5. Motion Planning

Lydia E. Kavraki, Steven M. LaValle

This chapter first provides a formulation of the geometric path planning problem in Sect. 5.1 and then introduces sampling-based planning in Sect. 5.2. Sampling-based planners are general techniques applicable to a wide set of problems and have been successful in dealing with hard planning instances. For specific, often simpler, planning instances, alternative approaches exist and are presented in Sect. 5.3. These approaches provide theoretical guarantees and for simple planning instances they outperform sampling-based planners. Section 5.4 considers problems that involve differential constraints, while Sect. 5.5 overviews several other extensions of the basic problem formulation and proposed solutions. Finally, Sect. 5.7 addresses some important and more advanced topics related to motion planning.

A fundamental robotics task is to plan collision-free motions for complex bodies from a start to a goal position among a collection of static obstacles. Although relatively simple, this geometric path planning problem is computationally hard [5.1]. Extensions of this formulation take into account additional problems that are inherited from mechanical and sensor limitations of real robots such as uncertainties, feedback, and differential constraints, which further complicate the development of automated planners. Modern algorithms have been fairly successful in addressing hard instances of the basic geometric problem and a lot of effort has been devoted to extend their capabilities to more challenging instances. These algorithms have had widespread success in applications beyond robotics, such as computer animation, virtual prototyping, and computational biology. There are many available surveys [5.2–4] and books [5.5–7] that cover modern motion planning techniques and applications.

5.1 Motion Planning Concepts

This section provides a description of the fundamental motion planning problem or the geometric path planning problem. Extensions of this basic formulation to more complicated instances will be discussed later in the chapter and will be revisited throughout this book.

5.1.1 Configuration Space

In path planning, a complete description of the geometry of a robot $\mathcal{A}$ and of a workspace $\mathcal{W}$ is provided. The workspace $\mathcal{W} = \mathbb{R}^N$, in which $N = 2$ or $N = 3$, is a static environment populated with obstacles. The goal is to find a collision-free path for $\mathcal{A}$ to move from an initial position and orientation to a goal position and orientation.

To achieve this, a complete specification of the location of every point on the robot geometry, or a *configuration* $\boldsymbol{q}$, must be provided. The *configuration space*, or C-space ($\boldsymbol{q} \in \mathcal{C}$), is the space of all possible configurations. The C-space represents the set of all transformations that can be applied to a robot given its kinematics as described in Chap. 1 (Kinematics). It was recognized early on in motion planning research [5.8, 9] that the C-space is a useful way to abstract planning problems in a unified way. The advantage of this abstraction is that a robot with a complex geometric shape is mapped to a single point in the C-space. The number of degrees of freedom of a robot system is the dimension of the C-space, or the minimum number of parameters needed to specify a configuration.

Let the closed set $\mathcal{O} \subset \mathcal{W}$ represent the *(workspace) obstacle region*, which is usually expressed as a collection of polyhedra, three-dimensional (3-D) triangles, or piecewise-algebraic surfaces. Let the closed set $\mathcal{A}(\boldsymbol{q}) \subset \mathcal{W}$ denote the set of points occupied by the robot when at configuration $\boldsymbol{q} \in \mathcal{C}$; this set is usually modeled using the same primitives as used for $\mathcal{O}$. The *C-space obstacle region*, $\mathcal{C}_{\text{obs}}$, is defined as

$$\mathcal{C}_{\text{obs}} = \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O} \neq \emptyset\}. \tag{5.1}$$

Since $\mathcal{O}$ and $\mathcal{A}(\boldsymbol{q})$ are closed sets in $\mathcal{W}$, the obstacle region is a closed set in $\mathcal{C}$. The set of configurations that avoid collision is $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obs}}$, and is called the *free space*.

Simple Examples of C-spaces

Translating Planar Rigid Bodies. The robot's configuration can be specified by a reference point (x, y) on the planar rigid body relative to some fixed coordinate frame. Therefore the C-space is equivalent to $\mathbb{R}^2$. Figure 5.1 gives an example of a C-space for a triangular robot and a single polygonal obstacle. The obstacle region in the C-space can be traced by sliding the robot around the workspace obstacle to find the constraints on all $\boldsymbol{q} \in \mathcal{C}$. Motion planning for the robot is now equivalent to motion planning for a point in the C-space.

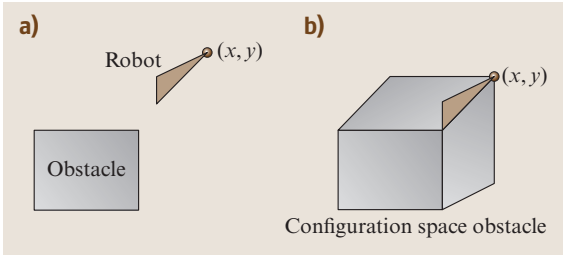

Fig. 5.1a,b A robot translating in the plane: **(a)** a triangular robot moves in a workspace with a single rectangular obstacle. **(b)** The C-space obstacle

Planar Arms. Figure 5.2 gives an example of a two-joint planar arm. The bases of both links are pinned, so that they can only rotate around the joints, and there are no joint limits. For this arm, specifying the rotational parameters θ_1 and θ_2 provides the configuration. Each joint angle θ_i corresponds to a point on the unit circle $\mathbb{S}^1$ and the C-space is $\mathbb{S}^1 \times \mathbb{S}^1 = T^2$, the two-dimensional torus shown in Fig. 5.2. For a higher number of links without joint limits, the C-space can be similarly defined as:

$$\mathcal{C} = \mathbb{S}^1 \times \mathbb{S}^1 \times \cdots \times \mathbb{S}^1 \,. \tag{5.2}$$

If a joint has limits, then each corresponding $\mathbb{S}^1$ is often replaced with $\mathbb{R}$, even though it is a finite interval. If the base of the planar arm is mobile and not pinned,

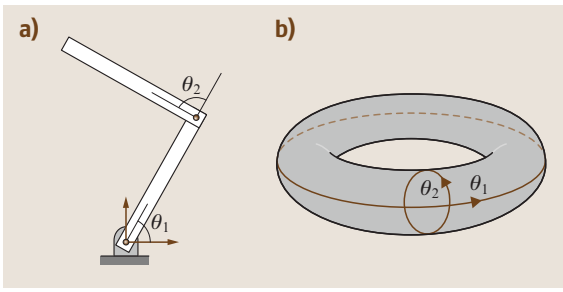

Fig. 5.2 **(a)** A two-joint planar arm in which the links are pinned and there are no joint limits. **(b)** The C-space

then the additional translation parameters must also be considered in the arm's configuration:

$$\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \cdots \times \mathbb{S}^1 \,. \tag{5.3}$$

Additional examples of C-spaces are provided in Sect. 5.6.1, where topological properties of configuration spaces are discussed.

5.1.2 Geometric Path Planning Problem

The basic motion planning problem, also known as the *piano mover's problem* [5.1], is defined as follows.
Given:

1. A *workspace* $\mathcal{W}$, where either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$.
2. An *obstacle region* $\mathcal{O} \subset \mathcal{W}$.
3. A *robot* defined in $\mathcal{W}$. Either a rigid body $\mathcal{A}$ or a collection of m links: $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_m$.
4. The *configuration space* $\mathcal{C}$ ($\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$ are then defined).
5. An *initial configuration* $\boldsymbol{q}_{\text{I}} \in \mathcal{C}_{\text{free}}$.
6. A *goal configuration* $\boldsymbol{q}_{\text{G}} \in \mathcal{C}_{\text{free}}$. The initial and goal configuration are often called a *query* $(\boldsymbol{q}_{\text{I}}, \boldsymbol{q}_{\text{G}})$.

Compute a (continuous) path, $\tau : [0, 1] \to \mathcal{C}_{\text{free}}$, *such that* $\tau(0) = \boldsymbol{q}_{\text{I}}$ *and* $\tau(1) = \boldsymbol{q}_{\text{G}}$.

5.1.3 Complexity of Motion Planning

The main complications in motion planning are that it is not easy to directly compute $\mathcal{C}_{\text{obs}}$ and $\mathcal{C}_{\text{free}}$, and the dimensionality of the C-space is often quite high. In terms of computational complexity, the piano mover's problem was studied early on and it was shown to be PSPACE-hard by *Reif* [5.1]. A series of polynomial-time algorithms for problems with fixed dimension suggested an exponential dependence on the problem dimensionality [5.10, 11]. A single exponential-time algorithm in the C-space dimensionality was proposed by *Canny* and showed that the problem is PSPACE-complete [5.12]. Although impractical, the algorithm serves as an upper bound on the general version of the basic motion planning problem. It applies computational algebraic geometry techniques for modeling the C-space in order to construct a *roadmap*, a one-dimensional (1-D) subspace that captures the connectivity of $\mathcal{C}_{\text{free}}$. Additional details about such techniques can be found in Sect. 5.6.3.

The complexity of the problem motivated work in path planning research. One direction was to study subclasses of the general problem for which polynomial time algorithms exist [5.13]. Even some simpler, special cases of motion planning, however, are at least as challenging, for example, the case of a finite number of translating, axis-aligned rectangles in $\mathbb{R}^2$ is PSPACE-hard as well [5.14]. Some extensions of motion planning are even harder. For example, a certain form of planning under uncertainty in 3-D polyhedral environment is NEXPTIME-hard [5.15]. The hardest problems in NEXPTIME are believed to require doubly exponential time to solve.

A different direction was the development of alternative planning paradigms that were practical under realistic assumptions. Many combinatorial approaches can efficiently construct 1-D roadmaps for specific two-dimensional (2-D) or 3-D problems. Potential field-based approaches define vector fields which can be followed by a robot towards the goal. Both approaches, however, do not scale well in the general case. They will be described in Sect. 5.3. An alternative paradigm, sampling-based planning, is a general approach that has been shown to be successful in practice for many challenging problems. It avoids the exact geometric modeling of the C-space but it cannot provide the guarantees of a complete algorithm. Complete and exact algorithms are able to detect that no path can be found. Instead sampling-based planning offers a lower level of completeness guarantee. This paradigm is described in the following section.

5.2 Sampling-Based Planning

Sampling-based planners are described first because they are the method of choice for a very general class of problems. The following section will describe other planners, some of which were developed before the sampling-based framework. The key idea in sampling-based planning is to exploit advances in collision detection algorithms that compute whether a single configuration is collision free. Given this simple primitive, a planner samples different configurations to construct a data structure that stores 1-D C-space curves, which represent collision-free paths. In this way, sampling-based planners do not access the C-space obstacles directly but only through the collision detector and the constructed data struc-

ture. Using this level of abstraction, the planners are applicable to a wide range of problems by tailoring the collision detector to specific robots and applications.

A standard for sampling-based planners is to provide a weaker, but still interesting, form of completeness: *if a solution path exists, the planner will eventually find it*. Giving up on the stronger form of completeness, which also requires failure to be reported in finite time, these techniques are able to solve in practice problems with more than three degrees of freedom that complete approaches cannot address. More details on this weaker form of completeness are provided in Sect. 5.6.2.

Different planners follow different approaches on how to sample configurations and what kind of data structures they construct. Section 5.6.2 provides a deeper insight on sampling issues. A typical classification of sampling-based planners is between multi-query and single-query approaches:

- In the first category, the planners construct a roadmap, an undirected graph G that is precomputed once so as to *map the connectivity properties of* $\mathcal{C}_{\text{free}}$. After this step, multiple queries in the same environment can be answered using only the constructed roadmap. Such planners are described in Sect. 5.2.1.
- Planners in the second category build tree data structures on the fly given a planning query. They attempt to focus on exploring the part of the C-space that will lead to solving a specific query as fast as possible. They are described in Sect. 5.2.2.

Both approaches, however, make similar use of a collision checking primitive. The objective of a collision detector is to report all geometric contacts between objects given their geometries and transformations [5.16–18]. The availability of packages that were able to answer collision queries in a fraction of a second was critical to the development of sampling-based planners. Modern planners use collision detectors as a *black box*. Initially the planner provides the geometries of all the involved objects and specifies which of them are mobile. Then, in order to validate a robot configuration, a planner provides the corresponding robot transformation and a collision detector responds on whether the objects collide or not. Many packages represent the geometric models hierarchically, avoid computing all-pairwise interactions, and conduct a binary search to evaluate collisions. Except from configurations, a planner must also validate entire paths. Some collision detectors return distance-from-collision information, which can be used to infer that entire neighborhoods in $\mathcal{C}$ are valid. It is often more expensive, however, to extract this information; instead paths are usually validated point by point using a small stepping size either incrementally or by employing binary search. Some collision detectors are incremental by design, which means that they can be faster by reusing information from a previous query [5.16].

5.2.1 Multi-Query Planners: Mapping the Connectivity of $\mathcal{C}_{\text{free}}$

Planners that aim to answer multiple queries for a certain static environment use a preprocessing phase during which they attempt to map the connectivity properties of $\mathcal{C}_{\text{free}}$ onto a roadmap. This roadmap has the form of a graph G, with vertices as configurations and edges as paths. A union of 1-D curves is a roadmap G if it satisfies the following properties:

1. *Accessibility*: From any $\boldsymbol{q} \in \mathcal{C}_{\text{free}}$, it is simple and efficient to compute a path $\tau : [0, 1] \to \mathcal{C}_{\text{free}}$ such that $\tau(0) = \boldsymbol{q}$ and $\tau(1) = \boldsymbol{s}$, in which $\boldsymbol{s}$ may be any point in $S(G)$. $S(G)$ is the *swath* of G, the union of all configurations reached by all edges and vertices. This means that it is always possible to connect a planning query pair $\boldsymbol{q}_{\text{I}}$ and $\boldsymbol{q}_{\text{G}}$ to some $\boldsymbol{s}_{\text{I}}$ and $\boldsymbol{s}_{\text{G}}$, respectively, in $S(G)$.
2. *Connectivity preserving*: The second condition requires that, if there exists a path $\tau : [0, 1] \to \mathcal{C}_{\text{free}}$ such that $\tau(0) = \boldsymbol{q}_{\text{G}}$ and $\tau(1) = \boldsymbol{q}_{\text{G}}$, then there also exists a path $\tau' : [0, 1] \to S(G)$, such that $\tau'(0) = \boldsymbol{s}_{\text{I}}$ and $\tau'(1) = \boldsymbol{s}_{\text{G}}$. Thus, solutions are not missed because G fails to capture the connectivity of $\mathcal{C}_{\text{free}}$.

The probabilistic roadmap method (PRM) approach [5.19] attempts to approximate such a roadmap G in a computationally efficient way. The preprocessing phase of PRM, which can be extended to sampling-based roadmaps in general, follows these steps:

1. *Initialization*: Let $G(V, E)$ represent an undirected *graph*, which is initially empty. Vertices of G will correspond to collision-free configurations, and edges to collision-free paths that connect vertices.
2. *Configuration sampling*: A configuration $\alpha(i)$ is sampled from $\mathcal{C}_{\text{free}}$ and added to the vertex set V. $\alpha(\cdot)$ is an infinite, dense sample sequence and $\alpha(i)$ is the i-th point in that sequence.

3. *Neighborhood computation*: Usually, a *metric* is defined in the C-space, $\rho : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$. Vertices $\boldsymbol{q}$ already in V are then selected as part of $\alpha(i)$'s neighborhood if they have small distance according to ρ.
4. *Edge consideration*: For those vertices $\boldsymbol{q}$ that do not belong in the same connected component of G with $\alpha(i)$ the algorithm attempts to connect them with an edge.
5. *Local planning method*: Given $\alpha(i)$ and $\boldsymbol{q} \in \mathcal{C}_{\text{free}}$ a module is used that attempts to construct a path $\tau_s : [0, 1] \to \mathcal{C}_{\text{free}}$ such that $\tau(0) = \alpha(i)$ and $\tau(1) = \boldsymbol{q}$. Using collision detection, τ_s must be checked to ensure that it does not cause a collision.
6. *Edge insertion*: Insert τ_s into E, as an edge from $\alpha(i)$ to $\boldsymbol{q}$.
7. *Termination*: The algorithm is typically terminated when a predefined number of collision-free vertices N has been added in the roadmap.

The algorithm is incremental in nature. Computation can be repeated by starting from an already existing graph. A general sampling-based roadmap is summarized in Algorithm 1.

Algorithm 5.1
SAMPLING-BASED ROADMAP
N: number of nodes to include in the roadmap

```
G.init(); i ← 0;
while i < N do
  if α(i) ∈ C_free then
    G.add_vertex(α(i)); i ← i + 1;
    for q ∈ NEIGHBORHOOD(α(i),G) do
      if CONNECT (α(i), q) then
        G.add_edge (α(i), q);
      endif
    end for
  endif
end while
```

An illustration of the algorithm's behavior is depicted in Fig. 5.3. To solve a query, $\boldsymbol{q}_{\text{I}}$ and $\boldsymbol{q}_{\text{G}}$ are connected to the roadmap, and graph search is performed.

For the original PRM [5.19], the configuration $\alpha(i)$ was produced using random sampling. For the connection step between $\boldsymbol{q}$ and $\alpha(i)$, the algorithm used straight line paths in the C-space. In some cases a connection was not attempted if $\boldsymbol{q}$ and $\alpha(i)$ were in the same connected component. There have been many subsequent works that try to improve the roadmap quality while using fewer samples. Methods for concentrating samples at or near the boundary of $\mathcal{C}_{\text{free}}$ are presented in [5.20, 21]. Methods that move samples as far from the boundary as possible appear in [5.22, 23]. Deterministic sampling techniques, including grids, appear in [5.24]. A method of pruning vertices based on mutual visibility that leads to a dramatic reduction in the number of roadmap vertices appears in [5.25]. Theoretical analysis of sampling-based roadmaps appears in [5.24, 26, 27] and is briefly discussed in Sect. 5.6.2. An experimental comparison of sampling-based roadmap variants appears in [5.28]. One difficulty in these roadmap approaches is identifying narrow passages. One proposal is to use the *bridge test* for identifying these [5.29]. For other PRM-based works, see [5.30–34]. Extended discussion of the topic can be found in [5.5, 7].

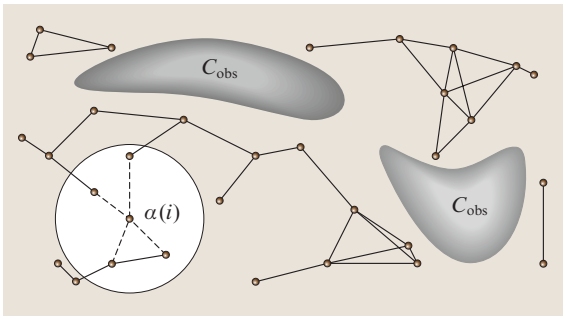

Fig. 5.3 The sampling-based roadmap is constructed incrementally by attempting to connect each new sample, $\alpha(i)$, to nearby vertices in the roadmap

5.2.2 Single-Query Planners: Incremental Search

Single-query planning methods focus on a single initial–goal configuration pair. They probe and search the continuous C-space by extending tree data structures initialized at these known configurations and eventually connecting them. Most single-query methods conform to the following template:

1. *Initialization*: Let $G(V, E)$ represent an undirected *search graph*, for which the vertex set V contains a vertex for one (usually $\boldsymbol{q}_{\text{I}}$) or more configurations in $\mathcal{C}_{\text{free}}$, and the edge set E is empty. Vertices of G are collision-free configurations, and edges are collision-free paths that connect vertices.
2. *Vertex selection method*: Choose a vertex $\boldsymbol{q}_{\text{cur}} \in V$ for expansion.

3. *Local planning method*: For some $\boldsymbol{q}_{\text{new}} \in \mathcal{C}_{\text{free}}$, which may correspond to an existing vertex in V but on a different tree or a sampled configuration, attempt to construct a path $\tau_s : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$ such that $\tau(0) = \boldsymbol{q}_{\text{cur}}$ and $\tau(1) = \boldsymbol{q}_{\text{new}}$. Using collision detection, 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 $\boldsymbol{q}_{\text{cur}}$ to $\boldsymbol{q}_{\text{new}}$. If $\boldsymbol{q}_{\text{new}}$ is not already in V, then it is inserted.
5. *Check for a solution*: Determine whether G encodes a solution path.
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.

During execution, G may be organized into one or more trees. This leads to: (1) *unidirectional* methods, which involve a single tree, usually rooted at $\boldsymbol{q}_{\text{I}}$ [5.35], (2) *bidirectional* methods, which involve two trees, typically rooted at $\boldsymbol{q}_{\text{I}}$ and $\boldsymbol{q}_{\text{G}}$ [5.35], and (3) *multidirectional* methods, which may have more than two trees [5.36,37]. The motivation for using more than one tree is that a single tree may become trapped trying to find an exit through a narrow opening. Traveling in the opposite direction, however, may be easier. As more trees are considered it becomes more complicated to determine which connections should be made between trees.

Rapidly Exploring Dense Trees

The important idea with this family of techniques is that the algorithm must incrementally explore the properties of the C-space. An algorithm that achieves this objective is the rapidly exploring random tree (RRT) [5.35], which can be generalized to the rapidly exploring dense tree (RDT) for any dense sampling, deterministic or random [5.7]. The basic idea is to induce a *Voronoi bias* in the exploration process by selecting for expansion the point in the tree that is closest to $\alpha(i)$ in each iteration. Using random samples, the probability that a vertex is chosen is proportional to the volume of its Voronoi region. The tree construction is outlined as:

Algorithm 5.2
RAPIDLY EXPLORING DENSE TREES
k: the exploration steps of the algorithm

```
G.init(q_I);
for i = 1 to k do
  G.add_vertex(α(i));
  q_n ← NEAREST(S(G), α(i));
  G.add_edge(q_n, α(i));
end for
```

The tree starts at $\boldsymbol{q}_{\text{I}}$, and in each iteration, an edge and vertex are added.

So far, the problem of reaching $\boldsymbol{q}_{\text{G}}$ has not been explained. There are several ways to use RDTs in a planning algorithm. One approach is to bias $\alpha(i)$ so that $\boldsymbol{q}_{\text{G}}$ is frequently chosen (perhaps once every 50 iterations). A more efficient approach is to develop a bidirectional search by growing two trees, one from each of $\boldsymbol{q}_{\text{I}}$ and $\boldsymbol{q}_{\text{G}}$. Roughly half of the time is spent expanding each tree in the usual way, while the other half is spend attempting to connect the trees. The simplest way to connect trees is to let the newest vertex of one tree be a substitute for $\alpha(i)$ in extending the other. This tricks one RDT into attempting to connect to the other while using the basic expansion algorithm [5.38]. Several works have extended, adapted, or applied RDTs in various applications [5.37, 39–42]. Detailed descriptions can be found in [5.5, 7].

Other Tree Algorithms

Planners based on the idea of expansive spaces are presented in [5.43–45]. In this case, the algorithm forces exploration by choosing vertices for expansion that have fewer points in a neighborhood around them. In [5.46], additional performance is obtained by self-tuning random walks, which focus virtually all of their effort on exploration. Other successful tree-based algorithms include the path-directed subdivision tree algorithm [5.47] and some of its variants [5.48]. In the literature, it is sometimes hard to locate tree-based planners for ordinary path planning problems as many of them (including RRT) were designed and/or applied to more complex problems (see Sect. 5.4.4). Their performance is nevertheless excellent for a variety of path planing problems.

5.3 Alternative Approaches

Alternative approaches to the sampling-based paradigm include potential-field-based techniques and combinatorial methods that also produce roadmaps, such as cell decompositions. These algorithms are able to elegantly and efficiently solve a narrow class of problems, and are much preferred over the algorithms of Sect. 5.2 in these cases. Most of the combinatorial algorithms are of theoretical interest, whereas sampling-based algorithms are motivated primarily by performance issues in challenging applications. Nevertheless, given some abstractions, the combinatorial algorithms can be used to solve practical problems such as autonomous navigation of mobile planar robots.

5.3.1 Combinatorial Roadmaps

Several algorithms exist for the case in which $\mathcal{C} = \mathbb{R}^2$ and $\mathcal{C}_{\text{obs}}$ is polygonal. Most of these cannot be directly extended to higher dimensions; however, some of the general principles remain the same. The *maximum clearance roadmap* (or *retraction method* [5.49]) constructs a roadmap that keeps paths as far from the obstacles as possible. Paths are contributed to the roadmap from the three cases shown in Fig. 5.5, which correspond to all ways to pair together polygon features. The roadmap can be made naively in time $O(n^4)$ by generating all curves shown in Fig. 5.5 for all possible pairs, computing their intersections, and tracing out the roadmap. Several algorithms exist that provide better asymptotic running time [5.50], 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 [5.51].

An alternative is to compute a *shortest-path roadmap* [5.52], as shown in Fig. 5.6. This is different than the roadmap presented in the previous section because paths may actually touch the obstacles, which must be allowed for paths to be optimal. The roadmap vertices are the reflex vertices of $\mathcal{C}_{\text{obs}}$, which are vertices for which the interior angle is greater than π. An edge exists in the roadmap if and only if a pair of vertices is mutually visible and the line through them pokes into $\mathcal{C}_{\text{free}}$ when extended outward from each vertex (such lines are called *bitangents*). An $O(n^2 \lg n)$-time construction algorithm can be formed by using a radial sweep algorithm from each reflex vertex. It can theoretically be computed in time $O(n^2 + m)$, in which m is the total number of edges in the roadmap [5.53].

Figure 5.7 illustrates the *vertical cell decomposition* approach. The idea is to decompose $\mathcal{C}_{\text{free}}$ into cells that are trapezoids or triangles. Planning in each cell is trivial because it is convex. A roadmap is made by placing a point in the center of each cell and each boundary between cells. Any graph search algorithm can be used to find a collision-free path quickly. The cell decomposition can be constructed in $O(n \lg n)$ time using the *plane-sweep principle* [5.54,55]. Imagine that a vertical line sweeps from $x = -\infty$ to $x = \infty$, stopping at places where a polygon vertex is encountered. In these cases,

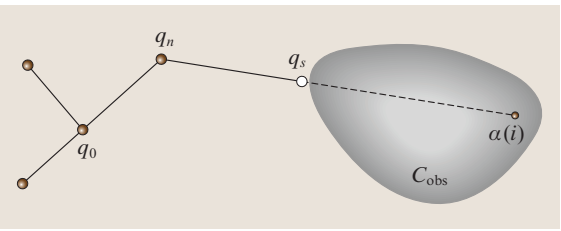

Fig. 5.4 If there is an obstacle, the edge travels up to the obstacle boundary, as far as allowed by the collision detection algorithm

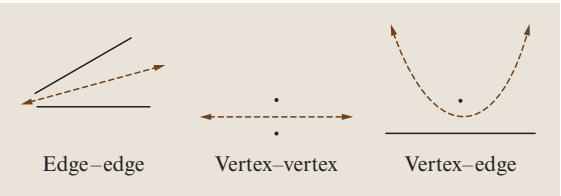

Fig. 5.5 Voronoi roadmap pieces are generated in one of three possible cases. The third case leads to a quadratic curve

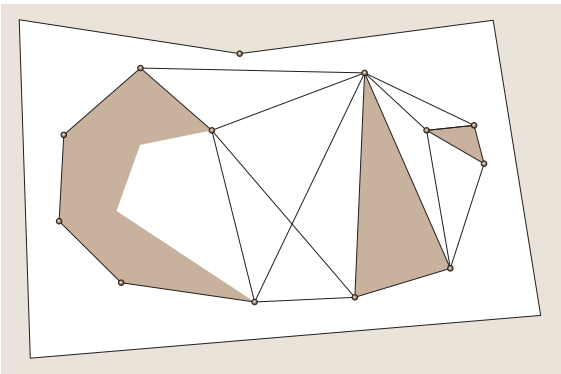

Fig. 5.6 The shortest-path roadmap includes edges between consecutive reflex vertices on $\mathcal{C}_{\text{obs}}$ and also bitangent edges

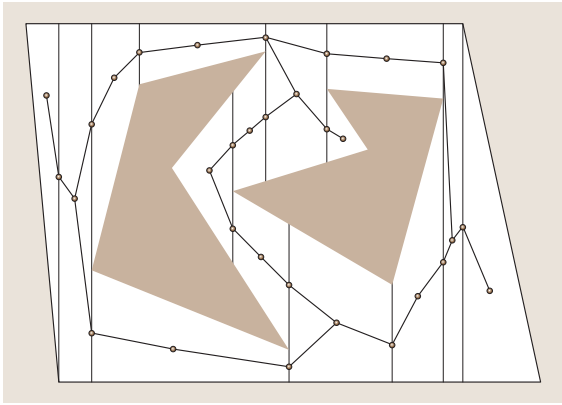

Fig. 5.7 The roadmap derived from the vertical cell decomposition

a cell boundary may be necessary above and/or below the vertex. The order in which segments stab the vertical line is maintained in a balanced search tree. This enables the determination of the vertical cell boundary limits in time $O(\lg n)$. The whole algorithm runs in time $O(n \lg n)$ because there are $O(n)$ vertices at which the sweep line can stop (also, the vertices need to be sorted at the outset, which requires time $O(n \lg n)$).

5.3.2 Roadmaps in Higher Dimensions

It would be convenient if the methods of Sect. 5.3.1 directly extend into higher dimensions. Although this unfortunately does not occur, some of the general ideas extend. To consider a cell decomposition in higher dimensions, there are two main requirements: (1) each cell should be simple enough that motion planning within a cell is trivial; (2) the cells should fit together nicely. A sufficient condition for the first requirement is that cells are convex; more general shapes may be allowed; however, the cells should not contain holes under any circumstances. For the second requirement, a sufficient condition is that the cells can be organized into a *singular complex*. This means that for any two d-dimensional cells for $d \leq n$, if the boundaries of the cells intersect, then the common boundary must itself be a complete cell (of lower dimension).

In two-dimensional polygonal C-spaces, triangulation methods define nice cell decompositions that are appropriate for motion planning. Finding good triangulations, which for example means trying to avoid thin triangles, is given considerable attention in computational geometry [5.55]. Determining the decomposition of a polygonal obstacle region with holes that uses the smallest number of convex cells is *NP*-hard [5.56]. Therefore, we are willing to tolerate nonoptimal decompositions.

In three-dimensional C-spaces, if $\mathcal{C}_{\text{obs}}$ is polyhedral, then the vertical decomposition method directly extends

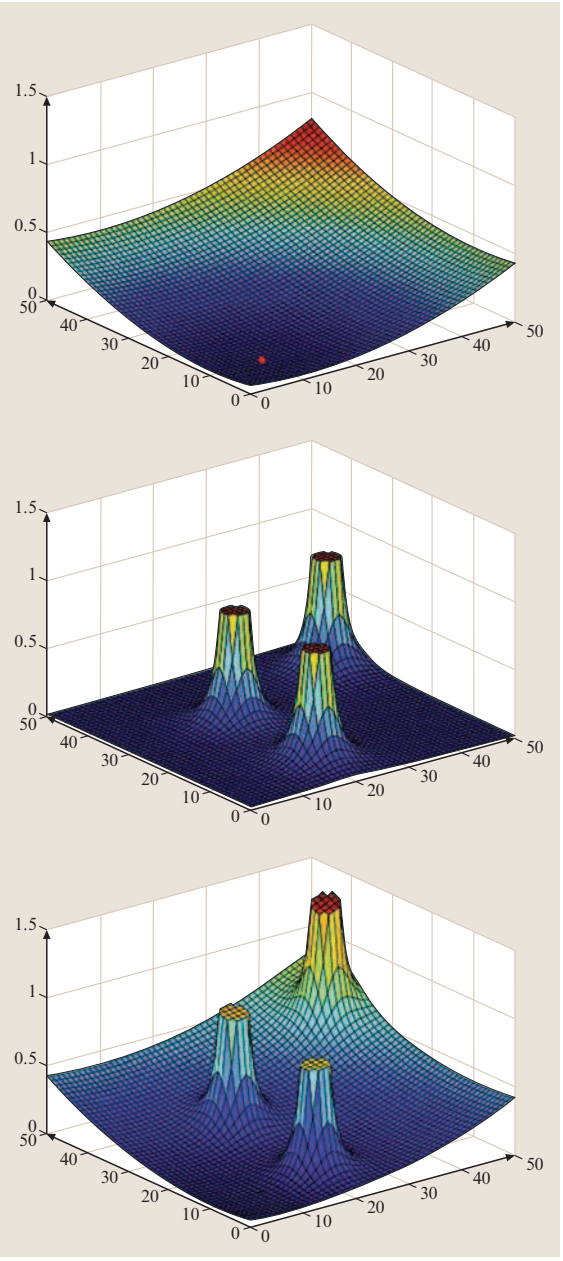

Fig. 5.8 An attractive and a repulsive component define a potential function

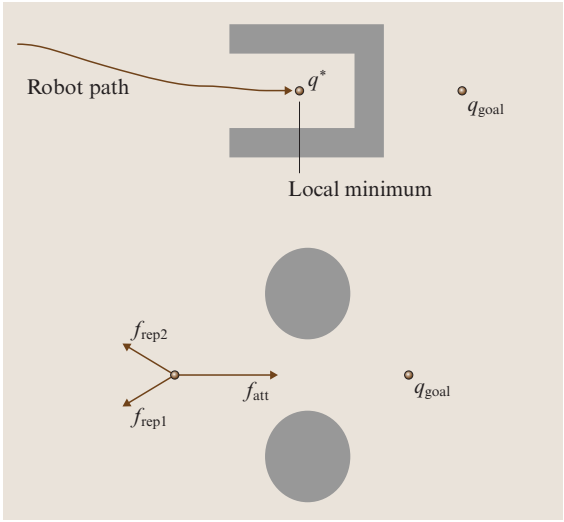

Fig. 5.9 Two examples of the local minimum problem with potential functions

by applying the plane sweep recursively, for example, the critical events may occur at each z coordinate, at which point changes a 2-D vertical decomposition over the x and y coordinates are maintained. The polyhedral case is obtained for a translating polyhedral robot among polyhedral obstacles in $\mathbb{R}^3$; however, for most interesting problems, $\mathcal{C}_{obs}$ becomes nonlinear. Suppose $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$, which corresponds to a robot that can translate and rotate in the plane. Suppose the robot and obstacles are polygonal. For the case of a line-segment robot, an $O(n^5)$ algorithm that is not too difficult to implement is given in [5.57]. The approaches for more general models and C-spaces are extremely difficult to use in practice; they are mainly of theoretical interest and are summarized in Sect. 5.6.3.

5.3.3 Potential Fields

A different approach for motion planning is inspired from obstacle avoidance techniques [5.58]. It does not explicitly construct a roadmap, but instead constructs a differentiable real-valued function $U : \mathbb{R}^m \to \mathbb{R}$, called a potential function, that guides the motion of the moving object. The potential is typically constructed so that it consists of an attractive component $U_a(\boldsymbol{q})$, which pulls the robot towards the goal, and a repulsive component $U_r(\boldsymbol{q})$, which pushes the robot away from the obstacles, as shown in Fig. 5.8. The gradient of the potential function is the vector $\nabla U(\boldsymbol{q}) = DU(\boldsymbol{q})^\top = \left[\frac{\partial U}{\partial \boldsymbol{q}_1}(\boldsymbol{q}), \ldots, \frac{\partial U}{\partial \boldsymbol{q}_m}(\boldsymbol{q})\right]^\top$, which points in the direction that locally maximally increases U. After the definition of U, a path can be computed by starting from $\boldsymbol{q}_I$ and applying *gradient descent*:

1. $\boldsymbol{q}(0) = \boldsymbol{q}_I$; $i = 0$;
2. **while** $\nabla U(\boldsymbol{q}(i)) \neq 0$ **do**
3. $\quad \boldsymbol{q}(i+1) = \boldsymbol{q}(i) + \nabla U(\boldsymbol{q}(i))$
4. $\quad i = i + 1$

However, this gradient-descent approach does not guarantee a solution to the problem. Gradient descent can only reach a local minimum of $U(\boldsymbol{q})$, which may not correspond to the goal state $\boldsymbol{q}_G$, as shown in Fig. 5.9.

A planner that makes uses of potential functions and attempts to avoid the issue of local minima is the *randomized potential planner* [5.59]. The idea is to combine potential functions with random walks by employing multiple planning modes. In one mode, gradient descent is applied until a local minimum is reached. Another mode uses random walks to try to escape local minima. A third mode performs backtracking whenever several attempts to escape a local minimum have failed. In many ways, this approach can be considered as a sampling-based planner. It also provides the weaker

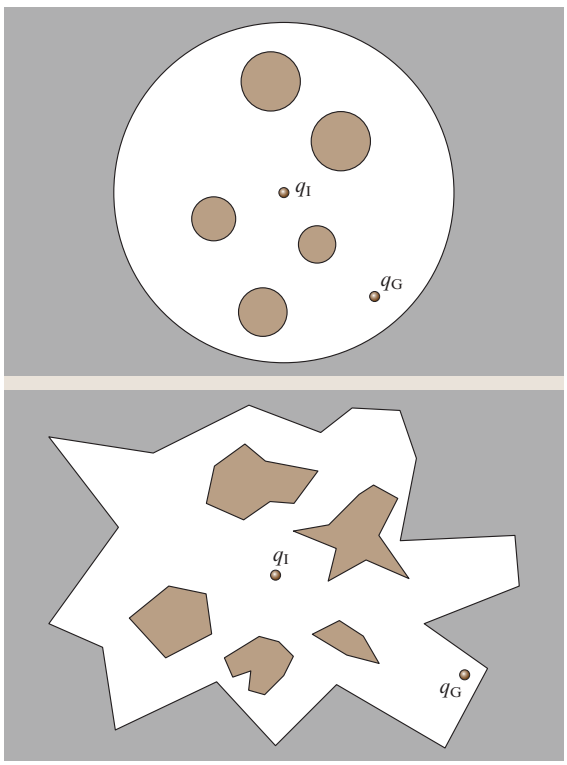

Fig. 5.10 Examples of sphere and star spaces

completeness guarantee but it requires parameter tuning. Recent sampling-based methods achieve better performance by spending more time exploring the space, rather than focusing heavily on a potential function.

The gradient of the potential function can be also used to define a *vector field*, which assigns a motion for the robot at any arbitrary configuration $\boldsymbol{q} \in \mathcal{C}$. This is an important advantage of the approach, beyond its computational efficiency, since it does not only compute a single path, but also a *feedback control* strategy. This makes the approach more robust against control and sensing errors. Most of the techniques in feedback motion planning are based on the idea of *navigation functions* [5.60], which are potential functions properly constructed so as to have a single minimum. A function $\phi : \mathcal{C}_{\text{free}} \rightarrow [0, 1]$ is called a navigation function if it:

- is smooth (or at least C^k for $k \geq 2$),
- has a unique minimum at $\boldsymbol{q}_{\text{G}}$ in the connected component of the free space that contains $\boldsymbol{q}_{\text{G}}$,
- is uniformly maximal on the free-space boundaries,
- and is Morse, which means that all its critical points, such as saddle points, are isolated and can be avoided with small random perturbations.

Navigation functions can be constructed for sphere boundary spaces centered at $\boldsymbol{q}_{\text{I}}$ that contain only spherical obstacles, as illustrated in Fig. 5.10. Then they can be extended to a large family of C-spaces that are diffeomorphic to sphere spaces, such as star-shaped spaces, as shown in Fig. 5.10. A more elaborate description of strategies for feedback motion planning will be presented in Chaps. 35, 36, and 37.

Putting the issue of local minima aside, another major challenge for such potential function based approaches is constructing and representing the C-space in the first place. This issue makes the applications of these techniques too complicated for high-dimensional problems.

5.4 Differential Constraints

Robot motions must usually conform to both global and local constraints. Global constraints on $\mathcal{C}$ have been considered in the form of obstacles and possibly joint limits. Local constraints are modeled with differential equations, and are therefore called *differential constraints*. These limit the velocities, and possibly accelerations, at every point due to kinematic considerations, such as wheels in contact, and dynamical considerations, such as the conservation of angular momentum.

5.4.1 Concepts and Terminology

Let $\dot{\boldsymbol{q}}$ denote a velocity vector. Differential constraints on $\mathcal{C}$ can be expressed either *implicitly* in the form $g_i(\boldsymbol{q}, \dot{\boldsymbol{q}}) = 0$ or *parametrically* in the form $\dot{\boldsymbol{x}} = f(\boldsymbol{q}, \boldsymbol{u})$. The implicit form is more general but often more difficult to understand and utilize. In the parametric form, a vector-valued equation indicates the velocity that is obtained for a given $\boldsymbol{q}$ and $\boldsymbol{u}$, in which $\boldsymbol{u}$ is an *input*, chosen from some *input space*, U. Let T denote an interval of time, starting at $t = 0$.

To model dynamics, the concepts are extended into a *phase space* X of the C-space. Usually each point $\boldsymbol{x} \in X$ represents both a configuration and velocity, $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. Both implicit and parametric representations are possible, yielding $g_i(\boldsymbol{x}, \dot{\boldsymbol{x}}) = 0$ and $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$, respectively. The latter is a common *control system* definition. Note that $\dot{\boldsymbol{x}} = (\dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$, which implies that acceleration constraints and full system dynamics can be expressed.

Planning in the state space X could lead to a straightforward definition of X_{obs} by declaring $\boldsymbol{x} \in X_{\text{obs}}$ if and only if $\boldsymbol{q} \in \mathcal{C}_{\text{obs}}$ for $\boldsymbol{x} = (\boldsymbol{q}, \dot{\boldsymbol{q}})$. However, another interesting possibility exists which provides some intuition about the difficulty of planning with dynamics. This possibility is based on the notion of a *region of inevitable collision*, which is defined as:

$$X_{\text{ric}} = \{\boldsymbol{x}(0) \in X \mid \text{for any } \tilde{u} \in \mathcal{U}_{\infty}, \ \exists t > 0 \text{ such that } \boldsymbol{x}(t) \in X_{\text{obs}}\}, \tag{5.4}$$

in which $\boldsymbol{x}(t)$ is the state at time t obtained by integrating the control function $\tilde{u} : T \rightarrow U$ from $\boldsymbol{x}(0)$. The set $\mathcal{U}_{\infty}$ is a predefined set of all possible control functions. X_{ric} denotes the set of states in which the robot is either in collision or, because of momentum, it cannot do anything to avoid collision. It can be considered as an invisible obstacle region that grows with speed.

Under the general heading of *planning under differential constraints*, there are many important categories of problems that have received considerable attention in research literature. The term *nonholonomic planning*

was introduced for wheeled mobile robots [5.61]. A simple example is that a car cannot move sideways, thereby making parallel parking more difficult. In general, a *nonholonomic constraint* is a differential equality constraint that cannot be integrated into a constraint that involves no derivatives. Typically, nonholonomic constraints that appear in robotics are *kinematic*, and arise from wheels in contact [5.62]. Nonholonomic constraints may also arise from dynamics.

If a planning problem involves constraints on at least velocity and acceleration, the problem is often referred to as *kinodynamic planning* [5.63]. Usually, the model expresses a *fully actuated system*, which means that it can be expressed as $\ddot{\boldsymbol{q}} = h(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u})$, in which U contains an open set that includes the origin of $\mathbb{R}^n$ (here, n is the dimension of both U and $\mathcal{C}$). It is possible for a problem to be nonholonomic, kinodynamic, both, or neither; however, in recent times, the terms are not used with much precision.

Trajectory planning is another important term, which has referred mainly to the problem of determining both a path and velocity function for a robot arm (e.g., PUMA 560). In the treatment below, all of these will be referred to as *planning under differential constraints*.

5.4.2 Discretization of Constraints

The only known methods for complete and optimal planning under differential constraints in the presence of obstacles are for the double integrator system with $X = \mathbb{R}$ [5.64] and $X = \mathbb{R}^2$ [5.65]. To develop planning algorithms in this context, several discretizations are often needed. For ordinary motion planning, only $\mathcal{C}$ needed to be discretized; with differential constraints, T and possibly also U require discretization, in addition to $\mathcal{C}$ (or X).

Discretization of the differential constraints is one of the most important issues. To solve challenging planning problems efficiently, it is often necessary to define *motion primitives* for the particular dynamical system [5.40,66,67]. One of the simplest ways to discretize the differential constraints is to construct a *discrete-time model*, which is characterized by three aspects:

1. The time interval T is partitioned into intervals of length Δt. This enables stages to be assigned, in which stage k indicates that $(k-1)\Delta t$ time has elapsed.
2. A finite subset U_{d} of the action space U is chosen. If U is already finite, then this selection may be $U_{\mathrm{d}} = U$.

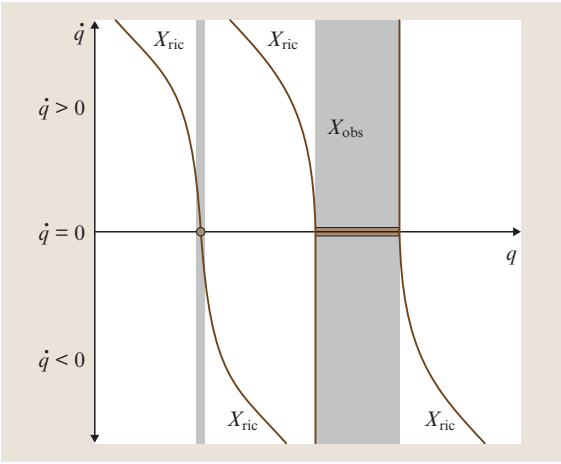

Fig. 5.11 The region of inevitable collision grows quadratically with the speed

3. The action $\boldsymbol{u}(t)$ must remain constant over each time interval.

From an initial state, $\boldsymbol{x}$, a *reachability tree* can be formed by applying all sequences of discretized actions. Figure 5.12 shows the path of this tree for the *Dubins car*, which is a kinematic model of a car that drives in the plane at unit speed and cannot move in reverse. The edges of the tree are circular arcs and line segments. For general systems, each trajectory segment in the tree is determined by numerical integration of $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ for a given $\boldsymbol{u}$. In general, this can be viewed as an *in-*

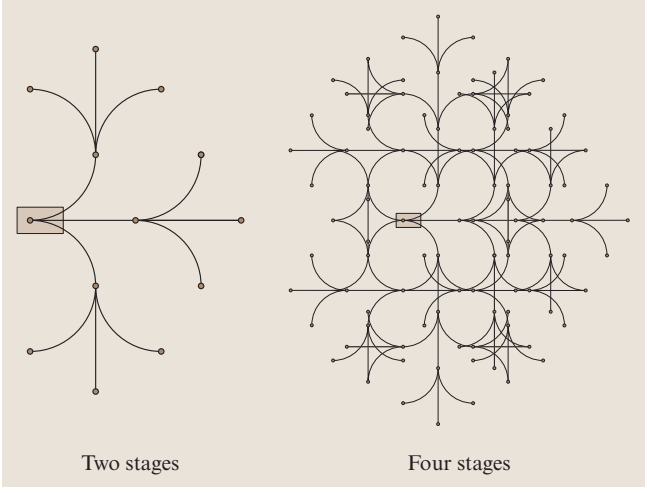

Fig. 5.12 A reachability tree for the Dubins car with three actions. The k-th stage produces 3^k new vertices

cremental simulator that takes an input and produces a trajectory segment according to $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$.

5.4.3 Decoupled Approach

A popular paradigm for trajectory planning and other problems that involve dynamics is to decouple the problem into first planning a path and then computing a timing function along the path by performing a search in the space spanned by $(s, \dot{s})$, in which s is the path parameter and $\dot{s}$ is its first derivative. This leads to a diagram such as the one shown in Fig. 5.13, in which the upper region S_{obs} must be avoided because the corresponding motion of the mechanical system violates the differential constraints. Most methods are based on early work in [5.68, 69], and determine a *bang-bang -control*, which means that they switch between accelerating and decelerating at full speed. This applies to determining time-optimal trajectories (optimal once constrained to the path). Dynamic programming can be used for more general problems [5.70].

For some problems and nonholonomic systems, *steering methods* have been developed to solve the two-point boundary value problem efficiently [5.62, 71]. This means that, for any pair of states, a trajectory that ignores obstacles but satisfies the differential constraints can be obtained. Moreover, for some systems, the complete set of optimal trajectories has been characterized [5.72, 73]. These control-based approaches enable straightforward adaptation of the sampling-based roadmap approach [5.74, 75]. One decoupled approach is to first plan a path that ignores differential constraints, and then incrementally transform it into one that obeys the constraints [5.62, 76].

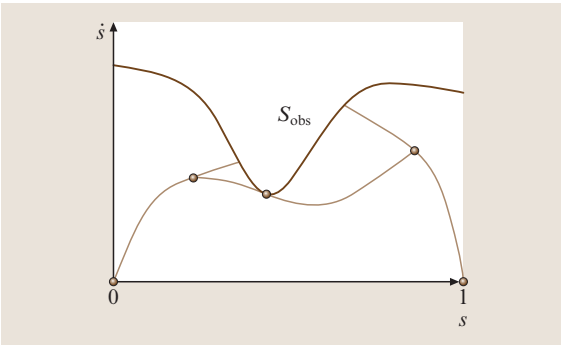

Fig. 5.13 An illustration of the bang–bang approach to computing a time-optimal trajectory. The solution trajectory is obtained by connecting the dots

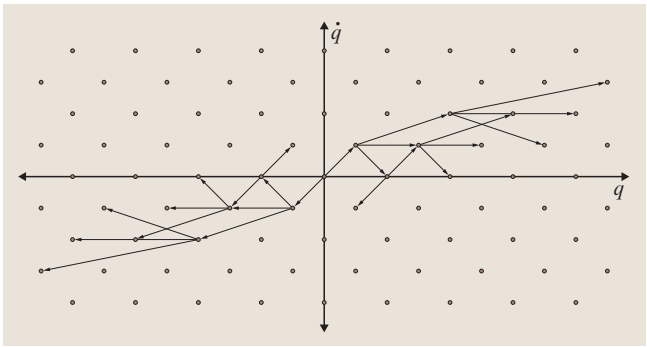

Fig. 5.14 Reachability graph from the origin, shown after three stages (the true edges are actually parabolic arcs when acceleration or deceleration occurs). Note that a lattice is obtained, but the distance traveled in one stage increases as $|\dot{\boldsymbol{q}}|$ increases

5.4.4 Kinodynamic Planning

Due to the great difficulty of planning under differential constraints, many successful planning algorithms that address kinodynamic problems directly in the phase space X are sampling based.

Sampling-based planning algorithms proceed by exploring one or more reachability trees. Many parallels can be drawn with searching on a grid; however, reachability trees are more complicated because they do not necessarily involve a regular lattice structure. The vertex set of reachability trees is dense in most cases. It is therefore not clear how to search a bounded region exhaustively at a fixed resolution. It is also difficult to design approaches that behave like a multiresolution grid, in which refinements can be made arbitrarily to ensure resolution completeness.

Many algorithms attempt to convert the reachability tree into a lattice. This is the basis of the original kinodynamic planning work [5.63], in which the discrete-time approximation to the double integrator, $\ddot{\boldsymbol{q}} = \boldsymbol{u}$, is forced onto a lattice as shown in Fig. 5.14. This enables an approximation algorithm to be developed that solves the kinodynamic planning problem in time polynomial in the approximation quality $1/\epsilon$ and the number of primitives that define the obstacles. Generalizations of the methods to fully actuated systems are described in [5.7]. Surprisingly, it is even possible to obtain a lattice for some underactuated, nonholonomic systems [5.77].

If the reachability tree does not form a lattice, then one approach is to force it to behave as a lattice by imposing a regular cell decomposition over X (or $\mathcal{C}$), and allowing no more than one vertex per cell to be expanded in the reachability graph; see Fig. 5.15. This idea was

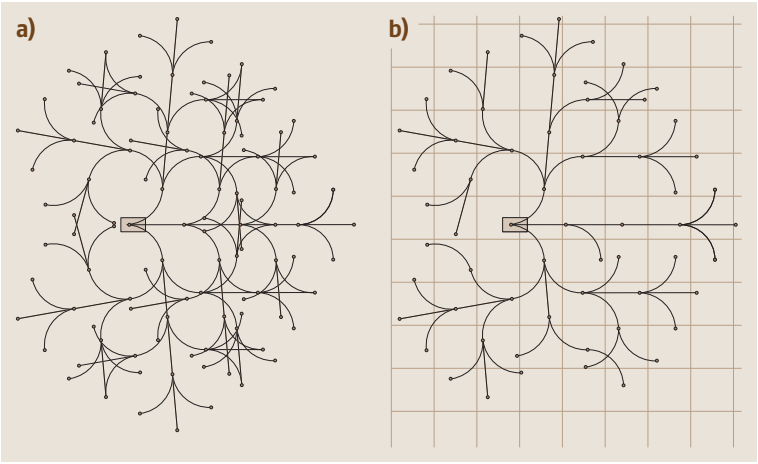

Fig. 5.15 **(a)** The first four stages of a dense reachability graph for the Dubins car; **(b)** one possible search graph, obtained by allowing at most one vertex per cell. Many branches are pruned away. In this simple example there are no cell divisions along the θ axis

introduced in [5.78]. In their version of this approach, the reachability graph is expanded by dynamic programming. Each cell is initially marked as being in collision or being collision free, but not yet visited. As cells are visited during the search, they become marked as such. If a potential new vertex lands in a visited cell, it is not saved. This has the effect of pruning the reachability tree.

Other related approaches do not try to force the reachability tree onto a lattice. RRTs were designed to expand the tree in a way that is biased toward covering as much new territory as possible in each iteration [5.79]. Planners that are based on the concept of expansive trees attempt to control the density of vertices in the tree by analyzing neighborhoods [5.44]. The path-directed subdivision tree planner expands a tree, while building an adaptive subdivision of the state space, so as to avoid resampling the same regions of the space [5.47, 80]. Such approaches can be biased to accelerate the expansion of the tree towards a goal, while still providing the weaker probabilistic completeness guarantee [5.48].

5.5 Extensions and Variations

A brief overview of other important extensions to the basic motion planning problem are presented in this section.

5.5.1 Closed Kinematic Chains

In many cases, the robot may be consist of links that form closed loops. This arises in many important applications, for example, if two arms grasp an object then a loop is formed and a humanoid robot forms a loop if both legs touch the ground. For *parallel robots*, loops are intentionally designed into the robot [5.81]; a classic example is the Stewart–Gough platform. To model closed-chain problems, the loops are broken so that a kinematic tree of links is obtained. The main complication is that constraints on $\mathcal{C}$ of the form $h(\boldsymbol{q}) = 0$ are introduced, which require that the loops are maintained. This causes great trouble for most planning algorithms because without loops a parameterization of $\mathcal{C}$ was available. The closure constraints restrict the planning to a lower-dimensional subset of $\mathcal{C}$ for which no parameterization is given. Computing a parameterization is generally difficult or impossible [5.82], although there has been recent progress for some special cases [5.83].

Sampling-based approaches can generally be adapted to handle closed chains. The main difficulty is that the samples $\alpha(i)$ over $\mathcal{C}$ are unlikely to be configurations that satisfy closure. In [5.84], both RRTs and PRMs were adapted to closed chains. RRTs performed much better because a costly optimization was required in the PRM to move samples onto the closure subspace; RRTs on the other hand do not require samples to lie in this subspace. By decomposing chains into active and passive links, followed by inverse kinematics computations, performance was dramatically improved for PRMs in [5.85]. This idea was further improved by the introduction of

the *random loop generator* (RLG). Based on this, some of the most challenging closed-chain planning problems ever solved appear in [5.86].

5.5.2 Manipulation Planning

In most forms of motion planning, the robot is not allowed to touch obstacles. Suppose instead that it is expected to interact with its environment by manipulating objects. The goal may be to bring an object from one place to another, or to rearrange a collection of objects. This leads to a kind of hybrid motion planning problem, which mixes discrete and continuous spaces. There are discrete modes that correspond to whether the robot is carrying a part [5.87]. In the *transit mode*, the robot moves toward a part. In the *transfer mode*, it carries the part. Transitions between modes require meeting specific grasping and stability requirement. One important variant of manipulation planning is *assembly planning*, in which the goal is to fit a collection of pieces together to make an assembled product [5.88]. Most motion planning work makes limiting assumptions on the kinds of interaction that can occur between the robot and the objects. For richer models of manipulation, see [5.89].

5.5.3 Time-Varying Problems

Suppose that the workspace contains moving obstacles whose trajectories are specified as a function of time. Let $T \subset \mathbb{R}$ denote the *time interval*, which may be *bounded* or *unbounded*. A state X is defined as $X = \mathcal{C} \times T$, in which $\mathcal{C}$ is the usual C-space of the robot. The obstacle region in X is characterized as

$$X_{\text{obs}} = \{(\boldsymbol{q}, t) \in X \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{O}(t) \neq \emptyset\}, \tag{5.5}$$

in which $\mathcal{O}(t)$ is a time-varying obstacle. Many planning algorithms can be adapted to X, which has only one more dimension than $\mathcal{C}$. The main complication is that time must always increase along a path through X.

For the easiest version of the problem, there is no bound on the robot speed. In this case, virtually any sampling-based algorithm can be adapted. Incremental searching and sampling methods apply with little modification, except that paths are directed so that forward time progress is made. Using bidirectional approaches is more difficult for time-varying problems because the goal is usually not a single point due to the time dependency. Sampling-based roadmaps can be adapted; however, a *directed roadmap* is needed, in which every edge must be directed to yield a time-monotonic path.

If the motion model is *algebraic* (i. e., expressed with polynomials) then X_{obs} is semi-algebraic. This enables cylindrical algebraic decomposition to apply. If X_{obs} is polyhedral, as depicted in Fig. 5.16, then vertical decomposition can be used. It is best to first sweep the plane along the T-axis, stopping at the critical times when the linear motion changes.

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 towards making a realistic model is to enforce a bound on the speed of the robot. Unfortunately, the problem is considerably more difficult. Even for piecewise-linear motions of obstacles in the plane, the problem has been established to be PSPACE-hard [5.90]. A complete algorithm based on the shortest-path roadmap is presented in [5.91].

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. A collision-free path in the absence of obstacles is first computed. A search in a two-dimensional space is then performed to determine the *timing function* (or *time scaling*) for the path.

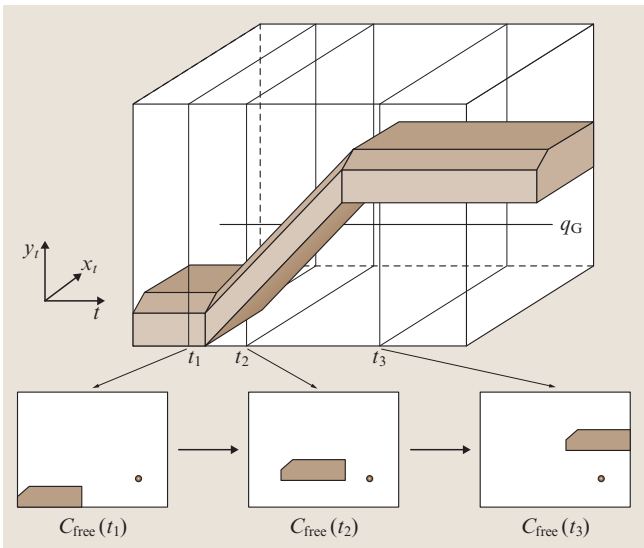

Fig. 5.16 A time-varying example with linear obstacle motion.

5.5.4 Multiple Robots

A simple extension to the basic motion planning problem can be made to handle multibody robots by including robot self-intersections; however, it is important to specify the pairs of bodies for which collision is unac-

ceptable, for example, consecutive links in a robot arm are allowed to touch.

Substantial attention has been devoted to the problem of planning for multiple robots. Suppose there are m robots. A state space is defined that considers the configurations of all robots simultaneously,

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m. \tag{5.6}$$

A state $\boldsymbol{x} \in X$ specifies all robot configurations, and may be expressed as $\boldsymbol{x} = (\boldsymbol{q}^1, \boldsymbol{q}^2, \ldots, \boldsymbol{q}^m)$. 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

$$X^i_{\text{obs}} = \{\boldsymbol{x} \in X \mid \mathcal{A}^i(\boldsymbol{q}^i) \cap \mathcal{O} \neq \emptyset\}. \tag{5.7}$$

This 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

$$X^{ij}_{\text{obs}} = \{\boldsymbol{x} \in X \mid \mathcal{A}^i(\boldsymbol{q}^i) \cap \mathcal{A}^j(\boldsymbol{q}^j) \neq \emptyset\}. \tag{5.8}$$

Both (5.7) and (5.8) will be combined in (5.9) to yield X_{obs}. The obstacle region in X is

$$X_{\text{obs}} = \left(\bigcup_{i=1}^{m} X^i_{\text{obs}} \right) \bigcup \left(\bigcup_{ij,\ i \neq j} X^{ij}_{\text{obs}} \right). \tag{5.9}$$

Once these definitions have been made, any general-purpose planning algorithm can be applied because X and X_{obs} appear no different from $\mathcal{C}$ and $\mathcal{C}_{\text{obs}}$, except that the dimension N may be very high. Approaches that plan directly in X are called *centralized*. The high dimensionality of X motivates the development of *decoupled* approaches that handle some aspects of the planning independently for each robot. Decoupled approaches are usually more efficient, but this usually comes at the expense of sacrificing completeness. An early decoupled approach is *prioritized planning* [5.92, 93], in which a path and timing function is computed for the i-th robot while treating the first $i-1$ robots as moving obstacles as they follow their paths. Another decoupled approach is *fixed-path* coordination [5.94], in which the paths are planned independently for each robot, and then their timing functions are determined by computing a collision-free path through an m-dimensional *coordination space*. Each axis in this space corresponds to the domain of the path of one robot. Figure 5.17 shows an example. The idea has been generalized to coordination on roadmaps [5.95, 96].

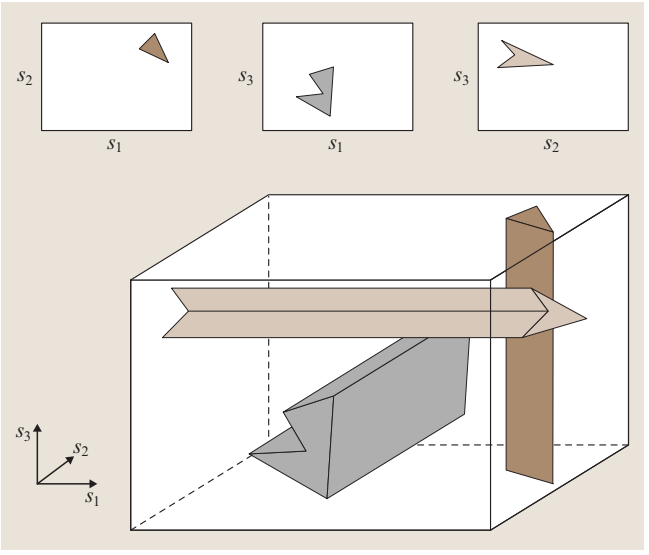

Fig. 5.17 The obstacles that arise from coordinating m robots are always cylindrical. The set of all $\frac{1}{2}m(m-1)$ axis-aligned 2-D projections completely characterizes X_{obs}

5.5.5 Uncertainty in Predictability

If the execution of the plan is not predictable, then feedback is needed. The uncertainty may be modeled either *implicitly*, which means that the plan is able to respond to unexpected future configurations, or *explicitly*, which means that the uncertainty is precisely characterized and analyzed in the development of a plan. Potential-function-based approaches are one way of achieving feedback motion planning.

A plan can be represented as a vector field over $\mathcal{C}_{\text{free}}$, in which each vector indicates the required velocity. The integral curves of the field should flow into the goal without leaving $\mathcal{C}_{\text{obs}}$. If dynamics are a concern, then the vector field can be tracked by an *acceleration-based control model*:

$$\boldsymbol{u} = K(f(\boldsymbol{q}) - \dot{\boldsymbol{q}}) + \nabla_{\dot{\boldsymbol{q}}} f(\boldsymbol{q}) \tag{5.10}$$

in which K is a scalar *gain constant*. Alternatively, a vector field may be designed directly on the phase space, X; however, there are not methods to compute such fields efficiently under general conditions. This can also be considered as a feedback control problem with implicit, nonlinear constraints on X.

If the uncertainty is modeled explicitly, then a *game against nature* is obtained, in which the uncertainty is caused by a special decision maker called *nature*. The decisions of nature can either be modeled *non-*

deterministically, which means that a set of possible actions is specified, or *probabilistically*, which means that a probability distribution or density is specified over the nature actions. Under nondeterministic uncertainty, *worst-case analysis* is usually performed to select a plan; under probabilistic uncertainty, *expected-case analysis* is usually performed. Numerous approaches exist for such problems, including value iteration, Dijkstra-like algorithms, and reinforcement learning algorithms [5.7].

5.5.6 Sensing Uncertainty

Consider solving tasks such as localization, map building, manipulation, target tracking, and pursuit evasion (hide-and-seek) with limited sensing. If the current configuration or state is not known during execution, then the problem is substantially more difficult. Information is obtained from sensors, and the problem naturally lives in an *information space* or *I-space* (see Chap. 11 of [5.7]). The state may include the configuration, velocities, or even the map of the environment (e.g., obstacles). The most basic I-space is the set of all histories that can be obtained during execution, based on all sensing observations, actions previously applied, and the initial conditions. The goal in developing efficient algorithms in this context is to determine information mappings that reduce the I-space size or complexity so that plans that use *information feedback* can be computed. The traditional way to use the information state is for estimating the state. This is sufficient for solving many tasks, but it is often not necessary. It may be possible to design and execute successful plans without ever knowing the current state. This can lead to more robust robot systems which may also be cheaper to manufacture due to weaker sensing requirements. For more material related to this topic, see any of the chapters in Part C of this book.

5.6 Advanced Issues

We cover here a series of more advanced issues, such as topics from topology and sampling theory, and how they influence the performance of motion planners. The last section is devoted to computational algebraic geometry techniques that achieve completeness in the general case. Rather than being a practical alternative, these techniques serve as an upper bound on the best asymptotic running time that could be obtained.

5.6.1 Topology of Configuration Spaces

Manifolds

One reason that the topology of a C-space is important is because it affects its representation. Another reason is that, if a path-planning algorithm can solve problems in a topological space, then that algorithm may carry over to topologically equivalent spaces.

The following definitions are important in order to describe the topology of C-space. A map $\phi : S \to T$ is called a *homeomorphism* if ϕ is a bijection and both ϕ and ϕ^{-1} are continuous. When such a map exists, S and T are said to be homeomorphic. A set S is an n-dimensional *manifold* if it is locally homeomorphic to $\mathbb{R}^n$, meaning that each point in S possesses a neighborhood that is homeomorphic to $\mathbb{R}^n$. For more details, see [5.97, 98].

In the vast majority of motion planning problems, the configuration space is a manifold. An example of a C-space that is not a manifold is the closed unit square: $[0, 1] \times [0, 1] \subset \mathbb{R}^2$, which is a manifold with boundary obtained by pasting the one-dimensional boundary on the two-dimensional open set $(0, 1) \times (0, 1)$. When a C-space is a manifold, then we can represent it with just n parameters, in which n is the dimension of the configuration space. Although an n-dimensional manifold can be represented using as few as n parameters, due to constraints it might be easier to use a representation that has higher number of parameters, e.g., the unit circle $\mathbb{S}^1$ can be represented as $\mathbb{S}^1 = \{(x, y) | x^2 + y^2 = 1\}$ by embedding $\mathbb{S}^1$ in $\mathbb{R}^2$. Similarly, the torus T^2 can be embedded in $\mathbb{R}^3$.

Representation

Embeddings into higher-dimensional spaces can facilitate many C-space operations. For example, the orientation of a rigid body in space can be represented by a $n \times n$ matrix of real numbers. The n^2 matrix entries must satisfy a number of smooth equality constraints, making the manifold of such matrices a *submanifold* of $\mathbb{R}^{m^2}$. One advantage is that these matrices can be multiplied to get another matrix in the manifold. For example, the orientation of a rigid-body in n-dimensional space ($n = 2$ or 3) is described by the set $SO(n)$, the set

of all $n \times n$ rotation matrices. The position and orientation of a rigid body is represented by the set $SE(n)$, the set of all $n \times n$ homogeneous transformation matrices. These matrix groups can be used to (1) represent rigid-body configurations, (2) change the reference frame for the representation of a configuration, and (3) displace a configuration.

There are numerous parameterizations of $SO(3)$ [5.99] but unit quaternions correctly preserve the $\mathcal{C}$-space topology as $\mathbb{S}^1$ represents 2-D rotations. Quaternions were introduced in Chap. 1. There is, however, a two-to-one correspondence between unit quaternions and 3-D rotation matrices. This causes a topological issue that is similar to the equivalence of 0 and 2π for 2-D rotations. One way to account for this is to declare antipodal (opposite) points on $\mathbb{S}^3$ to be equivalent. In planning, only the upper hemisphere of $\mathbb{S}^3$ is needed, and paths that cross the equator instantly reappear on the opposite side of $\mathbb{S}^3$, heading back into the northern hemisphere. In topology, this is called a real projective space: $\mathbb{RP}^3$. Hence, the C-space of a 3-D body capable only of rotation is $\mathbb{RP}^3$. If both translation and rotation are allowed, then $SE(3)$, the set of all 4×4 homogeneous transformation matrices, yields:

$$\mathcal{C} = \mathbb{R}^3 \times \mathbb{RP}^3, \tag{5.11}$$

which is six dimensional. A configuration $\boldsymbol{q} \in \mathcal{C}$ can be expressed using quaternions with seven coordinates, (x, y, z, a, b, c, d), in which $a^2 + b^2 + c^2 + d^2 = 1$.

5.6.2 Sampling Theory

Since the most successful paradigm for motion planning today is the sampling-based framework, presented in Sect. 5.2, sampling theory becomes relevant to the motion planning problem.

Metrics in Configuration/State Spaces

Virtually all sampling-based methods require some notion of distance on $\mathcal{C}$. For example, the sampling-based roadmap method selects candidate vertices to connect a new configuration given a distance-defined neighborhood. Similarly, the rapidly exploring dense trees expands the tree from the nearest node of the tree to a newly sampled configuration. Usually, a *metric*, $\rho : \mathcal{C} \times \mathcal{C} \to \mathbb{R}$, is defined, which satisfies the standard axioms: nonnegativity, reflexivity, symmetry, and the triangle inequality.

Two difficult issues that arise in constructing a metric are: (1) the topology of $\mathcal{C}$ must be respected, and (2) several different quantities, such as linear and angular displacements, must be compared in some way. To illustrate the second issue, consider defining a metric ρ_z for a space constructed as $Z = X \times Y$ as

$$\begin{aligned} \rho_z(z, z') &= \rho_z(x, y, x', y') \\ &= c_1 \rho_x(x, x') + c_2 \rho_y(y, y') \,. \end{aligned} \tag{5.12}$$

Above, c_1 and c_2 are arbitrary positive constants that indicate the relative weights of the two components. For a 2-D rotation, θ_i, expressed as $a_i = \cos\theta_i$ and $b_i = \sin\theta_i$, a useful metric is:

$$\rho(a_1, b_1, a_2, b_2) = \cos^{-1}(a_1 a_2 + b_1 b_2). \tag{5.13}$$

The 3-D equivalent is obtained by defining

$$\rho_0(\boldsymbol{h}_1, \boldsymbol{h}_2) = \cos^{-1}(a_1 a_2 + b_1 b_2 + c_1 c_2 + d_1 d_2) \,, \tag{5.14}$$

in which each $\boldsymbol{h}_i = (a_i, b_i, c_i, d_i)$ is a unit quaternion. The metric is defined as $\rho(\boldsymbol{h}_1, \boldsymbol{h}_2) = \min(\rho_0(\boldsymbol{h}_1, \boldsymbol{h}_2), \rho_0(\boldsymbol{h}_1, -\boldsymbol{h}_2))$, by respecting the required identification of antipodal points. This computes the shortest distance in $\mathbb{R}^4$, for a path constrained to the unit sphere.

In some algorithms, defining volume on $\mathcal{C}$ may also be important. In general, this leads to a *measure space*, for which the volume function (called the *measure*) must satisfy axioms that resemble the probability axioms, but without normalization. For transformation groups, one must be careful to define volumes in a way that is invariant with respect to transformations. Such volumes are called *Haar measures*. Defining volumes via balls using the metric definitions (5.13) and (5.14) actually satisfy this concern.

Probabilistic versus Deterministic Sampling

The C-space may be sampled *probabilistically* or *deterministically*. Either way, the requirement is usually that a dense sequence α of samples is obtained. This means that, in the limit as the number of samples tends to infinity, the samples become arbitrarily close to every point in $\mathcal{C}$. For probabilistic sampling, this denseness (with probability one) ensures *probabilistic completeness* of a planning algorithm. For deterministic sampling, it ensures *resolution completeness*, which means that, if a solution exists, the algorithm is guaranteed to find it; otherwise, it may run forever.

For probabilistic sampling, samples are selected randomly over $\mathcal{C}$, using a uniform probability density function. To obtain uniformity in a meaningful way, the Haar measure should be used. This is straightforward in many cases; $SO(3)$ however is tricky. A uniform

(with respect to Haar measure) random quaternion is selected as follows. Choose three points $u_1, u_2, u_3 \in [0, 1]$ uniformly at random, and let [5.100]

$$\boldsymbol{h} = \left(\sqrt{1-u_1}\sin 2\pi u_2,\ \sqrt{1-u_1}\cos 2\pi u_2,\ \sqrt{u_1}\sin 2\pi u_3,\ \sqrt{u_1}\cos 2\pi u_3\right). \tag{5.15}$$

Even though random samples are uniform in some sense, they are also required to have some irregularity to satisfy statistical tests. This has motivated the development of deterministic sampling schemes that offer better performance [5.101]. Instead of being concerned with randomness, deterministic sampling techniques are designed to optimize criteria, such as *discrepancy* and *dispersion*. Discrepancy penalizes regularity in the sample, which frequently causes trouble in numerical integration. Dispersion gives the radius of the largest empty (not containing samples) ball. Thus, driving dispersion down quickly means that the whole space is explored quickly. Deterministic samples may be *irregular* neighborhood structure (appearing much like random samples), or *regular* neighborhood structure, which means that points are arranged along a grid or lattice. For more details in the context of motion planning, see [5.7].

5.6.3 Computational Algebraic Geometry Techniques

Sampling-based algorithms provide good practical performance at the expense of achieving only a weaker form of completeness. On the other hand, complete algorithms, which are the focus of this section, are able to deduce that there is no solution to a planning problem.

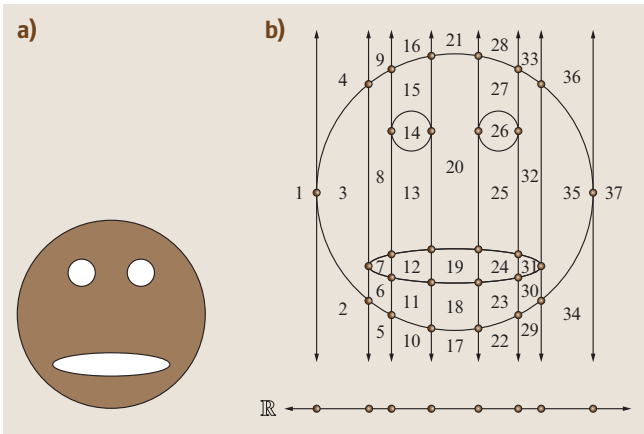

Fig. 5.18 **(a)** A face modeled with four algebraic primitives, and **(b)** a cylindrical algebraic decomposition of the face

Complete algorithms are able to solve virtually any motion planning problem as long as $\mathcal{C}_{\text{obs}}$ is represented by patches of algebraic surfaces. Formally, the model must be *semi-algebraic*, which means that it is formed from unions and intersections of roots of multivariate polynomials in q, and for computability, the polynomials must have rational coefficients (otherwise roots may not have finite representations). The set of all roots to polynomials with rational coefficients is called *real algebraic numbers* and has many nice computational properties. See [5.12, 102–104] for more information on the exact representation and calculation with real algebraic numbers. For a gentle introduction to algebraic geometry, see [5.82].

To use techniques based on algebraic geometry, the first step is to convert the models into the required polynomials. Suppose that the models, the robot, $\mathcal{A}$, and the obstacles $\mathcal{O}$ are semi-algebraic (this includes polyhedral models). For any number of attached 2-D or 3-D bodies, the kinematic transformations can be expressed using polynomials. Since polynomial transformations of polynomials yield polynomials, the transformed robot model is polynomial. The algebraic surfaces that comprise $\mathcal{C}_{\text{obs}}$ are computed by carefully considering all contact types, which characterize all ways to pair a robot feature (faces, edges, vertices) with an obstacle feature [5.6, 7, 9, 105]. This step already produces too many model primitives to be useful in most applications.

Once the semi-algebraic representation has been obtained, powerful techniques from algebraic geometry can be exploited. One of the most widely known algorithms, *cylindrical algebraic decomposition* [5.102, 106, 107], provides the information needed to solve the motion planning problem. It was originally designed to determine whether *Tarski sentences*, which involve quantifiers and polynomials, are satisfiable, and to find an equivalent expression that does not involve quantifiers. The decomposition produces a finite set of cells over which the signs of the polynomials remain fixed. This enables a systematic approach to satisfiability and quantifier elimination. It was recognized by *Schwartz* and *Sharir* [5.104] that it also solves motion planning.

The method is conceptually simple, but there are many difficult technical details. The decomposition is called cylindrical because the cells are organized into vertical columns of cells, see Fig. 5.18 for a 2-D example. There are two kinds of critical events, shown in Fig. 5.19. At critical points, rays are extended indef-

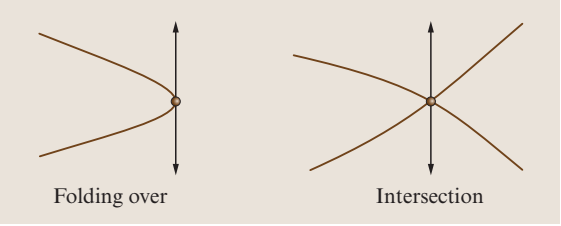

Fig. 5.19 Critical points occur either when the surface folds over in the vertical direction or when surfaces intersect

initely in both vertical directions. The decomposition differs from the vertical decomposition in Fig. 5.7 because there the rays were only extended until the next obstacle was hit. Here, columns of cells are obtained.

In n dimensions, each column represents a chain of cells. The first and last cells are n-dimensional and unbounded. The remaining cells are bounded and alternate between being $(n-1)$-dimensional and n dimensional. The bounded n-dimensional cells are bounded above and below by the roots of single multivariate polynomials. This makes it simple to describe the cells and their connectivity. To compute this cell decomposition, the algorithm constructs a cascading chain of projections. In the first step, $\mathcal{C}_{\text{obs}}$ is projected from $\mathbb{R}^n$ to $\mathbb{R}^{n-1}$. This is followed by a projection into $\mathbb{R}^{n-2}$. This repeats until $\mathbb{R}$ is obtained with a univariate polynomial that encodes the places at which all critical boundaries need to be placed. In a second phase of the algorithm, a series of liftings is performed. Each lifting takes the polynomials and cell decomposition over $\mathbb{R}^i$ and lifts them via columns of cells to $\mathbb{R}^{i+1}$. A single lifting is illustrated in Fig. 5.18b. The running time of the full algorithm depends on the particular methods used to perform the algebraic computations. The total running time required to use cylindrical algebraic decomposition for motion planning is bounded by $(md)^{O(1)^n}$, in which m is the number of polynomials to describe $\mathcal{C}_{\text{obs}}$ (a huge number), and d is the maximum algebraic degree. [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 the front of the whole formula.] The main point to remember is that the algorithm is doubly exponential in the dimension of $\mathcal{C}$ (even the number of cells is doubly exponential).

Although performing the cylindrical decomposition is sufficient for solving motion planning, it computes more information than is necessary. This motivates *Canny*'s roadmap algorithm [5.12], which 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 is only singly exponential in dimension [5.12].

The basic idea is to find silhouette curves in $\mathbb{R}^2$ of $\mathcal{C}_{\text{obs}}$ in $\mathbb{R}^n$. The method finds zero-dimensional critical points and one-dimensional critical curves. The critical curves become roadmap edges, and the critical points are places at which the algorithm recursively finds silhouettes of $(n-1)$-dimensional slices of $\mathcal{C}_{\text{obs}}$. These contribute more critical points and curves. The curves are added to the roadmap, and the algorithm recurses again on the critical points. The recursive iterations terminate at $n = 2$. Canny showed that the resulting union of critical curves preserves the connectivity of $\mathcal{C}_{\text{obs}}$ (and hence, $\mathcal{C}_{\text{free}}$). Some of the technical issues are: the algorithm works with a stratification of $\mathcal{C}_{\text{obs}}$ into manifolds; there are strong general position assumptions that are hard to meet; paths are actually considered along the boundary of $\mathcal{C}_{\text{free}}$; and the method does not produce a parameterized solution path. For improvements to Canny's algorithm and many other important details, see [5.102].

5.7 Conclusions and Further Reading

The brief survey given here hardly does justice to motion planning, which is a rich and active research field. For more details, we recommend consulting two recent textbooks [5.5, 7]. In addition, see the classic textbook of *Latombe* [5.6], the classic papers in [5.4], and the recent surveys in [5.2, 3]. Furthermore, consult the related handbook chapters that were indicated throughout this chapter.

References

5.1 J.H. Reif: Complexity of the mover's problem and generalizations, IEEE Symp. Found. Comput. Sci. (1979) pp. 421–427

5.2 H.H. Gonzalez-Banos, D. Hsu, J.C. Latombe: Motion planning: Recent developments. In: *Automous Mobile Robots: Sensing, Control, Decision-Making and Applications*, ed. by S.S. Ge, F.L. Lewis (CRC, Boca Raton 2006)

5.3 S.R. Lindemann, S.M. LaValle: Current issues in sampling-based motion planning. In: *Robotics Research: The Eleventh International Symposium*, ed. by P. Dario, R. Chatila (Springer, Berlin 2005) pp. 36–54

5.4 J.T. Schwartz, M. Sharir: A survey of motion planning and related geometric algorithms, Artif. Intell. J. **37**, 157–169 (1988)

5.5 H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, S. Thrun: *Principles of Robot Motion: Theory, Algorithms, and Implementations* (MIT Press, Cambridge 2005)

5.6 J.C. Latombe: *Robot Motion Planning* (Kluwer, Boston 1991)

5.7 S.M. LaValle: *Planning Algorithms* (Cambridge Univ. Press, Cambridge 2006)

5.8 S. Udupa: Collision detection and avoidance in computer controlled manipulators. Ph.D. Thesis (Dept. of Electical Engineering, California Institute of Technology 1977)

5.9 T. Lozano-Pérez: Spatial planning: A configuration space approach, IEEE Trans. Comput. **C-32**(2), 108–120 (1983)

5.10 J.T. Schwartz, M. Sharir: On the piano movers' problem: III. Coordinating the motion of several independent bodies, Int. J. Robot. Res. **2**(3), 97–140 (1983)

5.11 J.T. Schwartz, M. Sharir: On the piano movers' problem: V. The case of a rod moving in three-dimensional space amidst polyhedral obstacles, Commun. Pure Appl. Math. **37**, 815–848 (1984)

5.12 J.F. Canny: *The Complexity of Robot Motion Planning* (MIT Press, Cambridge 1988)

5.13 D. Halperin, M. Sharir: A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment, Discrete Comput. Geom. **16**, 121–134 (1996)

5.14 J.E. Hopcroft, J.T. Schwartz, M. Sharir: On the complexity of motion planning for multiple independent objects: PSPACE-hardness of the warehouseman's problem, Int. J. Robot. Res. **3**(4), 76–88 (1984)

5.15 J. Canny, J. Reif: New lower bound techniques for robot motion planning problems, IEEE Symp. Found. Comput. Sci. (1987) pp. 49–60

5.16 M.C. Lin, J.F. Canny: Efficient algorithms for incremental distance computation, IEEE Int. Conf. Robot. Autom. (1991)

5.17 P. Jiménez, F. Thomas, C. Torras: Collision detection algorithms for motion planning. In: *Robot Motion Planning and Control*, ed. by J.P. Laumond (Springer, Berlin 1998) pp. 1–53

5.18 M.C. Lin, D. Manocha: Collision and proximity queries. In: *Handbook of Discrete and Computational Geometry, 2nd Ed*, ed. by J.E. Goodman, J. O'Rourke (Chapman Hall/CRC, New York 2004) pp. 787–807

5.19 L.E. Kavraki, P. Svestka, J.C. Latombe, M.H. Overmars: Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Trans. Robot. Autom. **12**(4), 566–580 (1996)

5.20 N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones, D. Vallejo: OBPRM: an obstacle-based PRM for 3D workspaces, Workshop Algorith. Found. Robot. (1998) pp. 155–168

5.21 V. Boor, M.H. Overmars, A.F. van der Stappen: The Gaussian sampling strategy for probabilistic roadmap planners, IEEE Int. Conf. Robot. Autom. (1999) pp. 1018–1023

5.22 C. Holleman, L.E. Kavraki: A framework for using the workspace medial axis in PRM planners, IEEE Int. Conf. Robot. Autom. (2000) pp. 1408–1413

5.23 J.M. Lien, S.L. Thomas, N.M. Amato: A general framework for sampling on the medial axis of the free space, IEEE Int. Conf. Robot. Autom. (2003)

5.24 S.M. LaValle, M.S. Branicky, S.R. Lindemann: On the relationship between classical grid search and probabilistic roadmaps, Int. J. Robot. Res. **23**(7/8), 673–692 (2004)

5.25 T. Siméon, J.-P. Laumond, C. Nissoux: Visibility based probabilistic roadmaps for motion planning, Adv. Robot. **14**(6), 477–493 (2000)

5.26 J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, P. Raghavan: A random sampling scheme for robot path planning. In: *Proceedings International Symposium on Robotics Research*, ed. by G. Giralt, G. Hirzinger (Springer, New York 1996) pp. 249–264

5.27 A. Ladd, L.E. Kavraki: Measure theoretic analysis of probabilistic path planning, IEEE Trans. Robot. Autom. **20**(2), 229–242 (2004)

5.28 R. Geraerts, M. Overmars: Sampling techniques for probabilistic roadmap planners, Int. Conf. Intell. Auton. Syst. (2004)

5.29 D. Hsu, T. Jiang, J. Reif, Z. Sun: The bridge test for sampling narrow passages with probabilistic roadmap planners, IEEE Int. Conf. Robot. Autom. (2003)

5.30 R. Bohlin, L. Kavraki: Path planning using lazy PRM, IEEE Int. Conf. Robot. Autom. (2000)

5.31 B. Burns, O. Brock: Sampling-based motion planning using predictive models, IEEE/RSJ Int. Conf. Intell. Robot. Autom. (2005)

5.32 P. Isto: Constructing probabilistic roadmaps with powerful local planning and path optimization, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2002) pp. 2323–2328

5.33 P. Leven, S.A. Hutchinson: Using manipulability to bias sampling during the construction of probabilistic roadmaps, IEEE Trans. Robot. Autom. **19**(6), 1020–1026 (2003)

5.34 D. Nieuwenhuisen, M.H. Overmars: Useful cycles in probabilistic roadmap graphs, IEEE Int. Conf. Robot. Autom. (2004) pp. 446–452

5.35 S.M. LaValle, J.J. Kuffner: Rapidly-exploring random trees: progress and prospects. In: *Algorithmic and Computational Robotics: New Direction*, ed. by B.R. Donald, K.M. Lynch, D. Rus (A. K. Peters, Wellesley 2001) pp. 293–308

5.36 K.E. Bekris, B.Y. Chen, A. Ladd, E. Plaku, L.E. Kavraki: Multiple query probabilistic roadmap planning using single query primitives, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2003)

5.37 M. Strandberg: Augmenting RRT-planners with local trees, IEEE Int. Conf. Robot. Autom. (2004) pp. 3258–3262

5.38 J. J. Kuffner, S. M. LaValle: An efficient approach to path planning using balanced bidirectional RRT search, Techn. Rep. CMU-RI-TR-05-34 Robotics Institute, Carnegie Mellon University, Pittsburgh (2005)

5.39 J. Bruce, M. Veloso: Real-time randomized path planning for robot navigation, IEEE/RSJ Int. Conf. Intell. Robot. Autom. (2002)

5.40 E. Frazzoli, M.A. Dahleh, E. Feron: Real-time motion planning for agile autonomous vehicles, AIAA J. Guid. Contr. **25**(1), 116–129 (2002)

5.41 M. Kallmann, M. Mataric: Motion planning using dynamic roadmaps, IEEE Int. Conf. Robot. Autom. (2004)

5.42 A. Yershova, L. Jaillet, T. Simeon, S.M. LaValle: Dynamic-domain RRTs: efficient exploration by controlling the sampling domain, IEEE Int. Conf. Robot. Autom. (2005)

5.43 D. Hsu, J.C. Latombe, R. Motwani: Path planning in expansive configuration spaces, Int. J. Comput. Geom. Appl. **4**, 495–512 (1999)

5.44 D. Hsu, R. Kindel, J.C. Latombe, S. Rock: Randomized kinodynamic motion planning with moving obstacles. In: *Algorithmic and Computational Robotics: New Directions*, ed. by B.R. Donald, K.M. Lynch, D. Rus (A.K. Peters, Wellesley 2001)

5.45 G. Sánchez, J.-C. Latombe: A single-query bi-directional probabilistic roadmap planner with lazy collision checking, ISRR Int. Symp. Robot. Res. (2001)

5.46 S. Carpin, G. Pillonetto: Robot motion planning using adaptive random walks, IEEE Int. Conf. Robot. Autom. (2003) pp. 3809–3814

5.47 A. Ladd, L.E. Kavraki: Fast exploration for robots with dynamics, Workshop Algorithm. Found. Robot. (Zeist, Amsterdam 2004)

5.48 K.E. Bekris, L.E. Kavraki: Greedy but safe replanning under differential constraints, IEEE Int. Conf. Robot. Autom. (2007)

5.49 C. O'Dunlaing, C.K. Yap: A retraction method for planning the motion of a disc, J. Algorithms **6**, 104–111 (1982)

5.50 D. Leven, M. Sharir: Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams, Discrete Comput. Geom. **2**, 9–31 (1987)

5.51 M. Sharir: Algorithmic motion planning. In: *Handbook of Discrete and Computational Geometry*, 2nd edn., ed. by J. E. Goodman, J. O'Rourke (Chapman Hall/CRC Press, New York 2004) pp. 1037–1064

5.52 N.J. Nilsson: A mobile automaton: An application of artificial intelligence techniques, 1st Int. Conf. Artif. Intell. (1969) pp. 509–520

5.53 J. O'Rourke: Visibility. In: *Handbook of Discrete and Computational Geometry*, 2nd edn., ed. by J. E. Goodman, J. O'Rourke (Chapman Hall/CRC Press, New York 2004) pp. 643–663

5.54 B. Chazelle: Approximation and decomposition of shapes. In: *Algorithmic and Geometric Aspects of Robotics*, ed. by J.T. Schwartz, C.K. Yap (Lawrence Erlbaum, Hillsdale 1987) pp. 145–185

5.55 M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf: *Computational Geometry: Algorithms and Applications*, 2nd edn. (Springer, Berlin 2000)

5.56 J.M. Keil: Polygon decomposition. In: *Handbook on Computational Geometry*, ed. by J.R. Sack, J. Urrutia (Elsevier, New York 2000)

5.57 J.T. Schwartz, M. Sharir: On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, Commun. Pure Appl. Math. **36**, 345–398 (1983)

5.58 O. Khatib: Real-time obstacle avoidance for manipulators and mobile robots, Int. J. Robot. Res. **5**(1), 90–98 (1986)

5.59 J. Barraquand, J.-C. Latombe: Robot motion planning: A distributed representation approach, Int. J. Robot. Res. **10**(6), 628–649 (1991)

5.60 E. Rimon, D.E. Koditschek: Exact robot navigation using artificial potential fields, IEEE Trans. Robot. Autom. **8**(5), 501–518 (1992)

5.61 J.P. Laumond: Trajectories for mobile robots with kinematic and environment constraints, Int. Conf. Intell. Auton. Syst. (1986) pp. 346–354

5.62 J.P. Laumond, S. Sekhavat, F. Lamiraux: Guidelines in nonholonomic motion planning for mobile robots. In: *Robot Motion Planning and Control*, ed. by J.P. Laumond (Springer, Berlin 1998) pp. 1–53

5.63 B.R. Donald, P.G. Xavier, J. Canny, J. Reif: Kinodynamic planning, J. ACM **40**, 1048–1066 (1993)

5.64 C. O'Dunlaing: Motion planning with inertial constraints, Algorithmica **2**(4), 431–475 (1987)

5.65 J. Canny, A. Rege, J. Reif: An exact algorithm for kinodynamic planning in the plane, Discrete Comput. Geom. **6**, 461–484 (1991)

5.66 J. Go, T. Vu, J.J. Kuffner: Autonomous behaviors for interactive vehicle animations, SIGGRAPH Symp. Comput. Animat. (2004)

5.67 M. Pivtoraiko, A. Kelly: Generating near minimal spanning control sets for constrained motion planning in discrete state spaces, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2005)

5.68 J. Hollerbach: Dynamic scaling of manipulator trajectories, Tech. Rep. **700** (MIT A.I. Lab Memo, 1983)

5.69 K.G. Shin, N.D. McKay: Minimum-time control of robot manipulators with geometric path constraints, IEEE Trans. Autom. Contr. **30**(6), 531–541 (1985)

5.70 K.G. Shin, N.D. McKay: A dynamic programming approach to trajectory planning of robotic manipulators, IEEE Trans. Autom. Contr. **31**(6), 491–500 (1986)

5.71 S. Sastry: *Nonlinear Systems: Analysis, Stability, and Control* (Springer, Berlin 1999)

5.72 D.J. Balkcom, M.T. Mason: Time optimal trajectories for bounded velocity differential drive vehicles, Int. J. Robot. Res. **21**(3), 199–217 (2002)

5.73 P. Souères, J.-D. Boissonnat: Optimal trajectories for nonholonomic mobile robots. In: *Robot Motion Planning and Control*, ed. by J.P. Laumond (Springer, Berlin 1998) pp. 93–169

5.74 P. Svestka, M.H. Overmars: Coordinated motion planning for multiple car-like robots using probabilistic roadmaps, IEEE Int. Conf. Robot. Autom. (1995) pp. 1631–1636

5.75 S. Sekhavat, P. Svestka, J.-P. Laumond, M.H. Overmars: Multilevel path planning for nonholonomic robots using semiholonomic subsystems, Int. J. Robot. Res. **17**, 840–857 (1998)

5.76 P. Ferbach: A method of progressive constraints for nonholonomic motion planning, IEEE Int. Conf. Robot. Autom. (1996) pp. 2949–2955

5.77 S. Pancanti, L. Pallottino, D. Salvadorini, A. Bicchi: Motion planning through symbols and lattices, IEEE Int. Conf. Robot. Autom. (2004) pp. 3914–3919

5.78 J. Barraquand, J.-C. Latombe: Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles, Algorithmica **10**, 121–155 (1993)

5.79 S.M. LaValle, J.J. Kuffner: Randomized kinodynamic planning, IEEE Int. Conf. Robot. Autom. (1999) pp. 473–479

5.80 A. M. Ladd, L. E. Kavraki: Motion planning in the presence of drift underactuation and discrete system changes. In: *Robotics: Science and Systems I* ed. by (MIT Press, Boston 2005) pp. 233–241

5.81 J.-P. Merlet: *Parallel Robots* (Kluwer, Boston 2000)

5.82 D. Cox, J. Little, D. O'Shea: *Ideals, Varieties, and Algorithms* (Springer, Berlin 1992)

5.83 R.J. Milgram, J.C. Trinkle: The geometry of configuration spaces for closed chains in two and three dimensions, Homol. Homot. Appl. **6**(1), 237–267 (2004)

5.84 J. Yakey, S.M. LaValle, L.E. Kavraki: Randomized path planning for linkages with closed kinematic chains, IEEE Trans. Robot. Autom. **17**(6), 951–958 (2001)

5.85 L. Han, N.M. Amato: A kinematics-based probabilistic roadmap method for closed chain systems. In: *Algorithmic and Computational Robotics: New Directions*, ed. by B.R. Donald, K.M. Lynch, D. Rus (A.K. Peters, Wellesley 2001) pp. 233–246

5.86 J. Cortés: Motion Planning Algorithms for General Closed-Chain Mechanisms. Ph.D. Thesis (Institut National Polytechnique do Toulouse, Toulouse 2003)

5.87 R. Alami, J.-P. Laumond, T. Siméon: Two manipulation planning algorithms. In: *Algorithms for Robotic Motion and Manipulation*, ed. by J.P. Laumond, M. Overmars (A.K. Peters, Wellesley 1997)

5.88 L.E. Kavraki, M. Kolountzakis: Partitioning a planar assembly into two connected parts is NP-complete, Inform. Process. Lett. **55**(3), 159–165 (1995)

5.89 M.T. Mason: *Mechanics of Robotic Manipulation* (MIT Press, Cambridge 2001)

5.90 K. Sutner, W. Maass: Motion planning among time dependent obstacles, Acta Informatica **26**, 93–122 (1988)

5.91 J.H. Reif, M. Sharir: Motion planning in the presence of moving obstacles, J. ACM **41**, 764–790 (1994)

5.92 M.A. Erdmann, T. Lozano-Pérez: On multiple moving objects, Algorithmica **2**, 477–521 (1987)

5.93 J. van den Berg, M. Overmars: Prioritized motion planning for multiple robots, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (2005) pp. 2217–2222

5.94 T. Siméon, S. Leroy, J.-P. Laumond: Path coordination for multiple mobile robots: A resolution complete algorithm, IEEE Trans. Robot. Autom. **18**(1), 42–49 (2002)

5.95 R. Ghrist, J.M. O'Kane, S.M. LaValle: Pareto optimal coordination on roadmaps, Workshop Algorithm. Found. Robot. (2004) pp. 185–200

5.96 S.M. LaValle, S.A. Hutchinson: Optimal motion planning for multiple robots having independent goals, IEEE Trans. Robot. Autom. **14**(6), 912–925 (1998)

5.97 W.M. Boothby: *An Introduction to Differentiable Manifolds and Riemannian Geometry*, 2nd edn. (Academic, New York 2003)

5.98 A. Hatcher: *Algebraic Topology* (Cambridge Univ Press, Cambridge 2002)

5.99 G.S. Chirikjian, A.B. Kyatkin: *Engineering Applications of Noncommutative Harmonic Analysis* (CRC, Boca Raton 2001)

5.100 J. Arvo: Fast random rotation matrices. In: *Graphics Gems III*, ed. by D. Kirk (Academic, New York 1992) pp. 117–120

5.101 H. Niederreiter: *Random Number Generation and Quasi-Monte-Carlo Methods* (Society for Industrial and Applied Mathematics, Philadelphia 1992)

5.102 S. Basu, R. Pollack, M.-F. Roy: *Algorithms in Real Algebraic Geometry* (Springer, Berlin 2003)
5.103 B. Mishra: Computational real algebraic geometry. In: *Handbook of Discrete and Computational Geometry*, ed. by J.E. Goodman, J. O'Rourke (CRC, New York 1997) pp. 537–556
5.104 J.T. Schwartz, M. Sharir: On the piano movers' problem: II. General techniques for computing topological properties of algebraic manifolds, Commun. Pure Appl. Math. **36**, 345–398 (1983)
5.105 B.R. Donald: A search algorithm for motion planning with six degrees of freedom, Artif. Intell. J. **31**, 295–353 (1987)
5.106 D.S. Arnon: Geometric reasoning with logic and algebra, Artif. Intell. J. **37**(1-3), 37–60 (1988)
5.107 G.E. Collins: Quantifier elimination by cylindrical algebraic decomposition–twenty years of progress. In: *Quantifier Elimination and Cylindrical Algebraic Decomposition*, ed. by B.F. Caviness, J.R. Johnson (Springer, Berlin 1998) pp. 8–23