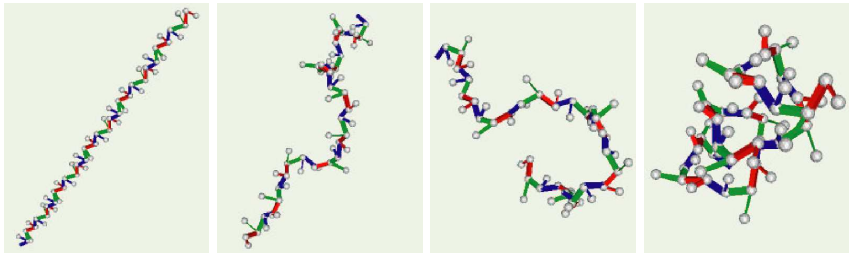


Figure 7.34: Protein folding for a polypeptide, computed by a sampling-based roadmap planning algorithm [18]

7.6 Coverage Planning

Imagine automating the motion of a lawnmower for an estate that has many obstacles, such as a house, trees, garage, and a complicated property boundary. What are the best zig-zag motions for the lawnmower? Can the amount of redundant traversals be minimized? Can the number of times the lawnmower needs to be stopped and rotated be minimized? This is one example of *coverage planning*, which is motivated by applications such as lawn mowing, automated farming, painting, vacuum cleaning, and mine sweeping. A survey of this area appears in [117]. Even for a region in $\mathcal{W} = \mathbb{R}^2$, finding an optimal-length solution to coverage planning is NP-hard, by reduction to the closely related Traveling Salesman Problem [28, 372]. Therefore, we are willing to tolerate approximate or even heuristic solutions to the general coverage problem, even in $\mathbb{R}^2$.

Boustrophedon decomposition One approach to the coverage problem is to decompose $\mathcal{C}_{free}$ into cells and perform boustrophedon (from the Greek "ox turning") motions in each cell as shown in Figure 7.35 [119]. It is assumed that the robot is a point in $\mathcal{W} = \mathbb{R}^2$, but it carries a *tool* of thickness ϵ that hangs evenly over the sides of the robot. This enables it to paint or mow part of $\mathcal{C}_{free}$ up to distance $\epsilon/2$ from either side of the robot as it moves forward. Such motions are often used in printers to reduce the number of carriage returns.

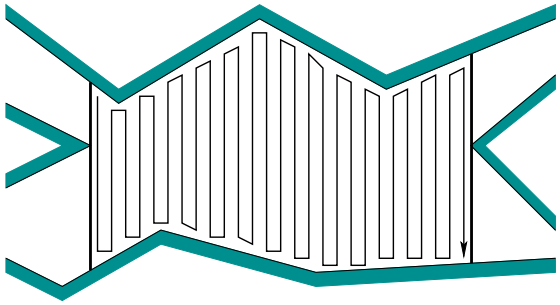
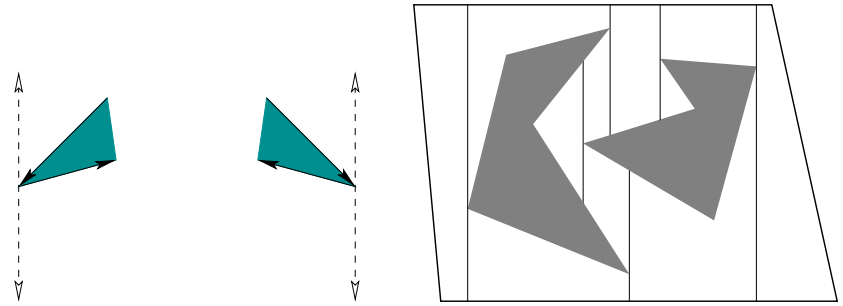


Figure 7.35: An example of the ox plowing motions.

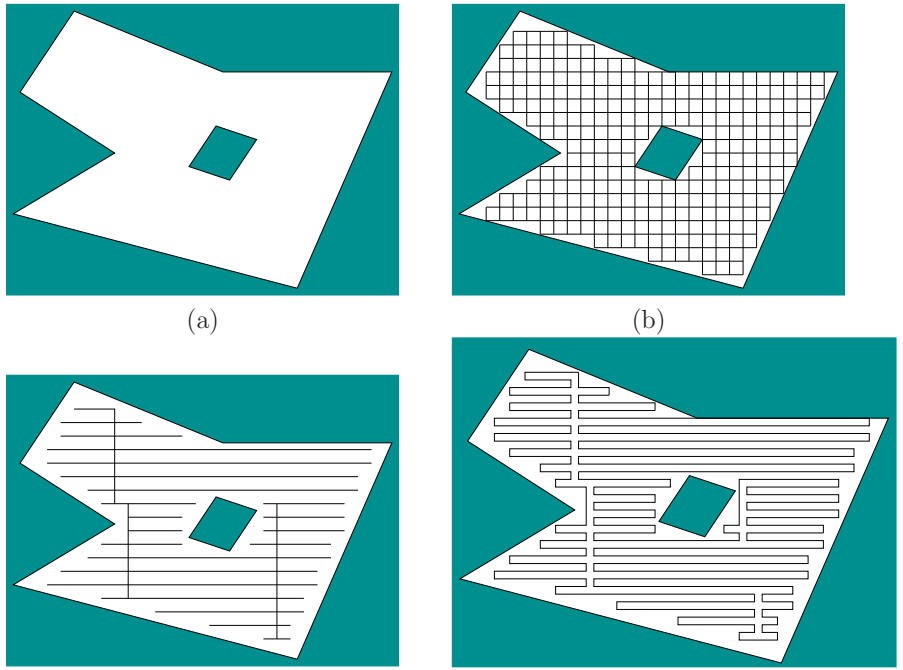


(a) (b)

Figure 7.36: (a) Only the first case from Figure 6.2 is needed: extend upward and downward. All other cases are neglected. (b) The resulting decomposition is shown, which has fewer cells than that of the vertical decomposition in Figure 6.3.

If $\mathcal{C}_{obs}$ is polygonal, a reasonable decomposition can be obtained by adapting the vertical decomposition method of Section 6.2.2. In that algorithm, critical events were defined for several cases, some of which are not relevant for the boustrophedon motions. The only events that need to be handled are shown in Figure 7.36a [116]. This produces a decomposition that has fewer cells, as shown in Figure 7.36b. Even though the cells are nonconvex, they can always be sliced nicely into vertical strips, which makes them suitable for boustrophedon motions. The original vertical decomposition could also be used, but the extra cell boundaries would cause unnecessary repositioning of the robot. A similar method, which furthermore optimizes the number of robot turns, is presented in [243].

Spanning tree covering An interesting approximate method was developed by Gabriely and Rimon; it places a tiling of squares inside of $\mathcal{C}_{free}$ and computes the spanning tree of the resulting connectivity graph [187, 188]. Suppose again that $\mathcal{C}_{free}$ is polygonal. Consider the example shown in Figure 7.37a. The first



(a) (b) (c) (d)

Figure 7.37: (a) An example used for spanning tree covering. (b) The first step is to tile the interior with squares. (c) The spanning tree of a roadmap formed from grid adjacencies. (d) The resulting coverage path.

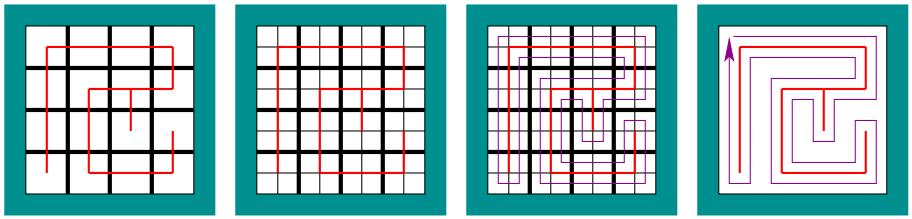


Figure 7.38: A circular path is made by doubling the resolution and following the perimeter of the spanning tree.

step is to tile the interior of $\mathcal{C}_{free}$ with squares, as shown in Figure 7.37b. Each square should be of width ϵ, for some constant $\epsilon > 0$. Next, construct a roadmap $\mathcal{G}$ by placing a vertex in the center of each square and by defining an edge that connects the centers of each pair of adjacent cubes. The next step is to compute a *spanning tree* of $\mathcal{G}$. This is a connected subgraph that has no cycles and touches every vertex of $\mathcal{G}$; it can be computed in $O(n)$ time, if $\mathcal{G}$ has n edges [354]. There are many possible spanning trees, and a criterion can be defined and optimized to induce preferences. One possible spanning tree is shown Figure 7.37c.

Once the spanning tree is made, the robot path is obtained by starting at a point near the spanning tree and following along its perimeter. This path can be precisely specified as shown in Figure 7.38. Double the resolution of the tiling, and form the corresponding roadmap. Part of the roadmap corresponds to the spanning tree, but also included is a loop path that surrounds the spanning tree. This path visits the centers of the new squares. The resulting path for the example of Figure 7.37a is shown in Figure 7.37d. In general, the method yields an optimal route, once the approximation is given. A bound on the uncovered area due to approximation is given in [187]. Versions of the method that do not require an initial map are also given in [187, 188]; this involves reasoning about information spaces, which are covered in Chapter 11.

7.7 Optimal Motion Planning

This section can be considered transitional in many ways. The main concern so far with motion planning has been *feasibility* as opposed to *optimality*. This placed the focus on finding *any* solution, rather than further requiring that a solution be optimal. In later parts of the book, especially as uncertainty is introduced, optimality will receive more attention. Even the most basic forms of decision theory (the topic of Chapter 9) center on making optimal choices. The requirement of optimality in very general settings usually requires an exhaustive search over the state space, which amounts to computing continuous cost-to-go functions. Once such functions are known, a feedback plan is obtained, which is much more powerful than having only a path. Thus, optimality also appears frequently in the design

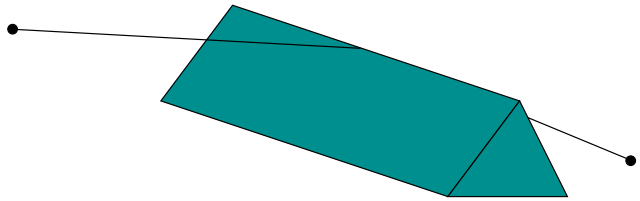


Figure 7.39: For a polyhedral environment, the shortest paths do not have to cross vertices. Therefore, the shortest-path roadmap method from Section 6.2.4 does not extend to three dimensions.

of feedback plans because it sometimes comes at no additional cost. This will become clearer in Chapter 8. The quest for optimal solutions also raises interesting issues about how to approximate a continuous problem as a discrete problem. The interplay between time discretization and space discretization becomes very important in relating continuous and discrete planning problems.

7.7.1 Optimality for One Robot

Euclidean shortest paths One of the most straightforward notions of optimality is the Euclidean shortest path in $\mathbb{R}^2$ or $\mathbb{R}^3$. Suppose that $\mathcal{A}$ is a rigid body that translates only in either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, which contains an obstacle region $\mathcal{O} \subset \mathcal{W}$. Recall that, ordinarily, $\mathcal{C}_{free}$ is an open set, which means that any path, $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, can be shortened. Therefore, shortest paths for motion planning must be defined on the closure $\mathrm{cl}(\mathcal{C}_{free})$, which allows the robot to make contact with the obstacles; however, their interiors must not intersect.

For the case in which $\mathcal{C}_{obs}$ is a polygonal region, the shortest-path roadmap method of Section 6.2.4 has already been given. This can be considered as a kind of multiple-query approach because the roadmap completely captures the structure needed to construct the shortest path for any query. It is possible to make a single-query algorithm using the *continuous Dijkstra paradigm* [227, 371]. This method propagates a *wavefront* from q_I and keeps track of critical events in maintaining the wavefront. As events occur, the wavefront becomes composed of *wavelets*, which are arcs of circles centered on obstacle vertices. The possible events that can occur are 1) a wavelet disappears, 2) a wavelet collides with an obstacle vertex, 3) a wavelet collides with another wavelet, or 4) a wavelet collides with a point in the interior of an obstacle edge. The method can be made to run in time $O(n \lg n)$ and uses $O(n \lg n)$ space. A roadmap is constructed that uses $O(n)$ space. See Section 8.4.3 for a related method.

Such elegant methods leave the impression that finding shortest paths is not very difficult, but unfortunately they do not generalize nicely to $\mathbb{R}^3$ and a polyhedral $\mathcal{C}_{obs}$. Figure 7.39 shows a simple example in which the shortest path does not have to cross a vertex of $\mathcal{C}_{obs}$. It may cross anywhere in the interior of an edge;

therefore, it is not clear where to draw the bitangent lines that would form the shortest-path roadmap. The lower bounds for this problem are also discouraging. It was shown in [91] that the 3D shortest-path problem in a polyhedral environment is NP-hard. Most of the difficulty arises because of the precision required to represent 3D shortest paths. Therefore, efficient polynomial-time approximation algorithms exist [115, 393].

General optimality criteria It is difficult to even define optimality for more general C-spaces. What does it mean to have a shortest path in $SE(2)$ or $SE(3)$? Consider the case of a planar, rigid robot that can translate and rotate. One path could minimize the amount of rotation whereas another tries to minimize the amount of translation. Without more information, there is no clear preference. Ulam's distance is one possibility, which is to minimize the distance traveled by k fixed points [246]. In Chapter 13, differential models will be introduced, which lead to meaningful definitions of optimality. For example, the shortest paths for a slow-moving car are shown in Section 15.3; these require a precise specification of the constraints on the motion of the car (it is more costly to move a car sideways than forward).

This section formulates some optimal motion planning problems, to provide a smooth transition into the later concepts. Up until now, actions were used in Chapter 2 for discrete planning problems, but they were successfully avoided for basic motion planning by directly describing paths that map into $\mathcal{C}_{free}$. It will be convenient to use them once again. Recall that they were convenient for defining costs and optimal planning in Section 2.3.

To avoid for now the complications of differential equations, consider making an approximate model of motion planning in which every path must be composed of a sequence of shortest-path segments in $\mathcal{C}_{free}$. Most often these are line segments; however, for the case of $SO(3)$, circular arcs obtained by spherical linear interpolation may be preferable. Consider extending Formulation 2.3 from Section 2.3.2 to the problem of motion planning.

Let the C-space $\mathcal{C}$ be embedded in $\mathbb{R}^m$ (i.e., $\mathcal{C} \subset \mathbb{R}^m$). An action will be defined shortly as an m-dimensional vector. Given a scaling constant ϵ and a configuration q, an action u produces a new configuration, $q' = q + \epsilon u$. This can be considered as a *configuration transition equation*, $q' = f(q, u)$. The path segment represented by the action u is the shortest path (usually a line segment) between q and q'. Following Section 2.3, let π_K denote a *K-step plan*, which is a sequence (u_1, u_2, …, u_K) of K actions. Note that if π_K and q_I are given, then a sequence of states, q_1, q_2, …, q_{K+1}, can be derived using f. Initially, $q_1 = q_I$, and each following state is obtained by $q_{k+1} = f(q_k, u_k)$. From this a path, $\tau : [0, 1] \rightarrow \mathcal{C}$, can be derived.

An approximate optimal planning problem is formalized as follows:

Formulation 7.4 (Approximate Optimal Motion Planning)

1. The following components are defined the same as in Formulation 4.1: $\mathcal{W}$, $\mathcal{O}$, $\mathcal{A}$, $\mathcal{C}$, $\mathcal{C}_{obs}$, $\mathcal{C}_{free}$, and q_I. It is assumed that $\mathcal{C}$ is an n-dimensional manifold.

2. For each $q \in \mathcal{C}$, a possibly infinite *action space*, $U(q)$. Each $u \in U$ is an n-dimensional vector.

3. A positive constant $\epsilon > 0$ called the *step size*.

4. A set of *stages*, each denoted by k, which begins at $k = 1$ and continues indefinitely. Each stage is indicated by a subscript, to obtain q_k and u_k.

5. A *configuration transition function* $f(q, u) = q + \epsilon u$ in which $q + \epsilon u$ is computed by vector addition on $\mathbb{R}^m$.

6. Instead of a goal state, a goal region X_G is defined.

7. Let L denote a real-valued cost functional, which is applied to a K-step plan, π_K. This means that the sequence $(u_1, \ldots, u_K)$ of actions and the sequence $(q_1, \ldots, q_{K+1})$ of configurations may appear in an expression of L. Let $F = K + 1$. The *cost functional* is

$$L(\pi_K) = \sum_{k=1}^{K} l(q_k, u_k) + l_F(q_F). \tag{7.26}$$

   The final term $l_F(q_F)$ is outside of the sum and is defined as $l_F(q_F) = 0$ if $q_F \in X_G$ and $l_F(q_F) = \infty$ otherwise. As in Formulation 2.3, K is not necessarily a constant.

8. Each $U(q)$ contains the special *termination action* u_T, which behaves the same way as in Formulation 2.3. If u_T is applied to q_k at stage k, then the action is repeatedly applied forever, the configuration remains in q_k forever, and no more cost accumulates.

The task is to compute a sequence of actions that optimizes (7.26). Formulation 7.4 can be used to define a variety of optimal planning problems. The parameter ϵ can be considered as the resolution of the approximation. In many formulations it can be interpreted as a *time step*, $\epsilon = \Delta t$; however, note that no explicit time reference is necessary because the problem only requires constructing a path through $\mathcal{C}_{free}$. As ϵ approaches zero, the formulation approaches an exact optimal planning problem. To properly express the exact problem, differential equations are needed. This is deferred until Part IV.

Example 7.4 (Manhattan Motion Model) Suppose that in addition to u_T, the action set $U(q)$ contains $2n$ vectors in which only one component is nonzero and must take the value 1 or -1. For example, if $\mathcal{C} = \mathbb{R}^2$, then

$$U(q) = \{(1, 0), (-1, 0), (0, -1), (0, 1), u_T\}. \tag{7.27}$$

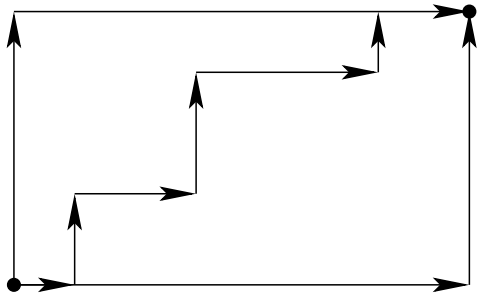


Figure 7.40: Under the Manhattan (L_1) motion model, all monotonic paths that follow the grid directions have equivalent length.

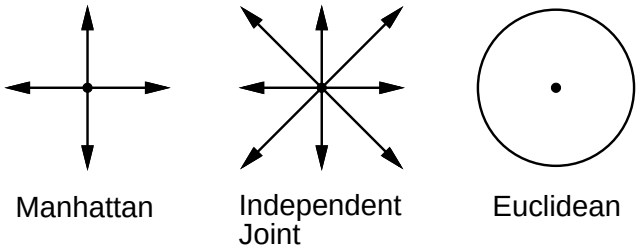


Figure 7.41: Depictions of the action sets, $U(q)$, for Examples 7.4, 7.5, and 7.6.

When used in the configuration transition equation, this set of actions produces "up," "down," "left," and "right" motions and a "terminate" command. This produces a topological graph according to the 1-neighborhood model, (5.37), which was given in Section 5.4.2. The action set for this example and the following two examples are shown in Figure 7.41 for comparison. The cost term $l(q_k, u_k)$ is defined to be 1 for all $q_k \in \mathcal{C}_{free}$ and u_k. If $q_k \in \mathcal{C}_{obs}$, then $l(q_k, u_k) = \infty$. Note that the set of configurations reachable by these actions lies on a grid, in which the spacing between 1-neighbors is ϵ. This corresponds to a convenient special case in which time discretization (implemented by ϵ) leads to a regular space discretization. Consider Figure 7.40. It is impossible to take a shorter path along a diagonal because the actions do not allow it. Therefore, all monotonic paths along the grid produce the same costs.

Optimal paths can be obtained by simply applying the dynamic programming algorithms of Chapter 2. This example provides a nice unification of concepts from Section 2.2, which introduced grid search, and Section 5.4.2, which explained how to adapt search methods to motion planning. In the current setting, only algorithms that produce optimal solutions on the corresponding graph are acceptable.

This form of optimization might not seem relevant because it does not represent the Euclidean shortest-path problem for $\mathbb{R}^2$. The next model adds more actions,

and does correspond to an important class of optimization problems in robotics. ■

Example 7.5 (Independent-Joint Motion Model) Now suppose that $U(q)$ includes u_T and the set of all 3^n vectors for which every element is either -1, 0, or 1. Under this model, a path can be taken along any diagonal. This still does not change the fact that all reachable configurations lie on a grid. Therefore, the standard grid algorithms can be applied once again. The difference is that now there are $3^n - 1$ edges emanating from every vertex, as opposed to $2n$ in Example 7.4. This model is appropriate for robots that are constructed from a collection of links attached by revolute joints. If each joint is operated independently, then it makes sense that each joint could be moved either forward, backward, or held stationary. This corresponds exactly to the actions. However, this model cannot nicely approximate Euclidean shortest paths; this motivates the next example. ■

Example 7.6 (Euclidean Motion Model) To approximate Euclidean shortest paths, let $U(q) = \mathbb{S}^{n-1} \cup \{u_T\}$, in which $\mathbb{S}^{n-1}$ is the m-dimensional unit sphere centered at the origin of $\mathbb{R}^n$. This means that in k stages, any piecewise-linear path in which each segment has length ϵ can be formed by a sequence of inputs. Therefore, the set of reachable states is no longer confined to a grid. Consider taking $\epsilon = 1$, and pick any point, such as $(\pi, \pi) \in \mathbb{R}^2$. How close can you come to this point? It turns out that the set of points reachable with this model is dense in $\mathbb{R}^n$ if obstacles are neglected. This means that we can come arbitrarily close to any point in $\mathbb{R}^n$. Therefore, a finite grid cannot be used to represent the problem. Approximate solutions can still be obtained by numerically computing an optimal cost-to-go function over $\mathcal{C}$. This approach is presented in Section 8.5.2.

One additional issue for this problem is the precision defined for the goal. If the goal region is very small relative to ϵ, then complicated paths may have to be selected to arrive precisely at the goal. ■

Example 7.7 (Weighted-Region Problem) In outdoor and planetary navigation applications, it does not make sense to define obstacles in the crisp way that has been used so far. For each patch of terrain, it is more convenient to associate a cost that indicates the estimated difficulty of its traversal. This is sometimes considered as a "grayscale" model of obstacles. The model can be easily captured in the cost term $l(q_k, u_k)$. The action spaces can be borrowed from Examples 7.4 or 7.5. Stentz's algorithm [450], which is introduced in Section 12.3.2, generates optimal navigation plans for this problem, even assuming that the terrain is initially unknown. Theoretical bounds for optimal weighted-region planning problems are given in [372, 373]. An approximation algorithm appears

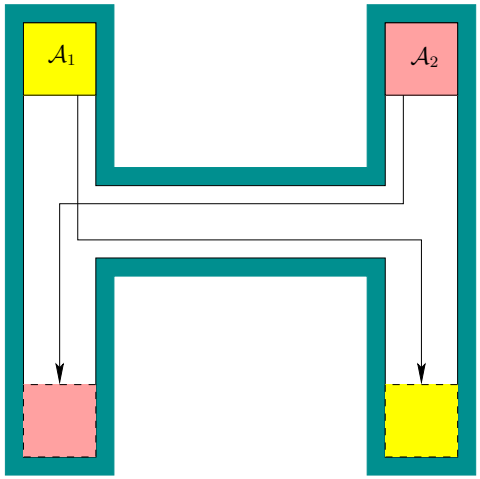


Figure 7.42: There are two Pareto-optimal coordination plans for this problem, depending on which robot has to wait.

in [412]. ■

7.7.2 Multiple-Robot Optimality

Suppose that there are two robots as shown in Figure 7.42. There is just enough room to enable the robots to translate along the corridors. Each would like to arrive at the bottom, as indicated by arrows; however, only one can pass at a time through the horizontal corridor. Suppose that at any instant each robot can either be *on* or *off*. When it is *on*, it moves at its maximum speed, and when it is *off*, it is stopped.[4] Now suppose that each robot would like to reach its goal as quickly as possible. This means each would like to minimize the total amount of time that it is *off*. In this example, there appears to be only two sensible choices: 1) $\mathcal{A}_1$ stays *on* and moves straight to its goal while $\mathcal{A}_2$ is *off* just long enough to let $\mathcal{A}_1$ pass, and then moves to its goal. 2) The opposite situation occurs, in which $\mathcal{A}_2$ stays *on* and $\mathcal{A}_1$ must wait. Note that when a robot waits, there are multiple locations at which it can wait and still yield the same time to reach the goal. The only important information is how long the robot was *off*.

Thus, the two interesting plans are that either $\mathcal{A}_2$ is *off* for some amount of time, $t_{off} > 0$, or $\mathcal{A}_1$ is *off* for time t_{off}. Consider a vector of costs of the form (L_1, L_2), in which each component represents the cost for each robot. The costs of the plans could be measured in terms of time wasted by waiting. This yields

[4]This model allows infinite acceleration. Imagine that the speeds are slow enough to allow this approximation. If this is still not satisfactory, then jump ahead to Chapter 13.

$(0, t_{off})$ and $(t_{off}, 0)$ for the cost vectors associated with the two plans (we could equivalently define cost to be the total time traveled by each robot; the time on is the same for both robots and can be subtracted from each for this simple example). The two plans are better than or equivalent to any others. Plans with this property are called *Pareto optimal* (or *nondominated*). For example, if $\mathcal{A}_2$ waits 1 second too long for $\mathcal{A}_1$ to pass, then the resulting costs are $(0, t_{off} + 1)$, which is clearly worse than $(0, t_{off})$. The resulting plan is not Pareto optimal. More details on Pareto optimality appear in Section 9.1.1.

Another way to solve the problem is to scalarize the costs by mapping them to a single value. For example, we could find plans that optimize the average wasted time. In this case, one of the two best plans would be obtained, yielding t_{off} average wasted time. However, no information is retained about which robot had to make the sacrifice. Scalarizing the costs usually imposes some kind of artificial preference or prioritization among the robots. Ultimately, only one plan can be chosen, which might make it seem inappropriate to maintain multiple solutions. However, finding and presenting the alternative Pareto-optimal solutions could provide valuable information if, for example, these robots are involved in a complicated application that involves many other time-dependent processes. Presenting the Pareto-optimal solutions is equivalent to discarding all of the worse plans and showing the best alternatives. In some applications, priorities between robots may change, and if a scheduler of robots has access to the Pareto-optimal solutions, it is easy to change priorities by switching between Pareto-optimal plans without having to generate new plans each time.

Now the Pareto-optimality concept will be made more precise and general. Suppose there are m robots, $\mathcal{A}^1$, ..., $\mathcal{A}^m$. Let γ refer to a motion plan that gives the paths and timing functions for all robots. For each $\mathcal{A}^i$, let L_i denote its cost functional, which yields a value $L_i(\gamma) \in [0, \infty]$ for a given plan, γ. An m-dimensional vector, $L(\gamma)$, is defined as

$$L(\gamma) = (L_1(\gamma), L_2(\gamma), \ldots, L_m(\gamma)). \tag{7.28}$$

Two plans, γ and γ', are called *equivalent* if $L(\gamma) = L(\gamma')$. A plan γ is said to *dominate* a plan γ' if they are not equivalent and $L_i(\gamma) \leq L_i(\gamma')$ for all i such that $1 \leq i \leq m$. A plan is called *Pareto optimal* if it is not dominated by any others. Since many Pareto-optimal plans may be equivalent, the task is to determine one representative from each equivalence class. This will be called finding the *unique* Pareto-optimal plans. For the example in Figure 7.42, there are two unique Pareto-optimal plans, which were already given.

Scalarization For the motion planning problem, a Pareto-optimal solution is also optimal for a scalar cost functional that is constructed as a linear combination of the individual costs. Let α_1, ..., α_m be positive real constants, and let

$$l(\gamma) = \sum_{i=1}^{m} \alpha_i L_i(\gamma). \tag{7.29}$$

It is easy to show that any plan that is optimal with respect to (7.29) is also a Pareto-optimal solution [309]. If a Pareto optimal solution is generated in this way, however, there is no easy way to determine what alternatives exist.

Computing Pareto-optimal plans Since optimization for one robot is already very difficult, it may not be surprising that computing Pareto-optimal plans is even harder. For some problems, it is even possible that a continuum of Pareto-optimal solutions exist (see Example 9.3), which is very discouraging. Fortunately, for the problem of coordinating robots on topological graphs, as considered in Section 7.2.2, there is only a finite number of solutions [197]. A grid-based algorithm, which is based on dynamic programming and computes all unique Pareto-optimal coordination plans, is presented in [309]. For the special case of two polygonal robots moving on a tree of piecewise-linear paths, a complete algorithm is presented in [114].

Further Reading

This chapter covered some of the most direct extensions of the basic motion planning problem. Extensions that involve uncertainties are covered throughout Part III, and the introduction of differential constraints to motion planning is the main focus of Part IV. Numerous other extensions can be found by searching through robotics research publications or the Internet.

The treatment of time-varying motion planning in Section 7.1 assumes that all motions are predictable. Most of the coverage is based on early work [80, 260, 410, 411]; other related work includes [185, 186, 278, 408, 438, 445]. To introduce uncertainties into this scenario, see Chapter 10. The logic-based representations of Section 2.4 have been extended to *temporal logics* to allow time-varying aspects of discrete planning problems (see Part IV of [193]).

For more on multiple-robot motion planning, see [10, 27, 32, 165, 166, 177, 182, 209, 309, 396, 441]. Closely related is the problem of planning for modular reconfigurable robots [97, 112, 196, 290, 484]. In both contexts, nonpositive curvature (NPC) is an important condition that greatly simplifies the structure of optimal paths [76, 196, 197]. For points moving on a topological graph, the topology of $\mathcal{C}_{free}$ is described in [3]. Over the last few years there has also been a strong interest in the coordination of a team or swarm of robots [85, 121, 149, 157, 160, 173, 180, 349].

The complexity of assembly planning is studied in [203, 264, 379, 475]. The problem is generally NP-hard; however, for some special cases, polynomial-time algorithms have been developed [7, 219, 476, 477]. Other works include [100, 217, 233, 236, 285].

Hybrid systems have attracted widespread interest over the past decade. Most of this work considers how to design control laws for piecewise-smooth systems [74, 325]. Early sources of hybrid control literature appear in [210]. The manipulation planning framework of Section 7.3.2 is based on [11, 12, 86]. The manipulation planning framework presented in this chapter ignores grasping issues. For analyses and algorithms for grasping, see [142, 251, 352, 397, 402, 403, 414, 415, 455]. Manipulation on a microscopic scale is considered in [70].

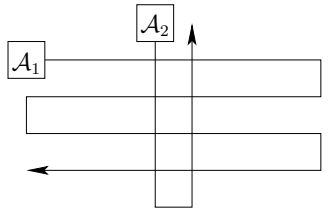


Figure 7.43: Two translating robots moving along piecewise-linear paths.

To read beyond Section 7.4 on sampling-based planning for closed kinematic chains, see [133, 135, 222, 479]. A complete planner for some closed chains is presented in [363]. For related work on inverse kinematics, see [162, 360]. The power of redundant degrees of freedom in robot systems was shown in [81].

Section 7.5 is a synthesis of several applications. The application of motion planning techniques to problems in computational biology is a booming area; see [18, 19, 26, 134, 263, 305, 308, 342, 488] for some representative papers. The knot-planning coverage is based on [298]. The box-folding presentation is based on [348]. A robotic system and planning technique for creating origami is presented in [46].

The coverage planning methods presented in Section 7.6 are based on [119] and [187, 188]. A survey of coverage planning appears in [117]. Other references include [4, 5, 84, 189, 228, 243, 478]. For discrete environments, approximation algorithms for the problem of optimally visiting all states in a goal set (the *orienteering problem*) are presented and analyzed in [64, 102].

Beyond two dimensions, optimal motion planning is extremely difficult. See Section 8.5.2 for dynamic programming-based approximations. See [115, 393] for approximate shortest paths in $\mathbb{R}^3$. The weighted region problem is considered in [372, 373]. Pareto-optimal coordination is considered in [114, 197, 309].

Exercises

1. Consider the obstacle region, (7.1), in the state space for time-varying motion planning.

   (a) To ensure that X_{obs} is polyhedral, what kind of paths should be allowed? Show how the model primitives H_i that define $\mathcal{O}$ are expressed in general, using t as a parameter.

   (b) Repeat the exercise, but for ensuring that X_{obs} is semi-algebraic.

2. Propose a way to adapt the sampling-based roadmap algorithm of Section 5.6 to solve the problem of time-varying motion planning with bounded speed.

3. Develop an efficient algorithm for computing the obstacle region for two translating polygonal robots that each follow a linear path.

4. Sketch the coordination space for the two robots moving along the fixed paths shown in Figure 7.43.

5. Suppose there are two robots, and each moves on its own roadmap of three paths. The paths in each roadmap are arranged end-to-end in a triangle.

   (a) Characterize the fixed-roadmap coordination space that results, including a description of its topology.

   (b) Now suppose there are n robots, each on a triangular roadmap, and characterize the fixed-roadmap coordination space.

6. Consider the state space obtained as the Cartesian product of the C-spaces of n identical robots. Suppose that each robot is labeled with a unique integer. Show that X can be partitioned nicely into $n!$ regions in which X_{obs} appears identical and the only difference is the labels (which indicate the particular robots that are in collision).

7. Suppose there are two robots, and each moves on its own roadmap of three paths. The paths in one roadmap are arranged end-to-end in a triangle, and the paths in the other are arranged as a Y. Characterize the fixed-roadmap coordination space that results, including a description of its topology.

8. Design an efficient algorithm that takes as input a graph representation of the connectivity of a linkage and computes an active-passive decomposition. Assume that all links are revolute. The algorithm should work for either 2D or 3D linkages (the dimension is also an input). Determine the asymptotic running time of your algorithm.

9. Consider the problem of coordinating the motion of two robots that move along precomputed paths but in the presence of predictable moving obstacles. Develop a planning algorithm for this problem.

10. Consider a manipulator in $\mathcal{W} = \mathbb{R}^2$ made of four links connected in a chain by revolute joints. There is unit distance between the joints, and the first joint is attached at $(0, 0)$ in $\mathcal{W} = \mathbb{R}^2$. Suppose that the end of the last link, which is position $(1, 0)$ in its body frame, is held at $(0, 2) \in \mathcal{W}$.

    (a) Use kinematics expressions to express the closure constraints for a configuration $q \in \mathcal{C}$.

    (b) Convert the closure constraints into polynomial form.

    (c) Use differentiation to determine the constraints on the allowable velocities that maintain closure at a configuration $q \in \mathcal{C}$.

Implementations

11. Implement the vertical decomposition algorithm to solve the path-tuning problem, as shown in Figure 7.5.

12. Use grid-based sampling and a search algorithm to compute collision-free motions of three robots moving along predetermined paths.

13. Under the conditions of Exercise 12, compute Pareto-optimal coordination strategies that optimize the time (number of stages) that each robot takes to reach its goal. Design a wavefront propagation algorithm that keeps track of the complete (ignoring equivalent strategies) set of minimal Pareto-optimal coordination strategies at each reached state. Avoid storing entire plans at each discretized state.

14. To gain an appreciation of the difficulties of planning for closed kinematic chains, try motion planning for a point on a torus among obstacles using only the implicit torus constraint given by (6.40). To simplify collision detection, the obstacles can be a collection of balls in $\mathbb{R}^3$ that intersect the torus. Adapt a sampling-based planning technique, such as the bidirectional RRT, to traverse the torus and solve planning problems.

15. Implement the spanning-tree coverage planning algorithm of Section 7.6.

16. Develop an RRT-based planning algorithm that causes the robot to chase an unpredictable moving target in a planar environment that contains obstacles. The algorithm should run quickly enough so that replanning can occur during execution. The robot should execute the first part of the most recently computed path while simultaneously computing a better plan for the next time increment.

17. Modify Exercise 16 so that the robot assumes the target follows a predictable, constant-velocity trajectory until some deviation is observed.

18. Show how to handle unexpected obstacles by using a fast enough planning algorithm. For simplicity, suppose the robot is a point moving in a polygonal obstacle region. The robot first computes a path and then starts to execute it. If the obstacle region changes, then a new path is computed from the robot's current position. Use vertical decomposition or another algorithm of your choice (provided it is fast enough). The user should be able to interactively place or move obstacles during plan execution.

19. Use the manipulation planning framework of Section 7.3.2 to develop an algorithm that solves the famous Towers of Hanoi problem by a robot that carries the rings [86]. For simplicity, suppose a polygonal robot moves polygonal parts in $\mathcal{W} = \mathbb{R}^2$ and rotation is not allowed. Make three pegs, and initially place all parts on one peg, sorted from largest to smallest. The goal is to move all of the parts to another peg while preserving the sorting.

20. Use grid-based approximation to solve optimal planning problems for a point robot in the plane. Experiment with using different neighborhoods and metrics. Characterize the combinations under which good and bad approximations are obtained.

Chapter 8

Feedback Motion Planning

So far in Part II it has been assumed that a continuous path sufficiently solves a motion planning problem. In many applications, such as computer-generated animation and virtual prototyping, there is no need to challenge this assumption because models in a virtual environment usually behave as designed. In applications that involve interaction with the physical world, future configurations may not be predictable. A traditional way to account for this in robotics is to use the refinement scheme that was shown in Figure 1.19 to design a feedback control law that attempts to follow the computed path as closely as possible. Sometimes this is satisfactory, but it is important to recognize that this approach is highly decoupled. Feedback and dynamics are neglected in the construction of the original path; the computed path may therefore not even be usable.

Section 8.1 motivates the consideration of feedback in the context of motion planning. Section 8.2 presents the main concepts of this chapter, but only for the case of a discrete state space. This requires less mathematical concepts than the continuous case, making it easier to present feedback concepts. Section 8.3 then provides the mathematical background needed to extend the feedback concepts to continuous state spaces (which includes C-spaces). Feedback motion planning methods are divided into complete methods, covered in Section 8.4, and sampling-based methods, covered in Section 8.5.

8.1 Motivation

For most problems involving the physical world, some form of feedback is needed. This means the actions of a plan should depend in some way on information gathered during execution. The need for feedback arises from the unpredictability of future states. In this chapter, every state space will be either discrete, or $X = \mathcal{C}$, which is a configuration space as considered in Chapter 4.

Two general ways to model uncertainty in the predictability of future states are

1. **Explicitly:** Develop models that explicitly account for the possible ways that the actual future state can drift away from the planned future state. A planning algorithm must take this uncertainty directly into account. Such explicit models of uncertainty are introduced and incorporated into the planning model in Part III.

2. **Implicitly:** The model of state transitions indicates that no uncertainty is possible; however, a feedback plan is constructed to ensure that it knows which action to apply, just in case it happens to be in some unexpected state during execution. This approach is taken in this chapter.

	Open Loop	Feedback
Free motions	Traditional motion planning	Chapter 8
Dynamics	Chapters 14 and 15	Traditional control theory

Figure 8.1: By separating the issue of dynamics from feedback, two less-investigated topics emerge.

The implicit way to handle this uncertainty may seem strange at first; therefore, some explanation is required. It will be seen in Part III that explicitly modeling uncertainty is extremely challenging and complicated. The requirements for expressing reliable models are much stronger; the complexity of the problem increases, making algorithm design more difficult and leading to greater opportunities to make modeling errors. The implicit way of handling uncertainty in predictability arose in control theory [63, 66, 356]. It is well known that a feedback control law is needed to obtain reliable performance, yet it is peculiar that the formulation of dynamics used in most contexts does not explicitly account for this. Classical control theory has always assumed that feedback is crucial; however, only in modern branches of the field, such as *stochastic control* and *robust control*, does this uncertainty get explicitly modeled. Thus, there is a widely accepted and successful practice of designing feedback control laws that use state feedback to implicitly account for the fact that future states may be unpredictable. Given the widespread success of this control approach across numerous applications over the past century, it seems valuable to utilize this philosophy in the context of motion planning as well (if you still do not like it, then jump to Chapter 10).

Due to historical reasons in the development of feedback control, it often seems that feedback and dynamics are inseparable. This is mainly because control theory was developed to reliably alter the behavior of dynamical systems. In traditional motion planning, neither feedback nor dynamics is considered. A solution path is considered *open loop*, which means there is no feedback of information during execution to *close* the loop. Dynamics are also not handled because the additional complications of differential constraints and higher dimensional phase spaces arise (see Part IV).

By casting history aside and separating feedback from dynamics, four separate topics can be made, as shown in Figure 8.1. The topic of open-loop planning that

involves dynamics has received increasing attention in recent years. This is the focus throughout most of Part IV. Those fond of classical control theory may criticize it for failing to account for feedback; however, such open-loop trajectories (paths in a phase space) are quite useful in applications that involve simulations. Furthermore, a trajectory that accounts for dynamics is more worthwhile in a decoupled approach than using a path that ignores dynamics, which has been an acceptable practice for decades. These issues will be elaborated upon further in Part IV.

The other interesting topic that emerges in Figure 8.1 is to develop feedback plans for problems in which there are no explicit models of dynamics or other differential constraints. If it was reasonable to solve problems in classical motion planning by ignoring differential constraints, one should certainly feel no less guilty designing feedback motion plans that still neglect differential constraints.[1] This uses the implicit model of uncertainty in predictability without altering any of the other assumptions previously applied in traditional motion planning.

Even if there are no unpredictability issues, another important use of feedback plans is for problems in which the initial state is not known. A feedback plan indicates what action to take from every state. Therefore, the specification of an initial condition is not important. The analog of this in graph algorithms is the single-destination shortest-path problem, which indicates how to arrive at a particular vertex optimally from any other vertex. Due to this connection, the next section presents feedback concepts for discrete state spaces, before extending the ideas to continuous spaces, which are needed for motion planning.

For these reasons, feedback motion planning is considered in this chapter. As a module in a decoupled approach used in robotics, feedback motion plans are at least as useful as a path computed by the previous techniques. We expect feedback solutions to be more reliable in general, when used in the place of open-loop paths computed by traditional motion planning algorithms.

8.2 Discrete State Spaces

This section is provided mainly to help to explain similar concepts that are coming in later sections. The presentation is limited to discrete spaces, which are much simpler to formulate and understand. Following this, an extension to configuration spaces and other continuous state spaces can be made. The discussion here is also relevant background for the feedback planning concepts that will be introduced in Section 8.4.1. In that case, uncertainty will be explicitly modeled. The resulting formulation and concepts can be considered as an extension of this section.

[1] Section 8.4.4 will actually consider some simple differential constraints, such as acceleration bounds; the full treatment of differential constraints is deferred until Part IV.

8.2.1 Defining a Feedback Plan

Consider a discrete planning problem similar to the ones defined in Formulations 2.1 and 2.3, except that the initial state is not given. Due to this, the cost functional cannot be expressed only as a function of a plan. It is instead defined in terms of the *state history* and *action history*. At stage k, these are defined as

$$\tilde{x}_k = (x_1, x_2, \ldots, x_k) \tag{8.1}$$

and

$$\tilde{u}_k = (u_1, u_2, \ldots, u_k), \tag{8.2}$$

respectively. Sometimes, it will be convenient to alternatively refer to $\tilde{x}_k$ as the *state trajectory*.

The resulting formulation is

Formulation 8.1 (Discrete Optimal Feedback Planning)

1. A finite, nonempty *state space* X.
2. For each state, $x \in X$, a finite *action space* $U(x)$.
3. A *state transition function* f that produces a state, $f(x, u) \in X$, for every $x \in X$ and $u \in U(x)$. Let U denote the union of $U(x)$ for all $x \in X$.
4. A set of *stages*, each denoted by k, that begins at $k = 1$ and continues indefinitely.
5. A *goal set*, $X_G \subset X$.
6. Let L denote a stage-additive *cost functional*,

$$L(\tilde{x}_F, \tilde{u}_K) = \sum_{k=1}^{K} l(x_k, u_k) + l_F(x_F), \tag{8.3}$$

in which $F = K + 1$.

There is one other difference in comparison to the formulations of Chapter 2. The state space is assumed here to be finite. This facilitates the construction of a feedback plan, but it is not necessary in general.

Consider defining a plan that solves Formulation 8.1. If the initial condition is given, then a sequence of actions could be specified, as in Chapter 2. Without having the initial condition, one possible approach is to determine a sequence of actions for each possible initial state, $x_1 \in X$. Once the initial state is given, the appropriate action sequence is known. This approach, however, wastes memory. Suppose some x is given as the initial state and the first action is applied, leading to the next state x'. What action should be applied from x'? The second action in the sequence at x can be used; however, we can also imagine that x' is now the

initial state and use its first action. This implies that keeping an action sequence for every state is highly redundant. It is sufficient at each state to keep only the first action in the sequence. The application of that action produces the next state, at which the next appropriate action is stored. An execution sequence can be imagined from an initial state as follows. Start at some state, apply the action stored there, arrive at another state, apply its action, arrive at the next state, and so on, until the goal is reached.

It therefore seems appropriate to represent a feedback plan as a function that maps every state to an action. Therefore, a *feedback plan* π is defined as a function $\pi : X \rightarrow U$. From every state, $x \in X$, the plan indicates which action to apply. If the goal is reached, then the termination action should be applied. This is specified as part of the plan: $\pi(x) = u_T$, if $x \in X_G$. A feedback plan is called a *solution* to the problem if it causes the goal to be reached from every state that is reachable from the goal.

If an initial state x_1 and a feedback plan π are given, then the state and action histories can be determined. This implies that the execution cost, (8.3), also can be determined. It can therefore be alternatively expressed as $L(\pi, x_1)$, instead of $L(\tilde{x}_F, \tilde{u}_K)$. This relies on future states always being predictable. In Chapter 10, it will not be possible to make this direct correspondence due to uncertainties (see Section 10.1.3).

Feasibility and optimality The notions of feasible and optimal plans need to be reconsidered in the context of feedback planning because the initial condition is not given. A plan π is called a solution to the *feasible* planning problem if from *every* $x \in X$ from which X_G is reachable the goal set is indeed reached by executing π from x. This means that the cost functional is ignored (an alternative to Formulation 8.1 can be defined in which the cost functional is removed). For convenience, π will be called a *feasible* feedback plan.

Now consider optimality. From a given state x, it is clear that an optimal plan exists using the concepts of Section 2.3. Is it possible that a different optimal plan needs to be associated with every $x \in X$ that can reach X_G? It turns out that only one plan is needed to encode optimal paths from every initial state to X_G. Why is this true? Suppose that the optimal cost-to-go is computed over X using Dijkstra's algorithm or value iteration, as covered in Section 2.3. Every cost-to-go value at some $x \in X$ indicates the cost received under the implementation of the optimal open-loop plan from x. The first step in this optimal plan can be determined by (2.19), which yields a new state $x' = f(x, u)$. From x', (2.19) can be applied once again to determine the next optimal action. The cost at x' represents both the optimal cost-to-go if x' is the initial condition and also the optimal cost-to-go when continuing on the optimal path from x. The two must be equivalent because of the dynamic programming principle. Since all such costs must coincide, a single feedback plan can be used to obtain the optimal cost-to-go from every initial condition.

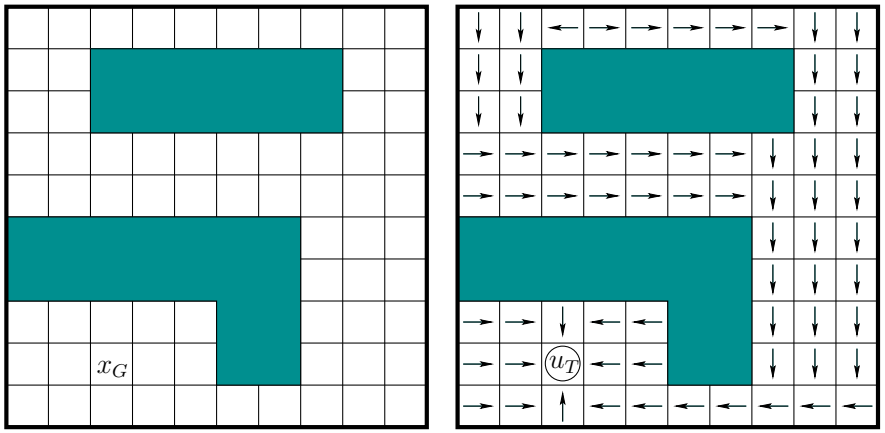


Figure 8.2: a) A 2D grid-planning problem. b) A solution feedback plan.

A feedback plan π is therefore defined as *optimal* if from *every* $x \in X$, the total cost, $L(\pi, x)$, obtained by executing π is the lowest among all possible plans. The requirement that this holds for every initial condition is important for feedback planning.

Example 8.1 (Feedback Plan on a 2D Grid) This example uses the 2D grid model explained in Example 2.1. A robot moves on a grid, and the possible actions are up ($\uparrow$), down ($\downarrow$), left ($\leftarrow$), right ($\rightarrow$), and terminate (u_T); some directions are not available from some states. A solution feedback plan is depicted in Figure 8.2. Many other possible solutions plans exist. The one shown here happens to be optimal in terms of the number of steps to the goal. Some alternative feedback plans are also optimal (figure out which arrows can be changed). To apply the plan from any initial state, simply follow the arrows to the goal. In each stage, the application of the action represented by the arrow leads to the next state. The process terminates when u_T is applied at the goal. ■

8.2.2 Feedback Plans as Navigation Functions

It conveniently turns out that tools for computing a feedback plan were already given in Chapter 2. Methods such as Dijkstra's algorithm and value iteration produce information as a side effect that can be used to represent a feedback plan. This section explains how this information is converted into a feedback plan. To achieve this, a feedback plan will be alternatively expressed as a potential function over the state space (recall potential functions from Section 5.4.3). The

potential values are computed by planning algorithms and can be used to recover the appropriate actions during execution. In some cases, an *optimal* feedback plan can even be represented using potential functions.

Navigation functions Consider a (discrete) *potential function*, defined as $\phi : X \rightarrow [0, \infty]$. The potential function can be used to define a feedback plan through the use of a *local operator*, which is a function that selects the action that reduces the potential as much as possible. First, consider the case of a feasible planning problem. The potential function, ϕ, defines a feedback plan by selecting u through the *local operator*,

$$u^* = \operatorname*{argmin}_{u \in U(x)} \Big\{ \phi(f(x,u)) \Big\}, \tag{8.4}$$

which means that $u^* \in U(x)$ is chosen to reduce ϕ as much as possible. The local operator yields a kind of *greedy* descent of the potential. Note that the action u^* may not be unique. In the continuous-space analog to this, the corresponding local operator performs a descent along the negative gradient (often referred to as *gradient descent*).

In the case of optimal planning, the local operator is defined as

$$u^* = \operatorname*{argmin}_{u \in U(x)} \Big\{ l(x,u) + \phi(f(x,u)) \Big\}, \tag{8.5}$$

which looks similar to the dynamic programming condition, (2.19). It becomes identical to (2.19) if ϕ is interpreted as the optimal cost-to-go. A simplification of (8.5) can be made if the planning problem is *isotropic*, which means that the cost is the same in every direction: $l(x,u) = l(x,u')$ for all $u, u' \in U(x) \setminus \{u_T\}$. In this case, the cost term $l(x,u)$ does not affect the minimization in (8.5). A common example in which this assumption applies is if the cost functional counts the number of stages required to reach the goal. The costs of particular actions chosen along the way are not important. Using the isotropic property, (8.5) simplifies back to (8.4).

When is a potential function useful? Many useless potential functions can be defined that fail to reach the goal, or cause states to cycle indefinitely, and so on. The most desirable potential function is one that for any initial state causes arrival in X_G, if it is reachable. This requires only a few simple properties. A potential function that satisfies these will be called a *navigation function*.[2]

Suppose that the cost functional is isotropic. Let $x' = f(x, u^*)$, which is the state reached after applying the action $u^* \in U(x)$ that was selected by (8.4). A potential function, ϕ, is called a *(feasible) navigation function* if

1. $\phi(x) = 0$ for all $x \in X_G$.

[2]This term was developed for continuous configuration spaces in [284, 416]; it will be used more broadly in this book but still retains the basic idea.

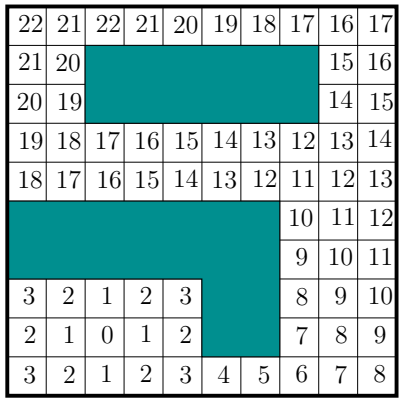


Figure 8.3: The cost-to-go values serve as a navigation function.

2. $\phi(x) = \infty$ if and only if no point in X_G is reachable from x.

3. For every reachable state, $x \in X \setminus X_G$, the local operator produces a state x' for which $\phi(x') < \phi(x)$.

The first condition requires the goal to have zero potential (this condition is actually not necessary but is included for convenience). The second condition requires that ∞ serves as a special indicator that the goal is not reachable from some state. The third condition means that the potential function has no local minima except at the goal. This means that the execution of the resulting feedback plan will progress without cycling and the goal region will eventually be reached.

An *optimal navigation function* is defined as the optimal cost-to-go, G^*. This means that in addition to the three properties above, the navigation function must also satisfy the principle of optimality:

$$\phi(x) = \min_{u \in U(x)} \Big\{ l(x,u) + \phi(f(x,u)) \Big\}, \tag{8.6}$$

which is just (2.18) with G^* replaced by ϕ. See Section 15.2.1 for more on this connection.

Example 8.2 (Navigation Function on a 2D Grid) Return to the planning problem in Example 8.1. Assume that an isotropic cost model is used: $l(x,u) = 1$ if $u \neq u_T$. Figure 8.3 shows a navigation function. The numbers shown in the tiles represent ϕ. Verify that ϕ satisfies the three requirements for a navigation function.

At any state, an action is applied that reduces the potential value. This corresponds to selecting the action using (8.4). The process may be repeated from any state until X_G is reached. This example clearly illustrates how a navigation function can be used as an alternative definition of a feedback plan. ■

Example 8.3 (Airport Terminal) You may have found yourself using a navigation function to find the exit after arriving in an unfamiliar airport terminal. Many terminals are tree-structured, with increasing gate numbers as the distance to the terminal exit increases. If you wish to leave the terminal, you should normally walk toward the lower numbered gates. ■

Computing navigation functions There are many ways to compute navigation functions. The cost-to-go function determined by Dijkstra's algorithm working backward from X_G yields an *optimal navigation function*. The third condition of a navigation function under the anisotropic case is exactly the stationary dynamic programming equation, (2.18), if the navigation function ϕ is defined as the optimal cost-to-go G^*. It was mentioned previously that the optimal actions can be recovered using only the cost-to-go. This was actually an example of using a navigation function, and the resulting procedure could have been considered as a feedback plan.

If optimality is not important, then virtually any backward search algorithm from Section 2.2 can be used, provided that it records the distance to the goal from every reached state. The distance does not have to be optimal. It merely corresponds to the cost obtained if the current vertex in the search tree is traced back to the root vertex (or back to any vertex in X_G, if there are multiple goal states).

If the planning problem does not even include a cost functional, as in Formulation 2.1, then a cost functional can be invented for the purposes of constructing a navigation function. At each $x \in X$ from which X_G is reachable, the number of edges in the search graph that would be traversed from x to X_G can be stored as the cost. If Dijkstra's algorithm is used to construct the navigation function, then the resulting feedback plan yields executions that are shortest in terms of the number of stages required to reach the goal.

The navigation function itself serves as the representation of the feedback plan, by recovering the actions from the local operator. Thus, a function, $\pi : X \rightarrow U$, can be recovered from a navigation function, $\phi : X \rightarrow [0, \infty]$. Likewise, a navigation function, ϕ, can be constructed from π. Therefore, the π and ϕ can be considered as interchangeable representations of feedback plans.

8.2.3 Grid-Based Navigation Functions for Motion Planning

To consider feedback plans for continuous spaces, vector fields and other basic definitions from differential geometry will be needed. These will be covered in Section 8.3; however, before handling such complications, we first will describe how to use the ideas presented so far in Section 8.2 as a discrete approximation to feedback motion planning.

WAVEFRONT PROPAGATION ALGORITHM

1. Initialize $W_0 = X_G$; $i = 0$.
2. Initialize $W_{i+1} = \emptyset$.
3. For every $x \in W_i$, assign $\phi(x) = i$ and insert all unexplored neighbors of x into W_{i+1}.
4. If $W_{i+1} = \emptyset$, then terminate; otherwise, let $i := i + 1$ and go to Step 2.

Figure 8.4: The wavefront propagation algorithm is a specialized version of Dijkstra's algorithm that optimizes the number of stages to reach the goal.

Examples 8.1 and 8.2 have already defined feedback plans and navigation functions for 2D grids that contain obstacles. Imagine that this model is used to approximate a motion planning problem for which $\mathcal{C} \subset \mathbb{R}^2$. Section 5.4.2 showed how to make a topological graph that approximates the motion planning problem with a grid of samples. The motions used in Example 8.1 correspond to the 1-neighborhood definition, (5.37). This idea was further refined in Section 7.7.1 to model approximate optimal motion planning by moving on a grid; see Formulation 7.4. By choosing the Manhattan motion model, as defined in Example 7.4, a grid with the same motions considered in Example 8.1 is produced.

To construct a navigation function that may be useful in mobile robotics, a high-resolution (e.g., 50 to 100 points per axis) grid is usually required. In Section 5.4.2, only a few points per axis were needed because feedback was not assumed. It was possible in some instances to find a collision-free path by investigating only a few points per axis. During the execution of a feedback plan, it is assumed that the future states of the robot are not necessarily predictable. Wherever the robot may end up, the navigation function in combination with the local operator must produce the appropriate action. If the current state (or configuration) is approximated by a grid, then it is important to reduce the approximation error as much as possible. This is accomplished by setting the grid resolution high. In the feedback case, the grid can be viewed as "covering" the whole configuration space, whereas in Section 5.4.2 the grid only represented a topological graph of paths that cut across the space.[3]

Wavefront propagation algorithms Once the approximation has been made, any of the methods discussed in Section 8.2.2 can be used to compute a navigation function. An optimal navigation function can be easily computed using Dijkstra's

[3]Difficulty in distinguishing between these two caused researchers for many years to believe that grids yield terrible performance for the open-loop path planning problems of Chapter 5. This was mainly because it was assumed that a high-resolution grid was necessary. For many problems, however, they could terminate early after only considering a few points per axis.

algorithm from the goal. If each motion has unit cost, then a useful simplification can be made. Figure 8.4 describes a wavefront propagation algorithm that computes an optimal navigation function. It can be considered as a special case of Dijkstra's algorithm that avoids explicit construction of the priority queue. In Dijkstra's algorithm, the cost of the smallest element in the queue is monotonically nondecreasing during the execution of the algorithm. In the case of each motion having unit cost, there will be many states in the queue that have the same cost. Dijkstra's algorithm could remove in parallel all elements that have the same, smallest cost. Suppose the common, smallest cost value is i. These states are organized into a *wavefront*, W_i. The initial wavefront is W_0, which represents the states in X_G. The algorithm can immediately assign an optimal cost-to-go value of 1 to every state that can be reached in one stage from any state in W_0. These must be optimal because no other cost value is optimal. The states that receive cost 1 can be organized into the wavefront W_1. The unexplored neighbors of W_1 are assigned cost 2, which also must be optimal. This process repeats inductively from i to $i+1$ until all reachable states have been reached. In the end, the optimal cost-to-go is computed in $O(n)$ time, in which n is the number of reachable grid states. For any states that were not reached, the value $\phi(x) = \infty$ can be assigned. The navigation function shown in Figure 8.3 can actually be computed using the wavefront propagation algorithm.

Maximum clearance One problem that typically arises in mobile robotics is that optimal motion plans bring robots too close to obstacles. Recall from Section 6.2.4 that the shortest Euclidean paths for motion planning in a polygonal environment must be allowed to touch obstacle vertices. This motivated the maximum clearance roadmap, which was covered in Section 6.2.3. A grid-based approximate version of the maximum clearance roadmap can be made. Furthermore, a navigation function can be defined that guides the robot onto the roadmap, then travels along the roadmap, and finally deposits the robot at a specified goal. In [304], the resulting navigation function is called *NF2*.

Assume that there is a single goal state, $x_G \in X$. The computation of a maximum clearance navigation function proceeds as follows:

1. Instead of X_G, assign W_0 to be the set of all states from which motion in at least one direction is blocked. These are the states on the boundary of the discretized collision-free space.

2. Perform wavefront iterations that propagate costs in waves outward from the obstacle boundaries.

3. As the wavefronts propagate, they will meet approximately at the location of the maximum clearance roadmap for the original, continuous problem. Mark any state at which two wavefront points arrive from opposing directions as a *skeleton state*. It may be the case that the wavefronts simply touch each other, rather than arriving at a common state; in this case, one of the two touching states is chosen as the skeleton state. Let S denote the set of all skeleton states.

4. After the wavefront propagation ends, connect x_G to the skeleton by inserting x_G and all states along the path to the skeleton into S. This path can be found using any search algorithm.

5. Compute a navigation function ϕ_1 over S by treating all other states as if they were obstacles and using the wavefront propagation algorithm. This navigation function guides any point in S to the goal.

6. Treat S as a goal region and compute a navigation function ϕ_2 using the wavefront propagation algorithm. This navigation function guides the state to the nearest point on the skeleton.

7. Combine ϕ_1 and ϕ_2 as follows to obtain ϕ. For every $x \in S$, let $\phi(x) = \phi_1(x)$. For every remaining state, the value $\phi(x) = \phi_1(x') + \phi_2(x)$ is assigned, in which x' is the nearest state to x such that $x' \in S$. The state x' can easily be recorded while ϕ_2 is computed.

If $\mathcal{C}_{free}$ is multiply connected, then there may be multiple ways to each x_G by traveling around different obstacles (the paths are not homotopic). The method described above does not take into account the problem that one route may have a tighter clearance than another. The given approach only optimizes the distance traveled along the skeleton; it does not, however, maximize the nearest approach to an obstacle, if there are multiple routes.

Dial's algorithm Now consider generalizing the wavefront propagation idea. Wavefront propagation can be applied to any discrete planning problem if $l(x, u) = 1$ for any $x \in X$ and $u \in U(x)$ (except $u = u_T$). It is most useful when the transition graph is sparse (imagine representing the transition graph using an adjacency matrix). The grid problem is a perfect example where this becomes important. More generally, if the cost terms assume integer values, then *Dial's algorithm* [148] results, which is a generalization of wavefront propagation, and a specialization of Dijkstra's algorithm. The idea is that the priority queue can be avoided by assigning the alive vertices to buckets that correspond to different possible cost-to-go values. In the wavefront propagation case, there are never more than two buckets needed at a time. Dial's algorithm allows all states in the smallest cost bucket to be processed in parallel. The scheme was enhanced in [460] to yield a linear-time algorithm.

Other extensions Several ideas from this section can be generalized to produce other navigation functions. One disadvantage of the methods discussed so far is that undesirable staircase motions (as shown in Figure 7.40) are produced. If the

2-neighborhood, as defined in (5.38), is used to define the action spaces, then the motions will generally be shorter. Dial's algorithm can be applied to efficiently compute an optimal navigation function in this case.

A grid approximation can be made to higher dimensional configuration spaces. Since a high resolution is needed, however, it is practical only for a few dimensions (e.g., 3 or 4). If the 1-neighborhood is used, then wavefront propagation can be easily applied to compute navigation functions. Dial's algorithm can be adapted for general k-neighborhoods.

Constructing navigation functions over grids may provide a practical solution in many applications. In other cases it may be unacceptable that staircase motions occur. In many cases, it may not even be possible to compute the navigation function quickly enough. Factors that influence this problem are 1) very high accuracy, and a hence high-resolution grid may be necessary; 2) the dimension of the configuration space may be high; and 3) the environment may be frequently changing, and a real-time response is required. To address these issues, it is appealing to abandon grid approximations. This will require defining potential functions and velocities directly on the configuration space. Section 8.3 presents the background mathematical concepts to make this transition.

8.3 Vector Fields and Integral Curves

To consider feedback motion plans over continuous state spaces, including configuration spaces, we will need to define a vector field and the trajectory that is obtained by integrating the vector field from an initial point. A vector field is ideal for characterizing a feedback plan over a continuous state space. It can be viewed as providing the continuous-space analog to the feedback plans on grids, as shown in Figure 8.2b.

This section presents two alternative presentations of the background mathematical concepts. Section 8.3.1 assumes that $X = \mathbb{R}^n$, which leads to definitions that appear very similar to those you may have learned in basic calculus and differential equations courses. Section 8.3.2 covers the more general case of vector fields on manifolds. This requires significantly more technical concepts and builds on the manifold definitions of Section 4.1.2.

Some readers may have already had some background in differentiable manifolds. If, however, you are seeing it for the first time, then it may be difficult to comprehend on the first reading. In addition to rereading, here are two other suggestions. First, try studying background material on this subject, which is suggested at the end of the chapter. Second, disregard the manifold technicalities in the subsequent sections and pretend that $X = \mathcal{C} = \mathbb{R}^n$. Nearly everything will make sense without the additional technicalities. Imagine that a manifold is defined as a cube, $[0, 1]^n$, with some sides identified, as in Section 4.1.2. The concepts that were presented for $\mathbb{R}^n$ can be applied everywhere except at the boundary of the cube. For example, if $\mathbb{S}^1$ is defined as $[0, 1]/\sim$, and a function f is defined on

$\mathbb{S}^1$, how can we define the derivative at $f(0)$? The technical definitions of Section 8.3.2 fix this problem. Sometimes, the technicalities can be avoided in practice by cleverly handling the identification points.

8.3.1 Vector Fields on $\mathbb{R}^n$

This section revisits some basic concepts from introductory courses such as calculus, linear algebra, and differential equations. You may have learned most of these for $\mathbb{R}^2$ and $\mathbb{R}^3$. We eventually want to describe velocities in $\mathbb{R}^n$ and on manifolds, and then use the notion of a vector field to express a feedback plan in Section 8.4.1.

Vector spaces Before defining a vector field, it is helpful to be precise about what is meant by a *vector*. A *vector space* (or *linear space*) is defined as a set, V, that is closed under two algebraic operations called *vector addition* and *scalar multiplication* and satisfies several axioms, which will be given shortly. The vector space used in this section is $\mathbb{R}^n$, in which the scalars are real numbers, and a vector is represented as a sequence of n real numbers. Scalar multiplication multiplies each component of the vector by the scalar value. Vector addition forms a new vector by adding each component of two vectors.

A *vector space* V can be defined over any field $\mathbb{F}$ (recall the definition from Section 4.4.1). The field $\mathbb{F}$ represents the *scalars*, and V represents the *vectors*. The concepts presented below generalize the familiar case of the vector space $\mathbb{R}^n$. In this case, $V = \mathbb{R}^n$ and $\mathbb{F} = \mathbb{R}$. In the definitions that follow, you may make these substitutions, if desired. We will not develop vector spaces that are more general than this; the definitions are nevertheless given in terms of V and $\mathbb{F}$ to clearly separate scalars from vectors. The *vector addition* is denoted by $+$, and the *scalar multiplication* is denoted by $\cdot$. These operations must satisfy the following axioms (a good exercise is to verify these for the case of $\mathbb{R}^n$ treated as a vector space over the field $\mathbb{R}$):

1. (**Commutative Group Under Vector Addition**) The set V is a commutative group with respect to vector addition, $+$.

2. (**Associativity of Scalar Multiplication**) For any $v \in V$ and any $\alpha, \beta \in \mathbb{F}$, $\alpha(\beta v) = (\alpha\beta)v$.

3. (**Distributivity of Scalar Sums**) For any $v \in V$ and any $\alpha, \beta \in \mathbb{F}$, $(\alpha + \beta)v = \alpha v + \beta v$.

4. (**Distributivity of Vector Sums**) For any $v, w \in V$ and any $\alpha \in \mathbb{F}$, $\alpha(v + w) = \alpha v + \alpha w$.

5. (**Scalar Multiplication Identity**) For any $v \in V$, $1v = v$ for the multiplicative identity $1 \in \mathbb{F}$.

The first axiom allows vectors to be added in any order. The rest of the axioms require that the scalar multiplication interacts with vectors in the way that we would expect from the familiar vector space $\mathbb{R}^n$ over $\mathbb{R}$.

A *basis* of a vector space V is defined as a set, $v_1,\ldots,v_n$, of vectors for which every $v \in V$ can be uniquely written as a *linear combination*:

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n, \tag{8.7}$$

for some $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$. This means that every vector has a unique representation as a linear combination of basis elements. In the case of $\mathbb{R}^3$, a familiar basis is [0 0 1], [0 1 0], and [1 0 0]. All vectors can be expressed as a linear combination of these three. Remember that a basis is not necessarily unique. From linear algebra, recall that any three linearly independent vectors can be used as a basis for $\mathbb{R}^3$. In general, the basis must only include linearly independent vectors. Even though a basis is not necessarily unique, the number of vectors in a basis is the same for any possible basis over the same vector space. This number, n, is called the *dimension* of the vector space. Thus, we can call $\mathbb{R}^n$ an n-dimensional vector space over $\mathbb{R}$.

Example 8.4 (The Vector Space $\mathbb{R}^n$ Over $\mathbb{R}$) As indicated already, $\mathbb{R}^n$ can be considered as a vector space. A natural basis is the set of n vectors in which, for each $i \in \{1, \ldots, n\}$, a unit vector is constructed as follows. Let $x_i = 1$ and $x_j = 0$ for all $j \neq i$. Since there are n basis vectors, $\mathbb{R}^n$ is an n-dimensional vector space. The basis is not unique. Any set of n linearly independent vectors may be used, which is familiar from linear algebra, in which nonsingular $n \times n$ matrices are used to transform between them. ■

To illustrate the power of these general vector space definitions, consider the following example.

Example 8.5 (A Vector Space of Functions) The set of all continuous, real-valued functions $f : [0, 1] \to \mathbb{R}$, for which

$$\int_0^1 f(x)dx \tag{8.8}$$

is finite, forms a vector space over $\mathbb{R}$. It is straightforward to verify that the vector space axioms are satisfied. For example, if two functions f_1 and f_2 are added, the integral remains finite. Furthermore, $f_1 + f_2 = f_2 + f_1$, and all of the group axioms are satisfied with respect to addition. Any function f that satisfies (8.8) can be multiplied by a scalar in $\mathbb{R}$, and the integral remains finite. The axioms that involve scalar multiplication can also be verified.

It turns out that this vector space is infinite-dimensional. One way to see this is to restrict the functions to the set of all those for which the Taylor series exists and converges to the function (these are called *analytic functions*). Each function can be expressed via a Taylor series as a polynomial that may have an infinite number of terms. The set of all monomials, x, x^2, x^3, and so on, represents a basis. Every continuous function can be considered as an infinite vector of coefficients; each coefficient is multiplied by one of the monomials to produce the function. This provides a simple example of a *function space*; with some additional definitions, this leads to a *Hilbert space*, which is crucial in functional analysis, a subject that characterizes spaces of functions [420, 422]. ■

The remainder of this chapter considers only finite-dimensional vector spaces over $\mathbb{R}$. It is important, however, to keep in mind the basic properties of vector spaces that have been provided.

Vector fields A vector field looks like a "needle diagram" over $\mathbb{R}^n$, as depicted in Figure 8.5. The idea is to specify a direction at each point $p \in \mathbb{R}^n$. When used to represent a feedback plan, it indicates the direction that the robot needs to move if it finds itself at p.

For every $p \in \mathbb{R}^n$, associate an n-dimensional vector space called the *tangent space* at p, which is denoted as $T_p(\mathbb{R}^n)$. Why not just call it a vector space at p? The use of the word "tangent" here might seem odd; it is motivated by the generalization to manifolds, for which the tangent spaces will be "tangent" to points on the manifold.

A *vector field*[4] $\vec{V}$ on $\mathbb{R}^n$ is a function that assigns a vector $v \in T_p(\mathbb{R}^n)$ to every $p \in \mathbb{R}^n$. What is the range of this function? The vector $\vec{V}(p)$ at each $p \in \mathbb{R}^n$ actually belongs to a different tangent space. The range of the function is therefore the union

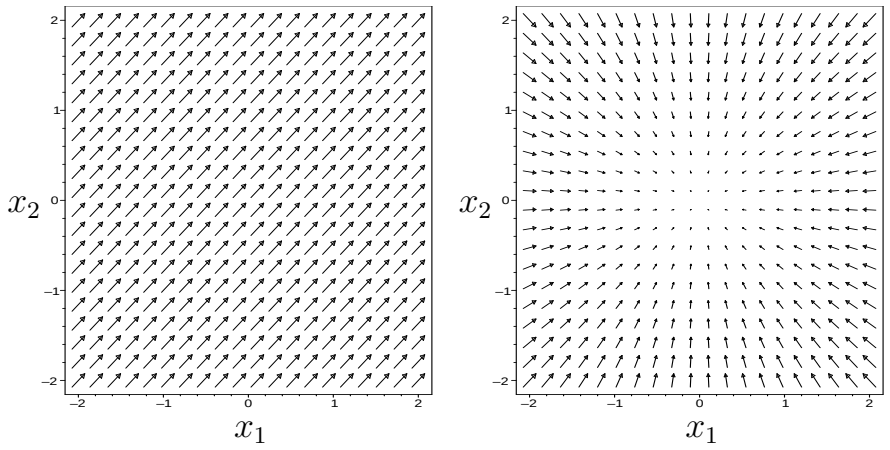
$$T(\mathbb{R}^n) = \bigcup_{p \in \mathbb{R}^n} T_p(\mathbb{R}^n), \tag{8.9}$$

which is called the *tangent bundle* on $\mathbb{R}^n$. Even though the way we describe vectors from $T_p(\mathbb{R}^n)$ may appear the same for any $p \in \mathbb{R}^n$, each tangent space is assumed to produce distinct vectors. To maintain distinctness, a point in the tangent bundle can be expressed with $2n$ coordinates, by specifying p and v together. This will become important for defining phase space concepts in Part IV. In the present setting, it is sufficient to think of the range of $\vec{V}$ as $\mathbb{R}^n$ because $T_p(\mathbb{R}^n) = \mathbb{R}^n$ for every $p \in \mathbb{R}^n$.

A vector field can therefore be expressed using n real-valued functions on $\mathbb{R}^n$. Let $f_i(x_1, \ldots, x_n)$ for i from 1 to n denote such functions. Using these, a vector field is specified as

$$f(x) = [f_1(x_1, \ldots, x_n) \;\; f_2(x_1, \ldots, x_n) \;\; \cdots \;\; f_n(x_1, \ldots, x_n)]. \tag{8.10}$$

[4] Unfortunately, the term *field* appears in two unrelated places: in the definition of a vector space and in the term *vector field*. Keep in mind that this is an accidental collision of terms.



(a) (b)

Figure 8.5: (a) A constant vector field, $f(x, y) = [1 \quad 1]$. (b) A vector field, $f(x, y) = [-x \quad -y]$ in which all vectors point to the origin.

In this case, it appears that a vector field is a function f from $\mathbb{R}^n$ into $\mathbb{R}^n$. Therefore, standard function notation will be used from this point onward to denote a vector field.

Now consider some examples of vector fields over $\mathbb{R}^2$. Let a point in $\mathbb{R}^2$ be represented as $p = (x, y)$. In standard vector calculus, a vector field is often specified as $[f_1(x, y) \quad f_2(x, y)]$, in which f_1 and f_2 are functions on $\mathbb{R}^2$

Example 8.6 (Constant Vector Field) Figure 8.5a shows a *constant vector field*, which assigns the vector $[1 \;\; 2]$ to every $(x, y) \in \mathbb{R}^2$. ■

Example 8.7 (Inward Flow) Figure 8.5b depicts a vector field that assigns $[-x \quad -y]$ to every $(x, y) \in \mathbb{R}^2$. This causes all vectors to point to the origin. ■

Example 8.8 (Swirl) The vector field in Figure 8.6 assigns $[(y - x) \quad (-x - y)]$ to every $(x, y) \in \mathbb{R}^2$. ■

Due to obstacles that arise in planning problems, it will be convenient to sometimes restrict the domain of a vector field to an open subset of $\mathbb{R}^n$. Thus, for any open subset $O \subset \mathbb{R}^n$, a vector field $f : O \to \mathbb{R}^n$ can be defined.

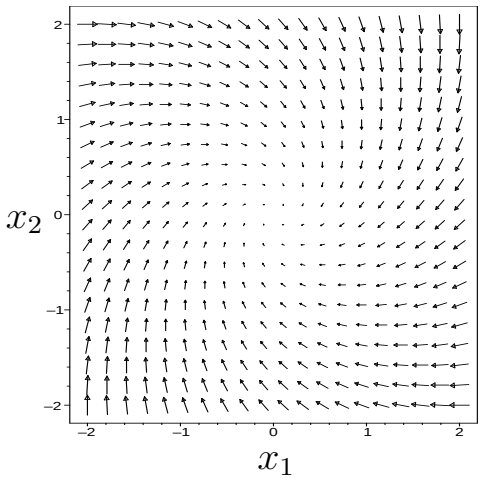


Figure 8.6: A swirling vector field, $f(x, y) = [(y - x) \;\; (-x - y)]$.

Smoothness A function f_i from a subset of $\mathbb{R}^n$ into $\mathbb{R}$ is called a *smooth function* if derivatives of any order can be taken with respect to any variables, at any point in the domain of f_i. A vector field is said to be *smooth* if every one of its n defining functions, $f_1, \ldots, f_n$, is smooth. An alternative name for a smooth function is a C^∞ *function*. The superscript represents the order of differentiation that can be taken. For a C^k *function*, its derivatives can be taken at least up to order k. A C^0 *function* is an alternative name for a continuous function. The notion of a homeomorphism can be extended to a *diffeomorphism*, which is a homeomorphism that is a smooth function. Two topological spaces are called *diffeomorphic* if there exists a diffeomorphism between them.

Vector fields as velocity fields We now give a particular interpretation to vector fields. A vector field expressed using (8.10) can be used to define a set of first-order differential equations as

$$\begin{aligned} \frac{dx_1}{dt} &= f_1(x_1, \ldots, x_n) \\ \frac{dx_2}{dt} &= f_2(x_1, \ldots, x_n) \\ &\vdots \\ \frac{dx_n}{dt} &= f_n(x_1, \ldots, x_n). \end{aligned} \tag{8.11}$$

Each equation represents the derivative of one coordinate with respect to time. For any point $x \in \mathbb{R}^n$, a *velocity vector* is defined as

$$\frac{dx}{dt} = \left[\frac{dx_1}{dt} \quad \frac{dx_2}{dt} \quad \cdots \quad \frac{dx_n}{dt} \right]. \tag{8.12}$$

This enables f to be interpreted as a *velocity field*.

It is customary to use the short notation $\dot{x} = dx/dt$. Each velocity component can be shortened to $\dot{x}_i = dx_i/dt$. Using f to denote the vector of functions f_1, ..., f_n, (8.11) can be shorted to

$$\dot{x} = f(x). \tag{8.13}$$

The use of f here is an intentional coincidence with the use of f for the state transition equation. In Part IV, we will allow vector fields to be parameterized by actions. This leads to a continuous-time state transition equation that looks like $\dot{x} = f(x, u)$ and is very similar to the transition equations defined over discrete stages in Chapter 2.

The differential equations expressed in (8.11) are often referred to as *autonomous* or *stationary* because f does not depend on time. A time-varying vector field could alternatively be defined, which yields $\dot{x} = f(x(t), t)$. This will not be covered, however, in this chapter.

An integral curve If a vector field f is given, then a velocity vector is defined at each point using (8.10). Imagine a point that starts at some $x_0 \in \mathbb{R}^n$ at time $t = 0$ and then moves according to the velocities expressed in f. Where should it travel? Its *trajectory* starting from x_0 can be expressed as a function $\tau : [0, \infty) \to \mathbb{R}^n$, in which the domain is a time interval, $[0, \infty)$. A trajectory represents an *integral curve* (or *solution trajectory*) of the differential equations with initial condition $\tau(0) = x_0$ if

$$\frac{d\tau}{dt}(t) = f(\tau(t)) \tag{8.14}$$

for every time $t \in [0, \infty)$. This is sometimes expressed in integral form as

$$\tau(t) = x_0 + \int_0^t f(\tau(s)) ds \tag{8.15}$$

and is called a solution to the differential equations in the *sense of Caratheodory*. Intuitively, the integral curve starts at x_0 and flows along the directions indicated by the velocity vectors. This can be considered as the continuous-space analog of following the arrows in the discrete case, as depicted in Figure 8.2b.

Example 8.9 (Integral Curve for a Constant Velocity Field) The simplest case is a constant vector field. Suppose that a constant field $x_1 = 1$ and $x_2 = 2$ is defined on $\mathbb{R}^2$. The integral curve from $(0, 0)$ is $\tau(t) = (t, 2t)$. It can be easily seen that (8.14) holds for all $t \geq 0$. ■

Example 8.10 (Integral Curve for a Linear Velocity Field) Consider a velocity field on $\mathbb{R}^2$. Let $\dot{x}_1 = -2x_1$ and $\dot{x}_2 = -x_2$. The function $\tau(t) = (e^{-2t}, e^{-t})$ represents the integral curve from $(1, 1)$. At $t = 0$, $\tau(0) = (1, 1)$, which is the initial state. If can be verified that for all $t > 0$, (8.14) holds. This is a simple example of a linear velocity field. In general, if each f_i is a linear function of the coordinate variables x_1, ..., x_n, then a linear velocity field is obtained. The integral curve is generally found by determining the eigenvalues of the matrix A when the velocity field is expressed as $\dot{x} = Ax$. See [106] for numerous examples. ■

A basic result from differential equations is that a unique integral curve exists to $\dot{x} = f(x)$ if f is smooth. An alternative condition is that a unique solution exists if f satisfies a *Lipschitz condition*. This means that there exists some constant $c \in (0, \infty)$ such that

$$\|f(x) - f(x')\| \leq c\|x - x'\| \tag{8.16}$$

for all $x, x' \in X$, and $\| \cdot \|$ denotes the Euclidean norm (vector magnitude). The constant c is often called a *Lipschitz constant*. Note that if f satisfies the Lipschitz condition, then it is continuous. Also, if all partial derivatives of f over all of X can be bounded by a constant, then f is Lipschitz. The expression in (8.16) is preferred, however, because it is more general (it does not even imply that f is differentiable everywhere).

Piecewise-smooth vector fields It will be important to allow vector fields that are smooth only over a finite number of patches. At a *switching boundary* between two patches, a discontinuous jump may occur. For example, suppose that an $(n-1)$-dimensional switching boundary, $S \subset \mathbb{R}^n$, is defined as

$$S = \{x \in \mathbb{R}^n | \; s(x) = 0\}, \tag{8.17}$$

in which s is a function $s : \mathbb{R}^n \to \mathbb{R}$. If $\mathbb{R}^n$ has dimension n and s is not singular, then S has dimension $n - 1$. Define

$$S_+ = \{x \in \mathbb{R}^n | \; s(x) > 0\} \tag{8.18}$$

and

$$S_- = \{x \in \mathbb{R}^n | \; s(x) < 0\}. \tag{8.19}$$

The definitions are similar to the construction of implicit models using geometric primitives in Section 3.1.2. Suppose that $f(x)$ is smooth over S_+ and S_- but experiences a discontinuous jump at S. Such differential equations model *hybrid systems* in control theory [74, 210, 325]. The task there is to design a *hybrid control system*. Can we still determine a solution trajectory in this case? Under special conditions, we can obtain what is called a solution to the differential equations in the *sense of Filipov* [174, 426].

Let $B(x, \delta)$ denote an open ball of radius δ centered at x. Let $f(B(x, \delta))$ denote the set

$$f(B(x, \delta)) = \{x' \in X \mid \exists x'' \in B(x, \delta) \text{ for which } x' = f(x'')\}. \tag{8.20}$$

Let X_0 denote any subset of $\mathbb{R}^n$ that has measure zero (i.e., $\mu(X_0) = 0$). Let hull(A) denote the convex hull of a set, A, of points in $\mathbb{R}^n$. A path $\tau : [0, t_f] \rightarrow \mathbb{R}^n$ is called a *solution in the sense of Filipov* if for almost all $t \in [0, t_f]$,

$$\frac{d\tau}{dt}(t) \in \bigcap_{\delta > 0} \left\{ \bigcap_{X_0} \text{hull}(f(B(\tau(t), \delta) \setminus X_0)) \right\}, \tag{8.21}$$

in which the intersections are taken over all possible $\delta > 0$ and sets, X_0, of measure zero. The expression (8.21) is actually called a *differential inclusion* [39] because a set of choices is possible for $\dot{x}$. The "for almost all" requirement means that the condition can even fail to hold on a set of measure zero in $[0, t_f]$. Intuitively, it says that almost all of the velocity vectors produced by τ must point "between" the velocity vectors given by f in the vicinity of $\tau(x(t))$. The "between" part comes from using the convex hull. Filipov's sense of solution is an incredible generalization of the solution concept in the sense of Caratheodory. In that case, every velocity vector produced by τ must agree with $f(x(t))$, as given in (8.14). The condition in (8.21) allows all sorts of sloppiness to appear in the solution, even permitting f to be discontinuous.

Many bizarre vector fields can yield solutions in the sense of Filipov. The switching boundary model is relatively simple among those permitted by Filipov's condition. Figure 8.7 shows various cases that can occur at the switching boundary S. For the case of consistent flow, solutions occur as you may intuitively expect. Filipov's condition, (8.21), requires that at S the velocity vector of τ points between vectors before and after crossing S (for example, it can point down, which is the average of the two directions). The magnitude must also be between the two magnitudes. For the inward flow case, the integral curve moves along S, assuming the vectors inside of S point in the same direction (within the convex hull) as the vectors on either side of the boundary. In applications that involve physical systems, this may lead to oscillations around S. This can be alleviated by regularization, which thickens the boundary [426] (the subject of *sliding-mode control* addresses this issue [159]). The outward flow case can lead to nonuniqueness if the initial state lies in S. However, trajectories that start outside of S will not cross S, and there will be no such troubles. If the flow is tangent on both sides of a boundary, then other forms of nonuniqueness may occur. The tangent-flow case will be avoided in this chapter.

8.3.2 Smooth Manifolds

The manifold definition given in Section 4.1.2 is often called a *topological manifold*. A manifold defined in this way does not necessarily have enough axioms to ensure

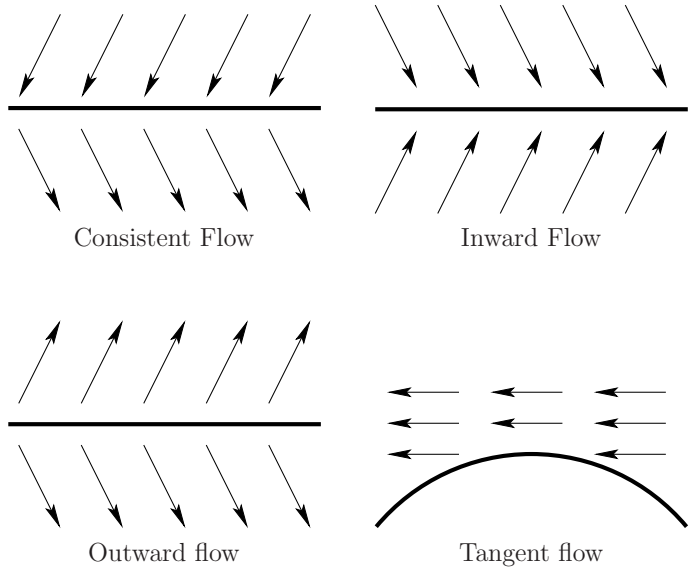


Figure 8.7: Different kinds of flows around a switching boundary.

that calculus operations, such as differentiation and integration, can be performed. We would like to talk about velocities on the configuration space $\mathcal{C}$ or in general for a continuous state space X. As seen in Chapter 4, the configuration space could be a manifold such as $\mathbb{RP}^3$. Therefore, we need to define some more qualities that a manifold should possess to enable calculus. This leads to the notion of a *smooth manifold*.

Assume that M is a topological manifold, as defined in Section 4.1.2. For example, M could represent $SO(3)$, the set of all rotation matrices for $\mathbb{R}^3$. A simpler example that will be helpful to keep in mind is $M = \mathbb{S}^2$, which is a sphere in $\mathbb{R}^3$. We want to extend the concepts of Section 8.3.1 from $\mathbb{R}^n$ to manifolds. One of the first definitions will be the tangent space $\mathbb{T}_p(M)$ at a point $p \in M$. As you might imagine intuitively, the tangent vectors are tangent to a surface, as shown in Figure 8.8. These will indicate possible velocities with which we can move along the manifold from p. This is more difficult to define for a manifold than for $\mathbb{R}^n$ because it is easy to express any point in $\mathbb{R}^n$ using n coordinates, and all local coordinate frames for the tangent spaces at every $p \in \mathbb{R}^n$ are perfectly aligned with each other. For a manifold such as $\mathbb{S}^2$, we must define tangent spaces in a way that is not sensitive to coordinates and handles the fact that the tangent plane rotates as we move around on $\mathbb{S}^2$.

First think carefully about what it means to assign coordinates to a manifold. Suppose M has dimension n and is embedded in $\mathbb{R}^m$. For $M = SO(3)$, $n = 3$ and $m = 9$. For $M = \mathbb{S}^2$, $n = 2$ and $m = 3$. The number of coordinates should be n,

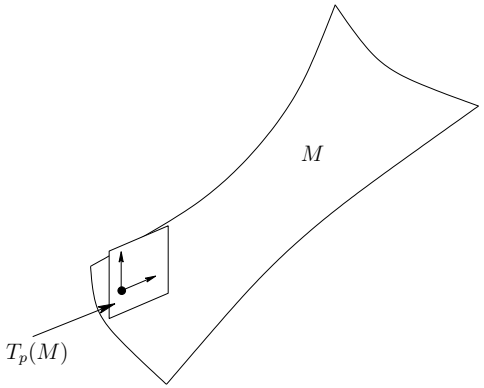


Figure 8.8: Intuitively, the tangent space is a linear approximation to the manifold in a neighborhood around p.

the dimension of M; however, manifolds embedded in $\mathbb{R}^m$ are often expressed as a subset of $\mathbb{R}^m$ for which some equality constraints must be obeyed. We would like to express some part of M in terms of coordinates in $\mathbb{R}^n$.

Coordinates and parameterizations For any open set $U \subseteq M$ and function $\phi : U \to \mathbb{R}^n$ such that ϕ is a homeomorphism onto a subset of $\mathbb{R}^n$, the pair (U, ϕ) is called a *coordinate neighborhood* (or *chart* in some literature). The values $\phi(p)$ for some $p \in U$ are called the *coordinates* of p.

Example 8.11 (Coordinate Neighborhoods on $\mathbb{S}^1$) A simple example can be obtained for the circle $M = \mathbb{S}^1$. Suppose M is expressed as the unit circle embedded in $\mathbb{R}^2$ (the set of solutions to $x^2 + y^2 = 1$). Let (x, y) denote a point in $\mathbb{R}^2$. Let U be the subset of $\mathbb{S}^1$ for which $x > 0$. A coordinate function $\phi : U \to (-\pi/2, \pi/2)$, can be defined as $\phi(x, y) = \tan^{-1}(y/x)$.

Let $W = \phi(U)$ (the range of ϕ) for some coordinate neighborhood (U, ϕ). Since U and W are homeomorphic via ϕ, the inverse function ϕ^{-1} can also be defined. It turns out that the inverse is the familiar idea of a *parameterization*. Continuing Example 8.11, ϕ^{-1} yields the mapping $\theta \mapsto (\cos\theta, \sin\theta)$, which is the familiar parameterization of the circle but restricted to $\theta \in (-\pi/2, \pi/2)$. ■

To make differentiation work at a point $p \in M$, it will be important to have a coordinate neighborhood defined over an open subset of M that contains p. This is mainly because defining derivatives of a function at a point requires that an open set exists around the point. If the coordinates appear to have no boundary, then this will be possible. It is unfortunately not possible to cover all of M with a single coordinate neighborhood, unless $M = \mathbb{R}^n$ (or M is at least homeomorphic to $\mathbb{R}^n$). We must therefore define multiple neighborhoods for which the domains

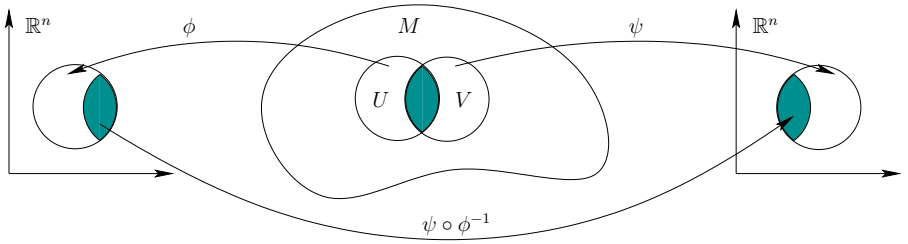


Figure 8.9: An illustration of a change of coordinates.

cover all of M. Since every domain is an open set, some of these domains must overlap. What happens in this case? We may have two or more alternative coordinates for the same point. Moving from one set of coordinates to another is the familiar operation used in calculus called a *change of coordinates*. This will now be formalized.

Suppose that (U, ϕ) and (V, ψ) are coordinate neighborhoods on some manifold M, and $U \cap V \neq \emptyset$. Figure 8.9 indicates how to change coordinates from ϕ to ψ. This change of coordinates is expressed using function composition as $\psi \circ \phi^{-1} : \mathbb{R}^n \to \mathbb{R}^n$ (ϕ^{-1} maps from $\mathbb{R}^n$ into M, and ψ maps from a subset of M to $\mathbb{R}^n$).

Example 8.12 (Change of Coordinates) Consider changing from Euler angles to quaternions for $M = SO(3)$. Since $SO(3)$ is a 3D manifold, $n = 3$. This means that any coordinate neighborhood must map a point in $SO(3)$ to a point in $\mathbb{R}^3$. We can construct a coordinate function $\phi : SO(3) \to \mathbb{R}^3$ by computing Euler angles from a given rotation matrix. The functions are actually defined in (3.47), (3.48), and (3.49). To make this a coordinate neighborhood, an open subset U of M must be specified.

We can construct another coordinate function $\psi : SO(3) \to \mathbb{R}^3$ by using quaternions. This may appear to be a problem because quaternions have four components; however, the fourth component can be determined from the other three. Using (4.24) to (4.26), the a, b, and c coordinates can be determined.

Now suppose that we would like to change from Euler angles to quaternions in the overlap region $U \cap V$, in which V is an open set on which the coordinate neighborhood for quaternions is defined. The task is to construct a change of coordinates, $\psi \circ \phi^{-1}$. We first have to invert ϕ over U. This means that we instead need a parameterization of M in terms of Euler angles. This is given by (3.42), which yields a rotation matrix, $\phi^{-1}(\alpha, \beta, \gamma) \in SO(3)$ for α, β, and γ. Once this matrix is determined, then ψ can be applied to it to determine the quaternion parameters, a, b, and c. This means that we have constructed three real-valued functions, f_1, f_2, and f_3, which yield $a = f_1(\alpha, \beta, \gamma)$, $b = f_2(\alpha, \beta, \gamma)$, and $c = f_3(\alpha, \beta, \gamma)$. Together, these define $\psi \circ \phi^{-1}$. ■

There are several reasons for performing coordinate changes in various contexts. Example 8.12 is motivated by a change that frequently occurs in motion planning. Imagine, for example, that a graphics package displays objects using quaternions, but a collision-detection algorithm uses Euler angles. It may be necessary in such cases to frequently change coordinates. From studies of calculus, you may recall changing coordinates to simplify an integral. In the definition of a smooth manifold, another motivation arises. Since coordinate neighborhoods are based on homeomorphisms of open sets, several may be required just to cover all of M. For example, even if we decide to use quaternions for $SO(3)$, several coordinate neighborhoods that map to quaternions may be needed. On the intersections of their domains, a change of coordinates is necessary.

Now we are ready to define a smooth manifold. Changes of coordinates will appear in the manifold definition, and they must satisfy a smoothness condition. A *smooth structure*[5] on a (topological) manifold M is a family[6] $\mathcal{U} = \{U_\alpha, \phi_\alpha\}$ of coordinate neighborhoods such that:

1. The union of all U_α contains M. Thus, it is possible to obtain coordinates in $\mathbb{R}^n$ for any point in M.

2. For any (U, ϕ) and (V, ψ) in $\mathcal{U}$, if $U \cap V \neq \emptyset$, then the changes of coordinates, $\psi \circ \phi^{-1}$ and $\phi \circ \psi^{-1}$, are smooth functions on $U \cap V$. The changes of coordinates must produce diffeomorphisms on the intersections. In this case, the coordinate neighborhoods are called *compatible*.

3. The family $\mathcal{U}$ is maximal in the sense that if some (U, ϕ) is compatible with every coordinate neighborhood in $\mathcal{U}$, then (U, ϕ) must be included in $\mathcal{U}$.

A well-known theorem (see [73], p. 54) states that if a set of compatible neighborhoods covers all of M, then a unique smooth structure exists that contains them.[7] This means that a differential structure can often be specified by a small number of neighborhoods, and the remaining ones are implied.

A manifold, as defined in Section 4.1.2, together with a smooth structure is called a *smooth manifold*.[8]

Example 8.13 ($\mathbb{R}^n$ as a Smooth Manifold) We should expect that the concepts presented so far apply to $\mathbb{R}^n$, which is the most straightforward family of manifolds. A single coordinate neighborhood $\mathbb{R}^n \rightarrow \mathbb{R}^n$ can be used, which is the identity map. For all integers $n \in \{1, 2, 3\}$ and $n > 4$, this is the only possible smooth structure on $\mathbb{R}^n$. It is truly amazing that for $\mathbb{R}^4$, there are uncountably many incompatible smooth structures, called *exotic* $\mathbb{R}^4$ [155]. There is no need to worry, however; just use the one given by the identity map for $\mathbb{R}^4$. ■

[5] Alternative names are *differentiable structure* and C^∞ *structure*.

[6] In literature in which the coordinate neighborhoods are called *charts*, this family is called an *atlas*.

[7] This is under the assumption that M is Hausdorff and has a countable basis of open sets, which applies to the manifolds considered here.

[8] Alternative names are *differentiable manifold* and C^∞ *manifold*.

Example 8.14 ($\mathbb{S}^n$ as a Smooth Manifold) One way to define $\mathbb{S}^n$ as a smooth manifold uses $2(n+1)$ coordinate neighborhoods and results in simple expressions. Let $\mathbb{S}^n$ be defined as

$$\mathbb{S}^n = \{(x_1, \ldots, x_{n+1}) \in \mathbb{R}^{n+1} | \; x_1^2 + \cdots + x_{n+1}^2 = 1\}. \tag{8.22}$$

The domain of each coordinate neighborhood is defined as follows. For each i from 1 to $n+1$, there are two neighborhoods:

$$U_i^+ = \{(x_1, \ldots, x_{n+1}) \in \mathbb{R}^{n+1} | \; x_i > 0\} \tag{8.23}$$

and

$$U_i^- = \{(x_1, \ldots, x_{n+1}) \in \mathbb{R}^{n+1} | \; x_i < 0\}. \tag{8.24}$$

Each neighborhood is an open set that covers half of $\mathbb{S}^n$ but misses the great circle at $x_i = 0$. The coordinate functions can be defined by projection down to the $(n-1)$-dimensional hyperplane that contains the great circle. For each i,

$$\phi_i^+(x_1, \ldots, x_{n+1}) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \tag{8.25}$$

over U_i^+. Each ϕ_i^- is defined the same way, but over U_i^-. Each coordinate function is a homeomorphism from an open subset of $\mathbb{S}^n$ to an open subset of $\mathbb{R}^n$, as required. On the subsets in which the neighborhoods overlap, the changes of coordinate functions are smooth. For example, consider changing from ϕ_i^+ to ϕ_j^- for some $i \neq j$. The change of coordinates is a function $\phi_j^- \circ (\phi_i^+)^{-1}$. The inverse of ϕ_i^+ is expressed as

$$\begin{aligned}&(\phi_i^+)^{-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \\ &(x_1, \ldots, x_{i-1}, 1 - \sqrt{1 - x_1^2 - \cdots - x_{i-1}^2 - x_{i+1}^2 - \cdots - x_n^2}, x_{i+1}, \ldots, x_{n+1}).\end{aligned} \tag{8.26}$$

When composed with ϕ_j^-, the jth coordinate is dropped. This yields

$$\begin{aligned}\phi_k^- \circ (\phi_i^+)^{-1}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \\ (x_1, \ldots, x_{i-1}, 1 - \sqrt{1 - x_1^2 - \cdots - x_{i-1}^2 - x_{i+1}^2 - \cdots - x_n^2}, \\ x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n),\end{aligned} \tag{8.27}$$

which is a smooth function over the domain U_i^+. Try visualizing the changes of coordinates for the circle $\mathbb{S}^1$ and sphere $\mathbb{S}^2$.

The smooth structure can alternatively be defined using only two coordinate neighborhoods by using *stereographic projection*. For $\mathbb{S}^2$, one coordinate function maps almost every point $x \in \mathbb{S}^2$ to $\mathbb{R}^2$ by drawing a ray from the north pole to

x and mapping to the point in the $x_3 = 0$ plane that is crossed by the ray. The only excluded point is the north pole itself. A similar mapping can be constructed from the south pole. ■

Example 8.15 ($\mathbb{RP}^n$ as a Smooth Manifold) This example is particularly important because $\mathbb{RP}^3$ is the same manifold as $SO(3)$, as established in Section 4.2.2. Recall from Section 4.1.2 that $\mathbb{RP}^n$ is defined as the set of all lines in $\mathbb{R}^{n+1}$ that pass through the origin. This means that for any $\alpha \in \mathbb{R}$ such that $\alpha \neq 0$, and any $x \in \mathbb{R}^{n+1}$, both x and αx are identified. In projective space, scale does not matter.

A smooth structure can be specified by only $n+1$ coordinate neighborhoods. For each i from 1 to $n+1$, let

$$\phi_i(x_1, \ldots, x_{n+1}) = (x_1/x_i, \ldots, x_{i-1}/x_i, x_{i+1}/x_i, \ldots, x_n/x_i), \tag{8.28}$$

over the open set of all points in $\mathbb{R}^{n+1}$ for which $x_i \neq 0$. The inverse coordinate function is given by

$$\phi_i^{-1}(z_1, \ldots, z_n) = (z_1, \ldots, z_{i-1}, 1, z_i, \ldots, z_{n+1}). \tag{8.29}$$

It is not hard to verify that these simple transformations are smooth on overlapping neighborhoods.

A smooth structure over $SO(3)$ can be derived as a special case because $SO(3)$ is topologically equivalent to $\mathbb{RP}^3$. Suppose elements of $SO(3)$ are expressed using unit quaternions. Each (a, b, c, d) is considered as a point on $\mathbb{S}^3$. There are four coordinate neighborhoods. For example, one of them is

$$\phi_b(a, b, c, d) = (a/b,\ c/b,\ d/b), \tag{8.30}$$

which is defined over the subset of $\mathbb{R}^4$ for which $b \neq 0$. The inverse of $\phi_b(a, b, c, d)$ needs to be defined so that a point on $SO(3)$ maps to a point in $\mathbb{R}^4$ that has unit magnitude. ■

Tangent spaces on manifolds Now consider defining tangent spaces on manifolds. Intuitively, the tangent space $T_p(M)$ at a point p on an n-dimensional manifold M is an n-dimensional hyperplane in $\mathbb{R}^m$ that best approximates M around p, when the hyperplane origin is translated to p. This is depicted in Figure 8.8. The notion of a tangent was actually used in Section 7.4.1 to describe local motions for motion planning of closed kinematic chains (see Figure 7.22).

To define a tangent space on a manifold, we first consider a more complicated definition of the tangent space at a point in $\mathbb{R}^n$, in comparison to what was given in Section 8.3.1. Suppose that $M = \mathbb{R}^2$, and consider taking directional derivatives

of a smooth function $f : \mathbb{R}^2 \to \mathbb{R}$ at a point $p \in \mathbb{R}^2$. For some (unnormalized) direction vector, $v \in \mathbb{R}^2$, the directional derivative of f at p can be defined as

$$\nabla_v(f)\Big|_p = v_1 \frac{\partial f}{\partial x_1}\Big|_p + v_2 \frac{\partial f}{\partial x_2}\Big|_p. \tag{8.31}$$

The directional derivative used here does not normalize the direction vector (contrary to basic calculus). Hence, $\nabla_v(f) = \nabla(f) \cdot v$, in which "$\cdot$" denotes the inner product or dot product, and $\nabla(f)$ denotes the gradient of f. The set of all possible direction vectors that can be used in this construction forms a two-dimensional vector space that happens to be the tangent space $T_p(\mathbb{R}^2)$, as defined previously. This can be generalized to n dimensions to obtain

$$\nabla_v(f)\Big|_p = \sum_{i=1}^{n} v_i \frac{\partial f}{\partial x_i}\Big|_p, \tag{8.32}$$

for which all possible direction vectors represent the tangent space $T_p(\mathbb{R}^n)$. The set of all directions can be interpreted for our purposes as the set of possible velocity vectors.

Now consider taking (unnormalized) directional derivatives of a smooth function, $f : M \to \mathbb{R}$, on a manifold. For an n-dimensional manifold, the tangent space $T_p(M)$ at a point $p \in M$ can be considered once again as the set of all unnormalized directions. These directions must intuitively be tangent to the manifold, as depicted in Figure 8.8. There exists a clever way to define them without even referring to specific coordinate neighborhoods. This leads to a definition of $T_p(M)$ that is intrinsic to the manifold.

At this point, you may accept that $T_p(M)$ is an n-dimensional vector space that is affixed to M at p and oriented as shown in Figure 8.8. For the sake of completeness, however, a technical definition of $T_p(M)$ from differential geometry will be given; more details appear in [73, 437]. The construction is based on characterizing the set of all possible directional derivative operators. Let $C^\infty(p)$ denote the set of all smooth functions that have domains that include p. Now make the following identification. Any two functions $f, g \in C^\infty(p)$ are defined to be *equivalent* if there exists an open set $U \subset M$ such that for any $p \in U$, $f(p) = g(p)$. There is no need to distinguish equivalent functions because their derivatives must be the same at p. Let $\tilde{C}^\infty(p)$ denote C^∞ under this identification. A directional derivative operator at p can be considered as a function that maps from $\tilde{C}^\infty(p)$ to $\mathbb{R}$ for some direction. In the case of $\mathbb{R}^n$, the operator appears as ∇_v for each direction v. Think about the set of all directional derivative operators that can be made. Each one must assign a real value to every function in $\tilde{C}^\infty(p)$, and it must obey two axioms from calculus regarding directional derivatives. Let ∇_v denote a directional derivative operator at some $p \in M$ (be careful, however, because here v is not explicitly represented since there are no coordinates). The directional derivative operator must satisfy two axioms:

1. **Linearity:** For any $\alpha, \beta \in \mathbb{R}$ and $f, g \in \tilde{C}^\infty(p)$,

$$\nabla_v(\alpha f + \beta g) = \alpha \nabla_v f + \beta \nabla_v g. \tag{8.33}$$

2. **Leibniz Rule (or Derivation):** For any $f, g \in \tilde{C}^\infty(p)$,

$$\nabla_v(fg) = \nabla_v f \; g(p) + f(p) \nabla_v g. \tag{8.34}$$

You may recall these axioms from standard vector calculus as properties of the directional derivative. It can be shown that the set of all possible operators that satisfy these axioms forms an n-dimensional vector space [73]. This vector space is called the *tangent space*, $T_p(M)$, at p. This completes the definition of the tangent space without referring to coordinates.

It is helpful, however, to have an explicit way to express vectors in $T_p(M)$. A basis for the tangent space can be obtained by using coordinate neighborhoods. An important theorem from differential geometry states that if $F : M \to N$ is a diffeomorphism onto an open set $U \subset N$, then the tangent space, $T_p(M)$, is isomorphic to $T_{F(p)}(N)$. This means that by using a parameterization (the inverse of a coordinate neighborhood), there is a bijection between velocity vectors in $T_p(M)$ and velocity vectors in $T_{F(p)}(N)$. Small perturbations in the parameters cause motions in the tangent directions on the manifold N. Imagine, for example, making a small perturbation to three quaternion parameters that are used to represent $SO(3)$. If the perturbation is small enough, motions that are tangent to $SO(3)$ occur. In other words, the perturbed matrices will lie very close to $SO(3)$ (they will not lie in $SO(3)$ because $SO(3)$ is defined by nonlinear constraints on $\mathbb{R}^9$, as discussed in Section 4.1.2).

Example 8.16 (The Tangent Space for $\mathbb{S}^2$) The discussion can be made more concrete by developing the tangent space for $\mathbb{S}^2$, which is embedded in $\mathbb{R}^3$ as the set of all points $(x, y, z) \in \mathbb{R}^3$ for which $x^2 + y^2 + z^2 = 1$. A coordinate neighborhood can be defined that covers most of $\mathbb{S}^2$ by using standard spherical coordinates. Let f denote the coordinate function, which maps from (x, y, z) to angles (θ, ϕ). The domain of f is the open set defined by $\theta \in (0, 2\pi)$ and $\phi \in (0, \pi)$ (this excludes the poles). The standard formulas are $\theta = \text{atan2}(y, x)$ and $\phi = \cos^{-1} z$. The inverse, f^{-1}, yields a parameterization, which is $x = \cos\theta \sin\phi$, $y = \sin\theta \sin\phi$, and $z = \cos\phi$.

Now consider different ways to express the tangent space at some point $p \in \mathbb{S}^2$, other than the poles (a change of coordinates is needed to cover these). Using the coordinates (θ, ϕ), velocities can be defined as vectors in $\mathbb{R}^2$. We can imagine moving in the plane defined by θ and ϕ, provided that the limits $\theta \in (0, 2\pi)$ and $\phi \in (0, \pi)$ are respected.

We can also use the parameterization to derive basis vectors for the tangent space as vectors in $\mathbb{R}^3$. Since the tangent space has only two dimensions, we must obtain a plane that is "tangent" to the sphere at p. These can be found by taking

derivatives. Let f^{-1} be denoted as $x(\theta, \phi)$, $y(\theta, \phi)$, and $z(\theta, \phi)$. Two basis vectors for the tangent plane at p are

$$\left[\frac{dx(\theta,\phi)}{d\theta} \quad \frac{dy(\theta,\phi)}{d\theta} \quad \frac{dz(\theta,\phi)}{d\theta} \right] \tag{8.35}$$

and

$$\left[\frac{dx(\theta,\phi)}{d\phi} \quad \frac{dy(\theta,\phi)}{d\phi} \quad \frac{dz(\theta,\phi)}{d\phi} \right]. \tag{8.36}$$

Computing these derivatives and normalizing yields the vectors $[-\sin\theta \;\; \cos\theta \;\; 0]$ and $[\cos\theta\cos\phi \;\; \sin\theta\cos\phi \;\; -\sin\phi]$. These can be imagined as the result of making small perturbations of θ and ϕ at p. The vector space obtained by taking all linear combinations of these vectors is the tangent space at $\mathbb{R}^2$. Note that the direction of the basis vectors depends on $p \in \mathbb{S}^2$, as expected.

The tangent vectors can now be imagined as lying in a plane that is tangent to the surface, as shown in Figure 8.8. The normal vector to a surface specified as $g(x, y, z) = 0$ is ∇g, which yields $[x \;\; y \;\; z]$ after normalizing. This could alternatively be obtained by taking the cross product of the two vectors above and using the parameterization f^{-1} to express it in terms of x, y, and z. For a point $p = (x_0, y_0, z_0)$, the plane equation is

$$x_0(x - x_0) + y_0(y - y_0) + z_0(z - z_0) = 0. \tag{8.37}$$

■

Vector fields and velocity fields on manifolds The notation for a tangent space on a manifold looks the same as for $\mathbb{R}^n$. This enables the vector field definition and notation to extend naturally from $\mathbb{R}^n$ to smooth manifolds. A *vector field* on a manifold M assigns a vector in $T_p(M)$ for every $p \in M$. It can once again be imagined as a needle diagram, but now the needle diagram is spread over the manifold, rather than lying in $\mathbb{R}^n$.

The velocity field interpretation of a vector field can also be extended to smooth manifolds. This means that $\dot{x} = f(x)$ now defines a set of n differential equations over M and is usually expressed using a coordinate neighborhood of the smooth structure. If f is a smooth vector field, then a *solution trajectory*, $\tau : [0, \infty) \to M$, can be defined from any $x_0 \in M$. Solution trajectories in the sense of Filipov can also be defined, for the case of piecewise-smooth vector fields.

8.4 Complete Methods for Continuous Spaces

A complete feedback planning algorithm must compute a feedback solution if one exists; otherwise, it must report failure. Section 8.4.1 parallels Section 8.2 by

defining feedback plans and navigation functions for the case of a continuous state space. Section 8.4.2 indicates how to define a feasible feedback plan from a cell complex that was computed using cell decomposition techniques. Section 8.4.3 presents a combinatorial approach to computing an optimal navigation function and corresponding feedback plan in $\mathbb{R}^2$. Sections 8.4.2 and 8.4.3 allow the feedback plan to be a discontinuous vector field. In many applications, especially those in which dynamics dominate, some conditions need to be enforced on the navigation functions and their resulting vector fields. Section 8.4.4 therefore considers constraints on the allowable vector fields and navigation functions. This coverage includes navigation functions in the sense of Rimon-Koditschek [416], from which the term navigation function was introduced.

8.4.1 Feedback Motion Planning Definitions

Using the concepts from Section 8.3, we are now ready to define feedback motion planning over configuration spaces or other continuous state spaces. Recall Formulation 4.1, which defined the basic motion planning problem in terms of configuration space. The differences in the current setting are that there is no initial condition, and the requirement of a solution path is replaced by a solution vector field. The formulation here can be considered as a continuous-time adaptation to Formulation 8.1.

Formulation 8.2 (Feedback Motion Planning)

1. A *state space*, X, which is a smooth manifold. The state space will most often be $\mathcal{C}_{free}$, as defined in Section 4.3.1.[9]

2. For each state, $x \in X$, an *action space*, $U(x) = T_x(X)$. The zero velocity, $0 \in T_x(X)$, is designated as the termination action, u_T. Using this model, the robot is capable of selecting its velocity at any state.[10]

3. An unbounded *time interval*, $T = [0, \infty)$.

4. A *state transition (differential) equation*,

$$\dot{x} = u, \tag{8.38}$$

which is expressed using a coordinate neighborhood and yields the velocity, $\dot{x}$, directly assigned by the action u. The velocity produced by u_T is $0 \in T_x(X)$ (which means "stop").

[9]Note that X already excludes the obstacle region. For some problems in Part IV, the state space will be $X = \mathcal{C}$, which includes the obstacle region.

[10]This allows discontinuous changes in velocity, which is unrealistic in many applications. Additional constraints, such as imposing acceleration bounds, will also be discussed. For a complete treatment of differential constraints, see Part IV.

5. A *goal set*, $X_G \subset X$.

A *feedback plan*, π, for Formulation 8.2 is defined as a function π, which produces an action $u \in U(x)$ for each $x \in X$. A feedback plan can equivalently be considered as a vector field on X because each $u \in U(x)$ specifies a velocity vector (u_T specifies zero velocity). Since the initial state is not fixed, it becomes slightly more complicated to define what it means for a plan to be a solution to the problem. Let $X_r \subset X$ denote the set of all states from which X_G is *reachable*. More precisely, a state x_I belongs to X_r if and only if a continuous path $\tau : [0, 1] \rightarrow X$ exists for which $\tau(0) = x_I$ and $\tau(1) = x_G$ for some $x_G \in X_G$. This means that a solution path exists from x_I for the "open-loop" motion planning problem, which was considered in Chapter 4.

Solution concepts

A feedback plan, π, is called a *solution* to the problem in Formulation 8.2 if from all $x_I \in X_r$, the integral curves of π (considered as a vector field) arrive in X_G, at which point the termination action is applied. Some words of caution must be given about what it means to "arrive" in X_G. Notions of stability from control theory [271, 426] are useful for distinguishing different cases; see Section 15.1. If X_G is a small ball centered on x_G, then the ball will be reached after finite time using the inward vector field shown in Figure 8.5b. Now suppose that X_G is a single point, x_G. The inward vector field produces velocities that bring the state closer and closer to the origin, but when is it actually reached? It turns out that convergence to the origin in this case is only *asymptotic*; the origin is reached in the limit as the time approaches infinity. Such stability often arises in control theory from smooth vector fields. We may allow such asymptotic convergence to the goal (if the vector field is smooth and the goal is a point, then this is unavoidable). If any integral curves result in only asymptotic convergence to the goal, then a solution plan is called an *asymptotic solution plan*. Note that in general it may be impossible to require that π is a smooth (or even continuous) nonzero vector field. For example, due to the *hairy ball theorem* [419], it is known that no such vector field exists for $\mathbb{S}^n$ for any even integer n. Therefore, the strongest possible requirement is that π is smooth except on a set of measure zero; see Section 8.4.4. We may also allow solutions π for which *almost all* integral curves arrive in X_G.

However, it will be assumed by default in this chapter that a solution plan converges to x_G in finite time. For example, if the inward field is normalized to produce unit speed everywhere except the origin, then the origin will be reached in finite time. A constraint can be placed on the set of allowable vector fields without affecting the existence of a solution plan. As in the basic motion planning problem, the speed along the path is not important. Let a *normalized vector field* be any vector field for which either $\|f(x)\| = 1$ or $f(x) = 0$, for all $x \in X$. This means that all velocity vectors are either unit vectors or the zero vector, and the speed is no longer a factor. A normalized vector field provides either a direction of motion

or no motion. Note that any vector field f can be converted into a normalized vector field by dividing the velocity vector $f(x)$ by its magnitude (unless the magnitude is zero), for each $x \in X$.

In many cases, unit speed does not necessarily imply a constant speed in some true physical sense. For example, if the robot is a floating rigid body, there are many ways to parameterize position and orientation. The speed of the body is sensitive to this parameterization. Therefore, other constraints may be preferable instead of $\|f(x)\| = 1$; however, it is important to keep in mind that the constraint is imposed so that $f(x)$ provides a *direction* at x. The particular magnitude is assumed unimportant.

So far, consideration has been given only to a *feasible feedback motion planning problem*. An *optimal feedback motion planning problem* can be defined by introducing a cost functional. Let $\tilde{x}_t$ denote the function $\tilde{x}_t : [0, t] \rightarrow X$, which is called the *state trajectory* (or *state history*). This is a continuous-time version of the state history, which was defined previously for problems that have discrete stages. Similarly, let $\tilde{u}_t$ denote the *action trajectory* (or *action history*), $\tilde{u}_t : [0, t] \rightarrow U$. Let L denote a cost functional, which may be applied from any x_I to yield

$$L(\tilde{x}_{t_F}, \tilde{u}_{t_F}) = \int_0^{t_F} l(x(t), u(t))dt + l_F(x(t_F)), \tag{8.39}$$

in which t_F is the time at which the termination action is applied. The term $l(x(t), u(t))$ can alternatively be expressed as $l(x(t), \dot{x}(t))$ by using the state transition equation (8.38). A normalized vector field that optimizes (8.39) from all initial states that can reach the goal is considered as an *optimal feedback motion plan*.

Note that the state trajectory can be determined from an action history and initial state. In fact, we could have used action trajectories to define a solution path to the motion planning problem of Chapter 4. Instead, a solution was defined there as a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ to avoid having to introduce velocity fields on smooth manifolds. That was the only place in the book in which the action space seemed to disappear, and now you can see that it was only hiding to avoid inessential notation.

Navigation functions

As in Section 8.2.2, potential functions can be used to represent feedback plans, assuming that a local operator is developed that works for continuous state spaces. In the discrete case, the local operator selects an action that reduces the potential value. In the continuous case, the local operator must convert the potential function into a vector field. In other words, a velocity vector must be defined at each state. By default, it will be assumed here that the vector fields derived from the navigation function are not necessarily normalized.

Assume that $\pi(x) = u_T$ is defined for all $x \in X_G$, regardless of the potential function. Suppose that a potential function $\phi : X \rightarrow \mathbb{R}$ has been defined for which

the gradient

$$\nabla\phi = \left[\frac{\partial\phi}{\partial x_1} \;\; \frac{\partial\phi}{\partial x_2} \;\; \cdots \;\; \frac{\partial\phi}{\partial x_n} \right] \tag{8.40}$$

exists over all of $X \setminus X_G$. The corresponding feedback plan can then be defined as $\pi(x) = -\nabla\phi|_x$. This defines the local operator, which means that the velocity is taken in the direction of the steepest descent of ϕ. The idea of using potential functions in this way was proposed for robotics by Khatib [273, 274] and can be considered as a form of *gradient descent*, which is a general optimization technique.

It is also possible to work with potential functions for which the gradient does not exist everywhere. In these cases, a continuous-space version of (8.4) can be defined for a small, fixed Δt as

$$u^* = \operatorname*{argmin}_{u \in U(x)} \Big\{ \phi(x') \Big\}, \tag{8.41}$$

in which x' is the state obtained by integrating velocity u from x for time Δt. One problem is that Δt should be chosen to use the smallest possible neighborhood around ϕ. It is best to allow only potential functions for which Δt can be made arbitrarily small at every x without affecting the decision in (8.41). To be precise, this means that an infinite sequence of u^* values can be determined from a sequence of Δt values that converges to 0. A potential function should then be chosen to ensure after some point in the sequence, u^*, exists and the same u^* can be chosen to satisfy (8.41) as Δt approaches 0. A special case of this is if the gradient of ϕ exists; the infinite sequence in this case converges to the negative gradient.

A potential function, ϕ, is called a *navigation function* if the vector field that is derived from it is a solution plan. The optimal cost-to-go serves as an *optimal navigation function*. If multiple vector fields can be derived from the same ϕ, then every possible derived vector field must yield a solution feedback plan. If designed appropriately, the potential function can be viewed as a kind of "ski slope" that guides the state to X_G. If there are extra local minima that cause the state to become trapped, then X_G will not be reached. To be a navigation function, such local minima outside of X_G are not allowed. Furthermore, there may be additional requirements to ensure that the derived vector field satisfies additional constraints, such as bounded acceleration.

Example 8.17 (Quadratic Potential Function) As a simple example, suppose $X = \mathbb{R}^2$, there are no obstacles, and $q_{goal} = (0, 0)$. A quadratic function $\phi(x, y) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$ serves as a good potential function to guide the state to the goal. The feedback motion strategy is defined as $f = -\nabla\phi = [-x_1 \;\; -x_2]$, which is the inward vector field shown in Figure 8.5b.

If the goal is instead at some $(x_1', x_2') \in \mathbb{R}^2$, then a potential function that guides the state to the goal is $\phi(x_1, x_2) = (x_1 - x_1')^2 + (x_2 - x_2')^2$. ■

Suppose the state space represents a configuration space that contains point obstacles. The previous function ϕ can be considered as an attractive potential because the configuration is attracted to the goal. One can also construct a repulsive potential that repels the configuration from the obstacles to avoid collision. Let ϕ_a denote the attractive component and ϕ_r denote a repulsive potential that is summed over all obstacle points. A potential function of the form $\phi = \phi_a + \phi_r$ can be defined to combine both effects. The robot should be guided to the goal while avoiding obstacles. The problem is that it is difficult in general to ensure that the potential function will not contain multiple local minima. The configuration could become trapped at a local minimum that is not in the goal region. This was an issue with the planner from Section 5.4.3.

8.4.2 Vector Fields Over Cell Complexes

This section describes how to construct a piecewise-smooth vector field over a cell complex. Only normalized vector fields will be considered. It is assumed that each cell in the complex has a simple shape over which it is easy to define a patch of the vector field. In many cases, the cell decomposition techniques that were introduced in Chapter 6 for motion planning can be applied to construct a feedback plan.

Suppose that an n-dimensional state space X has been decomposed into a cell complex, such as a simplicial complex or singular complex, as defined in Section 6.3.1. Assume that the goal set is a single point, x_G. Defining a feedback plan π over X requires placing a vector field on X for which all integral curves lead to x_G (if x_G is reachable). This is accomplished by defining a smooth vector field for each n-cell. Each $(n-1)$-cell is a switching boundary, as considered in Section 8.3.1. This leads directly to solution trajectories in the sense of Filipov. If π is allowed to be discontinuous, then it is actually not important to specify values on any of the cells of dimension $n-1$ or less.

A hierarchical approach is taken to the construction of π:

1. Define a discrete planning problem over the n-cells. The cell that contains x_G is designated as the goal, and a discrete navigation function is defined over the cells.

2. Define a vector field over each n-cell. The field should cause all states in the cell to flow into the next cell as prescribed by the discrete navigation function.

One additional consideration that is important in applications is to try to reduce the effect of the discontinuity across the boundary as much as possible. It may be possible to eliminate the discontinuity, or even construct a smooth transition between n-cells. This issue will not be considered here, but it is nevertheless quite important [127, 334].

The approach will now be formalized. Suppose that a cell complex has been defined over a continuous state space, X. Let $\check{X}$ denote the set of n-cells, which can be interpreted as a finite state space. A discrete planning problem will be defined over $\check{X}$. To avoid confusion with the original continuous problem, the prefix *super* will be applied to the discrete planning components. Each superstate $\check{x} \in \check{X}$ corresponds to an n-cell. From each $\check{x}$, a superaction, $\check{u} \in \check{U}(\check{x})$ exists for each neighboring n-cell (to be neighboring, the two cells must share an $(n-1)$-dimensional boundary). Let the goal superstate $\check{x}_g$ be the n-cell that contains x_G. Assume that the cost functional is defined for the discrete problem so that every action (other than u_T) produces a unit cost. Now the concepts from Section 8.2 can be applied to the discrete problem. A discrete navigation function, $\check{\phi} : \check{X} \rightarrow \mathbb{R}$, can be computed using Dijkstra's algorithm (or another algorithm, particularly if optimality is not important). Using the discrete local operator from Section 8.2.2, this results in a discrete feedback plan, $\check{\pi} : \check{X} \rightarrow \check{U}$.

Based on the discrete feedback plan, there are two kinds of n-cells. The first is the goal cell, $\check{x}_g$, for which a vector field needs to be defined so that all integral curves lead to X_g in finite time.[11] A termination action can be applied when x_G is actually reached. The remaining n-cells are of the second kind. For each cell $\check{x}$, the boundary that is shared with the cell reached by applying $\check{u} = \check{\pi}(\check{x})$ is called the *exit face*. The vector field over the n-cell $\check{x}$ must be defined so that all integral curves lead to the exit face. When the exit face is reached, a transition will occur into the next n-cell. If the n-cells are convex, then defining this transition is straightforward (unless there are additional requirements on the field, such as smoothness at the boundary). For more complicated cells, one possibility is to define a vector field that retracts all points onto a single curve in the cell.

A simple example of the approach is illustrated for the case of $X = \mathcal{C}_{free} \subset \mathbb{R}^2$, in which the boundary of $\mathcal{C}_{free}$ is polygonal. This motion planning problem was considered in Section 6.2, but without feedback. Suppose that a triangulation of X has been computed, as described in Section 6.3.2. An example was shown in Figure 6.16. A discrete feedback plan is shown for a particular goal state in Figure 8.10. Each 2-cell (triangle) is labeled with an arrow that points to the next cell.

For the cell that contains x_G, a normalized version of the inward vector field shown in Figure 8.5b can be formed by dividing each nonzero vector by its magnitude. It can then be translated to move its origin to x_G. For each remaining 2-cell, a vector field must be constructed that flows into the appropriate neighboring cell. Figure 8.11 illustrates a simple way to achieve this. An outward vector field can be made by negating the field shown in Figure 8.5b to obtain $f = [x \quad y]$. This field can be normalized and translated to move the origin to the triangle vertex that is not incident to the exit edge. This is called the *repulsive vertex* in Figure 8.11. This generates a vector field that pushes all points in the triangle to the ext edge. If the fields are constructed in this way for each triangle, then the global

[11] This is possible in finite time, even if X_g is a single point, because the vector field is not continuous. Otherwise, only asymptotic convergence may be possible.

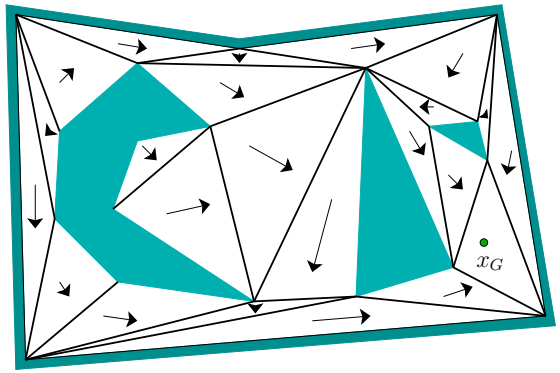


Figure 8.10: A triangulation is used to define a vector field over X. All solution trajectories lead to the goal.

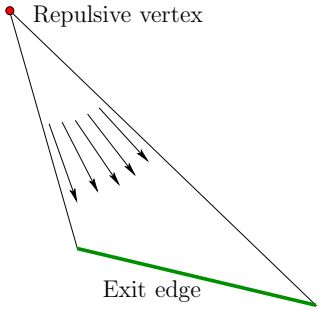


Figure 8.11: A vector field can be defined for each triangle by repelling from a vertex that opposes the exit edge.

vector field represents a solution feedback plan for the problem. Integral curves (in the sense of Filipov) lead to x_G in finite time.

8.4.3 Optimal Navigation Functions

The vector fields developed in the last section yield feasible trajectories, but not necessarily optimal trajectories unless the initial and goal states are in the same convex n-cell. If $X = \mathbb{R}^2$, then it is possible to make a continuous version of Dijkstra's algorithm [371]. This results in an exact cost-to-go function over X based on the Euclidean shortest path to a goal, x_G. The cost-to-go function serves as the navigation function, from which the feedback plan is defined by using a local steepest descent.

Suppose that X is bounded by a simple polygon (no holes). Assume that only normalized vector fields are allowed. The cost functional is assumed to be

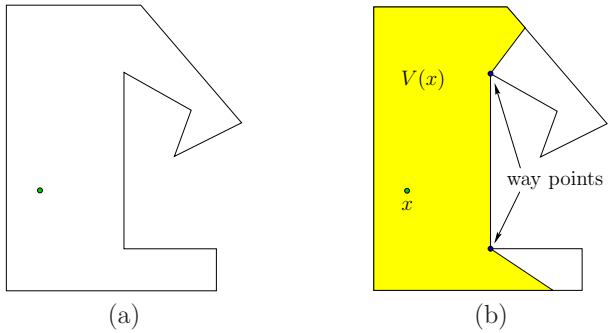


Figure 8.12: (a) A point, x, in a simple polygon. (b) The visibility polygon, $V(x)$.

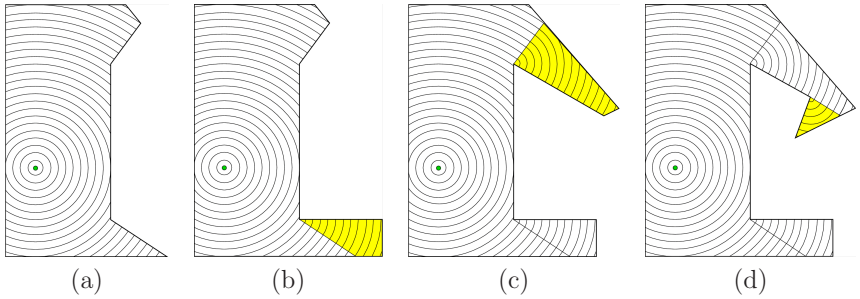


Figure 8.13: The optimal navigation function is computed in four iterations. In each iteration, the navigation function is extended from a new way point.

the Euclidean distance traveled along a state trajectory. Recall from Section 6.2.4 that for optimal path planning, $X = \text{cl}(\mathcal{C}_{free})$ must be used. Assume that $\mathcal{C}_{free}$ and $\text{cl}(\mathcal{C}_{free})$ have the same connectivity.[12] This technically interferes with the definition of tangent spaces from Section 8.3 because each point of X must be contained in an open neighborhood. Nevertheless, we allow vectors along the boundary, provided that they "point" in a direction tangent to the boundary. This can be formally defined by considering boundary regions as separate manifolds.

Consider computing the optimal cost-to-go to a point x_G for a problem such as that shown in Figure 8.12a. For any $x \in X$, let the *visibility polygon* $V(x)$ refer to the set of all points visible from x, which is illustrated in Figure 8.12b. A point x' lies in $V(x)$ if and only if the line segment from x' to x is contained in X. This implies that the cost-to-go from x' to x is just the Euclidean distance from

[12] This precludes a choice of $\mathcal{C}_{free}$ for which adding the boundary point enables a homotopically distinct path to be made through the boundary point. An example of this is when two square obstacles in $\mathbb{R}^2$ contact each other only at a pair of corners.

x' to x. The optimal navigation function can therefore be immediately defined over $V(x_G)$ as

$$\phi(x) = \|x - x_G\|. \tag{8.42}$$

Level sets at regularly spaced values of this navigation function are shown in Figure 8.13a.

How do we compute the optimal cost-to-go values for the points in $X \setminus V(x_G)$? For the segments on the boundary of $V(x)$ for any $x \in X$, some edges are contained in the boundary of X, and others cross the interior of X. For the example in Figure 8.12b, there are two edges that cross the interior. For each segment that crosses the interior, let the closer of the two vertices to x be referred to as a *way point*. Two way points are indicated in Figure 8.12b. The way points of $V(x_G)$ are places through which some optimal paths must cross. Let $W(x)$ for any $x \in X$ denote the set of way points of $V(x)$.

A straightforward algorithm proceeds as follows. Let Z_i denote the set of points over which ϕ has been defined, in the ith iteration of the algorithm. In the first iteration, $Z_1 = V(x_G)$, which is the case shown in Figure 8.13a. The way points of $V(x_G)$ are placed in a queue, Q. In each following iteration, a way point x is removed from Q. Let Z_i denote the domain over which ϕ is defined so far. The domain of ϕ is extended to include all new points visible from x. These new points are $V(x) \setminus Z_i$. This yields $Z_{i+1} = Z_i \cup V(x)$, the extended domain of ϕ. The values of $\phi(x')$ for $x' \in Z_{i+1} \setminus Z_i$ are defined by

$$\phi(x') = \phi(x) + \|x' - x\|, \tag{8.43}$$

in which x is the way point that was removed from Q (the optimal cost-to-go value of x was already computed). The way points of $V(x)$ that do not lie in Z_{i+1} are added to Q. Each of these will yield new portions of X that have not yet been seen. The algorithm terminates when Q is empty, which implies that $Z_k = X$ for some k. The execution of the algorithm is illustrated in Figure 8.13.

The visibility polygon can be computed in time $O(n \lg n)$ if X is described by n edges. This is accomplished by performing a *radial sweep*, which is an adaptation of the method applied in Section 6.2.2 for vertical cell decomposition. The difference for computing $V(x)$ is that a ray anchored at x is swept radially (like a radar sweep). The segments that intersect the ray are sorted by their distance from x. For the algorithm that constructs the navigation function, no more than $O(n)$ visibility polygons are computed because each one is computed from a unique way point. This implies $O(n^2 \lg n)$ running time for the whole algorithm. Unfortunately, there is no extension to higher dimensions; recall from Section 7.7.1 that computing shortest paths in a 3D environment is NP-hard [91].

The algorithm given here is easy to describe, but it is not the most general, nor the most efficient. If X has holes, then the level set curves can collide by arriving from different directions when traveling around an obstacle. The queue, Q, described above can be sorted as in Dijkstra's algorithm, and special data structures are needed to identify when critical events occur as the cost-to-go is

propagated outward. It was shown in [227] that this can be done in time $O(n \lg n)$ and space $O(n \lg n)$.

8.4.4 A Step Toward Considering Dynamics

If dynamics is an important factor, then the discontinuous vector fields considered so far are undesirable. Due to momentum, a mechanical system cannot instantaneously change its velocity (see Section 13.3). In this context, vector fields should be required to satisfy additional constraints, such as smoothness or bounded acceleration. This represents only a step toward considering dynamics. Full consideration is given in Part IV, in which precise equations of motions of dynamical systems are expressed as part of the model. The approach in this section is to make vector fields that are "dynamics-ready" rather than carefully considering particular equations of motion.

A framework has been developed by defining a navigation function that satisfies some desired constraints over a simple region, such as a disc [416]. A set of transformations is then designed that are proved to preserve the constraints while adapting the navigation function to more complicated environments. For a given problem, a complete algorithm for constructing navigation functions is obtained by applying the appropriate series of transformations from some starting shape.

This section mostly focuses on constraints that are maintained under this transformation-based framework. Sections 8.4.2 and 8.4.3 worked with normalized vector fields. Under this constraint, virtually any vector field could be defined, provided that the resulting algorithm constructs fields for which integral curves exist in the sense of Filipov. In this section, we remove the constraint that vector fields must be normalized, and then consider other constraints. The velocity given by the vector field is now assumed to represent the true speed that must be executed when the vector field is applied as a feedback plan.

One implication of adding constraints to the vector field is that optimal solutions may not satisfy them. For example, the optimal navigation functions of Section 8.4.3 lead to discontinuous vector fields, which violate the constraints to be considered in this section. The required constraints restrict the set of allowable vector fields. Optimality must therefore be defined over the restricted set of vector fields. In some cases, an optimal solution may not even exist (see the discussion of open sets and optimality in Section 9.1.1). Therefore, this section focuses only on feasible solutions.

An acceleration-based control model

To motivate the introduction of constraints, consider a control model proposed in [127, 417]. The action space, defined as $U(x) = T_x(X)$ in Formulation 8.2, produces a velocity for each action $u \in U(x)$. Therefore, $\dot{x} = u$. Suppose instead that each action produces an acceleration. This can be expressed as $\ddot{x} = u$, in

which $\ddot{x}$ is an *acceleration vector*,

$$\ddot{x} = \frac{d\dot{x}}{dt} = \left[\frac{d^2x_1}{dt^2} \;\; \frac{d^2x_2}{dt^2} \;\; \cdots \;\; \frac{d^2x_n}{dt^2} \right]. \tag{8.44}$$

The velocity $\dot{x}$ is obtained by integration over time. The state trajectory, $\tilde{x} : T \rightarrow X$, is obtained by integrating (8.44) twice.

Suppose that a vector field is given in the form $\dot{x} = f(x)$. How can a feedback plan be derived? Consider how the velocity vectors specified by $f(x)$ change as x varies. Assume that $f(x)$ is smooth (or at least C^1), and let

$$\nabla_{\dot{x}} f(x) = [\nabla_{\dot{x}} f_1(x) \;\; \nabla_{\dot{x}} f_2(x) \;\; \cdots \;\; \nabla_{\dot{x}} f_n(x)], \tag{8.45}$$

in which $\nabla_{\dot{x}}$ denotes the unnormalized directional derivative in the direction of $\dot{x}$: $\nabla f_i \cdot \dot{x}$. Suppose that an initial state x_I is given, and that the initial velocity is $\dot{x} = f(x_I)$. The feedback plan can now be defined as

$$u = \nabla_{\dot{x}} f(x). \tag{8.46}$$

This is equivalent to the previous definition of a feedback plan from Section 8.4.1; the only difference is that now two integrations are needed (which requires both x and $\dot{x} = f(x_I)$ as initial conditions) and a differentiability condition must be satisfied for the vector field.

Now the relationship between $\dot{x}$ and $f(x)$ will be redefined. Suppose that $\dot{x}$ is the true measured velocity during execution and that $f(x)$ is the prescribed velocity, obtained from the vector field f. During execution, it is assumed that $\dot{x}$ and $f(x)$ are not necessarily the same, but the task is to keep them as close to each other as possible. A discrepancy between them may occur due to dynamics that have not been modeled. For example, if the field $f(x)$ requests that the velocity must suddenly change, a mobile robot may not be able to make a sharp turn due to its momentum.

Using the new interpretation, the difference, $f(x) - \dot{x}$, can be considered as a discrepancy or error. Suppose that a vector field f has been computed. A feedback plan becomes the *acceleration-based control* model

$$u = K(f(x) - \dot{x}) + \nabla_{\dot{x}} f(x), \tag{8.47}$$

in which K is a scalar *gain constant*. A larger value of K will make the control system more aggressively attempt to reduce the error. If K is too large, then acceleration or energy constraints may be violated. Note that if $\dot{x} = f(x)$, then $u = \nabla_{\dot{x}} f(x)$, which becomes equivalent to the earlier formulation.

Velocity and acceleration constraints

Considering the acceleration-based control model, some constraints can be placed on the set of allowable vector fields. A *bounded-velocity model* means that $\|\dot{x}\| <$

v_{max}, for some positive real value v_{max} called the *maximum speed*. This could indicate, for example, that the robot has a maximum speed for safety reasons. It is also possible to bound individual components of the velocity vector. For example, there may be separate bounds for the maximum angular and linear velocities of an aircraft. Intuitively, velocity bounds imply that the functions f_i, which define the vector field, cannot take on large values.

A *bounded-acceleration model* means that $\|\ddot{x}\| \leq a_{max}$, in which a_{max} is a positive real value called the *maximum acceleration*. Intuitively, acceleration bounds imply that the velocity cannot change too quickly while traveling along an integral curve. Using the control model $\ddot{x} = u$, this implies that $\|u\| \leq a_{max}$. It also imposes the constraint that vector fields must satisfy $\|\nabla_{\dot{x}} f(x)\| \leq a_{max}$ for all $\dot{x}$ and $x \in X$. The condition $\|u\| \leq a_{max}$ is very important in practice because higher accelerations are generally more expensive (bigger motors are required, more fuel is consumed, etc.). The action u may correspond directly to the torques that are applied to motors. In this case, each motor usually has an upper limit.

As has already been seen, setting an upper bound on velocity generally does not affect the existence of a solution. Imagine that a robot can always decide to travel more slowly. If there is also an upper bound on acceleration, then the robot can attempt to travel more slowly to satisfy the bound. Imagine slowing down in a car to make a sharp turn. If you would like to go faster, then it may be more difficult to satisfy acceleration constraints. Nevertheless, in most situations, it is preferable to go faster.

A discontinuous vector field fails to satisfy any acceleration bound because it essentially requires infinite acceleration at the discontinuity to cause a discontinuous jump in the velocity vector. If the vector field satisfies the Lipschitz condition (8.16) for some constant C, then it satisfies the acceleration bound if $C < a_{max}$.

In Chapter 13, we will precisely specify $U(x)$ at every $x \in X$, which is more general than imposing simple velocity and acceleration bounds. This enables virtually any physical system to be modeled.

Navigation function in the sense of Rimon-Koditschek

Now consider constructing a navigation function from which a vector field can be derived that satisfies constraints motivated by the acceleration-based control model, (8.47). As usual, the definition of a navigation function begins with the consideration of a potential function, $\phi : X \rightarrow \mathbb{R}$. What properties does a potential function need to have so that it may be considered as a navigation function as defined in Section 8.4.1 and also yield a vector field that satisfies an acceleration bound? Sufficient conditions will be given that imply that a potential function will be a navigation function that satisfies the bound.

To give the conditions, it will first be important to characterize extrema of multivariate functions. Recall from basic calculus that a function $f : \mathbb{R} \rightarrow \mathbb{R}$ has a critical point when the first derivative is zero. At such points, the sign of the second derivative indicates whether the critical point is a minimum or maximum.

These ideas can be generalized to higher dimensions. A *critical point* of ϕ is one for which $\nabla\phi = 0$. The *Hessian* of ϕ is defined as the matrix

$$H(\phi) = \begin{pmatrix} \frac{\partial^2\phi}{\partial^2 x_1^2} & \frac{\partial^2\phi}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2\phi}{\partial x_1 \partial x_n} \\ \frac{\partial^2\phi}{\partial x_2 \partial x_1} & \frac{\partial^2\phi}{\partial^2 x_2^2} & \cdots & \frac{\partial^2\phi}{\partial x_2 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2\phi}{\partial x_n \partial x_1} & \frac{\partial^2\phi}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2\phi}{\partial^2 x_n^2} \end{pmatrix}. \tag{8.48}$$

At each critical point, the Hessian gives some information about the extremum. If the rank of $H(\phi)$ at x is n, then the Hessian indicates the kind of extremum. If (8.48) is positive definite,[13] then the ϕ achieves a *local minimum* at x. If (8.48) is negative definite,[14] then the ϕ achieves a *local maximum* at x. In all other cases, x is a *saddle point*. If the rank of $H(\phi)$ at x is less than n, then the Hessian is *degenerate*. In this case the Hessian cannot classify the type of extremum. An example of this occurs when x lies in a plateau (there is no direction in which ϕ increases or decreases. Such behavior is obviously bad for a potential function because the local operator would not be able to select a direction.

Suppose that the navigation function is required to be smooth, to ensure the existence of a gradient at every point. This enables gradient descent to be performed. If X is not contractible, then it turns out there must exist some critical points other than x_G at which $\nabla\phi(x) = 0$. The critical points can even be used to infer the topology of X, which is the basic idea in the subject of *Morse theory* [366, 126]. Unfortunately, this implies that there does not exist a solution navigation function for such spaces because the definition in Section 8.4.1 required that the integral curve from any state that can reach x_G must reach it using the vector field derived from the navigation function. If the initial state is a critical point, the integral curve is constant (the state remains at the critical point). Therefore, under the smoothness constraint, the definition of a navigation function should be modified to allow critical points at a small number of places (only on a set that has measure zero). It is furthermore required that the set of states from which the integral curves arrive at each critical point (i.e., the domain of attraction of each critical point) has measure zero. From all possible initial states, except from a set of measure zero, the integral curves must reach x_G, if it is reachable. This is ensured in the following definition.

[13] Positive definite for an $n \times n$ matrix A means that for all $x \in \mathbb{R}^n$, $x^T Ax > 0$. If A is symmetric (which applies to $H(\phi)$), then this is equivalent to A having all positive eigenvalues.

[14] Negative definite means that for all $x \in \mathbb{R}^n$, $x^T Ax < 0$. If A is symmetric, then this is equivalent to A having all negative eigenvalues.

A function $\phi : X \to \mathbb{R}$ is called a *navigation function in the sense of Rimon-Koditschek* if [416]:

1. It is smooth (or at least C^2).
2. Among all values on the connected component of $\mathcal{C}_{free}$ that contains x_G, there is only one local minimum, which is at x_G.[15]
3. It is maximal and constant on $\partial\mathcal{C}_{free}$, the boundary of $\mathcal{C}_{free}$.
4. It is a Morse function [366], which means that at each critical point x (i.e., $\nabla\phi|_x = 0$), the Hessian of ϕ is not degenerate.[16] Such functions are known to exist on any smooth manifold.

If ϕ is smooth in the C^∞ sense, then by Sard's Theorem [126] the set of critical points has measure zero.

Methods for constructing navigation functions are outlined in [416] for a general family of problems in which $\mathcal{C}_{free}$ has a semi-algebraic description. The basic idea is to start with simple shapes over which a navigation function can be easily defined. One example of this is a spherical subset of $\mathbb{R}^n$, which contains spherical obstacles. A set of distorting transformations is then developed to adapt the navigation functions to other shapes while ensuring that the four properties above are maintained. One such transformation extends a ball into any visibility region (in the sense defined in Section 8.4.3). This is achieved by smoothly stretching out the ball into the shape of the visibility region. (Such regions are sometimes called *star-shaped*.) The transformations given in [416] can be combined to define navigation functions for a large family of configuration spaces. The main problem is that the configuration space obstacles and the connectivity of $\mathcal{C}_{free}$ are represented only implicitly, which makes it difficult to correctly apply the method to complicated high-dimensional problems. One of the advantages of the approach is that proving convergence to the goal is simplified. In many cases, Lyapunov stability analysis can be performed (see Section 15.1.1).

Harmonic potential functions

Another important family of navigation functions is constructed from harmonic functions [128, 129, 130, 250, 276]. A function ϕ is called a *harmonic function* if it satisfies the differential equation

$$\nabla^2\phi = \sum_{i=1}^{n} \frac{\partial^2\phi}{\partial x_i^2} = 0. \tag{8.49}$$

[15] Some authors do not include the global minimum as a local minimum. In this case, one would say that there are no local minima.

[16] Technically, to be Morse, the values of the function must also be distinct at each critical point.

There are many possible solutions to the equation, depending on the conditions along the boundary of the domain over which ϕ is defined. A simple disc-based example is given in [127] for which an analytical solution exists. Complicated navigation functions are generally defined by imposing constraints on ϕ along the boundary of $\mathcal{C}_{free}$. A *Dirichlet boundary condition* means that the boundary must be held to a constant value. Using this condition, a harmonic navigation function can be developed that guides the state into a goal region from anywhere in a simply connected state space. If there are interior obstacles, then a *Neumann boundary condition* forces the velocity vectors to be tangent to the obstacle boundary. By solving (8.49) under a combination of both boundary conditions, a harmonic navigation function can be constructed that avoids obstacles by moving parallel to their boundaries and eventually landing in the goal. It has been shown under general conditions that navigation functions can be produced [130, 129]; however, the main problems are that the boundary of $\mathcal{C}_{free}$ is usually not constructed explicitly (recall why this was avoided in Chapter 5) and that a numerical solution to (8.49) is expensive to compute. This can be achieved, for example, by using Gauss-Seidel iterations (as indicated in [130]), which are related to value iteration (see [59] for the distinction). A sampling-based approach to constructing navigation functions via harmonic functions is presented in [68]. Value iteration will be used to produce approximate, optimal navigation functions in Section 8.5.2.

8.5 Sampling-Based Methods for Continuous Spaces

The methods in Section 8.4 can be considered as the feedback-case analogs to the combinatorial methods of Chapter 6. Although such methods provide elegant solutions to the problem, the issue arises once again that they are either limited to lower dimensional problems or problems that exhibit some special structure. This motivates the introduction of sampling-based methods. This section presents the feedback-case analog to Chapter 5.

8.5.1 Computing a Composition of Funnels

Mason introduced the concept of a *funnel* as a metaphor for motions that converge to the same small region of the state space, regardless of the initial position [351]. As grains of sand in a funnel, they follow the slope of the funnel until they reach the opening at the bottom. A navigation function can be imagined as a funnel that guides the state into the goal. For example, the cost-to-go function depicted in Figure 8.13d can be considered as a complicated funnel that sends each piece of sand along an optimal path to the goal.

Rather than designing a single funnel, consider decomposing the state space into a collection of simple, overlapping regions. Over each region, a funnel can be designed that leads the state into another funnel; see Figure 8.14. As an example, the approach in [83] places a *Lyapunov function* (such functions are covered in

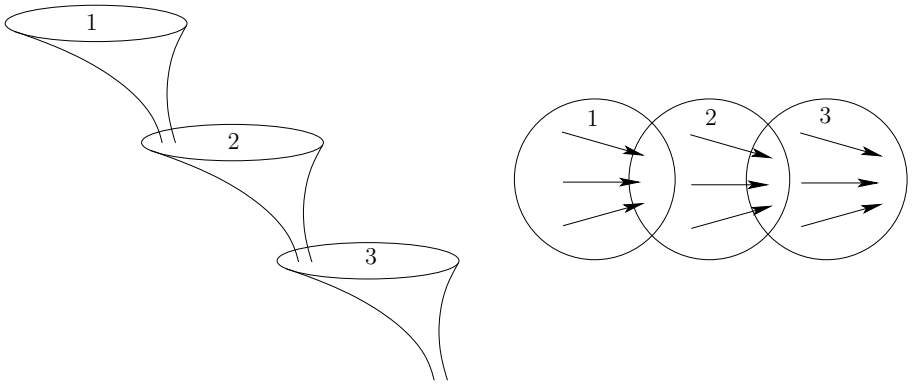


Figure 8.14: A navigation function and corresponding vector field can be designed as a composition of funnels.

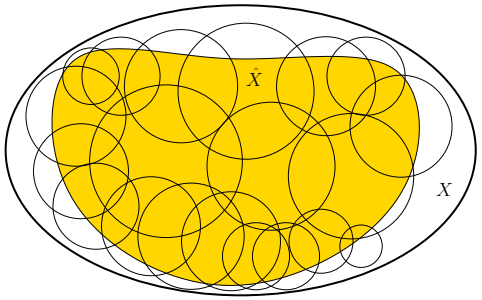


Figure 8.15: An approximate cover is shown. Every point of $\tilde{X}$ is contained in at least one neighborhood, and $\tilde{X}$ is a subset of X.

Section 15.1.2) over each funnel to ensure convergence to the next funnel. A feedback plan can be constructed by composing several funnels. Starting from some initial state in X, a sequence of funnels is visited until the goal is reached. Each funnel essentially solves the subgoal of reaching the next funnel. Eventually, a funnel is reached that contains the goal, and a navigation function on this funnel causes the goal to be reached. In the context of sensing uncertainty, for which the funnel metaphor was developed, the composition of funnels becomes the preimage planning framework [346], which is covered in Section 12.5.1. In this section, however, it is assumed that the current state is always known.

An approximate cover

Figure 8.15 illustrates the notion of an approximate cover, which will be used to represent the funnel domains. Let $\tilde{X}$ denote a subset of a state space X. A *cover* of $\tilde{X}$ is a collection $\mathcal{O}$ of sets for which

1. $O \subseteq X$ for each $O \in \mathcal{O}$.

2. $\tilde{X}$ is a subset of the union of all sets in the cover:

$$\tilde{X} \subseteq \bigcup_{O \in \mathcal{O}} O. \tag{8.50}$$

Let each $O \in \mathcal{O}$ be called a *neighborhood*. The notion of a cover was actually used in Section 8.3.2 to define a smooth manifold using a cover of coordinate neighborhoods.

In general, a cover allows the following:

1. Any number of neighborhoods may overlap (have nonempty intersection).

2. Any neighborhood may contain points that lie outside of $\tilde{X}$.

A cell decomposition, which was introduced in Section 6.3.1, is a special kind of cover for which the neighborhoods form a partition of $\tilde{X}$, and they must fit together nicely (recall Figure 6.15).

So far, no constraints have been placed on the neighborhoods. They should be chosen in practice to greatly simplify the design of a navigation function over each one. For the original motion planning problem, cell decompositions were designed to make the determination of a collision-free path trivial in each cell. The same idea applies here, except that we now want to construct a feedback plan. Therefore, it is usually assumed that the cells have a simple shape.

A cover is called *approximate* if $\tilde{X}$ is a strict subset of X. Ideally, we would like to develop an *exact cover*, which implies that $\tilde{X} = X$ and each neighborhood has some nice property, such as being convex. Developing such covers is possible in practice for state spaces that are either low-dimensional or exhibit some special structure. This was observed for the cell decomposition methods of Chapter 6.

Consider constructing an approximate cover for X. The goal should be to cover as much of X as possible. This means that $\mu(X \setminus \tilde{X})$ should be made as small as possible, in which μ denotes Lebesgue measure, as defined in Section 5.1.3. It is also desirable to ensure that $\tilde{X}$ preserves the connectivity of X. In other words, if a path between two points exists in X, then it should also exist in $\tilde{X}$.

Defining a feedback plan over a cover

The ideas from Section 8.4.2 can be adapted to define a feedback plan over $\tilde{X}$ using a cover. Let $\check{X}$ denote a discrete state space in which each superstate is a neighborhood. Most of the components of the associated discrete planning problems are the same as in Section 8.4.2. The only difference is in the definition of superactions because neighborhoods can overlap in a cover. For each neighborhood $O \in \mathcal{O}$, a superaction exists for each other neighborhood, $O' \in \mathcal{O}$ such that $O \cap O' \neq \emptyset$ (usually, their interiors overlap to yield $\text{int}(O) \cap \text{int}(O') \neq \emptyset$).

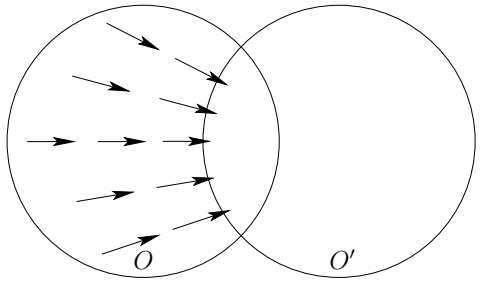


Figure 8.16: A transition from O to O' is caused by a vector field on O for which all integral curves lead into $O \cap O'$.

Note that in the case of a cell decomposition, this produces no superactions because it is a partition. To follow the metaphor of composing funnels, the domains of some funnels should overlap, as shown in Figure 8.14. A transition from one neighborhood, O, to another, O', is obtained by defining a vector field on O that sends all states from $O \setminus O'$ into $O \cap O'$; see Figure 8.16. Once O' is reached, the vector field of O is no longer followed; instead, the vector field of O' is used. Using the vector field of O', a transition may be applied to reach another neighborhood. Note that the jump from the vector field of O to that of O' may cause the feedback plan to be a discontinuous vector field on $\tilde{X}$. If the cover is designed so that $O \cap O'$ is large (if they intersect), then gradual transitions may be possible by blending the vector fields from O and O'.

Once the discrete problem has been defined, a discrete feedback plan can be computed over $\check{X}$, as defined in Section 8.2. This is converted into a feedback plan over X by defining a vector field on each neighborhood that causes the appropriate transitions. Each $\check{x} \in \check{X}$ can be interpreted both as a superstate and a neighborhood. For each $\check{x}$, the discrete feedback plan produces a superaction $\check{u} = \pi(\check{x})$, which yields a new neighborhood $\check{x}'$. The vector field over $\check{x} = O$ is then designed to send all states into $\check{x}' = O'$.

If desired, a navigation function ϕ over X can even be derived from a navigation function, $\check{\phi}$, over $\check{X}$. Suppose that $\check{\phi}$ is constructed so that every $\check{\phi}(\check{x})$ is distinct for every $\check{x} \in \check{X}$. Any navigation function can be easily transformed to satisfy this constraint (because $\check{X}$ is finite). Let ϕ_O denote a navigation function over some $O \in \mathcal{O}$. Assume that X_G is a point, x_G (extensions can be made to more general cases). For every neighborhood $O \in \mathcal{O}$ such that $x_G \notin O$, ϕ_O is defined so that performing gradient descent leads into the overlapping neighborhood for which $\check{\phi}(\check{x})$ is smallest. If O contains x_G, the navigation function ϕ_O simply guides the state to x_G.

The navigation functions over each $O \in \mathcal{O}$ can be easily pieced together to yield a navigation function over all of X. In places where multiple neighborhoods overlap, ϕ is defined to be the navigation function associated with the neighbor-

hood for which $\check{\phi}(\check{x})$ is smallest. This can be achieved by adding a large constant to each ϕ_O. Let c denote a constant for which $\phi_O(x) < c$ over all $O \in \mathcal{O}$ and $x \in O$ (it is assumed that each ϕ_O is bounded). Suppose that $\check{\phi}$ assumes only integer values. Let $\mathcal{O}(x)$ denote the set of all $O \in \mathcal{O}$ such that $x \in O$. The navigation function over X is defined as

$$\phi(x) = \min_{O \in \mathcal{O}(x)} \Big\{ \phi_O(x) + c\, \check{\phi}(O) \Big\}. \tag{8.51}$$

A sampling-based approach

There are numerous alternative ways to construct a cover. To illustrate the ideas, an approach called the *sampling-based neighborhood graph* is presented here [480]. Suppose that $X = \mathcal{C}_{free}$, which is a subset of some configuration space. As introduced in Section 5.4, let α be a dense, infinite sequence of samples in X. Assume that a collision detection algorithm is available that returns the distance, (5.28), between the robot and obstacles in the world. Such algorithms were described in Section 5.3.

An incremental algorithm is given in Figure 8.17. Initially, $\mathcal{O}$ is empty. In each iteration, if $\alpha(i) \in \mathcal{C}_{free}$ and it is not already contained in some neighborhood, then a new neighborhood is added to $\mathcal{O}$. The two main concerns are 1) how to define a new neighborhood, O, such that $O \subset \mathcal{C}_{free}$, and 2) when to terminate. At any given time, the cover is approximate. The union of all neighborhoods is $\tilde{X}$, which is a strict subset of X. In comparison to Figure 8.15, the cover is a special case in which the neighborhoods do not extend beyond $\tilde{X}$.

Defining new neighborhoods For defining new neighborhoods, it is important to keep them simple because during execution, the neighborhoods that contain the state x must be determined quickly. Suppose that all neighborhoods are open balls:

$$B(x, r) = \{x' \in X \mid \rho(x, x') < r\}, \tag{8.52}$$

in which ρ is the metric on $\mathcal{C}$. There are efficient algorithms for determining whether $x \in O$ for some $O \in \mathcal{O}$, assuming all of the neighborhoods are balls [365]. In practice, methods based on Kd-trees yield good performance [36, 38] (recall Section 5.5.2). A new ball, $B(x, r)$, can be constructed in Step 3 for $x = \alpha(i)$, but what radius can be assigned? For a point robot that translates in $\mathbb{R}^2$ or $\mathbb{R}^3$, the Hausdorff distance d between the robot and obstacles in $\mathcal{W}$ is precisely the distance to $\mathcal{C}_{obs}$ from $\alpha(i)$. This implies that we can set $r = d$, and $B(x, r)$ is guaranteed to be collision-free.

In a general configuration space, it is possible to find a value of r such that $B(x, r) \subseteq \mathcal{C}_{free}$, but in general $r < d$. This issue arose in Section 5.3.4 for checking path segments. The transformations of Sections 3.2 and 3.3 become important in the determination of r. For illustrative purposes, suppose that $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$,

INCREMENTAL COVER CONSTRUCTION

1. Initialize $\mathcal{O} = \emptyset$ and $i = 1$.
2. Let $x = \alpha(i)$, and let d be the distance returned by the collision detection algorithm applied at x.
3. If $d > 0$ (which implies that $x \in \mathcal{C}_{free}$) and $x \notin O$ for all $O \in \mathcal{O}$, then insert a new neighborhood, O_n, into $\mathcal{O}$. The neighborhood size and shape are determined from x and d.
4. If the termination condition is not satisfied, then let $i := i + 1$, and go to Step 1.
5. Remove any neighborhoods from $\mathcal{O}$ that are contained entirely inside of another neighborhood.

Figure 8.17: The cover is incrementally extended by adding new neighborhoods that are guaranteed to be collision-free.

which corresponds to a rigid robot, $\mathcal{A}$, that can translate and rotate in $\mathcal{W} = \mathbb{R}^2$. Each point $a \in \mathcal{A}$ is transformed using (3.35). Now imagine starting with some configuration $q = (x, y, \theta)$ and perturbing each coordinate by some Δx, Δy, and $\Delta\theta$. What is the maximum distance that a point on $\mathcal{A}$ could travel? Translation affects all points on $\mathcal{A}$ the same way, but rotation affects points differently. Recall Figure 5.12 from Section 5.3.4. Let $a_r \in \mathcal{A}$ denote the point that is furthest from the origin $(0, 0)$. Let r denote the distance from a_r to the origin. If the rotation is perturbed by some small amount, $\Delta\theta$, then the displacement of any $a \in \mathcal{A}$ is no more than $r\Delta\theta$. If all three configuration parameters are perturbed, then

$$(\Delta x)^2 + (\Delta y)^2 + (r\Delta\theta)^2 < d^2 \tag{8.53}$$

is the constraint that must be satisfied to ensure that the resulting ball is contained in $\mathcal{C}_{free}$. This is actually the equation of a solid ellipsoid, which becomes a ball if $r = 1$. This can be made into a ball by reparameterizing $SE(2)$ so that $\Delta\theta$ has the same affect as Δx and Δy. A transformation $h : \theta \mapsto r\theta$ maps θ into a new domain $Z = [0, 2\pi r)$. In this new space, the equation of the ball is

$$(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2 < d^2, \tag{8.54}$$

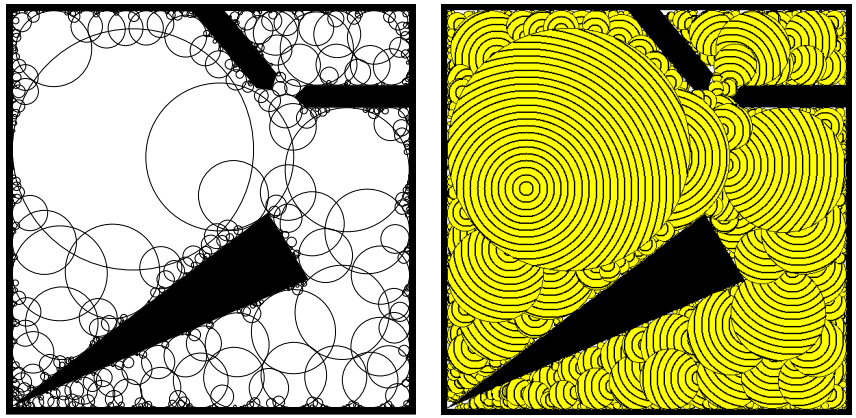
in which Δz represents the change in $z \in Z$. The reparameterized version of (3.35) is

$$T = \begin{pmatrix} \cos(\theta/r) & -\sin(\theta/r) & x_t \\ \sin(\theta/r) & \cos(\theta/r) & y_t \\ 0 & 0 & 1 \end{pmatrix}. \tag{8.55}$$

For a 3D rigid body, similar reparameterizations can be made to Euler angles or quaternions to generate six-dimensional balls. Extensions can be made to chains of bodies [480]. One of the main difficulties, however, is that the balls are not the largest possible. In higher dimensions the problem becomes worse because numerous balls are needed, and the radii constructed as described above tend to be much smaller than what is possible. The number of balls can be reduced by also allowing axis-aligned cylinders, but it still remains difficult to construct a cover over a large fraction of $\mathcal{C}_{free}$ in more than six dimensions.

Termination The sampling-based planning algorithms in Chapter 5 were designed to terminate upon finding a solution path. In the current setting, termination is complicated by the fact that we are interested in solutions from all initial configurations. Since α is dense, the volume of uncovered points in $\mathcal{C}_{free}$ tends to zero. After some finite number of iterations, it would be nice to measure the quality of the approximation and then terminate when the desired quality is achieved. This was also possible with the visibility sampling-based roadmap in Section 5.6.2. Using random samples, an estimate of the fraction of $\mathcal{C}_{free}$ can be obtained by recording the percentage of failures in obtaining a sample in $\mathcal{C}_{free}$ that is outside of the cover. For example, if a new neighborhood is created only once in 1000 iterations, then it can be estimated that 99.9 percent of $\mathcal{C}_{free}$ is covered. High-probability bounds can also be determined. Termination conditions are given in [480] that ensure with probability greater than P_c that at least a fraction $\alpha \in (0,1)$ of $\mathcal{C}_{free}$ has been covered. The constants P_c and α are given as parameters to the algorithm, and it will terminate when the condition has been satisfied using rigorous statistical tests. If deterministic sampling is used, then termination can be made to occur based on the dispersion, which indicates the largest ball in $\mathcal{C}_{free}$ that does not contain the center of another neighborhood. One problem with volume-based criteria, such as those suggested here, is that there is no way to ensure that the cover preserves the connectivity of $\mathcal{C}_{free}$. If two portions of $\mathcal{C}_{free}$ are connected by a narrow passage, the cover may miss a neighborhood that has very small volume yet is needed to connect the two portions.

Example 8.18 (2D Example of Computed Funnels) Figure 8.18 shows a 2D example that was computed using random samples and the algorithm in Figure 8.17. Note that once a cover is computed, it can be used to rapidly compute different navigation functions and vector fields for various goals. This example is mainly for illustrative purposes. For the case of a polygonal environment, constructing covers based on convex polygons would be more efficient. ■



(a) (b)

Figure 8.18: (a) A approximate cover for a 2D configuration space. (b) Level sets of a navigation function

.

8.5.2 Dynamic Programming with Interpolation

This section concludes Part II by solving the motion planning problem with value iteration, which was introduced in Section 2.3. It has already been applied to obtain discrete feedback plans in Section 8.2. It will now be adapted to continuous spaces by allowing interpolation first over a continuous state space and then by additionally allowing interpolation over a continuous action space. This yields a numerical approach to computing optimal navigation functions and feedback plans for motion planning. The focus will remain on backward value iteration; however, the interpolation concepts may also be applied in the forward direction. The approach here views optimal feedback motion planning as a discrete-time optimal control problem [22, 55, 79, 302].

Using interpolation for continuous state spaces

Consider a problem formulation that is identical to Formulation 8.1 except that X is allowed to be continuous. Assume that X is bounded, and assume for now that the action space, $U(x)$, it finite for all $x \in X$. Backward value iteration can be applied. The dynamic programming arguments and derivation are identical to those in Section 2.3. The resulting recurrence is identical to (2.11) and is repeated here for convenience:

$$G_k^*(x_k) = \min_{u_k \in U(x_k)} \Big\{ l(x_k, u_k) + G_{k+1}^*(x_{k+1}) \Big\}. \tag{8.56}$$

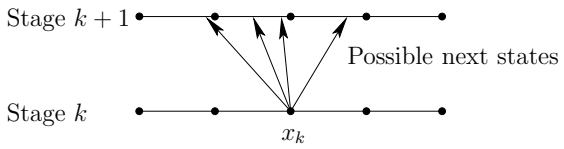


Figure 8.19: Even though x_k is a sample point, the next state, x_{k+1}, may land between sample points. For each $u_k \in U(x_k)$, interpolation may be needed for the resulting next state, $x_{k+1} = f(x_k, u_k)$.

The only difficulty is that $G_k^*(x_k)$ cannot be stored for every $x_k \in X$ because X is continuous. There are two general approaches. One is to approximate G_k^* using a parametric family of surfaces, such as polynomials or nonlinear basis functions derived from neural networks [60]. The other is to store G_k^* only over a finite set of sample points and use interpolation to obtain its value at all other points [301, 302].

Suppose that a finite set $S \subset X$ of samples is used to represent cost-to-go functions over X. The evaluation of (8.56) using interpolation is depicted in Figure 8.19. In general, the samples should be chosen to reduce the dispersion (defined in Section 5.2.3) as much as possible. This prevents attempts to approximate the cost-to-go function on large areas that contain no sample points. The rate of convergence ultimately depends on the dispersion [58] (in combination with Lipschitz conditions on the state transition equation and the cost functional). To simplify notation and some other issues, assume that S is a grid of regularly spaced points in $\mathbb{R}^n$.

First, consider the case in which $X = [0,1] \subset \mathbb{R}$. Let $S = \{s_0, s_1, \ldots, s_r\}$, in which $s_i = i/r$. For example, if $r = 3$, then $S = \{0, 1/3, 2/3, 1\}$. Note that this always yields points on the boundary of X, which ensures that for any point in $(0,1)$ there are samples both above and below it. Let i be the largest integer such that $s_i < x$. This implies that $s_{i+1} > x$. The samples s_i and s_{i+1} are called *interpolation neighbors* of x.

The value of G_{k+1}^* in (8.56) at any $x \in [0,1]$ can be obtained via *linear interpolation* as

$$G_{k+1}^*(x) \approx \alpha G_{k+1}^*(s_i) + (1-\alpha) G_{k+1}^*(s_{i+1}), \tag{8.57}$$

in which the coefficient $\alpha \in [0,1]$ is computed as

$$\alpha = 1 - \frac{x - s_i}{r}. \tag{8.58}$$

If $x = s_i$, then $\alpha = 1$, and (8.57) reduces to $G_{k+1}^*(s_i)$, as expected. If $x = s_{i+1}$, then $\alpha = 0$, and (8.57) reduces to $G_{k+1}^*(s_{i+1})$. At all points in between, (8.57) blends the cost-to-go values at s_i and s_{i+1} using α to provide the appropriate weights.

The interpolation idea can be naturally extended to multiple dimensions. Let X be a bounded subset of $\mathbb{R}^n$. Let S represent an n-dimensional grid of points

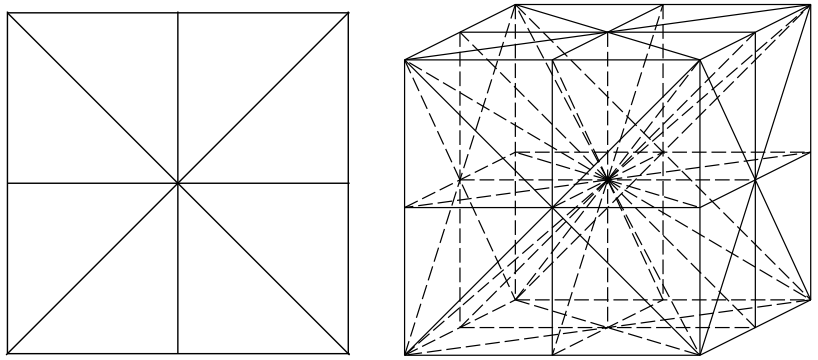


Figure 8.20: Barycentric subdivision can be used to partition each cube into simplexes, which allows interpolation to be performed in $O(n \lg n)$ time, instead of $O(2^n)$.

in $\mathbb{R}^n$. Each sample in S is denoted by $s(i_1, i_2, \ldots, i_n)$. For some $x \in X$, there are 2^n interpolation neighbors that "surround" it. These are the corners of an n-dimensional cube that contains x. Let $x = (x_1, \ldots, x_n)$. Let i_j denote the largest integer for which the jth coordinate of $s(i_1, i_2, \ldots, i_n)$ is less than x_j. The 2^n samples are all those for which either i_j or $i_j + 1$ appears in the expression $s(\cdot, \cdot, \ldots, \cdot)$, for each $j \in \{1, \ldots, n\}$. This requires that samples exist in S for all of these cases. Note that X may be a complicated subset of $\mathbb{R}^n$, provided that for any $x \in X$, all of the required 2^n interpolation neighbors are in S. Using the 2^n interpolation neighbors, the value of G_{k+1}^* in (8.56) on any $x \in X$ can be obtained via *multi-linear interpolation*. In the case of $n = 2$, this is expressed as

$$\begin{aligned} G_{k+1}^*(x) \approx \; & \alpha_1 \alpha_2 \, G_{k+1}^*(s(i_1, i_2)) + \\ & \alpha_1 (1 - \alpha_2) \, G_{k+1}^*(s(i_1, i_2 + 1)) + \\ & (1 - \alpha_1) \alpha_2 \, G_{k+1}^*(s(i_1 + 1, i_2)) + \\ & (1 - \alpha_1)(1 - \alpha_2) \, G_{k+1}^*(s(i_1 + 1, i_2 + 1)), \end{aligned} \tag{8.59}$$

in which α_1 and α_2 are defined similarly to α in (8.58) but are based on distances along the x_1 and x_2 directions, respectively. The expressions for multi-linear interpolation in higher dimensions are similar but are more cumbersome to express. Higher order interpolation, such a quadratic interpolation may alternatively be used [302].

Unfortunately, the number of interpolation neighbors grows exponentially with the dimension, n. Instead of using all 2^n interpolation neighbors, one improvement is to decompose the cube defined by the 2^n samples into simplexes. Each simplex has only $n+1$ samples as its vertices. Only the vertices of the simplex that contains

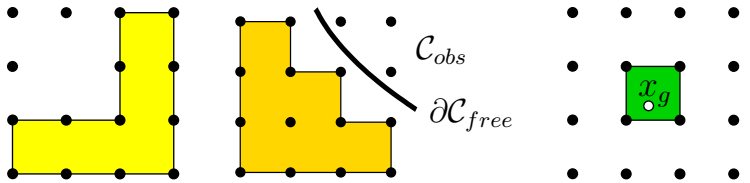


Figure 8.21: (a) An interpolation region, $R(S)$, is shown for a set of sample points, S. (b) The interpolation region that arises due to obstacles. (c) The interpolation region for goal points must not be empty.

x are declared to be the interpolation neighbors of x; this reduces the cost of evaluating $G^*_{k+1}(x)$ to $O(n)$ time. The problem, however, is that determining the simplex that contains x may be a challenging *point-location problem* (a common problem in computational geometry [146]). If barycentric subdivision is used to decompose the cube using the midpoints of all faces, then the point-location problem can be solved in $O(n \lg n)$ time [145, 310, 376], which is an improvement over the $O(2^n)$ scheme described above. Examples of this decomposition are shown for two and three dimensions in Figure 8.20. This is sometimes called the *Coxeter-Freudenthal-Kuhn triangulation*. Even though n is not too large due to practical performance considerations (typically, $n \leq 6$), substantial savings occur in implementations, even for $n = 3$.

It will be convenient to refer directly to the set of all points in X for which all required interpolation neighbors exist. For any finite set $S \subseteq X$ of sample points, let the *interpolation region* $R(S)$ be the set of all $x \in X \setminus S$ for which $G^*(x)$ can be computed by interpolation. This means that $x \in R(S)$ if and only if all interpolation neighbors of x lie in S. Figure 8.21a shows an example. Note that some sample points may not contribute any points to R. If a grid of samples is used to approximate G^*, then the volume of $X \setminus R(S)$ approaches zero as the sampling resolution increases.

Continuous action spaces Now suppose that $U(x)$ is continuous, in addition to X. Assume that $U(x)$ is both a closed and bounded subset of $\mathbb{R}^n$. Once again, the dynamic programming recurrence, (8.56), remains the same. The trouble now is that the *min* represents an optimization problem over an uncountably infinite number of choices. One possibility is to employ nonlinear optimization techniques to select the optimal $u \in U(x)$. The effectiveness of this depends heavily on $U(x)$, X, and the cost functional.

Another approach is to evaluate (8.56) over a finite set of samples drawn from $U(x)$. Again, it is best to choose samples that reduce the dispersion as much as possible. In some contexts, it may be possible to eliminate some actions from consideration by carefully utilizing the properties of the cost-to-go function and its representation via interpolation.

The connection to feedback motion planning

The tools have now been provided to solve motion planning problems using value iteration. The configuration space is a continuous state space; let $X = \mathcal{C}_{free}$. The action space is also continuous, $U(x) = T_x(X)$. For motion planning problems, $0 \in T_x(X)$ is only obtained only when u_T is applied. Therefore, it does not need to be represented separately. To compute optimal cost-to-go functions for motion planning, the main concerns are as follows:

1. The action space must be bounded.
2. A discrete-time approximation must be made to derive a state transition equation that works over stages.
3. The cost functional must be discretized.
4. The obstacle region, $\mathcal{C}_{obs}$, must be taken into account.
5. At least some interpolation region must yield $G^*(x) = 0$, which represents the goal region.

We now discuss each of these.

Bounding the action space Recall that using normalized vector fields does not alter the existence of solutions. This is convenient because $U(x)$ needs to be bounded to approximate it with a finite set of samples. It is useful to restrict the action set to obtain

$$U(x) = \{u \in \mathbb{R}^n \mid \|u\| \leq 1\}. \tag{8.60}$$

To improve performance, it is sometimes possible to use only those u for which $\|u\| = 1$ or $u = 0$; however, numerical instability problems may arise. A finite sample set for $U(x)$ should have low dispersion and always include $u = 0$.

Obtaining a state transition equation Value iterations occur over discrete stages; however, the integral curves of feedback plans occur over continuous time. Therefore, the time interval T needs to be sampled. Let Δt denote a small positive constant that represents a fixed interval of time. Let the stage index k refer to time $(k-1)\Delta t$. Now consider representing a velocity field $\dot{x}$ over $\mathbb{R}^n$. By definition,

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t + \Delta t) - x(t)}{\Delta t}. \tag{8.61}$$

In Section 8.3.1, a velocity field was defined by assigning some $u \in U(x)$ to each $x \in X$. If the velocity vector u is integrated from $x(t)$ over a small Δt, then a new state, $x(t + \Delta t)$, results. If u remains constant, then

$$x(t + \Delta t) = x(t) + \Delta t\, u, \tag{8.62}$$

which is called an *Euler approximation*. If a feedback plan is executed, then u is determined from x via $u = \pi(x(t))$. In general, this means that u could vary as the state is integrated forward. In this case, (8.62) is only approximate,

$$x(t + \Delta t) \approx x(t) + \Delta t\ \pi(x(t)). \tag{8.63}$$

The expression in (8.62) can be considered as a state transition equation that works over stages. Let $x_{k+1} = x(t + \Delta t)$ and $x_k = x(t)$. The transitions can now be expressed as

$$x_{k+1} = f(x_k, u) = x_k + \Delta t\ u. \tag{8.64}$$

The quality of the approximation improves as Δt decreases. Better approximations can be made by using more sample points along time. The most widely known approximations are the Runge-Kutta family. For optimal motion planning, it turns out that the direction vector almost always remains constant along the integral curve. For example, in Figure 8.13d, observe that piecewise-linear paths are obtained by performing gradient descent of the optimal navigation function. The direction vector is constant over most of the resulting integral curve (it changes only as obstacles are contacted). Therefore, approximation problems tend not to arise in motion planning problems. When approximating dynamical systems, such as those presented in Chapter 13, then better approximations are needed; see Section 14.3.2. One important concern is that Δt is chosen in a way that is compatible with the grid resolution. If Δt is so small that the actions do not change the state enough to yield new interpolation neighbors, then the interpolated cost-to-go values will remain constant. This implies that Δt must be chosen to ensure that $x(t + \Delta t)$ has a different set of interpolation neighbors than $x(t)$.

An interesting connection can be made to the approximate motion planning problem that was developed in Section 7.7. Formulation 7.4 corresponds precisely to the approximation defined here, except that ϵ was used instead of Δt because velocities were not yet considered (also, the initial condition was specified because there was no feedback). Recall the different possible action spaces shown in Figure 7.41. As stated in Section 7.7, if the Manhattan or independent-joint models are used, then the configurations remain on a grid of points. This enables discrete value iterations to be performed. A discrete feedback plan and navigation function, as considered in Section 8.2.3, can even be computed. If the Euclidean motion model is used, which is more natural, then the transitions allow a continuum of possible configurations. This case can finally be handled by using interpolation over the configuration space, as described in this section.

Approximating the cost functional A discrete cost functional must be derived from the continuous cost functional, (8.39). The final term is just assigned as $l_F(x_F) = l_F(x(t_f))$. The cost at each stage is

$$l_d(x_k, u_k) = \int_0^{\Delta t} l(x(t), u(t))dt, \tag{8.65}$$

and $l_d(x_k, u_k)$ is used in the place of $l(x_k, u_k)$ in (8.56). For many problems, the integral does not need to be computed repeatedly. To obtain Euclidean shortest paths, $l_d(x_k, u_k) = \|u_k\|$ can be safely assigned for all $x_k \in X$ and $u_k \in U(x_k)$. A reasonable approximation to (8.65) if Δt is small is $l(x(t), u(t))\Delta t$.

Handling obstacles A simple way to handle obstacles is to determine for each $x \in S$ whether $x \in \mathcal{C}_{obs}$. This can be computed and stored in an array before the value iterations are performed. For rigid robots, this can be efficiently computed using *fast Fourier transforms* [262]. For each $x \in \mathcal{C}_{obs}$, $G^*(x) = \infty$. No value iterations are performed on these states; their values must remain at infinity. During the evaluation of (8.59) (or a higher dimensional version), different actions are attempted. For each action, it is required that all of the interpolation neighbors of x_{k+1} lie in $\mathcal{C}_{free}$. If one of them lies in $\mathcal{C}_{obs}$, then that action produces infinite cost. This has the effect of automatically reducing the interpolation region, $R(S)$, to all cubes whose vertices all lie in $\mathcal{C}_{free}$, as shown in Figure 8.21b. All samples in $\mathcal{C}_{obs}$ are assumed to be deleted from S in the remainder of this section; however, the full grid is still used for interpolation so that infinite values represent the obstacle region.

Note that as expressed so far, it is possible that points in $\mathcal{C}_{obs}$ may lie in $R(S)$ because collision detection is performed only on the samples. In practice, either the grid resolution must be made fine enough to minimize the chance of this error occurring or distance information from a collision detection algorithm must be used to infer that a sufficiently large ball around each sample is collision free. If an interpolation region cannot be assured to lie in $\mathcal{C}_{free}$, then the resolution may have to be increased, at least locally.

Handling the goal region Recall that backward value iterations start with the final cost-to-go function and iterate backward. Initially, the final cost-to-go is assigned as infinity at all states except those in the goal. To properly initialize the final cost-to-go function, there must exist some subset of X over which the zero value can be obtained by interpolation. Let $G = S \cap X_G$. The requirement is that the interpolation region $R(G)$ must be nonempty. If this is not satisfied, then the grid resolution needs to be increased or the goal set needs to be enlarged. If X_g is a single point, then it needs to be enlarged, regardless of the resolution (unless an alternative way to interpolate near a goal point is developed). In the interpolation region shown in Figure 8.21c, all states in the vicinity of x_G yield an interpolated cost-to-go value of zero. If such a region did not exist, then all costs would remain at infinity during the evaluation of (8.59) from any state. Note that Δt must be chosen large enough to ensure that new samples can reach G.

Using G^* as a navigation function After the cost-to-go values stabilize, the resulting cost-to-go function, G^* can be used as a navigation function. Even though G^* is defined only over $S \subset X$, the value of the navigation function can

be obtained using interpolation over any point in $R(S)$. The optimal action is selected as the one that satisfies the min in (8.6). This means that the state trajectory does not have to visit the grid points as in the Manhattan model. A trajectory can visit any point in $R(S)$, which enables trajectories to converge to the true optimal solution as Δt and the grid spacing tend to zero.

Topological considerations So far there has been no explicit consideration of the topology of $\mathcal{C}$. Assuming that $\mathcal{C}$ is a manifold, the concepts discussed so far can be applied to any open set on which coordinates are defined. In practice, it is often convenient to use the manifold representations of Section 4.1.2. The manifold can be expressed as a cube, $[0,1]^n$, with some faces identified to obtain $[0,1]^n/\sim$. Over the interior of the cube, all of the concepts explained in this section work without modification. At the boundary, the samples used for interpolation must take the identification into account. Furthermore, actions, u_k, and next states, x_{k+1}, must function correctly on the boundary. One must be careful, however, in declaring that some solution is optimal, because Euclidean shortest paths depend on the manifold parameterization. This ambiguity is usually resolved by formulating the cost in terms of some physical quantity, such as time or energy. This often requires modeling dynamics, which will be covered in Part IV.

Value iteration with interpolation is extremely general. It is a generic algorithm for approximating the solution to optimal control problems. It can be applied to solve many of the problems in Part IV by restricting $U(x)$ to take into account complicated differential constraints. The method can also be extended to problems that involve explicit uncertainty in predictability. This version of value iteration is covered in Section 10.6.

Obtaining Dijkstra-like algorithms

For motion planning problems, it is expected that $x(t+\Delta t)$, as computed from (8.62), is always close to $x(t)$ relative to the size of X. This suggests the use of a Dijkstra-like algorithm to compute optimal feedback plans more efficiently. As discussed for the finite case in Section 2.3.3, many values remain unchanged during the value iterations, as indicated in Example 2.5. Dijkstra's algorithm maintains a data structure that focuses the computation on the part of the state space where values are changing. The same can be done for the continuous case by carefully considering the sample points [310].

During the value iterations, there are three kinds of sample points, just as in the discrete case (recall from Section 2.3.3):

1. **Dead:** The cost-to-go has stabilized to its optimal value.

2. **Alive:** The current cost-to-go is finite, but it is not yet known whether the value is optimal.

3. **Unvisited:** The cost-to-go value remains at infinity because the sample has not been reached.

The sets are somewhat harder to maintain for the case of continuous state spaces because of the interaction between the sample set S and the interpolated region $R(S)$.

Imagine the first value iteration. Initially, all points in G are set to zero values. Among the collection of samples S, how many can reach $R(G)$ in a single stage? We expect that samples very far from G will not be able to reach $R(G)$; this keeps their values are infinity. The samples that are close to G should reach it. It would be convenient to prune away from consideration all samples that are too far from G to lower their value. In every iteration, we eliminate iterating over samples that are too far away from those already reached. It is also unnecessary to iterate over the dead samples because their values no longer change.

To keep track of reachable samples, it will be convenient to introduce the notion of a backprojection, which will be studied further in Section 10.1. For a single state, $x \in X$, its *backprojection* is defined as

$$\mathrm{B}(x) = \{x' \in X \mid \exists u' \in U(x') \text{ such that } x = f(x', u')\}. \tag{8.66}$$

The backprojection of a set, $X' \subseteq X$, of points is just the union of backprojections for each point:

$$\mathrm{B}(X') = \bigcup_{x \in X'} \mathrm{B}(x). \tag{8.67}$$

Now consider a version of value iteration that uses backprojections to eliminate some states from consideration because it is known that their values cannot change. Let i refer to the number of stages considered by the current value iteration. During the first iteration, $i = 1$, which means that all one-stage trajectories are considered. Let S be the set of samples (assuming already that none lie in $\mathcal{C}_{obs}$). Let D_i and A_i refer to the *dead* and *alive* samples, respectively. Initially, $D_1 = G$, the set of samples in the goal set. The first set, A_1, of alive samples is assigned by using the concept of a frontier. The *frontier* of a set $S' \subseteq S$ of sample points is

$$\mathrm{Front}(S') = (\mathrm{B}(R(S')) \setminus S') \cap S. \tag{8.68}$$

This is the set of sample points that can reach $R(S')$ in one stage, excluding those already in S'. Figure 8.22 illustrates the frontier. Using (8.68), A_1 is defined as $A_1 = \mathrm{Front}(D_1)$.

Now the approach is described for iteration i. The cost-to-go update (8.56) is computed at all points in A_i. If $G^*_{k+1}(s) = G^*_k(s)$ for some $s \in A_i$, then s is declared dead and moved to D_{i+1}. Samples are never removed from the dead set; therefore, all points in D_i are also added to D_{i+1}. The next active set, A_{i+1}, includes all samples in A_i, excluding those that were moved to the dead set. Furthermore, all samples in $\mathrm{Front}(A_i)$ are added to A_{i+1} because these will produce a finite cost-to-go value in the next iteration. The iterations continue as usual until some

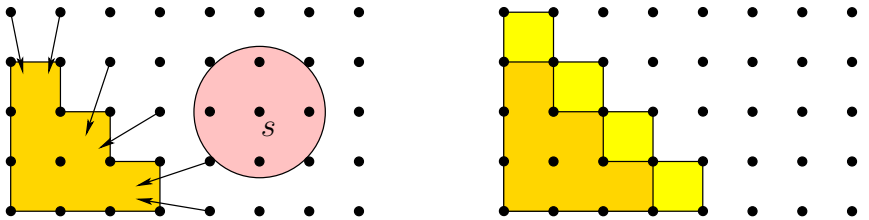


Figure 8.22: An illustration of the frontier concept: (a) the shaded disc indicates the set of all reachable points in one stage, from the sample on the left. The sample cannot reach in one stage the shaded region on the right, which represents $R(S')$. (b) The frontier is the set of samples that can reach $R(S')$. The inclusion of the frontier increases the interpolation region beyond $R(S')$.

stage, m, is reached for which A_m is empty, and D_m includes all samples from which the goal can be reached (under the approximation assumptions made in this section).

For efficiency purposes, an approximation to Front may be used, provided that the true frontier is a proper subset of the approximate frontier. For example, the frontier might add all new samples within a specified radius of points in S'. In this case, the updated cost-to-go value for some $s \in A_i$ may remain infinite. If this occurs, it is of course not added to D_{i+1}. Furthermore, it is deleted from A_i in the computation of the next frontier (the frontier should only be computed for samples that have finite cost-to-go values).

The approach considered so far can be expected to reduce the amount of computations in each value iteration by eliminating the evaluation of (8.56) on unnecessary samples. The same cost-to-go values are obtained in each iteration because only samples for which the value cannot change are eliminated in each iteration. The resulting algorithm still does not, however, resemble Dijkstra's algorithm because value iterations are performed over all of A_i in each stage.

To make a version that behaves like Dijkstra's algorithm, a queue Q will be introduced. The algorithm removes the smallest element of Q in each iteration. The interpolation version first assigns $G^*(s) = 0$ for each $s \in G$. It also maintains a set D of dead samples, which is initialized to $D = G$. For each $s \in \text{Front}(G)$, the cost-to-go update (8.56) is computed. The priority Q is initialized to $\text{Front}(G)$, and elements are sorted by their current cost-to-go values (which may not be optimal). The algorithm iteratively removes the smallest element from Q (because its optimal cost-to-go is known to be the current value) and terminates when Q is empty. Each time the smallest element, $s_s \in Q$, is removed, it is inserted into D. Two procedures are then performed: 1) Some elements in the queue need to have their cost-to-go values recomputed using (8.56) because the value $G^*(s_s)$ is

known to be optimal, and their values may depend on it. These are the samples in Q that lie in $\text{Front}(D)$ (in which D just got extended to include s_s). 2) Any samples in $\text{B}(R(D))$ that are not in Q are inserted into Q after computing their cost-to-go values using (8.56). This enables the active set of samples to grow as the set of dead samples grows. Dijkstra's algorithm with interpolation does not compute values that are identical to those produced by value iterations because G^*_{k+1} is not explicitly stored when G^*_k is computed. Each computed value is *some* cost-to-go, but it is only known to be the optimal when the sample is placed into D. It can be shown, however, that the method converges because computed values are no higher than what would have been computed in a value iteration. This is also the basis of dynamic programming using Gauss-Seidel iterations [59].

A specialized, wavefront-propagation version of the algorithm can be made for the special case of finding solutions that reach the goal in the smallest number of stages. The algorithm is similar to the one shown in Figure 8.4. It starts with an initial wavefront $W_0 = G$ in which $G^*(s) = 0$ for each $s \in G$. In each iteration, the optimal cost-to-go value i is increased by one, and the wavefront, W_{i+1}, is computed from W_i as $W_{i+1} = \text{Front}(W_i)$. The algorithm terminates at the first iteration in which the wavefront is empty.

Further Reading

There is much less related literature for this chapter in comparison to previous chapters. As explained in Section 8.1, there are historical reasons why feedback is usually separated from motion planning. Navigation functions [284, 416] were one of the most influential ideas in bringing feedback into motion planning; therefore, navigation functions were a common theme throughout the chapter. For other works that use or develop navigation functions, see [111, 149, 388]. The ideas of *progress measures* [164], Lyapunov functions (covered in Section 15.1.1), and cost-to-go functions are all closely related. For Lyapunov-based design of feedback control laws, see [150]. In the context of motion planning, the Error Detection and Recovery (EDR) framework also contains feedback ideas [154].

In [167], the *topological complexity* of C-spaces is studied by characterizing the minimum number of regions needed to cover $\mathcal{C} \times \mathcal{C}$ by defining a continuous path function over each region. This indicates limits on navigation functions that can be constructed, assuming that both q_I and q_G are variables (throughout this chapter, q_G was instead fixed). Further work in this direction includes [168, 169].

To gain better intuitions about properties of vector fields, [34] is a helpful reference, filled with numerous insightful illustrations. A good introduction to smooth manifolds that is particularly suited for control-theory concepts is [73]. Basic intuitions for 2D and 3D curves and surfaces can be obtained from [389]. Other sources for smooth manifolds and differential geometry include [2, 62, 126, 151, 437, 446, 470]. For discussions of piecewise-smooth vector fields, see [21, 325, 426, 487].

Sections 8.4.2 and 8.4.3 were inspired by [127, 334] and [371], respectively. Many difficulties were avoided because discontinuous vector fields were allowed in these approaches. By requiring continuity or smoothness, the subject of Section 8.4.4 was ob-

tained. The material is based mainly on [416, 417]. Other work on navigation functions includes [137, 339, 340].

Section 8.5.1 was inspired mainly by [83, 351], and the approach based on neighborhood graphs is drawn from [480].

Value iteration with interpolation, the subject of Section 8.5.2, is sometimes forgotten in motion planning because computers were not powerful enough at the time it was developed [55, 56, 301, 302]. Presently, however, solutions can be computed for challenging problems with several dimensions (e.g., 3 or 4). Convergence of discretized value iteration to solving the optimal continuous problem was first established in [58], based on Lipschitz conditions on the state transition equation and cost functional. Analyses that take interpolation into account, and general discretization issues, appear in [88, 156, 205, 295, 296]. A multi-resolution variant of value iteration was proposed in [377]. The discussion of Dijkstra-like versions of value iteration was based on [310, 461]. The level-set method is also closely related [278, 280, 279, 434].

Exercises

1. Suppose that a very fast path planning algorithm runs on board of a mobile robot (for example, it may find an answer in a few milliseconds, which is reasonable using trapezoidal decomposition in $\mathbb{R}^2$). Explain how this method can be used to simulate having a feedback plan on the robot. Explain the issues and trade-offs between having a fast on-line algorithm that computes open-loop plans vs. a better off-line algorithm that computes a feedback plan.

2. Use Dijkstra's algorithm to construct navigation functions on a 2D grid with obstacles. Experiment with adding a penalty to the cost functional for getting too close to obstacles.

3. If there are alternative routes, the NF2 algorithm does not necessarily send the state along the route that has the largest maximum clearance. Fix the NF2 algorithm so that it addresses this problem.

4. Tangent space problems:

   (a) For the manifold of unit quaternions, find basis vectors for the tangent space in $\mathbb{R}^4$ at any point.

   (b) Find basis vectors for the tangent space in $\mathbb{R}^9$, assuming that matrices in $SO(3)$ are parameterized with quaternions, as shown in (4.20).

5. Extend the algorithm described in Section 8.4.3 to make it work for polygons that have holes. See Example 8.16 for a similar problem.

6. Give a complete algorithm that uses the vertical cell decomposition for a polygonal obstacle region in $\mathbb{R}^2$ to construct a vector field that serves as a feedback plan. The vector field may be discontinuous.

7. Figure 8.23 depicts a 2D example for which X_{free} is an open annulus. Consider designing a vector field for which all integral curves flow into X_G and the vector field is continuous outside of X_G. Either give a vector field that achieves this or explain why it is not possible.

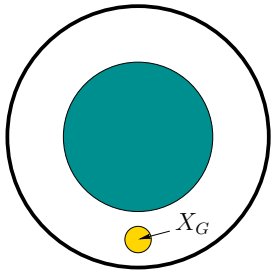


Figure 8.23: Consider designing a continuous vector field that flows into X_G.

8. Use the maximum-clearance roadmap idea from Section 6.2.3 to define a cell decomposition and feedback motion plan (vector field) that maximizes clearance. The vector field may be discontinuous.

9. Develop an algorithm that computes an exact cover for a polygonal configuration space and ensures that if two neighborhoods intersect, then their intersection always contains an open set (i.e., the overlap region is two-dimensional). The neighborhoods in the cover should be polygonal.

10. Using a distance measurement and Euler angles, determine the expression for a collision-free ball that can be inferred (make the ball as large as possible). This should generalize (8.54).

11. Using a distance measurement and quaternions, determine the expression for a collision-free ball (once again, make it as large as possible).

12. Generalize the multi-linear interpolation scheme in (8.59) from 2 to n dimensions.

13. Explain the convergence problems for value iteration that can result if $\|u\| = 1$ is used to constraint the set of allowable actions, instead of $\|u\| \leq 1$.

Implementations

14. Experiment with numerical methods for solving the function (8.49) in two dimensions under various boundary conditions. Report on the efficiency and accuracy of the methods. How well can they be applied in higher dimensions?

15. Implement value iteration with interpolation (it is not necessary to use the method in Figure 8.20) for a polygonal robot that translates and rotates among polygonal obstacles in $\mathcal{W} = \mathbb{R}^2$. Define the cost functional so that the distance traveled is obtained with respect to a weighted Euclidean metric (the weights that compare rotation to translation can be set arbitrarily).

16. Evaluate the efficiency of the interpolation method shown in Figure 8.20 applied to multi-linear interpolation given by generalizing (8.59) as in Exercise 12. You do not need to implement the full value iteration approach (alternatively, this could be done, which provides a better comparison of the overall performance).

17. Implement the method of Section 8.4.2 of computing vector fields on a triangulation. For given input polygons, have your program draw a needle diagram of the computed vector field. Determine how fast the vector field can be recomputed as the goal changes.

18. Optimal navigation function problems:

    (a) Implement the algorithm illustrated in Figure 8.13. Show the level sets of the optimal cost-to-go function.

    (b) Extend the algorithm and implementation to the case in which there are polygonal holes in X_{free}.

19. Adapt value iteration with interpolation so that a point robot moving in the plane can keep track of a predictable moving point called a *target*. The cost functional should cause a small penalty to be added if the target is not visible. Optimizing this should minimize the amount of time that the target is not visible. Assume that the initial configuration of the robot is given. Compute optimal feedback plans for the robot.

20. Try to experimentally construct navigation functions by adding potential functions that repel the state away from obstacles and attract the state toward x_G. For simplicity, you may assume that $X = \mathbb{R}^2$ and the obstacles are discs. Start with a single disc and then gradually construct more complicated obstacle regions. How difficult is it to ensure that the resulting potential function has no local minima outside of x_G?

Bibliography

[1] D. Aarno, D. Kragic, and H. I. Christensen. Artificial potential biased probabilistic roadmap method. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[2] R. Abraham, J. Marsden, and T. Ratiu. *Manifolds, Tensor Analysis, and Applications, 2nd Ed.* Springer-Verlag, Berlin, 1988.

[3] A. Abrams and R. Ghrist. Finding topology in a factory: Configuration spaces. *The American Mathematics Monthly*, 109:140–150, February 2002.

[4] E. U. Acar and H. Choset. Complete sensor-based coverage with extended-range detectors: A hierarchical decomposition in terms of critical points and Voronoi diagrams. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[5] E. U. Acar and H. Choset. Robust sensor-based coverage of unstructured environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[6] C. C. Adams. *The Knot Book: An Elementary Introduction to the Mathematical Theory of Knots.* W. H. Freeman, New York, 1994.

[7] P. Agarwal, M. de Berg, D. Halperin, and M. Sharir. Efficient generation of k-directional assembly sequences. In *ACM Symposium on Discrete Algorithms*, pages 122–131, 1996.

[8] P. K. Agarwal, N. Amenta, B. Aronov, and M. Sharir. Largest placements and motion planning of a convex polygon. In J.-P. Laumond and M. Overmars, editors, *Robotics: The Algorithmic Perspective.* A.K. Peters, Wellesley, MA, 1996.

[9] P. K. Agarwal, B. Aronov, and M. Sharir. Motion planning for a convex polygon in a polygonal environment. *Discrete and Computational Geometry*, 22:201–221, 1999.

[10] S. Akella and S. Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 624–631, 2002.

[11] R. Alami, J.-P. Laumond, and T. Siméon. Two manipulation planning algorithms. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation.* A.K. Peters, Wellesley, MA, 1997.

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

[13] G. Allgower and K. Georg. *Numerical Continuation Methods.* Springer-Verlag, Berlin, 1990.

[14] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. In *Proceedings ACM Symposium on Computational Geometry*, pages 281–289, 1990.

[15] 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 *Proceedings IEEE International Conference on Robotics & Automation*, pages 630–637, 1998.

[16] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[17] 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. *IEEE Transactions on Robotics & Automation*, 16(4):442–447, Aug 2000.

[18] N. M. Amato, K. A. Dill, and G. Song. Using motion planning to map protein folding landscapes and analyze folding kinetics of known native structures. In *Proceedings 6th ACM International Conference on Computational Molecular Biology (RECOMB)*, pages 2–11, 2002.

[19] N. M. Amato and G. Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, 2002.

[20] N. M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 113–120, 1996.

[21] F. Ancona and A. Bressan. Patchy vector fields and asymptotic stabilization. *ESAIM-Control, Optimisation and Calculus of Variations*, 4:445–471, 1999.

[22] B. D. Anderson and J. B. Moore. *Optimal Control: Linear-Quadratic Methods.* Prentice-Hall, Englewood Cliffs, NJ, 1990.

[23] J. Angeles. *Spatial Kinematic Chains. Analysis, Synthesis, and Optimisation*. Springer-Verlag, Berlin, 1982.

[24] J. Angeles. *Fundamentals of Robotic Mechanical Systems: Theory, Methods, and Algorithms*. Springer-Verlag, Berlin, 2003.

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

[26] 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.

[27] M. D. Ardema and J. M. Skowronski. Dynamic game applied to coordination control of two arm robotic system. In R. P. Hämäläinen and H. K. Ehtamo, editors, *Differential Games – Developments in Modelling and Computation*, pages 118–130. Springer-Verlag, Berlin, 1991.

[28] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering traveling salesman problem. *Discrete Applied Mathematics*, 55:194–218, 1994.

[29] B. Armstrong, O. Khatib, and J. Burdick. The explicit dynamic model and inertial parameters of the Puma 560 arm. In *Proceedings IEEE International Conference on Systems, Man, & Cyberetics*, pages 510–518, 1986.

[30] M. A. Armstrong. *Basic Topology*. Springer-Verlag, New York, 1983.

[31] D. S. Arnon. Geometric reasoning with logic and algebra. *Artificial Intelligence Journal*, 37(1-3):37–60, 1988.

[32] B. Aronov, M. de Berg, A. F. van der Stappen, P. Svestka, and J. Vleugels. Motion planning for multiple robots. *Discrete and Computational Geometry*, 22:505–525, 1999.

[33] B. Aronov and M. Sharir. On translational motion planning of a convex polyhedron in 3-space. *SIAM Journal on Computing*, 26(6):1875–1803, December 1997.

[34] D. K. Arrowsmith and C. M. Place. *Dynamical Systems: Differential Equations, Maps, and Chaotic Behaviour*. Chapman & Hall/CRC, New York, 1992.

[35] S. Arya and D. M. Mount. Algorithms for fast vector quantization. In *IEEE Data Compression Conference*, pages 381–390, March 1993.

[36] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings ACM-SIAM Symposium on Discrete Algorithms*, pages 271–280, 1993.

[37] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

[38] A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 632–637, 2002.

[39] J.-P. Aubin and A. Cellina. *Differential Inclusions*. Springer-Verlag, Berlin, 1984.

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

[41] F. Avnaim, J.-D. Boissonnat, and B. Faverjon. A practical exact planning algorithm for polygonal objects amidst polygonal obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1656–1660, 1988.

[42] J. Bañon. Implementation and extension of the ladder algorithm. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1548–1553, 1990.

[43] B. Baginski. The Z^3 method for fast path planning in dynamic environments. In *Proceedings IASTED Conference on Applications of Control and Robotics*, pages 47–52, 1996.

[44] B. Baginski. *Motion Planning for Manipulators with Many Degrees of Freedom – The BB-Method*. PhD thesis, Technical University of Munich, 1998.

[45] A. Baker. *Matrix Groups*. Springer-Verlag, Berlin, 2002.

[46] D. J. Balkcom and M. T. Mason. Introducing robotic origami folding. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[47] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for robot path planning. In G. Giralt and G. Hirzinger, editors, *Proceedings International Symposium on Robotics Research*, pages 249–264. Springer-Verlag, New York, 1996.

[48] J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1712–1717, 1990.

[49] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2328–2335, 1991.

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

[51] J. Basch, L. J. Guibas, D. Hsu, and A. T. Nguyen. Disconnection proofs for motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1765–1772, 2001.

[52] S. Basu, R. Pollack, and M. F. Roy. Computing roadmaps of semi-algebraic sets on a variety. *Journal of the American Society of Mathematics*, 3(1):55–82, 1999.

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

[54] K. E. Bekris, B. Y. Chen, A. Ladd, E. Plaku, and L. E. Kavraki. Multiple query probabilistic roadmap planning using single query primitives. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[55] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[56] R. E. Bellman and S. E. Dreyfus. *Applied Dynamic Programming*. Princeton University Press, Princeton, NJ, 1962.

[57] M. Bern. Triangulations and mesh generation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 563–582. Chapman and Hall/CRC Press, New York, 2004.

[58] D. P. Bertsekas. Convergence in discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, June 1975.

[59] D. P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II, 2nd Ed.* Athena Scientific, Belmont, MA, 2001.

[60] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[61] A. Beygelzimer, S. M. Kakade, and J. Langford. Cover trees for nearest neighbor. University of Pennsylvania, Available from http://www.cis.upenn.edu/∼skakade/papers/ml/cover_tree.pdf, 2005.

[62] R. L. Bishop and S. I. Goldberg. *Tensor Analysis on Manifolds*. Dover, New York, 1980.

[63] H. S. Black. Stabilized feedback amplifiers. *Bell Systems Technical Journal*, 13:1–18, 1934.

[64] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. In *Proceedings IEEE Symposium on Foundations of Computer Science*, 2003.

[65] L. Blum, F. Cucker, and M. Schub abd S. Smale. *Complexity and Real Computation*. Springer-Verlag, Berlin, 1998.

[66] H. Bode. Feedback: The history of an idea. In R. Bellman and R. Kalaba, editors, *Selected Papers on Mathematical Trends in Control Theory*, pages 106–123. Dover, New York, 1969.

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

[68] R. Bohlin. *Robot Path Planning*. PhD thesis, Chalmers University, Gothenburg, Sweden, 2002.

[69] R. Bohlin and L. Kavraki. Path planning using Lazy PRM. In *Proceedings IEEE International Conference on Robotics & Automation*, 2000.

[70] K.-F. Böhringer, B. R. Donald, and N. C. MacDonald. Upper and lower bounds for programmable vector fields with applications to MEMS and vibratory plate parts feeders. In J.-P. Laumond and M. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*. A.K. Peters, Wellesley, MA, 1997.

[71] J.-D. Boissonnat and M. Yvinec. *Algorithmic Geometry*. Cambridge University Press, Cambridge, U.K., 1998.

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

[73] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry. Revised 2nd Ed.* Academic, New York, 2003.

[74] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[75] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. RRTs for nonlinear, discrete, and hybrid planning and control. In *Proceedings IEEE Conference Decision & Control*, 2003.

[76] M. Bridson and A. Haefliger. *Metric Spaces of Non-Positive Curvature*. Springer-Verlag, Berlin, 1999.

[77] R. A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, & Cybernetics*, SMC-15(2):224–233, 1985.

[78] J. Bruce and M. Veloso. Real-time randomized path planning for robot navigation. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

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

[80] S. J. Buckley. Fast motion planning for multiple moving robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 322–326, 1989.

[81] J. W. Burdick. *Kinematic Analysis and Design of Redundant Manipulators*. PhD thesis, Stanford University, Stanford, CA, 1988.

[82] B. Burns and O. Brock. Sampling-based motion planning using predictive models. In *Proceedings IEEE International Conference on Robotics & Automation*, 2005.

[83] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, 1999.

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

[85] Z. J. Butler and D. Rus. Distributed motion planning for modular robots with unit-compressible modules. *International Journal of Robotics Research*, 22(9):699–716, 2003.

[86] S. Cambon, F. Gravot, and R. Alami. A robot task planner and merges symbolic and geometric reasoning. In *Proceedings European Conference on Artificial Intelligence*, 2004.

[87] S. Cameron. A comparison of two fast algorithms for computing the distance between convex polyhedra. *IEEE Transactions on Robotics & Automation*, 13(6):915–920, December 1997.

[88] F. Camilli and M. Falcone. Approximation of optimal control problems with state constraints: Estimates and applications. In B. S. Mordukhovich and H. J. Sussmann, editors, *Nonsmooth Analysis and Geometric Methods in Deterministic Optimal Control*, pages 23–57. Springer-Verlag, Berlin, 1996. Mathematics and its Applications, Vol. 78.

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

[90] J. Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993.

[91] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 49–60, 1987.

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

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

[94] S. Carpin and E. Pagello. On parallel RRTs for multi-robot systems. In *Proceedings 8th Conference of the Italian Association for Artificial Intelligence*, pages 834–841, 2002.

[95] S. Carpin and G. Pillonetto. Merging the adaptive random walks planner with the randomized potential field planner. In *Proceedings IEEE International Workshop on Robot Motion and Control*, pages 151–156, 2005.

[96] S. Carpin and G. Pillonetto. Robot motion planning using adaptive random walks. *IEEE Transactions on Robotics & Automation*, 21(1):129–136, 2005.

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

[98] S. Caselli and M. Reggiani. ERPP: An experience-based randomized path planner. In *Proceedings IEEE International Conference on Robotics & Automation*, 2000.

[99] D. Challou, D. Boley, M. Gini, and V. Kumar. A parallel formulation of informed randomized search for robot motion planning problems. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 709–714, 1995.

[100] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1012–1019, 1995.

[101] S. Charentus. *Modeling and Control of a Robot Manipulator Composed of Several Stewart Platforms*. PhD thesis, Université Paul Sabatier, Toulouse, France, 1990. In French.

[102] S. Chawla. *Graph Algorithms for Planning and Partitioning*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, June 2005.

[103] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[104] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6(5):485–524, 1991.

[105] B. Chazelle. *The Discrepancy Method*. Cambridge University Press, Cambridge, U.K., 2000.

[106] C.-T. Chen. *Linear System Theory and Design*. Holt, Rinehart, and Winston, New York, 1984.

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

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

[109] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 267–272, 2002.

[110] L. P. Chew and K. Kedem. A convex polygon among polygonal obstacles: Placement and high-clearance motion. *Computational Geometry: Theory and Applications*, 3:59–89, 1993.

[111] D. Chibisov, E. W. Mayr, and S. Pankratov. Spatial planning and geometric optimization: Combining configuration space and energy methods. In H. Hong and D. Wang, editors, *Automated Deduction in Geometry - ADG 2004*. Springer-Verlag, Berlin, 2006.

[112] G. Chirikjian, A. Pamecha, and I. Ebert-Uphoff. Evaluating efficiency of self-reconfiguration in a class of modular robots. *Journal of Robotic Systems*, 13(5):717–338, 1996.

[113] G. S. Chirikjian and A. B. Kyatkin. *Engineering Applications of Noncommutative Harmonic Analysis*. CRC Press, Boca Raton, FL, 2001.

[114] H. Chitsaz, J. M. O'Kane, and S. M. LaValle. Pareto-optimal coordination of two translating polygonal robots on an acyclic roadmap. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[115] J. Choi, J. Sellen, and C. K. Yap. Precision-sensitive Euclidean shortest path in 3-space. In *Proceedings ACM Symposium on Computational Geometry*, pages 350–359, 1995.

[116] H. Choset. Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots*, 9:247–253, 2000.

[117] H. Choset. Coverage for robotics – A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31:113–126, 2001.

[118] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA, 2005.

[119] H. Choset and P. Pignon. Cover path planning: The boustrophedron decomposition. In *Proceedings International Conference on Field and Service Robotics*, Canberra, Australia, December 1997.

[120] P. Choudhury and K. Lynch. Trajectory planning for second-order underactuated mechanical systems in presence of obstacles. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2002.

[121] C. M. Clark, S. M. Rock, and J.-C. Latombe. Motion planning for multiple mobile robots using dynamic networks. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[122] D. E. Clark, G. Jones, P. Willett P. W. Kenny, and R. C. Glen. Pharmacophoric pattern matching in files of three-dimensional chemical structures: Comparison of conformational searching algorithms for flexible searching. *Journal Chemical Information and Computational Sciences*, 34:197–206, 1994.

[123] K. L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for sb(s). Bell Labs. Available from http://cm.bell-labs.com/who/clarkson/Msb/readme.html, 2003.

[124] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Proceedings Second GI Conference on Automata Theory and Formal Languages*, pages 134–183, Berlin, 1975. Springer-Verlag. Lecture Notes in Computer Science, 33.

[125] G. E. Collins. Quantifier elimination by cylindrical algebraic decomposition–twenty years of progress. In B. F. Caviness and J. R. Johnson, editors,

Quantifier Elimination and Cylindrical Algebraic Decomposition, pages 8–23. Springer-Verlag, Berlin, 1998.

[126] L. Conlon. *Differentiable Manifolds, 2nd Ed.* Birkhäuser, Boston, MA, 2001.

[127] D. C. Conner, A. A. Rizzi, and H. Choset. Composition of local potential functions for global robot control and navigation. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3546–3551, 2003.

[128] C. Connolly and R. Grupen. The application of harmonic potential functions to robotics. *Journal of Robotic Systems*, 10(7):931–946, 1993.

[129] C. I. Connolly. Applications of harmonic functions to robotics. In *IEEE Symposium on Intelligent Control*, pages 498–502, 1992.

[130] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using Laplace's equation. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 2102–2106, May 1990.

[131] J. H. Conway and N. J. A. Sloane. *Sphere Packings, Lattices, and Groups.* Springer-Verlag, Berlin, 1999.

[132] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (2nd Ed.).* MIT Press, Cambridge, MA, 2001.

[133] J. Cortés. *Motion Planning Algorithms for General Closed-Chain Mechanisms.* PhD thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 2003.

[134] J. Cortés, T. Siméon M. Remaud-Siméon, and V. Tran. Geometric algorithms for the conformational analysis of long protein loops. *Journal of Computational Chemistry*, 25:956–967, 2004.

[135] J. Cortés, T. Siméon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains using PRM methods. In *Proceedings IEEE International Conference on Robotics & Automation*, 2002.

[136] M. G. Coutinho. *Dynamic Simulations of Multibody Systems.* Springer-Verlag, Berlin, 2001.

[137] N. Cowan. Composing navigation functions on Cartesian products of manifolds with boundary. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[138] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms.* Springer-Verlag, Berlin, 1992.

[139] H. S. M. Coxeter. *Regular Polytopes.* Dover, New York, 1973.

[140] J. J. Craig. *Introduction to Robotics.* Addison-Wesley, Reading, MA, 1989.

[141] J. C. Culberson. Sokoban is PSPACE-complete. In *Proceedings International Conference on Fun with Algorithms (FUN98)*, pages 65–76, Waterloo, Ontario, Canada, June 1998. Carleton Scientific.

[142] M. R. Cutkosky. *Robotic Grasping and Fine Manipulation.* Kluwer, Boston, MA, 1985.

[143] L. K. Dale and N. M. Amato. Probabilistic roadmap methods are embarrassingly parallel. In *Proceedings IEEE International Conference on Robotics & Automation*, 1999.

[144] J. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5:29–35, 1988.

[145] S. Davies. Multidimensional triangulation and interpolation for reinforcement learning. In *Proceedings Neural Information Processing Systems*, 1996.

[146] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.

[147] M. J. de Smith. *Distance and Path: The Development, Interpretation and Application of Distance Measurement in Mapping and Modelling.* PhD thesis, University College, University of London, London, 2003.

[148] R. Dial. Algorithm 360: Shortest path forest with topological ordering. *Communications of the ACM*, 12:632–633, 1969.

[149] D. V. Dimarogonas, M. M. Zavlanos, S. G. Loizou, and K. J. Kyriakopoulos. Decentralized motion control of multiple holonomic agents under input constraints. In *Proceedings IEEE Conference Decision & Control*, 2003.

[150] W. E. Dixon, A. Behal, D. M. Dawson, and S. Nagarkatti. *Nonlinear Control of Engineering Systems: A Lyapunov-Based Approach.* Birkhäuser, Boston, MA, 2003.

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

[152] B. R. Donald. Motion planning with six degrees of freedom. Technical Report AI-TR-791, Artificial Intelligence Lab., Massachusetts Institute of Technology, Cambridge, MA, 1984.

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

[154] B. R. Donald. A geometric approach to error detection and recovery for robot motion planning with uncertainty. *Artificial Intelligence Journal*, 37:223–271, 1988.

[155] S. K. Donaldson. Self-dual connections and the topology of smooth 4-manifold. *Bulletin of the American Mathematical Society*, 8:81–83, 1983.

[156] A. L. Dontchev. Discrete approximations in optimal control. In B. S. Mordukhovich and H. J. Sussmann, editors, *Nonsmooth Analysis and Geometric Methods in Deterministic Optimal Control*, pages 59–80. Springer-Verlag, Berlin, 1996. Mathematics and Its Applications, Vol. 78.

[157] G. E. Dullerud and R. D'Andrea. Distributed control of heterogeneous systems. *IEEE Transactions on Automatic Control*, 49(12):2113–2128, 2004.

[158] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.

[159] C. Edwards and S. K. Spurgeon. *Sliding Mode Control: Theory and Applications*. CRC Press, Ann Arbor, MI, 1998.

[160] M. Egerstedt and X. Hu. Formation constrained multi-agent control. *IEEE Transactions on Robotics & Automation*, 17(6):947–951, December 2001.

[161] S. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proceedings Eurographics*, 2001.

[162] I. Z. Emiris and B. Mourrain. Computer algebra methods for studying and computing molecular conformations. Technical report, INRIA, Sophia-Antipolis, France, 1997.

[163] A. G. Erdman, G. N. Sandor, and S. Kota. *Mechanism Design: Analysis and Synthesis, 4th Ed., Vol. 1*. Prentice Hall, Englewood Cliffs, NJ, 2001.

[164] M. A. Erdmann. Understanding action and sensing by designing action-based sensors. *International Journal of Robotics Research*, 14(5):483–509, 1995.

[165] M. A. Erdmann and T. Lozano-Pérez. On multiple moving objects. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1419–1424, 1986.

[166] M. A. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[167] M. Farber. Topological complexity of motion planning. *Discrete and Computational Geometry*, 29:211–221, 2003.

[168] M. Farber, S. Tabachnkov, and S. Yuzvinsky. Topological robotics: Motion planning in projective spaces. *International Mathematical Research Notices*, 34:1853–1870, 2003.

[169] M. Farber and S. Yuzvinsky. Topological robotics: Subspace arrangements and collision free motion planning. Technical Report math.AT/0210115, arXiv (on-line), 2004.

[170] B. Faverjon. Obstacle avoidance using an octree in the configuration space of a manipulator. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 504–512, 1984.

[171] B. Faverjon. Hierarchical object models for efficient anti-collision algorithms. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 333–340, 1989.

[172] B. Faverjon and P. Tournassoud. A local based method for path planning of manipulators with a high number of degrees of freedom. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1152–1159, 1987.

[173] R. Fierro, A. Das, V. Kumar, and J. P. Ostrowski. Hybrid control of formations of robots. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 157–162, 2001.

[174] A. F. Filippov. Differential equations with discontinuous right-hand sides. *American Mathematical Society Translations, Ser. 2*, 64:199–231, 1964.

[175] P. W. Finn, D. Halperin, L. E. Kavraki, J.-C. Latombe, R. Motwani, C. Shelton, and S. Venkatasubramanian. Geometric manipulation of flexible ligands. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry*, pages 67–78. Springer-Verlag, Berlin, 1996. Lecture Notes in Computer Science, 1148.

[176] G. F. Fishman. *Monte Carlo: Concepts, Algorithms, and Applications*. Springer-Verlag, Berlin, 1996.

[177] H. Flordal, M. Fabian, and K. Akesson. Automatic implementation and verification of coordinating PLC-code for robotcells. In *Proceedings IFAC Symposium of Information Control Problems in Manufacturing*, 2004.

[178] G. B. Folland. *Real Analysis: Modern Techniques and Their Applications*. Wiley, New York, 1984.

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

[180] E. Frazzoli and F. Bullo. Decentralized algorithms for vehicle routing in a stochastic time-varying environment. In *Proceedings IEEE Conference Decision & Control*, pages 3357–3363, 2004.

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

[182] E. Freund and H. Hoyer. Path finding in multi robot systems including obstacle avoidance. *International Journal of Robotics Research*, 7(1):42–70, February 1988.

[183] J. H. Friedman, J. L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[184] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. *Robotics: Control, Sensing, Vision, and Intelligence*. McGraw-Hill, New York, 1987.

[185] K. Fujimura and H. Samet. A hierarchical strategy for path planning among moving obstacles. Technical Report CAR-TR-237, Center for Automation Research, University of Maryland, November 1986.

[186] K. Fujimura and H. Samet. Planning a time-minimal motion among moving obstacles. *Algorithmica*, 10:41–63, 1993.

[187] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. Technical report, Dept. of Mechanical Engineering, Technion, Israel Institute of Technology, December 1999.

[188] Y. Gabriely and E. Rimon. Spanning-tree based coverage of continuous areas by a mobile robot. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1927–1933, 2001.

[189] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry: Theory and Applications*, 24(3):197–224, April 2003.

[190] J. Gallier. *Curves and Surfaces in Geometric Modeling*. Morgan Kaufmann, San Francisco, CA, 2000.

[191] R. Geraerts and M. Overmars. Sampling techniques for probabilistic roadmap planners. In *Proceedings International Conference on Intelligent Autonomous Systems*, 2004.

[192] R. Geraerts and M. H. Overmars. A comparative study of probabilistic roadmap planners. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, December 2002.

[193] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufman, San Francisco, CA, 2004.

[194] A. K. Ghose, M. E. Logan, A. M. Treasurywala, H. Wang, R. C. Wahl, B. E. Tomczuk, M. R. Gowravaram, E. P. Jaeger, and J. J. Wendoloski. Determination of pharmacophoric geometry for collagenase inhibitors using a novel computational method and its verification using molecular dynamics, NMR, and X-ray crystallography. *Journal of the American Chemical Society*, 117:4671–4682, 1995.

[195] S. K. Ghosh and D. M. Mount. An output sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20:888–910, 1991.

[196] R. Ghrist. Shape complexes for metamorphic robot systems. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, December 2002.

[197] R. Ghrist, J. M. O'Kane, and S. M. LaValle. Computing Pareto Optimal Coordinations on Roadmaps. *The International Journal of Robotics Research*, 24(11):997–1010, 2005.

[198] E. G. Gilbert and D. W. Johnson. Distance functions and their application to robot path planning in the presence of obstacles. *IEEE Transactions on Robotics & Automation*, 1(1):21–30, March 1985.

[199] E. G. Gilbert, D. W. Johnson, and S. S. Keerth. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics & Automation*, RA-4(2):193–203, Apr 1988.

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

[201] B. Glavina. *Planning collision free motions for manipulators through a combination of goal oriented search and the creation of intermediate random subgoals*. PhD thesis, Technical University of Munich, 1991. In German.

[202] J. Go, T. Vu, and J. J. Kuffner. Autonomous behaviors for interactive vehicle animations. In *Proceedings SIGGRAPH Symposium on Computer Animation*, 2004.

[203] M. Goldwasser and R. Motwani. Intractability of assembly sequencing: Unit disks in the plane. In F. Dehne, A. Rau-Chaplin, J.-R. Sack, and R. Tamassia, editors, *WADS '97 Algorithms and Data Structures*, pages 307–320. Springer-Verlag, Berlin, 1997. Lecture Notes in Computer Science, 1272.

[204] G. H. Golub and C. F. Van Loan. *Matrix Computations (3rd ed)*. Johns Hopkins University Press, Baltimore, MD, 1996.

[205] R. Gonzalez and E. Rofman. On deterministic control problems: An approximation procedure for the optimal cost, parts I, II. *SIAM Journal on Control & Optimization*, 23:242–285, 1985.

[206] J. E. Goodman and J. O'Rourke (eds). *Handbook of Discrete and Computational Geometry, 2nd Ed.* Chapman and Hall/CRC Press, New York, 2004.

[207] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings ACM SIGGRAPH*, 1996.

[208] V. E. Gough and S. G. Whitehall. Universal tyre test machine. In *Proceedings 9th International Technical Congress F.I.S.I.T.A.*, May 1962.

[209] E. J. Griffith and S. Akella. Coordinating multiple droplets in planar array digital microfluidic systems. *International Journal of Robotics Research*, 24(11):933–949, 2005.

[210] R. Grossman, A. Nerode, A. Ravn, and H. Rischel (eds). *Hybrid Systems*. Springer-Verlag, Berlin, 1993.

[211] L. Guibas and R. Seidel. Computing convolution by reciprocal search. In *Proceedings ACM Symposium on Computational Geometry*, pages 90–99, 1986.

[212] L. J. Guibas, D. Hsu, and L. Zhang. H-Walk: Hierarchical distance computation for moving convex bodies. In *Proceedings ACM Symposium on Computational Geometry*, pages 265–273, 1999.

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

[214] K. Gupta and X. Zhu. Practical motion planning for many degrees of freedom: A novel approach within sequential framework. *Journal of Robotic Systems*, 2(12):105–118, 1995.

[215] P. R. Halmos. *Measure Theory*. Springer-Verlag, Berlin, 1974.

[216] D. Halperin. Arrangements. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 529–562. Chapman and Hall/CRC Press, New York, 2004.

[217] D. Halperin, J.-C. Latombe, and R. H. Wilson. A general framework for assembly planning: the motion space approach. In *Proceedings ACM Symposium on Computational Geometry*, pages 9–18, 1998.

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

[219] D. Halperin and R. Wilson. Assembly partitioning along simple paths: the case of multiple translations. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1585–1592, 1995.

[220] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[221] J. M. Hammersley. Monte-Carlo methods for solving multivariable problems. *Annals of the New York Academy of Science*, 86:844–874, 1960.

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

[223] R. S. Hartenberg and J. Denavit. A kinematic notation for lower pair mechanisms based on matrices. *Journal of Applied Mechanics*, 77:215–221, 1955.

[224] R. S. Hartenberg and J. Denavit. *Kinematic Synthesis of Linkages*. McGraw-Hill, New York, 1964.

[225] R. Hartshorne. *Algebraic Geometry*. Springer-Verlag, Berlin, 1977.

[226] A. Hatcher. *Algebraic Topology*. Cambridge University Press, Cambridge, U.K., 2002. Available at http://www.math.cornell.edu/∼hatcher/AT/ATpage.html.

[227] J. Hershberger and S. Suri. Efficient computation of Euclidean shortest paths in the plane. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 508–517, 1995.

[228] S. Hert, S. Tiwari, and V. Lumelsky. A terrain-covering algorithm for an AUV. *Autonomous Robots*, 3:91–119, 1996.

[229] F. J. Hickernell. Lattice rules: How well do they measure up? In P. Bickel, editor, *Random and Quasi-Random Point Sets*, pages 109–166. Springer-Verlag, Berlin, 1998.

[230] F. J. Hickernell, H. S. Hong, P. L'Ecuyer, and C. Lemieux. Extensible lattice sequences for quasi-monte carlo quadrature. *SIAM Journal on Scientific Computing*, 22:1117–1138, 2000.

[231] M. W. Hirsch. *Differential Topology*. Springer-Verlag, Berlin, 1994.

[232] J. G. Hocking and G. S. Young. *Topology*. Dover, New York, 1988.

[233] R. L. Hoffman. Automated assembly in a CSG domain. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 210–215, 1989.

[234] C. M. Hoffmann. *Geometric and Solid Modeling*. Morgan Kaufmann, San Francisco, CA, 1989.

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

[236] L. S. Homem de Mello and A. C. Sanderson. Representations of mechanical assembly sequences. *IEEE Transactions on Robotics & Automation*, 7(2):211–227, 1991.

[237] J. Hopcroft, D. Joseph, and S. Whitesides. Movement problems for 2-dimensional linkages. In J .T .Schwartz, M. Sharir, and J. Hopcroft, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 282–329. Ablex, Norwood, NJ, 1987.

[238] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the "warehouseman's problem". *International Journal of Robotics Research*, 3(4):76–88, 1984.

[239] J. E. Hopcroft, J. D. Ullman, and R. Motwani. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 2000.

[240] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom: Random reflections at C-space obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3318–3323, San Diego, CA, April 1994.

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

[242] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications*, 4:495–512, 1999.

[243] W. Huang. Optimal line-sweep-based decompositions for coverage algorithms. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 27–32, 2001.

[244] T. W. Hungerford. *Algebra*. Springer-Verlag, Berlin, 1984.

[245] Y. K. Hwang and N. Ahuja. Gross motion planning–A survey. *ACM Computing Surveys*, 24(3):219–291, September 1992.

[246] C. Icking, G. Rote, E. Welzl, and C.-K. Yap. Shortest paths for line segments. *Algorithmica*, 10:182–200, 1992.

[247] P. Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 877–892. Chapman and Hall/CRC Press, New York, 2004.

[248] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings ACM Symposium on Theory of Computing*, pages 604–613, 1998.

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

[250] H. Jacob, S. Feder, and J. Slotine. Real-time path planning using harmonic potential functions in dynamic environment. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 874–881, 1997.

[251] Y.-B. Jia. Computation on parametric curves with an application in grasping. *International Journal of Robotics Research*, 23(7-8):825–855, 2004.

[252] P. Jiménez, F. Thomas, and C. Torras. Collision detection algorithms for motion planning. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 1–53. Springer-Verlag, Berlin, 1998.

[253] D. Jordan and M. Steiner. Configuration spaces of mechanical linkages. *Discrete and Computational Geometry*, 22:297–315, 1999.

[254] D. A. Joseph and W. H. Plantiga. On the complexity of reachability and motion planning questions. In *Proceedings ACM Symposium on Computational Geometry*, pages 62–66, 1985.

[255] S. Kagami, J. Kuffner, K. Nishiwaki, and K. Okada M. Inaba. Humanoid arm motion planning using stereo vision and RRT search. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[256] D. W. Kahn. *Topology: An Introduction to the Point-Set and Algebraic Areas*. Dover, New York, 1995.

[257] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. *Eurographics*, 22(3), 2003.

[258] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[259] M. H. Kalos and P. A. Whitlock. *Monte Carlo Methods*. Wiley, New York, 1986.

[260] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *International Journal of Robotics Research*, 5(3):72–89, 1986.

[261] L. Kauffman. *Knots and Applications*. World Scientific, River Edge, NJ, 1995.

[262] L. E. Kavraki. Computation of configuration-space obstacles using the Fast Fourier Transform. *IEEE Transactions on Robotics & Automation*, 11(3):408–413, 1995.

[263] L. E. Kavraki. Geometry and the discovery of new ligands. In J.-P. Laumond and M. H. Overmars, editors, *Algorithms for Robotic Motion and Manipulation*, pages 435–445. A.K. Peters, Wellesley, MA, 1997.

[264] L. E. Kavraki and M. Kolountzakis. Partitioning a planar assembly into two connected parts is NP-complete. *Information Processing Letters*, 55(3):159–165, 1995.

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

[266] Y. Ke and J. O'Rourke. Lower bounds on moving a ladder in two and three dimensions. *Discrete and Computational Geometry*, 3:197–217, 1988.

[267] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete and Computational Geometry*, 1:59–71, 1986.

[268] J. M. Keil. Polygon decomposition. In J. R. Sack and J. Urrutia, editors, *Handbook on Computational Geometry*. Elsevier, New York, 2000.

[269] J. F. Kenney and E. S. Keeping. *Mathematics of Statistics, Part 2, 2nd ed.* Van Nostrand, Princeton, NJ, 1951.

[270] L. Kettner. Designing a data structure for polyhedral surfaces. In *Proceedings ACM Symposium on Computational Geometry*, pages 146–154, 1998.

[271] H. K. Khalil. *Nonlinear Systems*. Macmillan, New York, 2002.

[272] W. Khalil and J. F. Kleinfinger. A new geometric notation for open and closed-loop robots. In *Proceedings IEEE International Conference on Robotics & Automation*, volume 3, pages 1174–1179, 1986.

[273] O. Khatib. *Commande dynamique dans l'espace opérational des robots manipulateurs en présence d'obstacles*. PhD thesis, Ecole Nationale de la Statistique et de l'Administration Economique, France, 1980.

[274] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[275] J. Kim and J. P. Ostrowski. Motion planning of aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[276] J.-O. Kim and P. Khosla. Real-time obstacle avoidance using harmonic potential functions. Technical report, Carnegie Mellon University, Pittsburgh, PA, 1990.

[277] J. T. Kimbrell. *Kinematic Analysis and Synthesis*. McGraw-Hill, New York, 1991.

[278] R. Kimmel, N. Kiryati, and A. M. Bruckstein. Multivalued distance maps for motion planning on surfaces with moving obstacles. *IEEE Transactions on Robotics & Automation*, 14(3):427–435, June 1998.

[279] R. Kimmel and J. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences, USA*, 95(15):8431–8435, 1998.

[280] R. Kimmel and J. Sethian. Optimal algorithm for shape from shading and path planning. *Journal of Mathematical Imaging and Vision*, 14(3):234–244, 2001.

[281] C. L. Kinsey. *Topology of Surfaces*. Springer-Verlag, Berlin, 1993.

[282] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings ACM Symposium on Theory of Computing*, pages 599–608, May 1997.

[283] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Ed.* Addison-Wesley, Reading, MA, 1998.

[284] D. E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1–6, 1987.

[285] D. E. Koditschek. An approach to autonomous robot assembly. *Robotica*, 12:137–155, 1994.

[286] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe. Planning motions with intentions. *Proceedings ACM SIGGRAPH*, pages 395–408, 1994.

[287] A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis*. Dover, New York, 1975.

[288] V. Koltun. Pianos are not flat: Rigid motion planning in three dimensions. In *Proceedings ACM-SIAM Symposium on Discrete Algorithms*, 2005.

[289] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Transactions on Robotics & Automation*, 7(3):267–277, 1991.

[290] K. Kotay, D. Rus, M. Vora, and C. McGray. The self-reconfiguring robotic molecule: Design and control algorithms. In P. K. Agarwal, L. E. Kavraki, and M. T. Mason, editors, *Robotics: The Algorithmic Perspective*. A.K. Peters, Natick, MA, 1998.

[291] K. Kozlowski, P. Dutkiewicz, and W. Wróblewski. *Modeling and Control of Robots*. Wydawnictwo Naukowe PWN, Warsaw, Poland, 2003. In Polish.

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

[293] J. J. Kuffner and S. M. LaValle. An efficient approach to path planning using balanced bidirectional RRT search. Technical Report CMU-RI-TR-05-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 2005.

[294] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, Princeton, NJ, 2002.

[295] H. J. Kushner. Numerical methods for continuous control problems in continuous time. *SIAM Journal on Control & Optimization*, 28:999–1048, 1990.

[296] H. J. Kushner and P. G. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time*. Springer-Verlag, Berlin, 1992.

[297] A. Ladd and L. E. Kavraki. Motion planning for knot untangling. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Nice, France, December 2002.

[298] A. Ladd and L. E. Kavraki. Fast exploration for robots with dynamics. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

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

[300] F. Lamiraux and L. Kavraki. Path planning for elastic plates under manipulation constraints. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 151–156, 1999.

[301] R. E. Larson. A survey of dynamic programming computational procedures. *IEEE Transactions on Automatic Control*, 12(6):767–774, December 1967.

[302] R. E. Larson and J. L. Casti. *Principles of Dynamic Programming, Part II*. Dekker, New York, 1982.

[303] A. Lasota and M. C. Mackey. *Chaos, Fractals, and Noise: Stochastic Aspects of Dynamics, 2nd Ed.* Springer-Verlag, Berlin, 1995.

[304] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[305] J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[306] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[307] S. M. LaValle, M. S. Branicky, and S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, 23(7/8):673–692, July/August 2004.

[308] S. M. LaValle, P. Finn, L. Kavraki, and J.-C. Latombe. A randomized kinematics-based approach to pharmacophore-constrained conformational search and database screening. *J. Computational Chemistry*, 21(9):731–747, 2000.

[309] S. M. LaValle and S. A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *IEEE Trans. on Robotics and Automation*, 14(6):912–925, December 1998.

[310] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.

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

[312] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In *Proceedings Workshop on the Algorithmic Foundations of Robotics*, 2000.

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

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

[315] A. R. Leach and I. D. Kuntz. Conformational analysis of flexible ligands in macromolecular receptor sites. *Journal of Computational Chemistry*, 13(6):730–748, 1992.

[316] D. T. Lee and R. L. Drysdale. Generalization of Voronoi diagrams in the plane. *SIAM Journal on Computing*, 10:73–87, 1981.

[317] D. H. Lehmer. Mathematical methods in large-scale computing units. In *Proceedings 2nd Symposium on Large-Scale Digital Computing Machinery*, pages 141–146, Cambridge, MA, 1951. Harvard University Press.

[318] 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, August 1990.

[319] D. Leven and M. Sharir. An efficient and simple motion planning algorithm for a ladder moving in a 2-dimensional space amidst polygonal barriers. *Journal of Algorithms*, 8:192–215, 1987.

[320] D. Leven and M. Sharir. Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete and Computational Geometry*, 2:9–31, 1987.

[321] P. Leven and S. A. Hutchinson. Real-time path planning in changing environments. *IEEE Transactions on Robotics & Automation*, 21(12):999–1030, December 2002.

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

[323] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1997.

[324] T.-Y. Li and Y.-C. Shie. An incremental learning approach to motion planning with roadmap management. In *Proceedings IEEE International Conference on Robotics & Automation*, 2002.

[325] D. Liberzon. *Switching in Systems and Control*. Birkhäuser, Boston, MA, 2003.

[326] J.-M. Lien, S. L. Thomas, and N. M. Amato. A general framework for sampling on the medial axis of the free space. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[327] M. C. Lin and J. F. Canny. Efficient algorithms for incremental distance computation. In *Proceedings IEEE International Conference on Robotics & Automation*, 1991.

[328] M. C. Lin and D. Manocha. Collision and proximity queries. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 787–807. Chapman and Hall/CRC Press, New York, 2004.

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

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

[331] S. R. Lindemann and S. M. LaValle. Current issues in sampling-based motion planning. In P. Dario and R. Chatila, editors, *Proceedings International Symposium on Robotics Research*. Springer-Verlag, Berlin, 2004.

[332] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[333] S. R. Lindemann and S. M. LaValle. Steps toward derandomizing RRTs. In *IEEE Fourth International Workshop on Robot Motion and Control*, 2004.

[334] S. R. Lindemann and S. M. LaValle. Smoothly blending vector fields for global robot navigation. In *Proceedings IEEE Conference Decision & Control*, pages 3353–3559, 2005.

[335] S. R. Lindemann, A. Yershova, and S. M. LaValle. Incremental grid sampling strategies in robotics. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, pages 297–312, 2004.

[336] A. Lingas. The power of non-rectilinear holes. In *Proceedings 9th International Colloquium on Automata, Languange, and Programming*, pages 369–383. Springer-Verlag, 1982. Lecture Notes in Computer Science, 140.

[337] F. Lingelbach. Path planning using probabilistic cell decomposition. In *Proceedings IEEE International Conference on Robotics & Automation*, 2004.

[338] 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.

[339] S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple holonomic vehicles. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[340] S. G. Loizou and K. J. Kyriakopoulos. Closed loop navigation for multiple non-holonomic vehicles. In *Proceedings IEEE International Conference on Robotics & Automation*, 2003.

[341] I. Lotan, F. Schwarzer, D. Halperin, and J.-C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proceedings ACM Symposium on Computational Geometry*, pages 43–52, 2002.

[342] I. Lotan, H. van den Bedem, A. M. Deacon, and J.-C. Latombe. Computing protein structures from electron density maps: The missing loop problem. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2004.

[343] T. Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man, & Cybernetics*, 11(10):681–698, 1981.

[344] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, C-32(2):108–120, 1983.

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

[346] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

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

[348] L. Lu and S. Akella. Folding cartons with fixtures: A motion planning approach. *IEEE Transactions on Robotics & Automation*, 16(4):346–356, Aug 2000.

[349] V. J. Lumelsky and K. R. Harinarayan. Decentralized motion planning for multiple mobile robots: The cocktail party model. *Autonomous Robots*, 4(1):121–135, 1997.

[350] D. Manocha and J. Canny. Real time inverse kinematics of general 6R manipulators. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 383–389, Nice, May 1992.

[351] M. T. Mason. The mechanics of manipulation. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 544–548, 1985.

[352] M. T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, Cambridge, MA, 2001.

[353] J. Matousek. *Geometric Discrepancy*. Springer-Verlag, Berlin, 1999.

[354] J. Matousek and J. Nesetril. *Invitation to Discrete Mathematics*. Oxford University Press, Oxford, U.K., 1998.

[355] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, January 1998.

[356] O. Mayr. *The Origins of Feedback Control*. MIT Press, Cambridge, MA, 1970.

[357] E. Mazer, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, November 1998.

[358] E. Mazer, G. Talbi, J. M. Ahuactzin, and P. Bessière. The Ariadne's clew algorithm. In *Proceedings International Conference of Society of Adaptive Behavior*, Honolulu, 1992.

[359] J. M. McCarthy. *Geometric Design of Linkages*. Springer-Verlag, Berlin, 2000.

[360] J.-P. Merlet. *Parallel Robots*. Kluwer, Boston, MA, 2000.

[361] N. C. Metropolis and S. M. Ulam. The Monte-Carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[362] A. N. Michel and C. J. Herget. *Applied Algebra and Functional Analysis*. Dover, New York, 1993.

[363] R. J. Milgram and J. C. Trinkle. Complete path planning for closed kinematic chains with spherical joints. *International Journal of Robotics Research*, 21(9):773–789, 2002.

[364] R. J. Milgram and J. C. Trinkle. The geometry of configuration spaces for closed chains in two and three dimensions. *Homology, Homotopy, and Applications*, 6(1):237–267, 2004.

[365] G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, 44(1):1–29, January 1997.

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

[367] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electronics Research Laboratory, 1997.

[368] B. Mirtich. Efficient algorithms for two-phase collision detection. In K. Gupta and A.P. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 203–223. Wiley, New York, 1998.

[369] B. Mishra. *Algorithmic Algebra*. Springer-Verlag, New York, 1993.

[370] B. Mishra. Computational real algebraic geometry. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 537–556. CRC Press, New York, 1997.

[371] J. S. B. Mitchell. Shortest paths among obstacles in the plane. *International Journal Computational Geometry & Applications*, 6(3):309–332, 1996.

[372] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 607–641. Chapman and Hall/CRC Press, New York, 2004.

[373] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem. *Journal of the ACM*, 38:18–73, 1991.

[374] M. E. Mortenson. *Geometric Modeling, 2nd Ed.* Wiley, New York, 1997.

[375] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, U.K., 1995.

[376] R. Munos and A. Moore. Barycentric interpolator for continuous space & time reinforcement learning. In *Proceedings Neural Information Processing Systems*, 1998.

[377] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2001.

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

[379] B. K. Natarajan. On planning assemblies. In *Proceedings ACM Symposium on Computational Geometry*, pages 299–308, 1988.

[380] W. S. Newman and M. S. Branicky. Real-time configuration space transforms for obstacle avoidance. *International Journal of Robotics Research*, 10(6):650–667, 1991.

[381] H. Niederreiter. *Random Number Generation and Quasi-Monte-Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[382] H. Niederreiter and C. P. Xing. Nets, (t,s)-sequences, and algebraic geometry. In P. Hellekalek and G. Larcher, editors, *Random and Quasi-Random Point Sets*, pages 267–302. Springer-Verlag, Berlin, 1998. Lecture Notes in Statistics, 138.

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

[384] N. J. Nilsson. A mobile automaton: An application of artificial intelligence techniques. In *1st International Conference on Artificial Intelligence*, pages 509–520, 1969.

[385] P. A. O'Donnell and T. Lozano-Pérez. Deadlock-free and collision-free coordination of two robot manipulators. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 484–489, 1989.

[386] C. O'Dunlaing, M. Sharir, and C. K. Yap. Retraction: A new approach to motion planning. In J .T .Schwartz, M. Sharir, and J. Hopcroft, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 193–213. Ablex, Norwood, NJ, 1987.

[387] C. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of Algorithms*, 6:104–111, 1982.

[388] P. Ögren. *Formations and Obstacle Avoidance in Mobile Robot Control*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2003.

[389] B. O'Neill. *Elementary Differential Geometry*. Academic, New York, 1966.

[390] J. O'Rourke and S. Suri. Polygons. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 583–606. Chapman and Hall/CRC Press, New York, 2004.

[391] M. H. Overmars and J. van Leeuwen. Dynamic multidimensional data structures based on Quad- and K-D trees. *Acta Informatica*, 17:267–285, 1982.

[392] B. Paden, A. Mees, and M. Fisher. Path planning using a Jacobian-based freespace generation algorithm. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1732–1737, 1989.

[393] C. H. Papadimitriou. An algorithm for shortest-path planning in three dimensions. *Information Processing Letters*, 20(5):259–263, 1985.

[394] A. Papantonopoulou. *Algebra: Pure and Applied*. Prentice Hall, Englewood Cliffs, NJ, 2002.

[395] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, 1981.

[396] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V (WAFR 2002)*, pages 221–237. Springer-Verlag, Berlin, 2002.

[397] J. Pertin-Troccaz. Grasping: A state of the art. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *The Robotics Review 1*. MIT Press, Cambridge, MA, 1989.

[398] J. Pettré, J.-P. Laumond, and T. Siméon. A 2-stages locomotion planner for digital actors. In *Proceedings Eurographics/SIGGRAPH Symposium on Computer Animation*, pages 258–264, 2003.

[399] L. Piegl. On NURBS: A survey. *IEEE Transactions on Computer Graphics & Applications*, 11(1):55–71, Jan 1991.

[400] C. Pisula, K. Hoff, M. Lin, and D. Manocha. Randomized path planning for a rigid body based on hardware accelerated Voronoi sampling. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2000.

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

[402] J. Ponce and B. Faverjon. On computing three-finger force-closure grasps of polygonal objects. *IEEE Transactions on Robotics & Automation*, 11(6):868–881, 1995.

[403] J. Ponce, S. Sullivan, A. Sudsang, J.-D. Boissonnat, and J.-P. Merlet. On computing four-finger equilibrium and force-closure grasps of polyhedral objects. *International Journal of Robotics Research*, 16(1):11–35, February 1997.

[404] H. Pottman and J. Wallner. *Computational Line Geometry*. Springer-Verlag, Berlin, 2001.

[405] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, Berlin, 1985.

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

[407] M. Rabin. Transaction protection by beacons. *Journal of Computation Systems Science*, 27(2):256–267, 1983.

[408] S. Ratering and M. Gini. Robot navigation in a known environment with unknown moving obstacles. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 25–30, 1993.

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

[410] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 144–154, 1985.

[411] J. H. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *Journal of the ACM*, 41:764–790, 1994.

[412] J. H. Reif and Z. Sun. An efficient approximation algorithm for weighted region shortest path problem. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 191–203. A.K. Peters, Wellesley, MA, 2001.

[413] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1977.

[414] E. Rimon and J. W. Burdick. Mobility of bodies in contact–I: A 2nd order mobility index for multiple-finger grasps. *IEEE Transactions on Robotics & Automation*, 14(5):696–708, 1998.

[415] E. Rimon and J. W. Burdick. Mobility of bodies in contact–II: How forces are generated by curvature effects. *IEEE Transactions on Robotics & Automation*, 14(5):709–717, 1998.

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

[417] A. A. Rizzi. Hybrid control as a method for robot motion programming. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 832–837, 1998.

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

[419] J. J. Rotman. *Introduction to Algebraic Topology*. Springer-Verlag, Berlin, 1988.

[420] H. L. Royden. *Real Analysis*. MacMillan, New York, 1988.

[421] W. Rudin. *Real Analysis*. McGraw-Hill, New York, 1987.

[422] W. Rudin. *Functional Analysis, 2nd Ed.* McGraw-Hill, New York, 1991.

[423] H. Sagan. *Space-Filling Curves*. Springer-Verlag, Berlin, 1994.

[424] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Proceedings International Symposium on Robotics Research*, 2001.

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

[426] S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, Berlin, 1999.

[427] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[428] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: II. General techniques for computing topological properties of algebraic manifolds. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[429] J. T. Schwartz and M. Sharir. On the Piano Movers' Problem: III. Coordinating the motion of several independent bodies. *International Journal of Robotics Research*, 2(3):97–140, 1983.

[430] J. T. Schwartz, M. Sharir, and J. Hopcroft. *Planning, Geometry, and Complexity of Robot Motion*. Ablex, Norwood, NJ, 1987.

[431] 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 (WAFR 2002)*. Springer-Verlag, Berlin, 2002.

[432] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. Springer-Verlag, Berlin, 1996.

[433] N. F. Sepetov, V. Krchnak, M. Stankova, S. Wade, K. S. Lam, and M. Lebl. Library of libraries: Approach to synthetic combinatorial library design and screening of "pharmacophore" motifs. *Proceedings of the National Academy of Sciences, USA*, 92:5426–5430, June 1995.

[434] J. A. Sethian. *Level set methods : Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, Cambridge, U.K., 1996.

[435] M. Sharir. Algorithmic motion planning. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 1037–1064. Chapman and Hall/CRC Press, New York, 2004.

[436] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, Cambridge, U.K., 1995.

[437] R. W. Sharpe. *Differential Geometry*. Springer-Verlag, Berlin, 1997.

[438] C. L. Shih, T.-T. Lee, and W. A. Gruver. A unified approach for robot motion planning with moving polyhedral obstacles. *IEEE Transactions on Systems, Man, & Cybernetics*, 20:903–915, 1990.

[439] K. Shoemake. Uniform random rotations. In D. Kirk, editor, *Graphics Gems III*, pages 124–132. Academic, New York, 1992.

[440] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6), 2000.

[441] T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics & Automation*, 18(1), February 2002.

[442] M. Sipser. *Introduction to the Theory of Computation*. PWS, Boston, MA, 1997.

[443] I. H. Sloan and S. Joe. *Lattice Methods for Multiple Integration*. Oxford Science, Englewood Cliffs, NJ, 1994.

[444] G. Song and N. M. Amato. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 26(2):282–304, 2002.

[445] R. Spence and S. A. Hutchinson. Dealing with unexpected moving obstacles by integrating potential field planning with inverse dynamics control. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1485–1490, 1992.

[446] M. Spivak. *Differential Geometry*. Publish or Perish, Houston, TX, 1979.

[447] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. Wiley, New York, 2005.

[448] R. L Sproull. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*, 6:579–589, 1991.

[449] L. A. Steen and J. A. Seebach Jr. *Counterexamples in Topology*. Dover, New York, 1996.

[450] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3310–3317, 1994.

[451] D. Stewart. A platform with six degrees of freedom. In *Institution of Mechanical Engineers, Proceedings 1965-66, 180 Part 1*, pages 371–386, 1966.

[452] M. Stilman and J. J. Kuffner. Navigation among movable obstacles: Real-time reasoning in complex environments. In *Proceedings 2004 IEEE International Conference on Humanoid Robotics (Humanoids'04)*, 2004.

[453] M. Strandberg. Augmenting RRT-planners with local trees. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3258–3262, 2004.

[454] M. Strandberg. *Robot Path Planning: An Object-Oriented Approach*. PhD thesis, Royal Institute of Technology (KTH), Stockholm, Sweden, 2004.

[455] A. Sudsang, J. Ponce, and N. Srinivasa. Grasping and in-hand manipulation: Geometry and algorithms. *Algorithmica*, 26:466–493, 2000.

[456] A. G. Sukharev. Optimal strategies of the search for an extremum. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 11(4), 1971. Translated from Russian, *Zh. Vychisl. Mat. i Mat. Fiz.*, 11, 4, 910-924, 1971.

[457] K. Sutner and W. Maass. Motion planning among time dependent obstacles. *Acta Informatica*, 26:93–122, 1988.

[458] S. Tezuka. *Uniform Random Numbers: Theory and Practice*. Kluwer, Boston, MA, 1995.

[459] S. Tezuka. Quasi-Monte Carlo: The discrepancy between theory and practice. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 124–140. Springer-Verlag, Berlin, 2002.

[460] M. Thorup. Undirected single source shortest paths in linear time. In *Proceedings IEEE Symposium on Foundations of Computer Science*, pages 12–21, 1997.

[461] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[462] S. Udupa. *Collision Detection and Avoidance in Computer Controlled Manipulators*. PhD thesis, Dept. of Electrical Engineering, California Institute of Technology, 1977.

[463] University of North Carolina. PQP: A proximity query package. GAMMA Research Group, Available from http://www.cs.unc.edu/∼geom/SSV/, 2005.

[464] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[465] J. van den Berg and M. Overmars. Roadmap-based motion planing in dynamic environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1598–1605, 2004.

[466] J. van den Berg and M. Overmars. Prioritized motion planning for multiple robots. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2217–2222, 2005.

[467] J. G. van der Corput. Verteilungsfunktionen I. *Akademie van Wetenschappen*, 38:813–821, 1935.

[468] G. Vegter. Computational topology. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry, 2nd Ed.*, pages 719–742. Chapman and Hall/CRC Press, New York, 2004.

[469] X. Wang and F. J. Hickernell. An historical overview of lattice point sets. In K.-T. Fang, F. J. Hickernell, and H. Niederreiter, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 158–167. Springer-Verlag, Berlin, 2002.

[470] F. W. Warner. *Foundations of Differentiable Manifolds and Lie Groups*. Springer-Verlag, Berlin, 1983.

[471] D. S. Watkins. *Fundamentals of Matrix Computations, 2nd Ed.* Wiley, New York, 2002.

[472] H. Weyl. Über die Gleichverteilung von Zahlen mod Eins. *Mathematische Annalen*, 77:313–352, 1916.

[473] H. Whitney. Local properties of analytic varieties. In S. Cairns, editor, *Differential and Combinatorial Topology*, pages 205–244. Princeton University Press, Princeton, NJ, 1965.

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

[475] R. Wilson, L. Kavraki, J.-C. Latombe, and T. Lozano-Pérez. Two-handed assembly sequencing. *International Journal of Robotics Research*, 14(4):335–350, 1995.

[476] R. H. Wilson. *On Geometric Assembly Planning*. PhD thesis, Stanford University, Stanford, CA, March 1992.

[477] R. H. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence Journal*, 71(2):371–396, 1994.

[478] S. C. Wong, L. Middleton, and B. A. MacDonald. Performance metrics for robot coverage tasks. In *Proceedings Australasian Conference on Robotics and Automation*, 2002.

[479] J. Yakey, S. M. LaValle, and L. E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17(6):951–958, December 2001.

[480] L. Yang and S. M. LaValle. The sampling-based neighborhood graph: A framework for planning and executing feedback motion strategies. *IEEE Transactions on Robotics and Automation*, 20(3):419–432, June 2004.

[481] A. Yershova, L. Jaillet, T. Simeon, and S. M. LaValle. Dynamic-domain RRTs: Efficient exploration by controlling the sampling domain. In *Proceedings IEEE International Conference on Robotics and Automation*, 2005.

[482] A. Yershova and S. M. LaValle. Deterministic sampling methods for spheres and SO(3). In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[483] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[484] M. Yim. *Locomotion with a Unit-Modular Reconfigurable Robot*. PhD thesis, Stanford University, Stanford, CA, December 1994. Stanford Technical Report STAN-CS-94-1536.

[485] T. Yoshikawa. *Foundations of Robotics: Analysis and Control*. MIT Press, Cambridge, MA, 1990.

[486] Y. Yu and K. Gupta. On sensor-based roadmap: A framework for motion planning for a manipulator arm in unknown environments. In *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1919–1924, 1998.

[487] M. Zefran and J. Burdick. Stabilization of systems with changing dynamics by means of switching. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 1090–1095, 1998.

[488] M. Zhang, R. A. White, L. Wang, R. N. Goldman, L. E. Kavraki, and B. Hassett. Improving conformational searches by geometric screening. *Bioinformatics*, 21(5):624–630, 2005.