Chapter 7

Extensions of Basic Motion Planning

This chapter presents many extensions and variations of the motion planning problem considered in Chapters 3 to 6. Each one of these can be considered as a "spin-off" that is fairly straightforward to describe using the mathematical concepts and algorithms introduced so far. Unlike the previous chapters, there is not much continuity in Chapter 7. Each problem is treated independently; therefore, it is safe to jump to whatever sections in the chapter you find interesting without fear of missing important details.

In many places throughout the chapter, a state space X will arise. This is consistent with the general planning notation used throughout the book. In Chapter 4, the C-space, $\mathcal{C}$, was introduced, which can be considered as a special state space: It encodes the set of transformations that can be applied to a collection of bodies. Hence, Chapters 5 and 6 addressed planning in $X = \mathcal{C}$. The C-space alone is insufficient for many of the problems in this chapter; therefore, X will be used because it appears to be more general. For most cases in this chapter, however, X is derived from one or more C-spaces. Thus, C-space and state space terminology will be used in combination.

7.1 Time-Varying Problems

This section brings time into the motion planning formulation. Although the robot has been allowed to move, it has been assumed so far that the obstacle region $\mathcal{O}$ and the goal configuration, $q_G \in \mathcal{C}_{free}$, are stationary for all time. It is now assumed that these entities may vary over time, although their motions are predictable. If the motions are not predictable, then some form of feedback is needed to respond to observations that are made during execution. Such problems are much more difficult and will be handled in Chapters 8 and throughout Part IV.

7.1.1 Problem Formulation

The formulation is designed to allow the tools and concepts learned so far to be applied directly. Let $T \subset \mathbb{R}$ denote the *time interval*, which may be *bounded* or *unbounded*. If T is bounded, then $T = [0, t_f]$, in which 0 is the *initial time* and t_f is the *final time*. If T is unbounded, then $T = [0, \infty)$. An initial time other than 0 could alternatively be defined without difficulty, but this will not be done here.

Let the state space X be defined as $X = \mathcal{C} \times T$, in which $\mathcal{C}$ is the usual C-space of the robot, as defined in Chapter 4. A state x is represented as $x = (q, t)$, to indicate the configuration q and time t components of the state vector. The planning will occur directly in X, and in many ways it can be treated as any C-space seen to far, but there is one critical difference: *Time marches forward*. Imagine a path that travels through X. If it first reaches a state $(q_1, 5)$, and then later some state $(q_2, 3)$, some traveling backward through time is required! There is no mathematical problem with allowing such time travel, but it is not realistic for most applications. Therefore, paths in X are forced to follow a constraint that they must move forward in time.

Now consider making the following time-varying versions of the items used in Formulation 4.1 for motion planning.

Formulation 7.1 (The Time-Varying Motion Planning Problem)

1. A *world* $\mathcal{W}$ in which either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. This is the same as in Formulation 4.1.

2. A *time interval* $T \subset \mathbb{R}$ that is either *bounded* to yield $T = [0, t_f]$ for some *final time*, $t_f > 0$, or *unbounded* to yield $T = [0, \infty)$.

3. A semi-algebraic, time-varying *obstacle region* $\mathcal{O}(t) \subset \mathcal{W}$ for every $t \in T$. It is assumed that the obstacle region is a finite collection of rigid bodies that undergoes continuous, time-dependent rigid-body transformations.

4. The *robot* $\mathcal{A}$ (or $\mathcal{A}_1$, ..., $\mathcal{A}_m$ for a linkage) and *configuration space* $\mathcal{C}$ definitions are the same as in Formulation 4.1.

5. The *state space* X is the Cartesian product $X = \mathcal{C} \times T$ and a state $x \in X$ is denoted as $x = (q, t)$ to denote the configuration q and time t components. See Figure 7.1. The obstacle region, X_{obs}, in the state space is defined as

$$X_{obs} = \{(q, t) \in X \mid \mathcal{A}(q) \cap \mathcal{O}(t) \neq \emptyset\}, \tag{7.1}$$

   and $X_{free} = X \setminus X_{obs}$. For a given $t \in T$, slices of X_{obs} and X_{free} are obtained. These are denoted as $\mathcal{C}_{obs}(t)$ and $\mathcal{C}_{free}(t)$, respectively, in which (assuming $\mathcal{A}$ is one body)

$$\mathcal{C}_{obs}(t) = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O}(t) \neq \emptyset\} \tag{7.2}$$

   and $\mathcal{C}_{free} = \mathcal{C} \setminus \mathcal{C}_{obs}$.

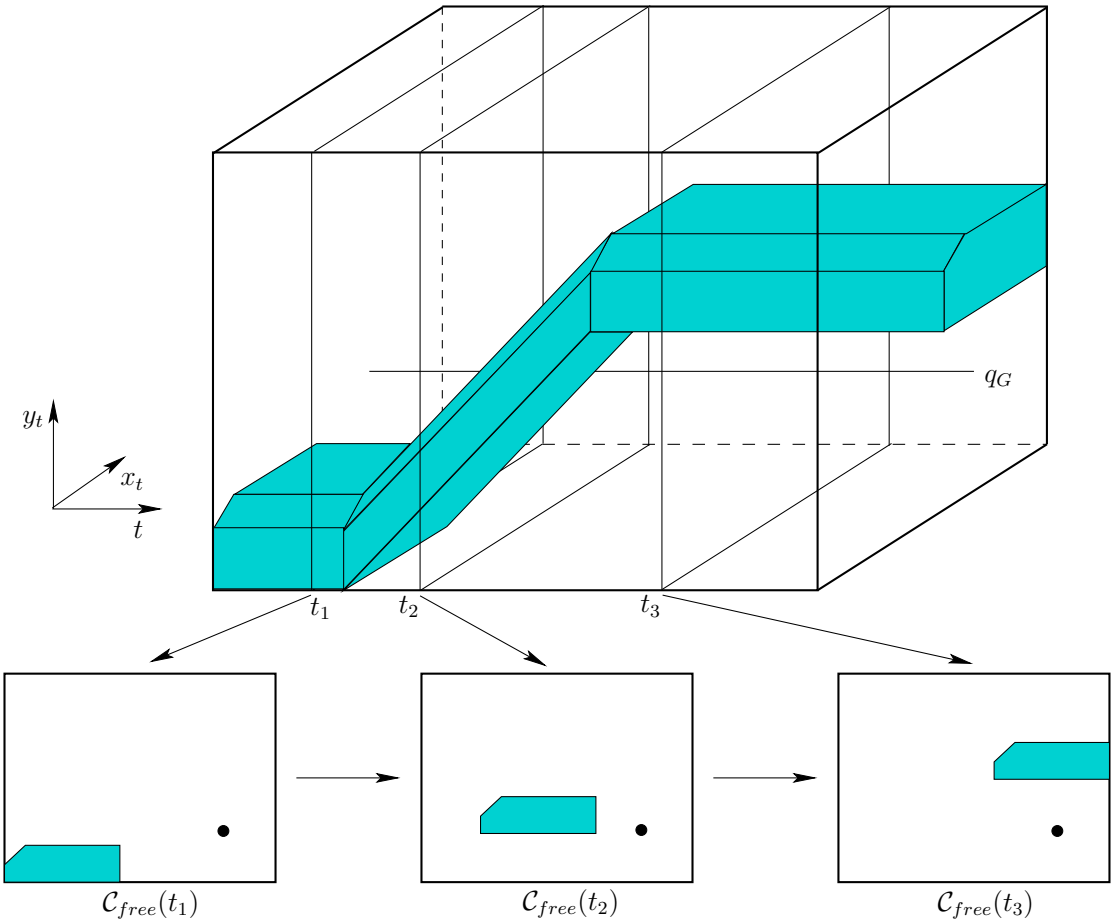


Figure 7.1: A time-varying example with piecewise-linear obstacle motion.

6. A state $x_I \in X_{free}$ is designated as the *initial state*, with the constraint that $x_I = (q_I, 0)$ for some $q_I \in \mathcal{C}_{free}(0)$. In other words, at the initial time the robot cannot be in collision.

7. A subset $X_G \subset X_{free}$ is designated as the *goal region*. A typical definition is to pick some $q_G \in \mathcal{C}$ and let $X_G = \{(q_G, t) \in X_{free} \mid t \in T\}$, which means that the goal is *stationary* for all time.

8. A complete algorithm must compute a continuous, time-monotonic *path*, $\tau[0,1] \rightarrow X_{free}$, such that $\tau(0) = x_I$ and $\tau(1) \in X_G$, or correctly report that such a path does not exist. To be *time-monotonic* implies that for any $s_1, s_2 \in [0,1]$ such that $s_1 < s_2$, we have $t_1 < t_2$, in which $(q_1, t_1) = \tau(s_1)$ and $(q_2, t_2) = \tau(s_2)$.

Example 7.1 (Piecewise-Linear Obstacle Motion) Figure 7.1 shows an example of a convex, polygonal robot $\mathcal{A}$ that translates in $\mathcal{W} = \mathbb{R}^2$. There is a single, convex, polygonal obstacle $\mathcal{O}$. The two of these together yield a convex, polygonal

C-space obstacle, $\mathcal{C}_{obs}(t)$, which is shown for times t_1, t_2, and t_3. The obstacle moves with a *piecewise-linear motion model*, which means that transformations applied to $\mathcal{O}$ are a piecewise-linear function of time. For example, let (x, y) be a fixed point on the obstacle. To be a linear motion model, this point must transform as $(x + c_1 t, y + c_2 t)$ for some constants $c_1, c_2 \in \mathbb{R}$. To be piecewise-linear, it may change to a different linear motion at a finite number of critical times. Between these critical times, the motion must remain linear. There are two critical times in the example. If $\mathcal{C}_{obs}(t)$ is polygonal, and a piecewise-linear motion model is used, then X_{obs} is polyhedral, as depicted in Figure 7.1. A stationary goal is also shown, which appears as a line that is parallel to the T-axis. ■

In the general formulation, there are no additional constraints on the path, τ, which means that the robot motion model allows infinite acceleration and unbounded speed. The robot velocity may change instantaneously, but the path through $\mathcal{C}$ must always be continuous. These issues did not arise in Chapter 4 because there was no need to mention time. Now it becomes necessary.[1]

7.1.2 Direct Solutions

Sampling-based methods Many sampling-based methods can be adapted from $\mathcal{C}$ to X without much difficulty. The time dependency of obstacle models must be taken into account when verifying that path segments are collision-free; the techniques from Section 5.3.4 can be extended to handle this. One important concern is the metric for X. For some algorithms, it may be important to permit the use of a pseudometric because symmetry is broken by time (going backward in time is not as easy as going forward).

For example, suppose that the C-space $\mathcal{C}$ is a metric space, $(\mathcal{C}, \rho)$. The metric can be extended across time to obtain a pseudometric, ρ_X, as follows. For a pair of states, $x = (q, t)$ and $x' = (q', t')$, let

$$\rho_X(x, x') = \begin{cases} 0 & \text{if } q = q' \\ \infty & \text{if } q \neq q' \text{ and } t' \leq t \\ \rho(q, q') & \text{otherwise.} \end{cases} \tag{7.3}$$

Using ρ_X, several sampling-based methods naturally work. For example, RDTs from Section 5.5 can be adapted to X. Using ρ_X for a single-tree approach ensures that all path segments travel forward in time. Using bidirectional approaches is more difficult for time-varying problems because X_G is usually not a single point. It is not clear which (q, t) should be the starting vertex for the tree from

[1]The infinite acceleration and unbounded speed assumptions may annoy those with mechanics and control backgrounds. In this case, assume that the present models approximate the case in which every body moves slowly, and the dynamics can be consequently neglected. If this is still not satisfying, then jump ahead to Part IV, where general nonlinear systems are considered. It is still helpful to consider the implications derived from the concepts in this chapter because the issues remain for more complicated problems that involve dynamics.

the goal; one possibility is to initialize the goal tree to an entire time-invariant segment. The sampling-based roadmap methods of Section 5.6 are perhaps the most straightforward to adapt. The notion of a *directed roadmap* is needed, in which every edge must be directed to yield a time-monotonic path. For each pair of states, (q, t) and (q', t'), such that $t \neq t'$, exactly one valid direction exists for making a potential edge. If $t = t'$, then no edge can be attempted because it would require the robot to instantaneously "teleport" from one part of $\mathcal{W}$ to another. Since forward time progress is already taken into account by the directed edges, a symmetric metric may be preferable instead of (7.3) for the sampling-based roadmap approach.

Combinatorial methods In some cases, combinatorial methods can be used to solve time-varying problems. If the motion model is *algebraic* (i.e., expressed with polynomials), then X_{obs} is semi-algebraic. This enables the application of general planners from Section 6.4, which are based on computational real algebraic geometry. The key issue once again is that the resulting roadmap must be directed with all edges being time-monotonic. For Canny's roadmap algorithm, this requirement seems difficult to ensure. Cylindrical algebraic decomposition is straightforward to adapt, provided that time is chosen as the last variable to be considered in the sequence of projections. This yields polynomials in $\mathbb{Q}[t]$, and $\mathbb{R}$ is nicely partitioned into time intervals and time instances. Connections can then be made for a cell of one cylinder to an adjacent cell of a cylinder that occurs later in time.

If X_{obs} is polyhedral as depicted in Figure 7.1, then vertical decomposition can be used. It is best to first sweep the plane along the time axis, stopping at the critical times when the linear motion changes. This yields nice sections, which are further decomposed recursively, as explained in Section 6.3.3, and also facilitates the connection of adjacent cells to obtain time-monotonic path segments. It is not too difficult to imagine the approach working for a four-dimensional state space, X, for which $\mathcal{C}_{obs}(t)$ is polyhedral as in Section 6.3.3, and time adds the fourth dimension. Again, performing the first sweep with respect to the time axis is preferable.

If X is not decomposed into cylindrical slices over each noncritical time interval, then cell decompositions may still be used, but be careful to correctly connect the cells. Figure 7.2 illustrates the problem, for which transitivity among adjacent cells is broken. This complicates sample point selection for the cells.

Bounded speed There has been no consideration so far of the speed at which the robot must move to avoid obstacles. It is obviously impractical in many applications if the solution requires the robot to move arbitrarily fast. One step toward making a realistic model is to enforce a bound on the speed of the robot. (More steps towards realism are taken in Chapter 13.) For simplicity, suppose $\mathcal{C} = \mathbb{R}^2$, which corresponds to a translating rigid robot, $\mathcal{A}$, that moves in $\mathcal{W} = \mathbb{R}^2$. A configuration, $q \in \mathcal{C}$, is represented as $q = (y, z)$ (since x already refers to the

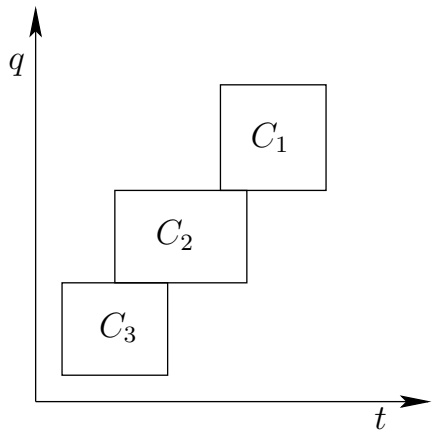


Figure 7.2: Transitivity is broken if the cells are not formed in cylinders over T. A time-monotonic path exists from C_1 to C_2, and from C_2 to C_3, but this does not imply that one exists from C_1 to C_3.

whole state vector). The *robot velocity* is expressed as $v = (\dot{y}, \dot{z}) \in \mathbb{R}^2$, in which $\dot{y} = dy/dt$ and $\dot{z} = dz/dt$. The *robot speed* is $\|v\| = \sqrt{\dot{y}^2 + \dot{z}^2}$. A *speed bound*, b, is a positive constant, $b \in (0, \infty)$, for which $\|v\| \leq b$.

In terms of Figure 7.1, this means that the slope of a solution path τ is bounded. Suppose that the domain of τ is $T = [0, t_f]$ instead of $[0, 1]$. This yields $\tau : T \rightarrow X$ and $\tau(t) = (y, z, t)$. Using this representation, $d\tau_1/dt = \dot{y}$ and $d\tau_2/dt = \dot{z}$, in which τ_i denotes the ith component of τ (because it is a vector-valued function). Thus, it can seen that b constrains the slope of $\tau(t)$ in X. To visualize this, imagine that only motion in the y direction occurs, and suppose $b = 1$. If τ holds the robot fixed, then the speed is zero, which satisfies any bound. If the robot moves at speed 1, then $d\tau_1/dt = 1$ and $d\tau_2/dt = 0$, which satisfies the speed bound. In Figure 7.1 this generates a path that has slope 1 in the yt plane and is horizontal in the zt plane. If $d\tau_1/dt = d\tau_2/dt = 1$, then the bound is exceeded because the speed is $\sqrt{2}$. In general, the velocity vector at any state (y, z, t) points into a cone that starts at (y, z) and is aligned in the positive t direction; this is depicted in Figure 7.3. At time $t + \Delta t$, the state must stay within the cone, which means that

$$\big(y(t + \Delta t) - y(t)\big)^2 + \big(z(t + \Delta t) - z(t)\big)^2 \leq b^2(\Delta t)^2. \tag{7.4}$$

This constraint makes it considerably more difficult to adapt the algorithms of Chapters 5 and 6. Even for piecewise-linear motions of the obstacles, the problem has been established to be PSPACE-hard [818, 819, 928]. A complete algorithm is presented in [819] that is similar to the shortest-path roadmap algorithm of Section 6.2.4. The sampling-based roadmap of Section 5.6 is perhaps one of the easiest of the sampling-based algorithms to adapt for this problem. The neighbors of point q, which are determined for attempted connections, must lie within the cone that represents the speed bound. If this constraint is enforced, a resolution complete or probabilistically complete planning algorithm results.

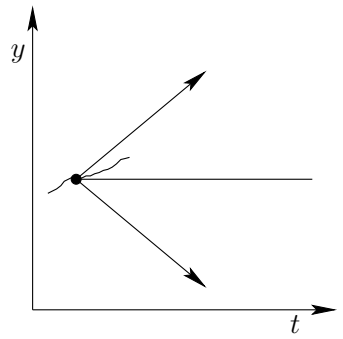


Figure 7.3: A projection of the cone constraint for the bounded-speed problem.

7.1.3 The Velocity-Tuning Method

An alternative to defining the problem in $\mathcal{C} \times T$ is to decouple it into a *path planning* part and a *motion timing* part [506]. Algorithms based on this method are not complete, but velocity tuning is an important idea that can be applied elsewhere. Suppose there are both *stationary obstacles* and *moving obstacles*. For the stationary obstacles, suppose that some path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ has been computed using any of the techniques described in Chapters 5 and 6.

The timing part is then handled in a second phase. Design a *timing function* (or *time scaling*), $\sigma : T \rightarrow [0,1]$, that indicates for time, t, the location of the robot along the path, τ. This is achieved by defining the composition $\phi = \tau \circ \sigma$, which maps from T to $\mathcal{C}_{free}$ via $[0,1]$. Thus, $\phi : T \rightarrow \mathcal{C}_{free}$. The configuration at time $t \in T$ is expressed as $\phi(t) = \tau(\sigma(t))$.

A 2D state space can be defined as shown in Figure 7.4. The purpose is to convert the design of σ (and consequently ϕ) into a familiar planning problem. The robot must move along its path from $\tau(0)$ to $\tau(1)$ while an obstacle, $\mathcal{O}(t)$, moves along its path over the time interval T. Let $S = [0,1]$ denote the domain of τ. A state space, $X = T \times S$, is shown in Figure 7.4b, in which each point (t, s) indicates the time $t \in T$ and the position along the path, $s \in [0,1]$. The obstacle region in X is defined as

$$X_{obs} = \{(t,s) \in X \mid \mathcal{A}(\tau(s)) \cap \mathcal{O}(t) \neq \emptyset\}. \tag{7.5}$$

Once again, X_{free} is defined as $X_{free} = X \setminus X_{obs}$. The task is to find a continuous path $g : [0,1] \rightarrow X_{free}$. If g is time-monotonic, then a position $s \in S$ is assigned for every time, $t \in T$. These assignments can be nicely organized into the timing function, $\sigma : T \rightarrow S$, from which ϕ is obtained by $\phi = \tau \circ \sigma$ to determine where the robot will be at each time. Being time-monotonic in this context means that the path must always progress from left to right in Figure 7.4b. It can, however, be nonmonotonic in the positive s direction. This corresponds to moving back and forth along τ, causing some configurations to be revisited.

Any of the methods described in Formulation 7.1 can be applied here. The dimension of X in this case is always 2. Note that X_{obs} is polygonal if $\mathcal{A}$ and $\mathcal{O}$ are both polygonal regions and their paths are piecewise-linear. In this case, the

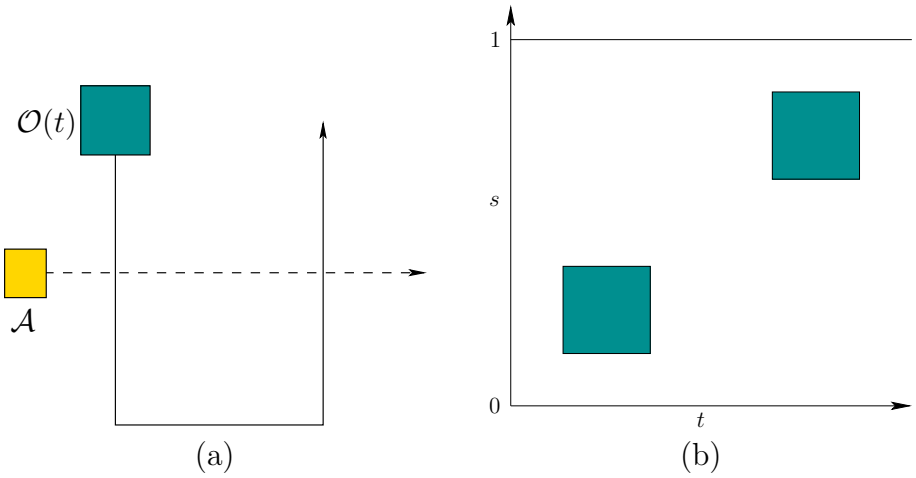


Figure 7.4: An illustration of path tuning. (a) If the robot follows its computed path, it may collide with the moving obstacle. (b) The resulting state space.

vertical decomposition method of Section 6.2.2 can be applied by sweeping along the time axis to yield a complete algorithm (it is complete after having committed to τ, but it is not complete for Formulation 7.1). The result is shown in Figure 7.5. The cells are connected only if it is possible to reach one from the other by traveling in the forward time direction. As an example of a sampling-based approach that may be preferable when X_{obs} is not polygonal, place a grid over X and apply one of the classical search algorithms described in Section 5.4.2. Once again, only path segments in X that move forward in time are allowed.

7.2 Multiple Robots

Suppose that multiple robots share the same world, $\mathcal{W}$. A path must be computed for each robot that avoids collisions with obstacles and with other robots. In Chapter 4, each robot could be a rigid body, $\mathcal{A}$, or it could be made of k attached bodies, $\mathcal{A}_1$, ..., $\mathcal{A}_k$. To avoid confusion, superscripts will be used in this section to denote different robots. The ith robot will be denoted by $\mathcal{A}^i$. Suppose there are m robots, $\mathcal{A}^1$, $\mathcal{A}^2$, ..., $\mathcal{A}^m$. Each robot, $\mathcal{A}^i$, has its associated C-space, $\mathcal{C}^i$, and its initial and goal configurations, q^i_{init} and q^i_{goal}, respectively.

7.2.1 Problem Formulation

A state space is defined that considers the configurations of all robots simultaneously,

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m. \tag{7.6}$$

A state $x \in X$ specifies all robot configurations and may be expressed as $x = (q^1, q^2, \ldots, q^m)$. The dimension of X is N, which is $N = \sum_{i=1}^{m} \dim(\mathcal{C}^i)$.

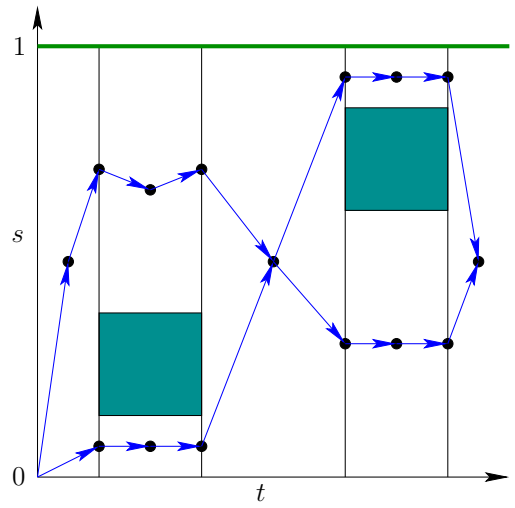


Figure 7.5: Vertical cell decomposition can solve the path tuning problem. Note that this example is not in general position because vertical edges exist. The goal is to reach the horizontal line at the top, which can be accomplished from any adjacent 2-cell. For this example, it may even be accomplished from the first 2-cell if the robot is able to move quickly enough.

There are two sources of obstacle regions in the state space: 1) *robot-obstacle* collisions, and 2) *robot-robot* collisions. For each i such that $1 \leq i \leq m$, the subset of X that corresponds to robot $\mathcal{A}^i$ in collision with the obstacle region, $\mathcal{O}$, is

$$X^i_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{O} \neq \emptyset\}. \tag{7.7}$$

This only models the robot-obstacle collisions.

For each pair, $\mathcal{A}^i$ and $\mathcal{A}^j$, of robots, the subset of X that corresponds to $\mathcal{A}^i$ in collision with $\mathcal{A}^j$ is

$$X^{ij}_{obs} = \{x \in X \mid \mathcal{A}^i(q^i) \cap \mathcal{A}^j(q^j) \neq \emptyset\}. \tag{7.8}$$

Both (7.7) and (7.8) will be combined in (7.10) later to yield X_{obs}.

Formulation 7.2 (Multiple-Robot Motion Planning)

1. The *world* $\mathcal{W}$ and *obstacle region* $\mathcal{O}$ are the same as in Formulation 4.1.

2. There are m *robots*, $\mathcal{A}^1$, ..., $\mathcal{A}^m$, each of which may consist of one or more bodies.

3. Each robot $\mathcal{A}^i$, for i from 1 to m, has an associated *configuration space*, $\mathcal{C}^i$.

4. The *state space* X is defined as the Cartesian product

$$X = \mathcal{C}^1 \times \mathcal{C}^2 \times \cdots \times \mathcal{C}^m. \tag{7.9}$$

The obstacle region in X is

$$X_{obs} = \left(\bigcup_{i=1}^{m} X_{obs}^i \right) \bigcup \left(\bigcup_{ij,\ i \neq j} X_{obs}^{ij} \right), \tag{7.10}$$

in which X_{obs}^i and X_{obs}^{ij} are the robot-obstacle and robot-robot collision states from (7.7) and (7.8), respectively.

5. A state $x_I \in X_{free}$ is designated as the *initial state*, in which $x_I = (q_I^1, \ldots, q_I^m)$. For each i such that $1 \leq i \leq m$, q_I^i specifies the initial configuration of $\mathcal{A}^i$.

6. A state $x_G \in X_{free}$ is designated as the *goal state*, in which $x_G = (q_G^1, \ldots, q_G^m)$.

7. The task is to compute a continuous path $\tau : [0, 1] \rightarrow X_{free}$ such that $\tau(0) = x_{init}$ and $\tau(1) \in x_{goal}$.

An ordinary motion planning problem? On the surface it may appear that there is nothing unusual about the multiple-robot problem because the formulations used in Chapter 4 already cover the case in which the robot consists of multiple bodies. They do not have to be attached; therefore, X can be considered as an ordinary C-space. The planning algorithms of Chapters 5 and 6 may be applied without adaptation. The main concern, however, is that the dimension of X grows linearly with respect to the number of robots. For example, if there are 12 rigid bodies for which each has $\mathcal{C}^i = SE(3)$, then the dimension of X is $6 \cdot 12 = 72$. Complete algorithms require time that is at least exponential in dimension, which makes them unlikely candidates for such problems. Sampling-based algorithms are more likely to scale well in practice when there many robots, but the dimension of X might still be too high.

Reasons to study multi-robot motion planning Even though multiple-robot motion planning can be handled like any other motion planning problem, there are several reasons to study it separately:

1. The motions of the robots can be decoupled in many interesting ways. This leads to several interesting methods that first develop some kind of partial plan for the robots independently, and then consider the plan interactions to produce a solution. This idea is referred to as *decoupled planning*.

2. The part of X_{obs} due to robot-robot collisions has a cylindrical structure, depicted in Figure 7.6, which can be exploited to make more efficient planning algorithms. Each X_{obs}^{ij} defined by (7.8) depends only on two robots. A point, $x = (q^1, \ldots, q^m)$, is in X_{obs} if there exists i, j such that $1 \leq i, j \leq m$ and $\mathcal{A}^i(q^i) \cap \mathcal{A}^j(q^j) \neq \emptyset$, regardless of the configurations of the other $m - 2$ robots. For some decoupled methods, this even implies that X_{obs} can be completely characterized by 2D projections, as depicted in Figure 7.9.

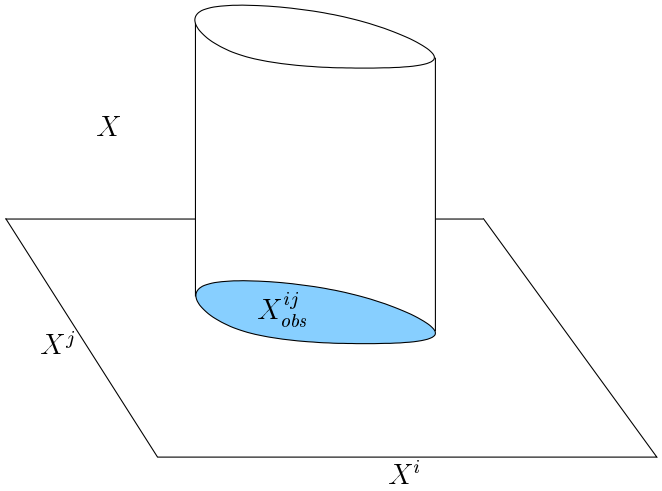


Figure 7.6: The set X_{obs}^{ij} and its cylindrical structure on X.

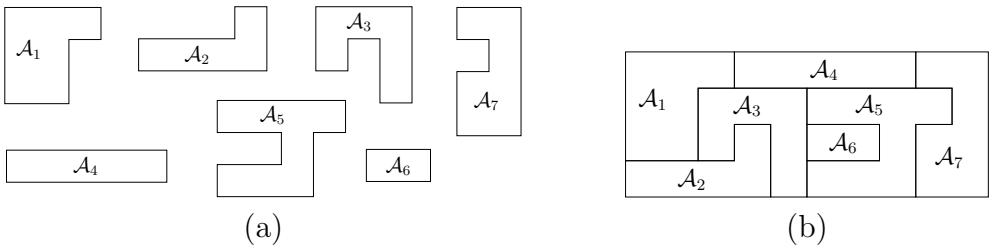


Figure 7.7: (a) A collection of pieces used to define an assembly planning problem; (b) assembly planning involves determining a sequence of motions that assembles the parts. The object shown here is assembled from the parts.

3. If optimality is important, then a unique set of issues arises for the case of multiple robots. It is not a standard optimization problem because the performance of each robot has to be optimized. There is no clear way to combine these objectives into a single optimization problem without losing some critical information. It will be explained in Section 7.7.2 that Pareto optimality naturally arises as the appropriate notion of optimality for multiple-robot motion planning.

Assembly planning One important variant of multiple-robot motion planning is called *assembly planning*; recall from Section 1.2 its importance in applications. In automated manufacturing, many complicated objects are assembled step-by-step from individual parts. It is convenient for robots to manipulate the parts one-by-one to insert them into the proper locations (see Section 7.3.2). Imagine a collection of parts, each of which is interpreted as a robot, as shown in Figure 7.7a. The goal is to assemble the parts into one coherent object, such as that shown in Figure 7.7b. The problem is generally approached by starting with the goal

configuration, which is tightly constrained, and working outward. The problem formulation may allow that the parts touch, but their interiors cannot overlap. In general, the assembly planning problem with arbitrarily many parts is NP-hard. Polynomial-time algorithms have been developed in several special cases. For the case in which parts can be removed by a sequence of straight-line paths, a polynomial-time algorithm is given in [973, 974].

7.2.2 Decoupled planning

Decoupled approaches first design motions for the robots while ignoring robot-robot interactions. Once these interactions are considered, the choices available to each robot are already constrained by the designed motions. If a problem arises, these approaches are typically unable to reverse their commitments. Therefore, completeness is lost. Nevertheless, decoupled approaches are quite practical, and in some cases completeness can be recovered.

Prioritized planning A straightforward approach to decoupled planning is to sort the robots by priority and plan for higher priority robots first [320, 951]. Lower priority robots plan by viewing the higher priority robots as moving obstacles. Suppose the robots are sorted as $\mathcal{A}^1$, ..., $\mathcal{A}^m$, in which $\mathcal{A}^1$ has the highest priority.

Assume that collision-free paths, $\tau_i : [0,1] \rightarrow \mathcal{C}^i_{free}$, have been computed for i from 1 to n. The prioritized planning approach proceeds inductively as follows:

> **Base case:** Use any motion planning algorithm from Chapters 5 and 6 to compute a collision-free path, $\tau_1 : [0,1] \rightarrow \mathcal{C}^1_{free}$ for $\mathcal{A}^1$. Compute a timing function, σ_1, for τ_1, to yield $\phi_1 = \tau_1 \circ \sigma_1 : T \rightarrow \mathcal{C}^1_{free}$.
>
> **Inductive step:** Suppose that ϕ_1, ..., ϕ_{i-1} have been designed for $\mathcal{A}^1$, ..., $\mathcal{A}^{i-1}$, and that these functions avoid robot-robot collisions between any of the first $i-1$ robots. Formulate the first $i-1$ robots as moving obstacles in $\mathcal{W}$. For each $t \in T$ and $j \in \{1, \ldots, i-1\}$, the configuration q^j of each $\mathcal{A}^j$ is $\phi_j(t)$. This yields $\mathcal{A}^j(\phi_j(t)) \subset \mathcal{W}$, which can be considered as a subset of the obstacle $\mathcal{O}(t)$. Design a path, τ_i, and timing function, σ_i, using any of the time-varying motion planning methods from Section 7.1 and form $\phi_i = \tau_i \circ \sigma_i$.

Although practical in many circumstances, Figure 7.8 illustrates how completeness is lost.

A special case of prioritized planning is to design all of the paths, τ_1, τ_2, ..., τ_m, in the first phase and then formulate each inductive step as a velocity tuning problem. This yields a sequence of 2D planning problems that can be solved easily. This comes at a greater expense, however, because the choices are even more constrained. The idea of preplanned paths, and even roadmaps, for all robots independently can lead to a powerful method if the coordination of the robots is approached more carefully. This is the next topic.

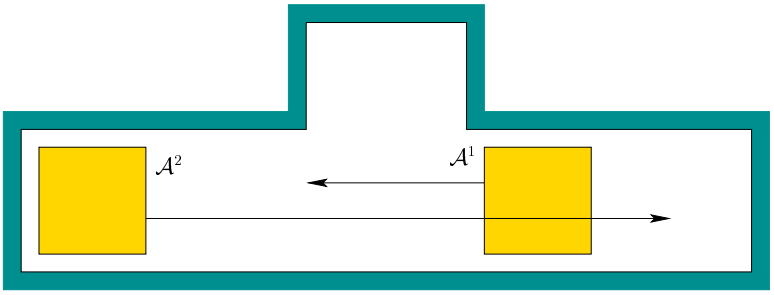


Figure 7.8: If $\mathcal{A}^1$ neglects the query for $\mathcal{A}^2$, then completeness is lost when using the prioritized planning approach. This example has a solution in general, but prioritized planning fails to find it.

Fixed-path coordination Suppose that each robot $\mathcal{A}^i$ is constrained to follow a path $\tau_i : [0,1] \rightarrow \mathcal{C}^i_{free}$, which can be computed using any ordinary motion planning technique. For m robots, an m-dimensional state space called a *coordination space* is defined that schedules the motions of the robots along their paths so that they will not collide [746]. One important feature is that time will only be *implicitly* represented in the coordination space. An algorithm must compute a path in the coordination space, from which explicit timings can be easily extracted.

For m robots, the *coordination space* X is defined as the m-dimensional unit cube $X = [0,1]^m$. Figure 7.9 depicts an example for which $m = 3$. The ith coordinate of X represents the domain, $S_i = [0,1]$, of the path τ_i. Let s_i denote a point in S_i (it is also the ith component of x). A state, $x \in X$, indicates the configuration of every robot. For each i, the configuration $q^i \in \mathcal{C}^i$ is given by $q^i = \tau_i(s_i)$. At state $(0, \ldots, 0) \in X$, every robot is in its initial configuration, $q^i_I = \tau_i(0)$, and at state $(1, \ldots, 1) \in X$, every robot is in its goal configuration, $q^i_G = \tau_i(1)$. Any continuous path, $h : [0,1] \rightarrow X$, for which $h(0) = (0, \ldots, 0)$ and $h(1) = (1, \ldots, 1)$, moves the robots to their goal configurations. The path h does not even need to be monotonic, in contrast to prioritized planning.

One important concern has been neglected so far. What prevents us from designing h as a straight-line path between the opposite corners of $[0,1]^m$? We have not yet taken into account the collisions between the robots. This forms an obstacle region X_{obs} that must be avoided when designing a path through X. Thus, the task is to design $h : [0,1] \rightarrow X_{free}$, in which $X_{free} = X \setminus X_{obs}$.

The definition of X_{obs} is very similar to (7.8) and (7.10), except that here the state-space dimension is much smaller. Each q^i is replaced by a single parameter. The cylindrical structure, however, is still retained, as shown in Figure 7.9. Each cylinder of X_{obs} is

$$X^{ij}_{obs} = \{(s_1, \ldots, s_m) \in X \mid \mathcal{A}^i(\tau_i(s_i)) \cap \mathcal{A}^j(\tau_j(s_j)) \neq \emptyset\}, \tag{7.11}$$

which are combined to yield

$$X_{obs} = \bigcup_{ij,\ i \neq j} X^{ij}_{obs}. \tag{7.12}$$

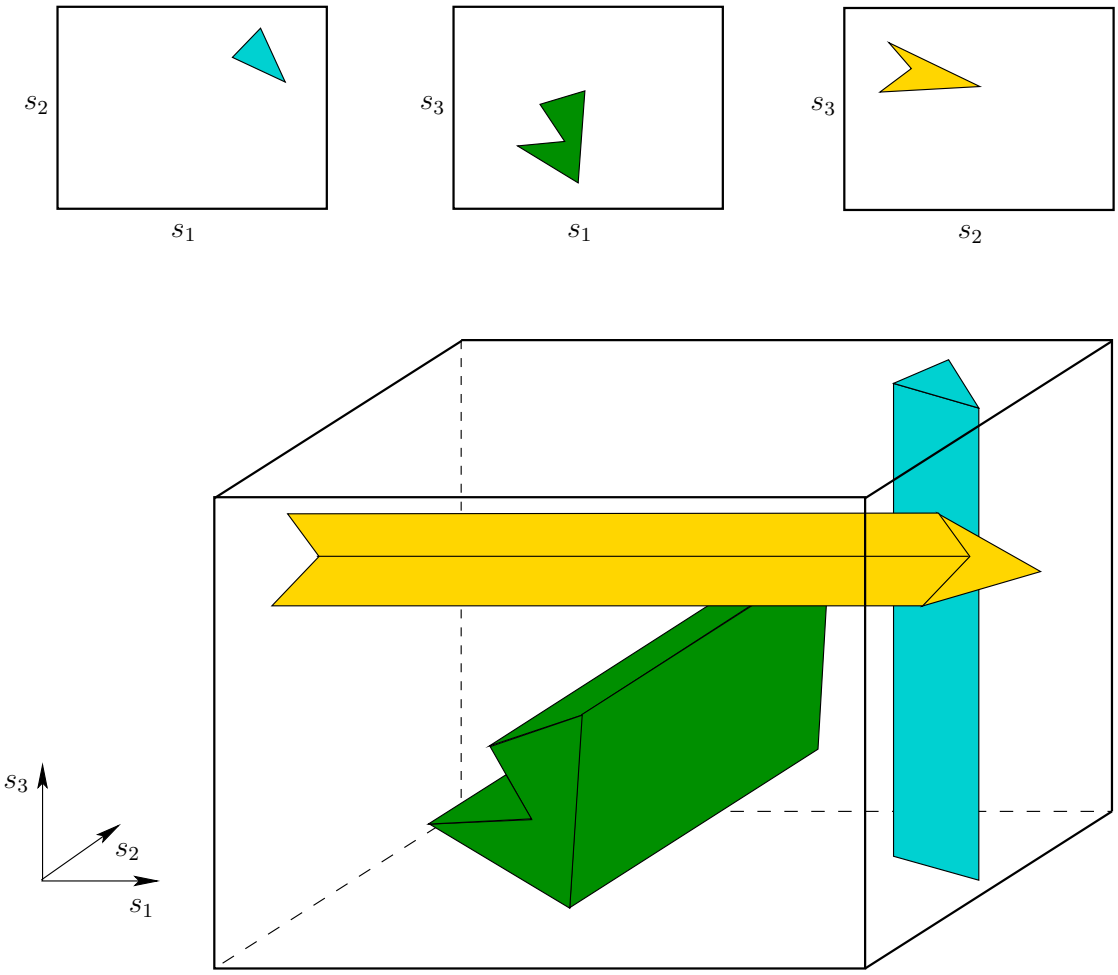


Figure 7.9: The obstacles that arise from coordinating m robots are always cylindrical. The set of all $\frac{1}{2}m(m-1)$ axis-aligned 2D projections completely characterizes X_{obs}.

Standard motion planning algorithms can be applied to the coordination space because there is no monotonicity requirement on h. If 1) $\mathcal{W} = \mathbb{R}^2$, 2) $m = 2$ (two robots), 3) the obstacles and robots are polygonal, and 4) the paths, τ_i, are piecewise-linear, then X_{obs} is a polygonal region in X. This enables the methods of Section 6.2, for a polygonal $\mathcal{C}_{obs}$, to directly apply after the representation of X_{obs} is explicitly constructed. For $m > 2$, the multi-dimensional version of vertical cell decomposition given for $m = 3$ in Section 6.3.3 can be applied. For general coordination problems, cylindrical algebraic decomposition or Canny's roadmap algorithm can be applied. For the problem of robots in $\mathcal{W} = \mathbb{R}^2$ that either translate or move along circular paths, a resolution complete planning method based on the exact determination of X_{obs} using special collision detection methods is given in [886].

For very challenging coordination problems, sampling-based solutions may

yield practical solutions. Perhaps one of the simplest solutions is to place a grid over X and adapt the classical search algorithms, as described in Section 5.4.2 [606, 746]. Other possibilities include using the RDTs of Section 5.5 or, if the multiple-query framework is appropriate, then the sampling-based roadmap methods of 5.6 are suitable. Methods for validating the path segments, which were covered in Section 5.3.4, can be adapted without trouble to the case of coordination spaces.

Thus far, the particular speeds of the robots have been neglected. For explanation purposes, consider the case of $m = 2$. Moving vertically or horizontally in X holds one robot fixed while the other moves at some maximum speed. Moving diagonally in X moves both robots, and their relative speeds depend on the slope of the path. To carefully regulate these speeds, it may be necessary to reparameterize the paths by distance. In this case each axis of X represents the distance traveled, instead of $[0, 1]$.

Fixed-roadmap coordination The fixed-path coordination approach still may not solve the problem in Figure 7.8 if the paths are designed independently. Fortunately, fixed-path coordination can be extended to enable each robot to move over a roadmap or topological graph. This still yields a coordination space that has only one dimension per robot, and the resulting planning methods are much closer to being complete, assuming each robot utilizes a roadmap that has many alternative paths. There is also motivation to study this problem by itself because of automated guided vehicles (AGVs), which often move in factories on a network of predetermined paths. In this case, coordinating the robots *is* the planning problem, as opposed to being a simplification of Formulation 7.2.

One way to obtain completeness for Formulation 7.2 is to design the independent roadmaps so that each robot has its own *garage* configuration. The conditions for a configuration, q^i, to be a *garage* for $\mathcal{A}^i$ are 1) while at configuration q^i, it is impossible for any other robots to collide with it (i.e., in all coordination states for which the ith coordinate is q^i, no collision occurs); and 2) q^i is always reachable by $\mathcal{A}^i$ from x_I. If each robot has a roadmap and a garage, and if the planning method for X is complete, then the overall planning algorithm is complete. If the planning method in X uses some weaker notion of completeness, then this is also maintained. For example, a resolution complete planner for X yields a resolution complete approach to the problem in Formulation 7.2.

Cube complex How is the coordination space represented when there are multiple paths for each robot? It turns out that a *cube complex* is obtained, which is a special kind of singular complex (recall from Section 6.3.1). The coordination space for m fixed paths can be considered as a singular m-simplex. For example, the problem in Figure 7.9 can be considered as a singular 3-simplex, $[0, 1]^3 \rightarrow X$. In Section 6.3.1, the domain of a k-simplex was defined using B^k, a k-dimensional ball; however, a cube, $[0, 1]^k$, also works because B^k and $[0, 1]^k$ are homeomorphic.

For a topological space, X, let a *k-cube* (which is also a singular k-simplex),

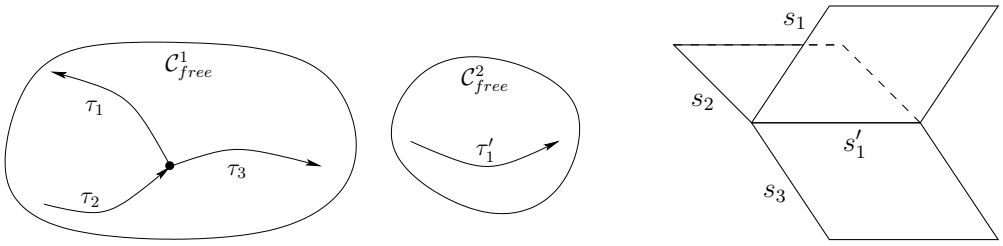


Figure 7.10: (a) An example in which $\mathcal{A}^1$ moves along three paths, and $\mathcal{A}^2$ moves along one. (b) The corresponding coordination space.

$\square_k$, be a continuous mapping $\sigma : [0,1]^k \rightarrow X$. A cube complex is obtained by connecting together k-cubes of different dimensions. Every k-cube for $k \geq 1$ has $2k$ *faces*, which are $(k-1)$-cubes that are obtained as follows. Let $(s_1, \ldots, s_k)$ denote a point in $[0,1]^k$. For each $i \in \{1, \ldots, k\}$, one face is obtained by setting $s_i = 0$ and another is obtained by setting $s_i = 1$.

The cubes must fit together nicely, much in the same way that the simplexes of a simplicial complex were required to fit together. To be a *cube complex*, $\mathcal{K}$ must be a collection of simplexes that satisfy the following requirements:

1. Any face, $\square_{k-1}$, of a cube $\square_k \in \mathcal{K}$ is also in $\mathcal{K}$.

2. The intersection of the images of any two k-cubes $\square_k, \square'_k \in \mathcal{K}$, is either empty or there exists some cube, $\square_i \in \mathcal{K}$ for $i < k$, which is a common face of both $\square_k$ and $\square'_k$.

Let $\mathcal{G}_i$ denote a topological graph (which may also be a roadmap) for robot $\mathcal{A}^i$. The graph edges are paths of the form $\tau : [0,1] \rightarrow \mathcal{C}^i_{free}$. Before covering formal definitions of the resulting complex, consider Figure 7.10a, in which $\mathcal{A}^1$ moves along three paths connected in a "T" junction and $\mathcal{A}^2$ moves along one path. In this case, three 2D fixed-path coordination spaces are attached together along one common edge, as shown in Figure 7.10b. The resulting cube complex is defined by three 2-cubes (i.e., squares), one 1-cube (i.e., line segment), and eight 0-cubes (i.e., corner points).

Now suppose more generally that there are two robots, $\mathcal{A}^1$ and $\mathcal{A}^2$, with associated topological graphs, $\mathcal{G}_1(V_1, E_1)$ and $\mathcal{G}_2(V_2, E_2)$, respectively. Suppose that $\mathcal{G}$ and $\mathcal{G}_2$ have n_1 and n_2 edges, respectively. A 2D cube complex, $\mathcal{K}$, is obtained as follows. Let τ_i denote the ith path of $\mathcal{G}_1$, and let σ_j denote the jth path of $\mathcal{G}_2$. A 2-cube (square) exists in $\mathcal{K}$ for every way to select an edge from each graph. Thus, there are $n_1 n_2$ 2-cubes, one for each pair (τ_i, σ_j), such that $\tau_i \in E_1$ and $\sigma_j \in E_2$. The 1-cubes are generated for pairs of the form (v_i, σ_j) for $v_i \in V_1$ and $\sigma_j \in E_2$, or (τ_i, v_j) for $\tau_i \in E_1$ and $v_j \in V_2$. The 0-cubes (corner points) are reached for each pair (v_i, v_j) such that $v_i \in V_1$ and $v_j \in V_2$.

If there are m robots, then an m-dimensional cube complex arises. Every m-cube corresponds to a unique combination of paths, one for each robot. The $(m-1)$-cubes are the faces of the m-cubes. This continues iteratively until the 0-cubes are reached.

Planning on the cube complex Once again, any of the planning methods described in Chapters 5 and 6 can be adapted here, but the methods are slightly complicated by the fact that X is a complex. To use sampling-based methods, a dense sequence should be generated over X. For example, if random sampling is used, then an m-cube can be chosen at random, followed by a random point in the cube. The local planning method (LPM) must take into account the connectivity of the cube complex, which requires recognizing when branches occur in the topological graph. Combinatorial methods must also take into account this connectivity. For example, a sweeping technique can be applied to produce a vertical cell decomposition, but the sweep-line (or sweep-plane) must sweep across the various m-cells of the complex.

7.3 Mixing Discrete and Continuous Spaces

Many important applications involve a mixture of discrete and continuous variables. This results in a state space that is a Cartesian product of the C-space and a finite set called the *mode space*. The resulting space can be visualized as having layers of C-spaces that are indexed by the modes, as depicted in Figure 7.11. The main application given in this section is manipulation planning; many others exist, especially when other complications such as dynamics and uncertainties are added to the problem. The framework of this section is inspired mainly from *hybrid systems* in the control theory community [409], which usually model mode-dependent dynamics. The main concern in this section is that the allowable robot configurations and/or the obstacles depend on the mode.

7.3.1 Hybrid Systems Framework

As illustrated in Figure 7.11, a hybrid system involves interaction between discrete and continuous spaces. The formal model will first be given, followed by some explanation. This formulation can be considered as a combination of the components from discrete feasible planning, Formulation 2.1, and basic motion planning, Formulation 4.1.

Formulation 7.3 (Hybrid-System Motion Planning)

1. The $\mathcal{W}$ and $\mathcal{C}$ components from Formulation 4.1 are included.
2. A nonempty *mode space*, M that is a finite or countably infinite set of *modes*.
3. A semi-algebraic *obstacle region* $\mathcal{O}(m)$ for each $m \in M$.

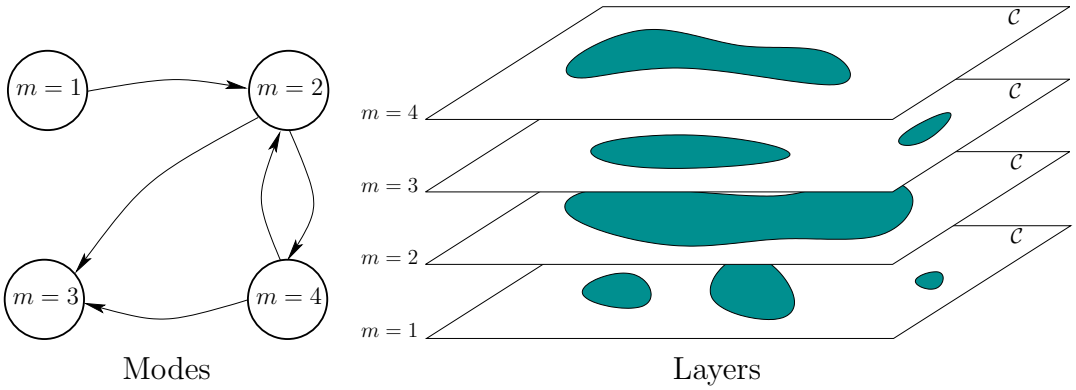


Figure 7.11: A hybrid state space can be imagined as having layers of C-spaces that are indexed by modes.

4. A semi-algebraic *robot* $\mathcal{A}(m)$, for each $m \in M$. It may be a rigid robot or a collection of links. It is assumed that the C-space is not mode-dependent; only the geometry of the robot can depend on the mode. The robot, transformed to configuration q, is denoted as $\mathcal{A}(q, m)$.

5. A *state space* X is defined as the Cartesian product $X = \mathcal{C} \times M$. A state is represented as $x = (q, m)$, in which $q \in \mathcal{C}$ and $m \in M$. Let

$$X_{obs} = \{(q, m) \in X \mid \mathcal{A}(q, m) \cap \mathcal{O}(m) \neq \emptyset\}, \tag{7.13}$$

   and $X_{free} = X \setminus X_{obs}$.

6. For each state, $x \in X$, there is a finite *action space*, $U(x)$. Let U denote the set of all possible actions (the union of $U(x)$ over all $x \in X$).

7. There is a *mode transition function* f_m that produces a mode, $f_m(x, u) \in M$, for every $x \in X$ and $u \in U(x)$. It is assumed that f_m is defined in a way that does not produce race conditions (oscillations of modes within an instant of time). This means that if q is fixed, the mode can change at most once. It then remains constant and can change only if q is changed.

8. There is a *state transition function*, f, that is derived from f_m by changing the mode and holding the configuration fixed. Thus, $f(x, u) = (q, f_m(x, u))$.

9. A configuration $x_I \in X_{free}$ is designated as the *initial state*.

10. A set $X_G \in X_{free}$ is designated as the *goal region*. A region is defined instead of a point to facilitate the specification of a goal configuration that does not depend on the final mode.

11. An algorithm must compute a (continuous) *path* $\tau : [0, 1] \rightarrow X_{free}$ and an *action trajectory* $\sigma : [0, 1] \rightarrow U$ such that $\tau(0) = x_I$ and $\tau(1) \in X_G$, or the

algorithm correctly reports that such a combination of a path and an action trajectory does not exist.

The obstacle region and robot may or may not be mode-dependent, depending on the problem. Examples of each will be given shortly. Changes in the mode depend on the action taken by the robot. From most states, it is usually assumed that a "do nothing" action exists, which leaves the mode unchanged. From certain states, the robot may select an action that changes the mode as desired. An interesting degenerate case exists in which there is only a single action available. This means that the robot has no control over the mode from that state. If the robot arrives in such a state, a mode change could unavoidably occur.

The solution requirement is somewhat more complicated because both a path and an action trajectory need to be specified. It is insufficient to specify a path because it is important to know what action was applied to induce the correct mode transitions. Therefore, σ indicates when these occur. Note that τ and σ are closely coupled; one cannot simply associate any σ with a path τ; it must correspond to the actions required to generate τ.

Example 7.2 (The Power of the Portiernia) In this example, a robot, $\mathcal{A}$, is modeled as a square that translates in $\mathcal{W} = \mathbb{R}^2$. Therefore, $\mathcal{C} = \mathbb{R}^2$. The obstacle region in $\mathcal{W}$ is mode-dependent because of two doors, which are numbered "1" and "2" in Figure 7.12a. In the upper left sits the *portiernia*,[2] which is able to give a key to the robot, if the robot is in a configuration as shown in Figure 7.12b. The portiernia only trusts the robot with one key at a time, which may be either for Door 1 or Door 2. The robot can return a key by revisiting the portiernia. As shown in Figures 7.12c and 7.12d, the robot can open a door by making contact with it, as long as it holds the correct key.

The set, M, of modes needs to encode which key, if any, the robot holds, and it must also encode the status of the doors. The robot may have: 1) the key to Door 1; 2) the key to Door 2; or 3) no keys. The doors may have the status: 1) both open; 2) Door 1 open, Door 2 closed; 3) Door 1 closed, Door 2 open; or 4) both closed. Considering keys and doors in combination yields 12 possible modes.

If the robot is at a portiernia configuration as shown in Figure 7.12b, then its available actions correspond to different ways to pick up and drop off keys. For example, if the robot is holding the key to Door 1, it can drop it off and pick up the key to Door 2. This changes the mode, but the door status and robot configuration must remain unchanged when f is applied. The other locations in which the robot may change the mode are when it comes in contact with Door 1 or Door 2. The mode changes only if the robot is holding the proper key. In all other configurations, the robot only has a single action (i.e., no choice), which keeps the mode fixed.

The task is to reach the configuration shown in the lower right with dashed lines. The problem is solved by: 1) picking up the key for Door 1 at the portiernia;

[2]This is a place where people guard the keys at some public facilities in Poland.

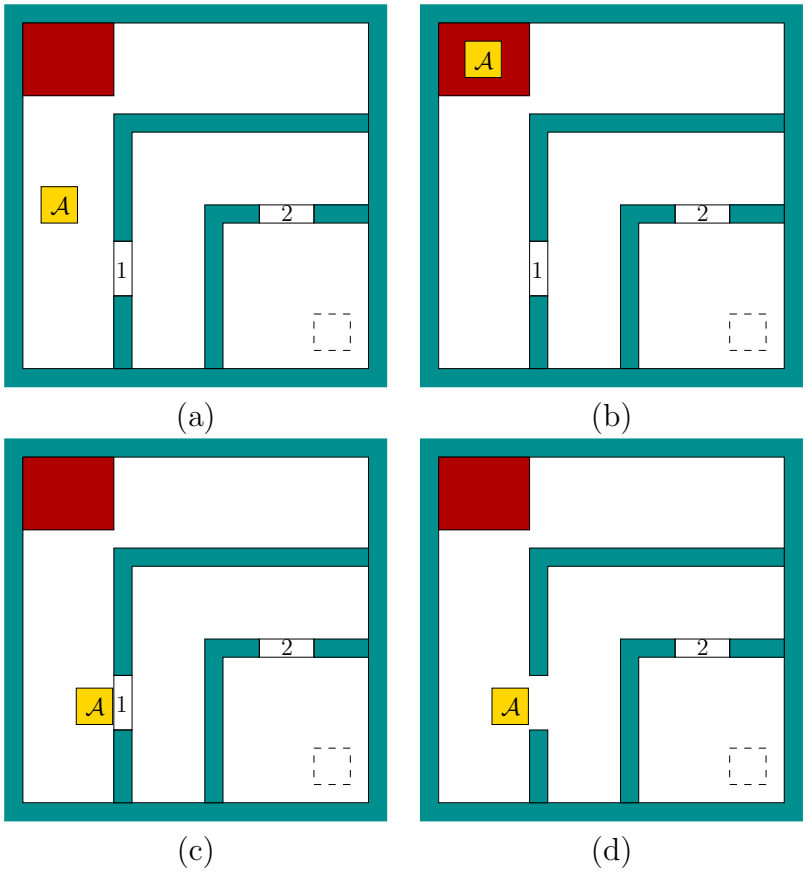


Figure 7.12: In the upper left (at the portiernia), the robot can pick up and drop off keys that open one of two doors. If the robot contacts a door while holding the correct key, then it opens.

2) opening Door 1; 3) swapping the key at the portiernia to obtain the key for Door 2; or 4) entering the innermost room to reach the goal configuration. As a final condition, we might want to require that the robot returns the key to the portiernia. ■

Example 7.2 allows the robot to change the obstacles in $\mathcal{O}$. The next example involves a robot that can change its shape. This is an illustrative example of a *reconfigurable robot*. The study of such robots has become a popular topic of research [209, 385, 552, 990]; the reconfiguration possibilities in that research area are much more complicated than the simple example considered here.

Example 7.3 (Reconfigurable Robot) To solve the problem shown in Figure 7.13, the robot must change its shape. There are two possible shapes, which correspond directly to the modes: *elongated* and *compressed*. Examples of each are shown in the figure. Figure 7.14 shows how $\mathcal{C}_{free}(m)$ appears for each of the

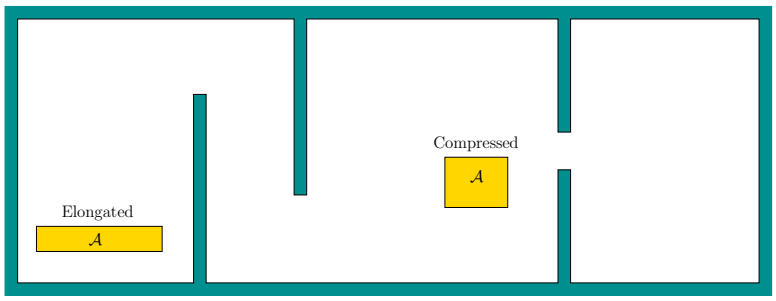
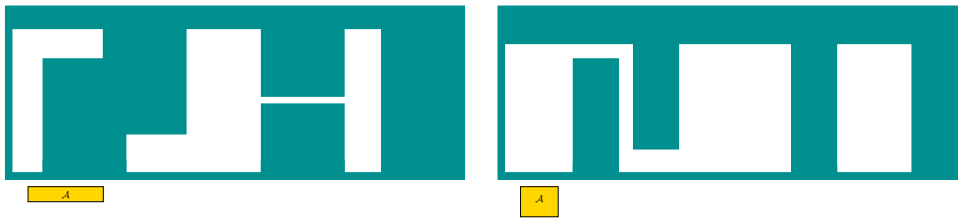


Figure 7.13: An example in which the robot must reconfigure itself to solve the problem. There are two modes: *elongated* and *compressed*.



Elongated mode Compressed mode

Figure 7.14: When the robot reconfigures itself, $\mathcal{C}_{free}(m)$ changes, enabling the problem to be solved.

two modes. Suppose the robot starts initially from the left while in the elongated mode and must travel to the last room on the right. This problem must be solved by 1) reconfiguring the robot into the compressed mode; 2) passing through the corridor into the center; 3) reconfiguring the robot into the elongated mode; and 4) passing through the corridor to the rightmost room. The robot has actions that directly change the mode by reconfiguring itself. To make the problem more interesting, we could require the robot to reconfigure itself in specific locations (e.g., where there is enough clearance, or possibly at a location where another robot can assist it).

The examples presented so far barely scratch the surface on the possible hybrid motion planning problems that can be defined. Many such problems can arise, for example, in the context making automated video game characters or digital actors. To solve these problems, standard motion planning algorithms can be adapted if they are given information about how to change the modes. Locations in X from which the mode can be changed may be expressed as subgoals. Much of the planning effort should then be focused on attempting to change modes, in addition to trying to directly reach the goal. Applying sampling-based methods requires the definition of a metric on X that accounts for both changes in the mode and the configuration. A wide variety of hybrid problems can be formulated, ranging from those that are impossible to solve in practice to those that are straightforward

extensions of standard motion planning. In general, the hybrid motion planning model is useful for formulating a hierarchical approach, as described in Section 1.4. One particularly interesting class of problems that fit this model, for which successful algorithms have been developed, will be covered next.

7.3.2 Manipulation Planning

This section presents an overview of manipulation planning; the concepts explained here are mainly due to [16, 17]. Returning to Example 7.2, imagine that the robot must carry a key that is so large that it changes the connectivity of $\mathcal{C}_{free}$. For the manipulation planning problem, the robot is called a *manipulator*, which interacts with a *part*. In some configurations it is able to *grasp* the part and move it to other locations in the environment. The *manipulation task* usually requires moving the part to a specified location in $\mathcal{W}$, without particular regard as to how the *manipulator* can accomplish the task. The model considered here greatly simplifies the problems of grasping, stability, friction, mechanics, and uncertainties and instead focuses on the geometric aspects (some of these issues will be addressed in Section 12.5). For a thorough introduction to these other important aspects of manipulation planning, see [681]; see also Sections 13.1.3 and 12.5.

Admissible configurations Assume that $\mathcal{W}$, $\mathcal{O}$, and $\mathcal{A}$ from Formulation 4.1 are used. For manipulation planning, $\mathcal{A}$ is called the *manipulator*, and let $\mathcal{C}^a$ refer to the *manipulator configuration space*. Let $\mathcal{P}$ denote a *part*, which is a rigid body modeled in terms of geometric primitives, as described in Section 3.1. It is assumed that $\mathcal{P}$ is allowed to undergo rigid-body transformations and will therefore have its own *part configuration space*, $\mathcal{C}^p = SE(2)$ or $\mathcal{C}^p = SE(3)$. Let $q^p \in \mathcal{C}^p$ denote a *part configuration*. The transformed part model is denoted as $\mathcal{P}(q^p)$.

The combined *configuration space*, $\mathcal{C}$, is defined as the Cartesian product

$$\mathcal{C} = \mathcal{C}^a \times \mathcal{C}^p, \tag{7.14}$$

in which each configuration $q \in \mathcal{C}$ is of the form $q = (q^a, q^p)$. The first step is to remove all configurations that must be avoided. Parts of Figure 7.15 show examples of these sets. Configurations for which the manipulator collides with obstacles are

$$\mathcal{C}^a_{obs} = \{(q^a, q^p) \in \mathcal{C} \mid \mathcal{A}(q^a) \cap \mathcal{O} \neq \emptyset\}. \tag{7.15}$$

The next logical step is to remove configurations for which the part collides with obstacles. It will make sense to allow the part to "touch" the obstacles. For example, this could model a part sitting on a table. Therefore, let

$$\mathcal{C}^p_{obs} = \{(q^a, q^p) \in \mathcal{C} \mid \text{int}(\mathcal{P}(q^p)) \cap \mathcal{O} \neq \emptyset\} \tag{7.16}$$

denote the open set for which the interior of the part intersects $\mathcal{O}$. Certainly, if the part penetrates $\mathcal{O}$, then the configuration should be avoided.

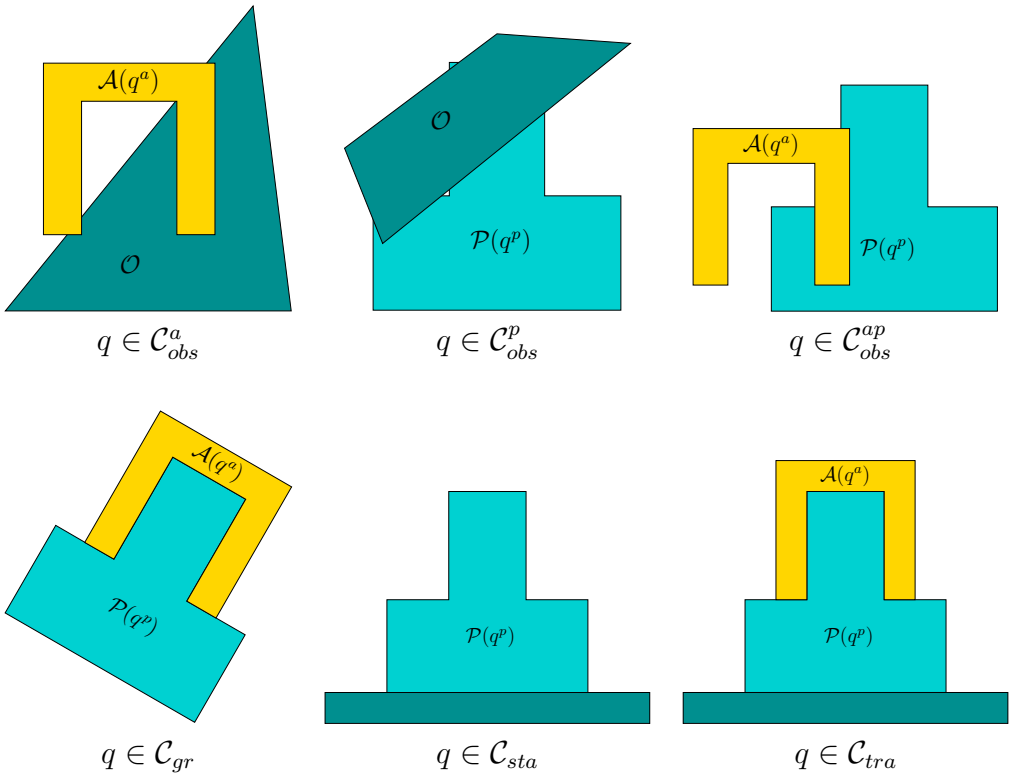


Figure 7.15: Examples of several important subsets of $\mathcal{C}$ for manipulation planning.

Consider $\mathcal{C} \setminus (\mathcal{C}_{obs}^a \cup \mathcal{C}_{obs}^p)$. The configurations that remain ensure that the robot and part do not inappropriately collide with $\mathcal{O}$. Next consider the interaction between $\mathcal{A}$ and $\mathcal{P}$. The manipulator must be allowed to touch the part, but penetration is once again not allowed. Therefore, let

$$\mathcal{C}_{obs}^{ap} = \{(q^a, q^p) \in \mathcal{C} \mid \mathcal{A}(q^a) \cap \mathrm{int}(\mathcal{P}(q^p)) \neq \emptyset\}. \tag{7.17}$$

Removing all of these bad configurations yields

$$\mathcal{C}_{adm} = \mathcal{C} \setminus (\mathcal{C}_{obs}^a \cup \mathcal{C}_{obs}^p \cup \mathcal{C}_{obs}^{ap}), \tag{7.18}$$

which is called the set of *admissible configurations*.

Stable and grasped configurations Two important subsets of $\mathcal{C}_{adm}$ are used in the manipulation planning problem. See Figure 7.15. Let $\mathcal{C}_{sta}^p$ denote the set of *stable part configurations*, which are configurations at which the part can safely rest without any forces being applied by the manipulator. This means that a part cannot, for example, float in the air. It also cannot be in a configuration from which it might fall. The particular stable configurations depend on properties such as the part geometry, friction, mass distribution, and so on. These issues are not considered here. From this, let $\mathcal{C}_{sta} \subseteq \mathcal{C}_{adm}$ be the corresponding *stable configurations*, defined as

$$\mathcal{C}_{sta} = \{(q^a, q^p) \in \mathcal{C}_{adm} \mid q^p \in \mathcal{C}_{sta}^p\}. \tag{7.19}$$

The other important subset of $\mathcal{C}_{adm}$ is the set of all configurations in which the robot is grasping the part (and is capable of carrying it, if necessary). Let this denote the *grasped configurations*, denoted by $\mathcal{C}_{gr} \subseteq \mathcal{C}_{adm}$. For every configuration, $(q^a, q^p) \in \mathcal{C}_{gr}$, the manipulator touches the part. This means that $\mathcal{A}(q^a) \cap \mathcal{P}(q^p) \neq \emptyset$ (penetration is still not allowed because $\mathcal{C}_{gr} \subseteq \mathcal{C}_{adm}$). In general, many configurations at which $\mathcal{A}(q^a)$ contacts $\mathcal{P}(q^p)$ will not necessarily be in $\mathcal{C}_{gr}$. The conditions for a point to lie in $\mathcal{C}_{gr}$ depend on the particular characteristics of the manipulator, the part, and the contact surface between them. For example, a typical manipulator would not be able to pick up a block by making contact with only one corner of it. This level of detail is not defined here; see [681] for more information about grasping.

We must always ensure that either $x \in \mathcal{C}_{sta}$ or $x \in \mathcal{C}_{gr}$. Therefore, let $\mathcal{C}_{free} = \mathcal{C}_{sta} \cup \mathcal{C}_{gr}$, to reflect the subset of $\mathcal{C}_{adm}$ that is permissible for manipulation planning.

The mode space, M, contains two modes, which are named the *transit mode* and the *transfer mode*. In the transit mode, the manipulator is not carrying the part, which requires that $q \in \mathcal{C}_{sta}$. In the transfer mode, the manipulator carries the part, which requires that $q \in \mathcal{C}_{gr}$. Based on these simple conditions, the only way the mode can change is if $q \in \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$. Therefore, the manipulator has two available actions only when it is in these configurations. In all other configurations the mode remains unchanged. For convenience, let $\mathcal{C}_{tra} = \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$ denote the set of *transition configurations*, which are the places in which the mode may change.

Using the framework of Section 7.3.1, the mode space, M, and C-space, $\mathcal{C}$, are combined to yield the *state space*, $X = \mathcal{C} \times M$. Since there are only two modes, there are only two copies of $\mathcal{C}$, one for each mode. State-based sets, X_{free}, X_{tra}, X_{sta}, and X_{gr}, are directly obtained from $\mathcal{C}_{free}$, $\mathcal{C}_{tra}$, $\mathcal{C}_{sta}$, and $\mathcal{C}_{gr}$ by ignoring the mode. For example,

$$X_{tra} = \{(q, m) \in X \mid q \in \mathcal{C}_{tra}\}. \tag{7.20}$$

The sets X_{free}, X_{sta}, and X_{gr} are similarly defined.

The task can now be defined. An *initial part configuration*, $q^p_{init} \in \mathcal{C}_{sta}$, and a *goal part configuration*, $q^p_{goal} \in \mathcal{C}_{sta}$, are specified. Compute a path $\tau : [0, 1] \rightarrow X_{free}$ such that $\tau(0) = q^p_{init}$ and $\tau(1) = q^p_{goal}$. Furthermore, the *action trajectory* $\sigma : [0, 1] \rightarrow U$ must be specified to indicate the appropriate mode changes whenever $\tau(s) \in X_{tra}$. A solution can be considered as an alternating sequence of *transit paths* and *transfer paths*, whose names follow from the mode. This is depicted in Figure 7.16.

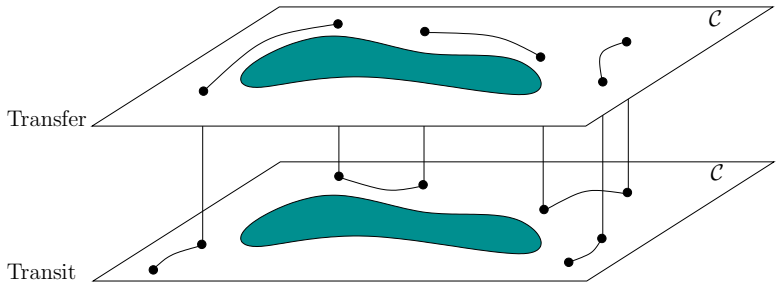


Figure 7.16: The solution to a manipulation planning problem alternates between the two layers of X. The transitions can only occur when $x \in X_{tra}$.

Manipulation graph The manipulation planning problem generally can be solved by forming a manipulation graph, $\mathcal{G}_m$ [16, 17]. Let a *connected component* of X_{tra} refer to any connected component of $\mathcal{C}_{tra}$ that is lifted into the state space by ignoring the mode. There are two copies of the connected component of $\mathcal{C}_{tra}$, one for each mode. For each connected component of X_{tra}, a vertex exists in $\mathcal{G}_m$. An edge is defined for each transfer path or transit path that connects two connected components of X_{tra}. The general approach to manipulation planning then is as follows:

1. Compute the connected components of X_{tra} to yield the vertices of $\mathcal{G}_m$.

2. Compute the edges of $\mathcal{G}_m$ by applying ordinary motion planning methods to each pair of vertices of $\mathcal{G}_m$.

3. Apply motion planning methods to connect the initial and goal states to every possible vertex of X_{tra} that can be reached without a mode transition.

4. Search $\mathcal{G}_m$ for a path that connects the initial and goal states. If one exists, then extract the corresponding solution as a sequence of transit and transfer paths (this yields σ, the actions that cause the required mode changes).

This can be considered as an example of hierarchical planning, as described in Section 1.4.

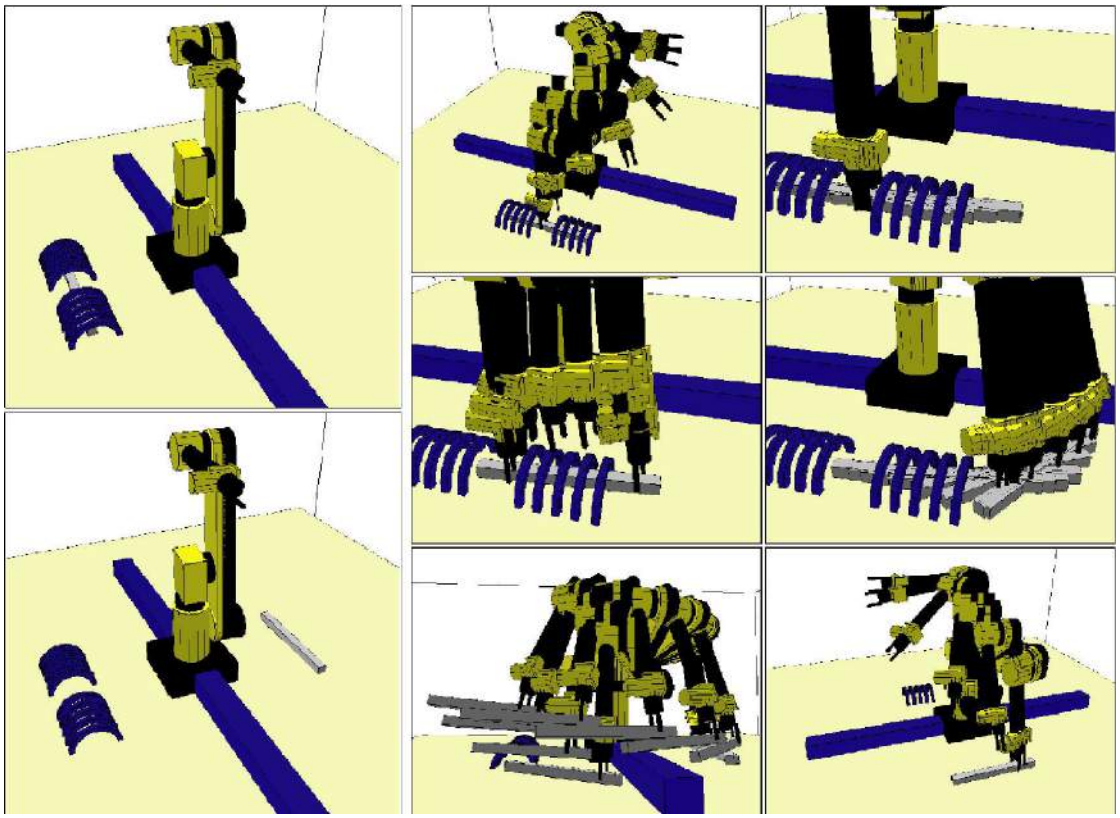


Figure 7.17: This example was solved in [244] using the manipulation planning framework and the visibility-based roadmap planner. It is very challenging because the same part must be regrasped in many places.

Multiple parts The manipulation planning framework nicely generalizes to multiple parts, $\mathcal{P}_1$, ..., $\mathcal{P}_k$. Each part has its own C-space, and $\mathcal{C}$ is formed by taking the Cartesian product of all part C-spaces with the manipulator C-space. The set $\mathcal{C}_{adm}$ is defined in a similar way, but now part-part collisions also have to be removed, in addition to part-manipulator, manipulator-obstacle, and part-obstacle collisions. The definition of $\mathcal{C}_{sta}$ requires that all parts be in stable configurations; the parts may even be allowed to stack on top of each other. The definition of $\mathcal{C}_{gr}$ requires that one part is grasped and all other parts are stable. There are still two modes, depending on whether the manipulator is grasping a part. Once again, transitions occur only when the robot is in $\mathcal{C}_{tra} = \mathcal{C}_{sta} \cap \mathcal{C}_{gr}$.

Figure 7.18: This manipulation planning example was solved in [915] and involves 90 movable pieces of furniture. Some of them must be dragged out of the way to solve the problem. Paths for two different queries are shown.

The task involves moving each part from one configuration to another. This is achieved once again by defining a manipulation graph and obtaining a sequence of transit paths (in which no parts move) and transfer paths (in which one part is carried and all other parts are fixed). Challenging manipulation problems solved by motion planning algorithms are shown in Figures 7.17 and 7.18.

Other generalizations are possible. A generalization to k robots would lead to 2^k modes, in which each mode indicates whether each robot is grasping the part. Multiple robots could even grasp the same object. Another generalization could allow a single robot to grasp more than one object.

7.4 Planning for Closed Kinematic Chains

This section continues where Section 4.4 left off. The subspace of $\mathcal{C}$ that results from maintaining kinematic closure was defined and illustrated through some ex-

amples. Planning in this context requires that paths remain on a lower dimensional variety for which a parameterization is not available. Many important applications require motion planning while maintaining these constraints. For example, consider a manipulation problem that involves multiple manipulators grasping the same object, which forms a closed loop as shown in Figure 7.19. A loop exists because both manipulators are attached to the ground, which may itself be considered as a link. The development of virtual actors for movies and video games also involves related manipulation problems. Loops also arise in this context when more than one human limb is touching a fixed surface (e.g., two feet on the ground). A class of robots called *parallel manipulators* are intentionally designed with internal closed loops [693]. For example, consider the Stewart-Gough platform [407, 914] illustrated in Figure 7.20. The lengths of each of the six arms, $\mathcal{A}_1$, ..., $\mathcal{A}_6$, can be independently varied while they remain attached via spherical joints to the ground and to the *platform*, which is $\mathcal{A}_7$. Each arm can actually be imagined as two links that are connected by a prismatic joint. Due to the total of 6 degrees of freedom introduced by the variable lengths, the platform actually achieves the full 6 degrees of freedom (hence, some six-dimensional region in $SE(3)$ is obtained for $\mathcal{A}_7$). Planning the motion of the Stewart-Gough platform, or robots that are based on the platform (the robot shown in Figure 7.27 uses a stack of several of these mechanisms), requires handling many closure constraints that must be maintained simultaneously. Another application is computational biology, in which the C-space of molecules is searched, many of which are derived from molecules that have closed, flexible chains of bonds [245].

7.4.1 Adaptation of Motion Planning Algorithms

All of the components from the general motion planning problem of Formulation 4.1 are included: $\mathcal{W}$, $\mathcal{O}$, $\mathcal{A}_1$, ..., $\mathcal{A}_m$, $\mathcal{C}$, q_I, and q_G. It is assumed that the robot is a collection of r links that are possibly attached in loops.

It is assumed in this section that $\mathcal{C} = \mathbb{R}^n$. If this is not satisfactory, there are two ways to overcome the assumption. The first is to represent $SO(2)$ and $SO(3)$ as $\mathbb{S}^1$ and $\mathbb{S}^3$, respectively, and include the circle or sphere equation as part of the constraints considered here. This avoids the topology problems. The other option is to abandon the restriction of using $\mathbb{R}^n$ and instead use a parameterization of $\mathcal{C}$ that is of the appropriate dimension. To perform calculus on such manifolds, a *smooth structure* is required, which is introduced in Section 8.3.2. In the presentation here, however, vector calculus on $\mathbb{R}^n$ is sufficient, which intentionally avoids these extra technicalities.

Closure constraints The closure constraints introduced in Section 4.4 can be summarized as follows. There is a set, $\mathcal{P}$, of polynomials f_1, ..., f_k that belong to $\mathbb{Q}[q_1, \ldots, q_n]$ and express the constraints for particular points on the links of the robot. The determination of these is detailed in Section 4.4.3. As mentioned previously, polynomials that force points to lie on a circle or sphere in the case of

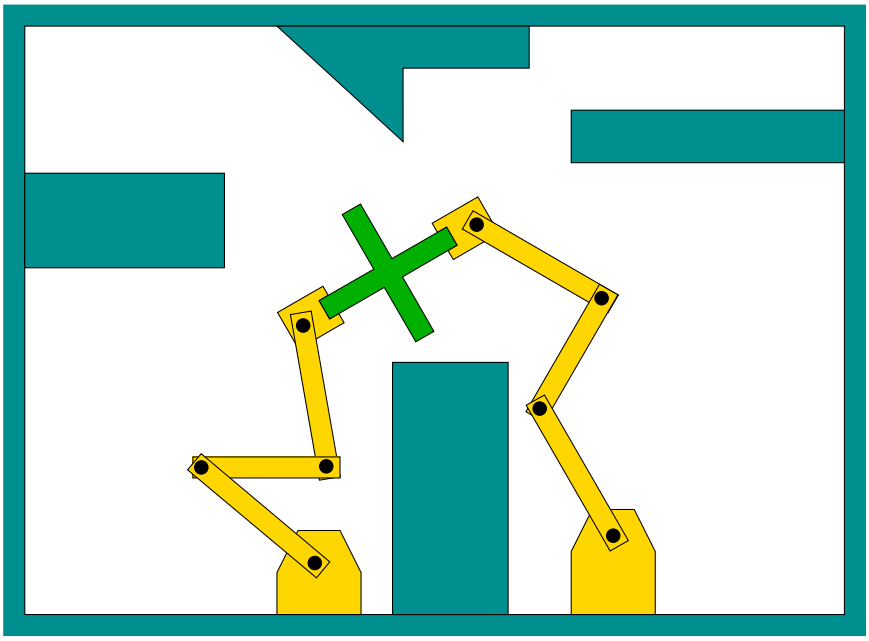


Figure 7.19: Two or more manipulators manipulating the same object causes closed kinematic chains. Each black disc corresponds to a revolute joint.

rotations may also be included in $\mathcal{P}$.

Let n denote the dimension of $\mathcal{C}$. The *closure space* is defined as

$$\mathcal{C}_{clo} = \{q \in \mathcal{C} \mid \forall f_i \in \mathcal{P}, f_i(q_1, \ldots, q_n) = 0\}, \tag{7.21}$$

which is an m-dimensional subspace of $\mathcal{C}$ that corresponds to all configurations that satisfy the closure constants. The obstacle set must also be taken into account. Once again, $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ are defined using (4.34). The *feasible space* is defined as $\mathcal{C}_{fea} = \mathcal{C}_{clo} \cap \mathcal{C}_{free}$, which are the configurations that satisfy closure constraints and avoid collisions.

The motion planning problem is to find a path $\tau : [0, 1] \rightarrow \mathcal{C}_{fea}$ such that $\tau(0) = q_I$ and $\tau(1) = q_G$. The new challenge is that there is no explicit parameterization of $\mathcal{C}_{fea}$, which is further complicated by the fact that $m < n$ (recall that m is the dimension of $\mathcal{C}_{clo}$).

Combinatorial methods Since the constraints are expressed with polynomials, it may not be surprising that the computational algebraic geometry methods of Section 6.4 can solve the general motion planning problem with closed kinematic chains. Either cylindrical algebraic decomposition or Canny's roadmap algorithm may be applied. As mentioned in Section 6.5.3, an adaptation of the roadmap algorithm that is optimized for problems in which $m < n$ is given in [76].

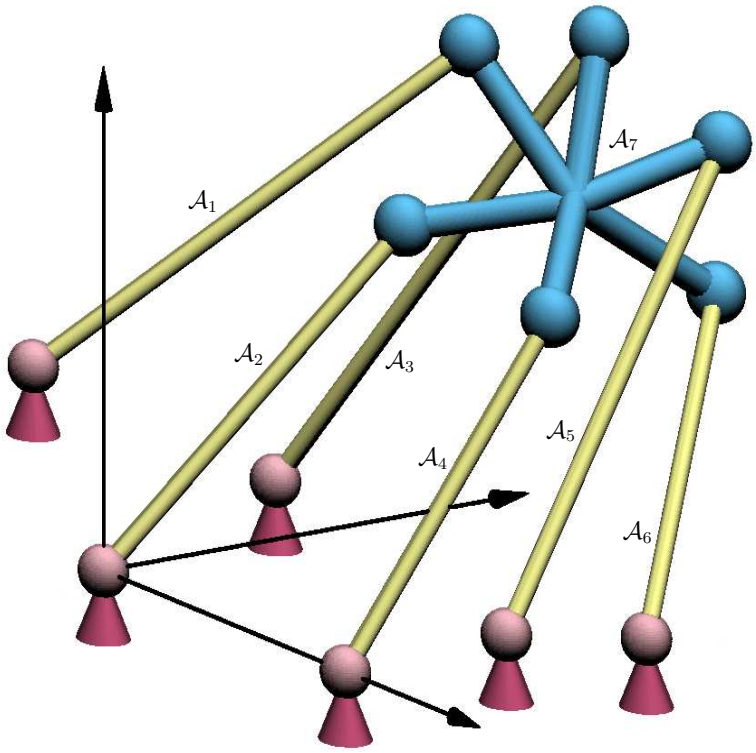


Figure 7.20: An illustration of the Stewart-Gough platform (adapted from a figure made by Frank Sottile).

Sampling-based methods Most of the methods of Chapter 5 are not easy to adapt because they require sampling in $\mathcal{C}_{fea}$, for which a parameterization does not exist. If points in a bounded region of $\mathbb{R}^n$ are chosen at random, the probability is zero that a point on $\mathcal{C}_{fea}$ will be obtained. Some incremental sampling and searching methods can, however, be adapted by the construction of a local planning method (LPM) that is suited for problems with closure constraints. The sampling-based roadmap methods require many samples to be generated directly on $\mathcal{C}_{fea}$. Section 7.4.2 presents some techniques that can be used to generate such samples for certain classes of problems, enabling the development of efficient sampling-based planners and also improving the efficiency of incremental search planners. Before covering these techniques, we first present a method that leads to a more general sampling-based planner and is easier to implement. However, if designed well, planners based on the techniques of Section 7.4.2 are more efficient.

Now consider adapting the RDT of Section 5.5 to work for problems with closure constraints. Similar adaptations may be possible for other incremental sampling and searching methods covered in Section 5.4, such as the randomized potential field planner. A dense sampling sequence, α, is generated over a bounded n-dimensional subset of $\mathbb{R}^n$, such as a rectangle or sphere, as shown in Figure 7.21.

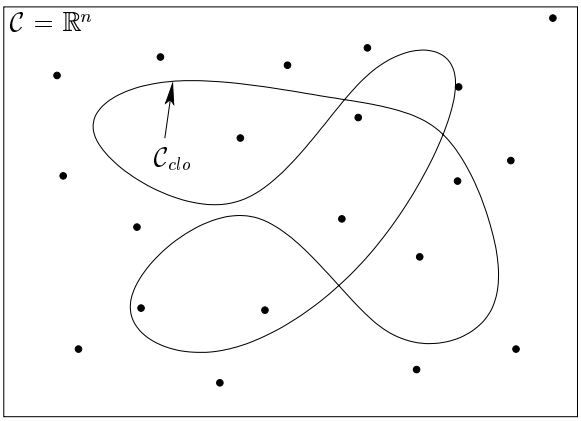


Figure 7.21: For the RDT, the samples can be drawn from a region in $\mathbb{R}^n$, the space in which $\mathcal{C}_{clo}$ is embedded.

The samples are not actually required to lie on $\mathcal{C}_{clo}$ because they do not necessarily become part of the topological graph, $\mathcal{G}$. They mainly serve to pull the search tree in different directions. One concern in choosing the bounding region is that it must include $\mathcal{C}_{clo}$ (at least the connected component that includes q_I) but it must not be unnecessarily large. Such bounds are obtained by analyzing the motion limits for a particular linkage.

Stepping along $\mathcal{C}_{clo}$ The RDT algorithm given Figure 5.21 can be applied directly; however, the STOPPING-CONFIGURATION function in line 4 must be changed to account for both obstacles and the constraints that define $\mathcal{C}_{clo}$. Figure 7.22 shows one general approach that is based on *numerical continuation* [18]. An alternative is to use inverse kinematics, which is part of the approach described in Section 7.4.2. The nearest RDT vertex, $q \in \mathcal{C}$, to the sample $\alpha(i)$ is first computed. Let $v = \alpha(i) - q$, which indicates the direction in which an edge would be made from q if there were no constraints. A local motion is then computed by projecting v into the tangent plane[3] of $\mathcal{C}_{clo}$ at the point q. Since $\mathcal{C}_{clo}$ is generally nonlinear, the local motion produces a point that is not precisely on $\mathcal{C}_{clo}$. Some numerical tolerance is generally accepted, and a small enough step is taken to ensure that the tolerance is maintained. The process iterates by computing v with respect to the new point and moving in the direction of v projected into the new tangent plane. If the error threshold is surpassed, then motions must be executed in the normal direction to return to $\mathcal{C}_{clo}$. This process of executing tangent and normal motions terminates when progress can no longer be made, due either to the alignment of the tangent plane (nearly perpendicular to v) or to an obstacle. This finally yields q_s, the stopping configuration. The new path followed in $\mathcal{C}_{fea}$ is no longer a "straight line" as was possible for some problems in Section 5.5; therefore, the approximate methods in Section 5.5.2 should be used to create intermediate

[3]Tangent planes are defined rigorously in Section 8.3.

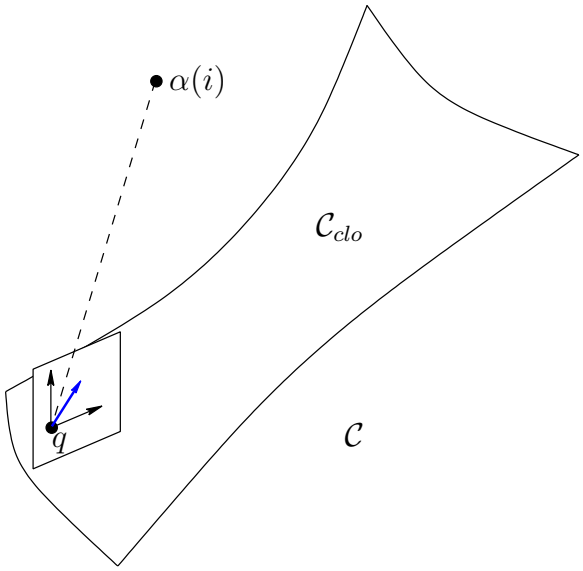


Figure 7.22: For each sample $\alpha(i)$ the nearest point, $q_n \in \mathcal{C}$, is found, and then the local planner generates a motion that lies in the local tangent plane. The motion is the project of the vector from q_n to $\alpha(i)$ onto the tangent plane.

vertices along the path.

In each iteration, the tangent plane computation is computed at some $q \in \mathcal{C}_{clo}$ as follows. The differential configuration vector dq lies in the tangent space of a constraint $f_i(q) = 0$ if

$$\frac{\partial f_i(q)}{\partial q_1} dq_1 + \frac{\partial f_i(q)}{\partial q_2} dq_2 + \cdots + \frac{\partial f_i(q)}{\partial q_n} dq_n = 0. \tag{7.22}$$

This leads to the following homogeneous system for all of the k polynomials in $\mathcal{P}$ that define the closure constraints

$$\begin{pmatrix} \frac{\partial f_1(q)}{\partial q_1} & \frac{\partial f_1(q)}{\partial q_2} & \cdots & \frac{\partial f_1(q)}{\partial q_n} \\ \frac{\partial f_2(q)}{\partial q_1} & \frac{\partial f_2(q)}{\partial q_2} & \cdots & \frac{\partial f_2(q)}{\partial q_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_k(q)}{\partial q_1} & \frac{\partial f_k(q)}{\partial q_2} & \cdots & \frac{\partial f_k(q)}{\partial q_n} \end{pmatrix} \begin{pmatrix} dq_1 \\ dq_2 \\ \vdots \\ dq_n \end{pmatrix} = \mathbf{0}. \tag{7.23}$$

If the rank of the matrix is $m \leq n$, then m configuration displacements can be chosen independently, and the remaining $n - m$ parameters must satisfy (7.23). This can be solved using linear algebra techniques, such as singular value decomposition (SVD) [399, 961], to compute an orthonormal basis for the tangent space at q. Let $e_1, \ldots, e_m$, denote these n-dimensional basis vectors. The components

of the motion direction are obtained from $v = \alpha(i) - q_n$. First, construct the inner products, $a_1 = v \cdot e_1$, $a_2 = v \cdot e_2$, ..., $a_m = v \cdot e_m$. Using these, the projection of v in the tangent plane is the n-dimensional vector w given by

$$w = \sum_{i}^{m} a_i e_i, \tag{7.24}$$

which is used as the direction of motion. The magnitude must be appropriately scaled to take sufficiently small steps. Since $\mathcal{C}_{clo}$ is generally curved, a linear motion leaves $\mathcal{C}_{clo}$. A motion in the inward normal direction is then required to move back onto $\mathcal{C}_{clo}$.

Since the dimension m of $\mathcal{C}_{clo}$ is less than n, the procedure just described can only produce numerical approximations to paths in $\mathcal{C}_{clo}$. This problem also arises in implicit curve tracing in graphics and geometric modeling [454]. Therefore, each constraint $f_i(q_1, \ldots, q_n) = 0$ is actually slightly weakened to $|f_i(q_1, \ldots, q_n)| < \epsilon$ for some fixed tolerance $\epsilon > 0$. This essentially "thickens" $\mathcal{C}_{clo}$ so that its dimension is n. As an alternative to computing the tangent plane, motion directions can be sampled directly inside of this thickened region without computing tangent planes. This results in an easier implementation, but it is less efficient [979].

7.4.2 Active-Passive Link Decompositions

An alternative sampling-based approach is to perform an *active-passive decomposition*, which is used to generate samples in $\mathcal{C}_{clo}$ by directly sampling *active* variables, and computing the closure values for *passive* variables using inverse kinematics methods. This method was introduced in [432] and subsequently improved through the development of the *random loop generator* in [244, 246]. The method serves as a general framework that can adapt virtually any of the methods of Chapter 5 to handle closed kinematic chains, and experimental evidence suggests that the performance is better than the method of Section 7.4.1. One drawback is that the method requires some careful analysis of the linkage to determine the best decomposition and also bounds on its mobility. Such analysis exists for very general classes of linkages [244].

Active and passive variables In this section, let $\mathcal{C}$ denote the C-space obtained from all joint variables, instead of requiring $\mathcal{C} = \mathbb{R}^n$, as in Section 7.4.1. This means that $\mathcal{P}$ includes only polynomials that encode closure constraints, as opposed to allowing constraints that represent rotations. Using the tree representation from Section 4.4.3, this means that $\mathcal{C}$ is of dimension n, arising from assigning one variable for each revolute joint of the linkage in the absence of any constraints. Let $q \in \mathcal{C}$ denote this vector of configuration variables. The *active-passive decomposition* partitions the variables of q to form two vectors, q^a, called the *active variables* and q^p, called the *passive variables*. The values of passive variables are always determined from the active variables by enforcing the closure

constraints and using inverse kinematics techniques. If m is the dimension of $\mathcal{C}_{clo}$, then there are always m active variables and $n - m$ passive variables.

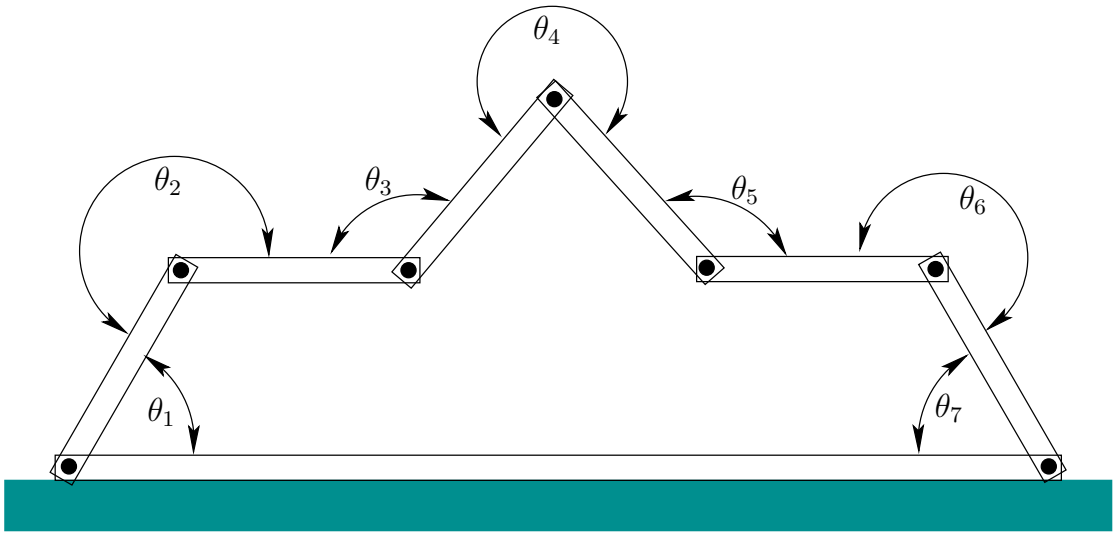


Figure 7.23: A chain of links in the plane. There are seven links and seven joints, which are constrained to form a loop. The dimension of $\mathcal{C}$ is seven, but the dimension of $\mathcal{C}_{clo}$ is four.

Temporarily, suppose that the linkage forms a single loop as shown in Figure 7.23. One possible decomposition into active q^a and passive q^p variables is given in Figure 7.24. If constrained to form a loop, the linkage has four degrees of freedom, assuming the bottom link is rigidly attached to the ground. This means that values can be chosen for four active joint angles q^a and the remaining three q^p can be derived from solving the inverse kinematics. To determine q^p, there are three equations and three unknowns. Unfortunately, these equations are nonlinear and fairly complicated. Nevertheless, efficient solutions exist for this case, and the 3D generalization [675]. For a 3D loop formed of revolute joints, there are six passive variables. The number, 3, of passive links in $\mathbb{R}^2$ and the number 6 for $\mathbb{R}^3$ arise from the dimensions of $SE(2)$ and $SE(3)$, respectively. This is the freedom that is stripped away from the system by enforcing the closure constraints. Methods for efficiently computing inverse kinematics in two and three dimensions are given in [30]. These can also be applied to the RDT stepping method in Section 7.4.1, instead of using continuation.

If the maximal number of passive variables is used, there is at most a finite number of solutions to the inverse kinematics problem; this implies that there are often several choices for the passive variable values. It could be the case that for some assignments of active variables, there are no solutions to the inverse kinematics. An example is depicted in Figure 7.25. Suppose that we want to generate samples in $\mathcal{C}_{clo}$ by selecting random values for q^a and then using inverse kinematics for q^p. What is the probability that a solution to the inverse kinematics exists? For the example shown, it appears that solutions would not exist in most

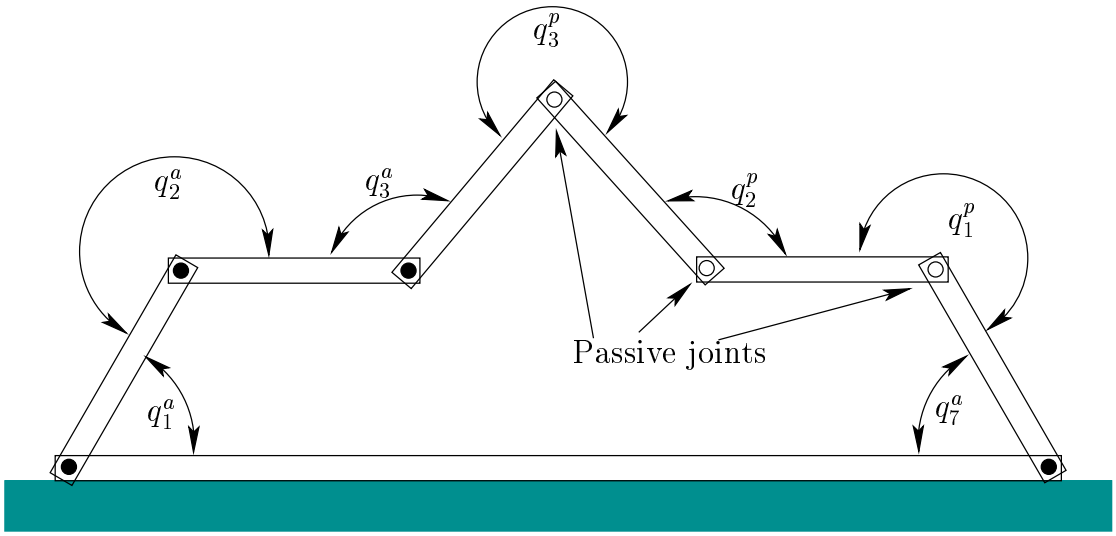


Figure 7.24: Three of the joint variables can be determined automatically by inverse kinematics. Therefore, four of the joints be designated as *active*, and the remaining three will be passive.

trials.

Loop generator The *random loop generator* greatly improves the chance of obtaining closure by iteratively restricting the range on each of the active variables. The method requires that the active variables appear sequentially along the chain (i.e., there is no interleaving of active and passive variables). The m coordinates of q^a are obtained sequentially as follows. First, compute an interval, I_1, of allowable values for q_1^a. The interval serves as a loose bound in the sense that, for any value $q_1^a \notin I_1$, it is known for certain that closure cannot be obtained. This is ensured by performing a careful geometric analysis of the linkage, which will be explained shortly. The next step is to generate a sample in $q_1^a \in I_1$, which is accomplished in [244] by picking a random point in I_1. Using the value q_1^a, a bounding interval I_2 is computed for allowable values of q_2^a. The value q_2^a is obtained by sampling in I_2. This process continues iteratively until I_m and q_m^a are obtained, unless it terminates early because some $I_i = \emptyset$ for $i < m$. If successful termination occurs, then the active variables q^a are used to find values q^p for the passive variables. This step still might fail, but the probability of success is now much higher. The method can also be applied to linkages in which there are multiple, common loops, as in the Stewart-Gough platform, by breaking the linkage into a tree and closing loops one at a time using the loop generator. The performance depends on how the linkage is decomposed [244].

Computing bounds on joint angles The main requirement for successful application of the method is the ability to compute bounds on how far a chain of links can travel in $\mathcal{W}$ over some range of joint variables. For example, for a planar

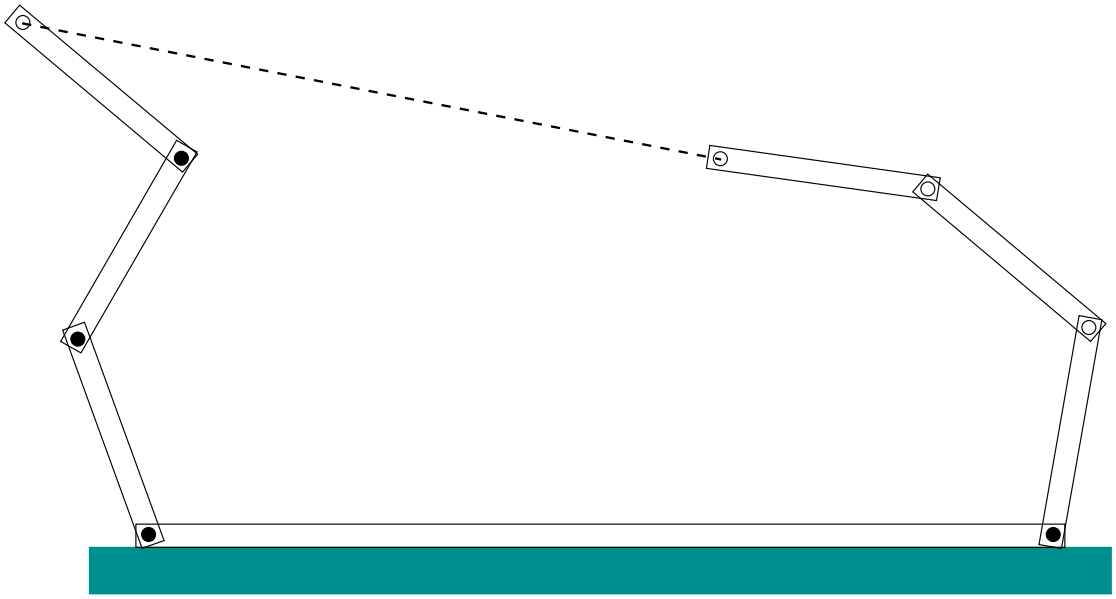
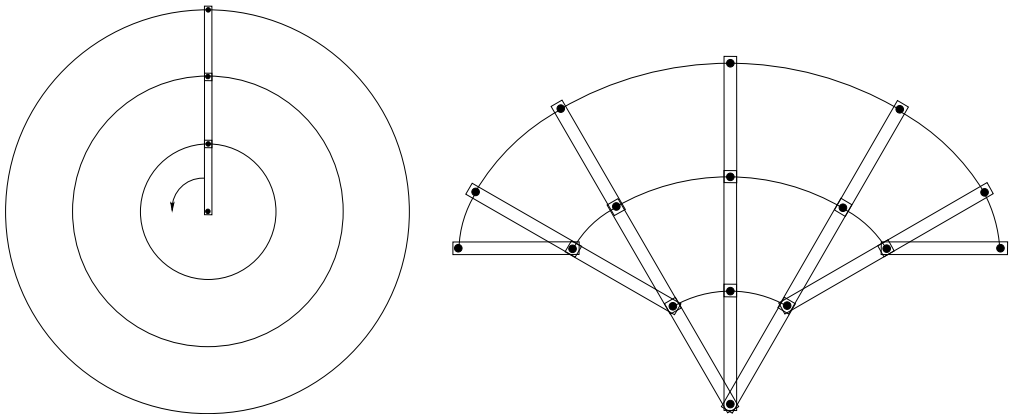


Figure 7.25: In this case, the active variables are chosen in a way that makes it impossible to assign passive variables that close the loop.

chain that has revolute joints with no limits, the chain can sweep out a circle as shown in Figure 7.26a. Suppose it is known that the angle between links must remain between $-\pi/6$ and $\pi/6$. A tighter bounding region can be obtained, as shown in Figure 7.26b. Three-dimensional versions of these bounds, along with many necessary details, are included in [244]. These bounds are then used to compute I_i in each iteration of the sampling algorithm.

Now that there is an efficient method that generates samples directly in $\mathcal{C}_{clo}$, it is straightforward to adapt any of the sampling-based planning methods of Chapter 5. In [244] many impressive results are obtained for challenging problems that have the dimension of $\mathcal{C}$ up to 97 and the dimension of $\mathcal{C}_{clo}$ up to 25; see Figure 7.27. These methods are based on applying the new sampling techniques to the RDTs of Section 5.5 and the visibility sampling-based roadmap of Section 5.6.2. For these algorithms, the local planning method is applied to the active variables, and inverse kinematics algorithms are used for the passive variables in the path validation step. This means that inverse kinematics and collision checking are performed together, instead of only collision checking, as described in Section 5.3.4.

One important issue that has been neglected in this section is the existence of *kinematic singularities*, which cause the dimension of $\mathcal{C}_{clo}$ to drop in the vicinity of certain points. The methods presented here have assumed that solving the motion planning problem does not require passing through a singularity. This assumption is reasonable for robot systems that have many extra degrees of freedom, but it is important to understand that completeness is lost in general because the sampling-based methods do not explicitly handle these degeneracies. In a



(a) (b)

Figure 7.26: (a) If any joint angle is possible, then the links sweep out a circle in the limit. (b) If there are limits on the joint angles, then a tighter bound can be obtained for the reachability of the linkage.

sense, they occur below the level of sampling resolution. For more information on kinematic singularities and related issues, see [693].

7.5 Folding Problems in Robotics and Biology

A growing number of motion planning applications involve some form of folding. Examples include automated carton folding, computer-aided drug design, protein folding, modular reconfigurable robots, and even robotic origami. These problems are generally modeled as a linkage in which all bodies are connected by revolute joints. In robotics, self-collision between pairs of bodies usually must be avoided. In biological applications, energy functions replace obstacles. Instead of crisp obstacle boundaries, energy functions can be imagined as "soft" obstacles, in which a real value is defined for every $q \in \mathcal{C}$, instead of defining a set $\mathcal{C}_{obs} \subset \mathcal{C}$. For a given threshold value, such energy functions can be converted into an obstacle region by defining $\mathcal{C}_{obs}$ to be the configurations that have energy above the threshold. However, the energy function contains more information because such thresholds are arbitrary. This section briefly shows some examples of folding problems and techniques from the recent motion planning literature.

Carton folding An interesting application of motion planning to the automated folding of boxes is presented in [661]. Figure 7.28 shows a carton in its original flat form and in its folded form. As shown in Figure 7.29, the problem can be modeled as a tree of bodies connected by revolute joints. Once this model has been formulated, many methods from Chapters 5 and 6 can be adapted for this problem. In [661], a planning algorithm optimized particularly for box folding is presented.

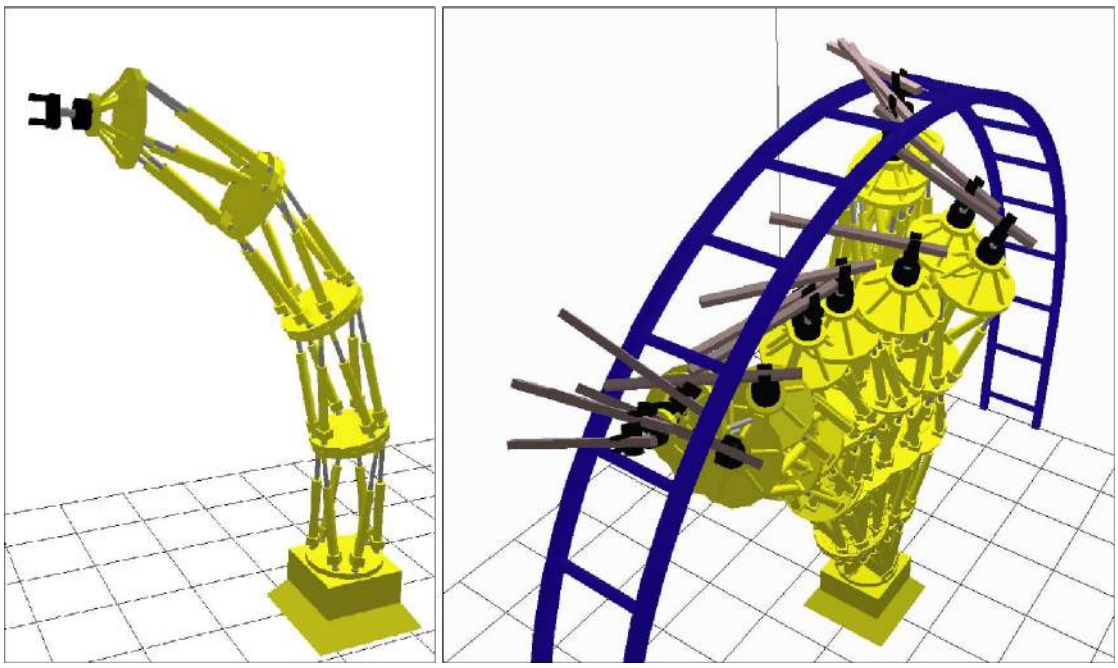


Figure 7.27: Planning for the Logabex LX4 robot [187]. This solution was computed in less than a minute by applying active-passive decomposition to an RDT-based planner [244]. In this example, the dimension of $\mathcal{C}$ is 97 and the dimension of $\mathcal{C}_{clo}$ is 25.

It is an adaptation of an approximate cell decomposition algorithm developed for kinematic chains in [658]. Its complexity is exponential in the degrees of freedom of the carton, but it gives good performance on practical examples. One such solution that was found by motion planning is shown in Figure 7.30. To use these solutions in a factory, the manipulation problem has to be additionally considered. For example, as demonstrated in [661], a manipulator arm robot can be used in combination with a well-designed set of fixtures. The fixtures help hold the carton in place while the manipulator applies pressure in the right places, which yields the required folds. Since the feasibility with fixtures depends on the particular folding path, the planning algorithm generates all possible distinct paths from the

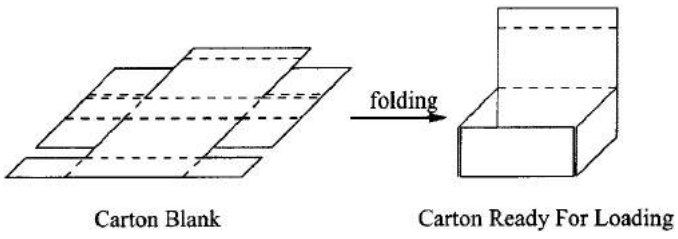


Figure 7.28: An important packaging problem is to automate the folding of a perforated sheet of cardboard into a carton.

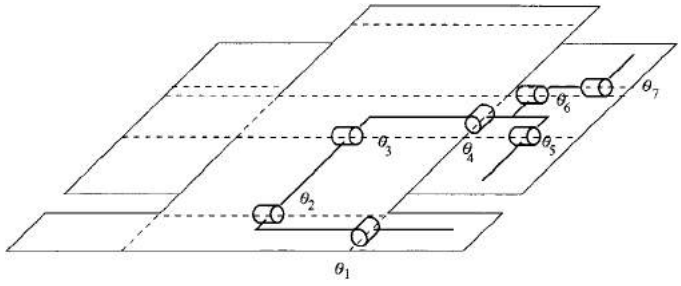


Figure 7.29: The carton can be cleverly modeled as a tree of bodies that are attached by revolute joints.

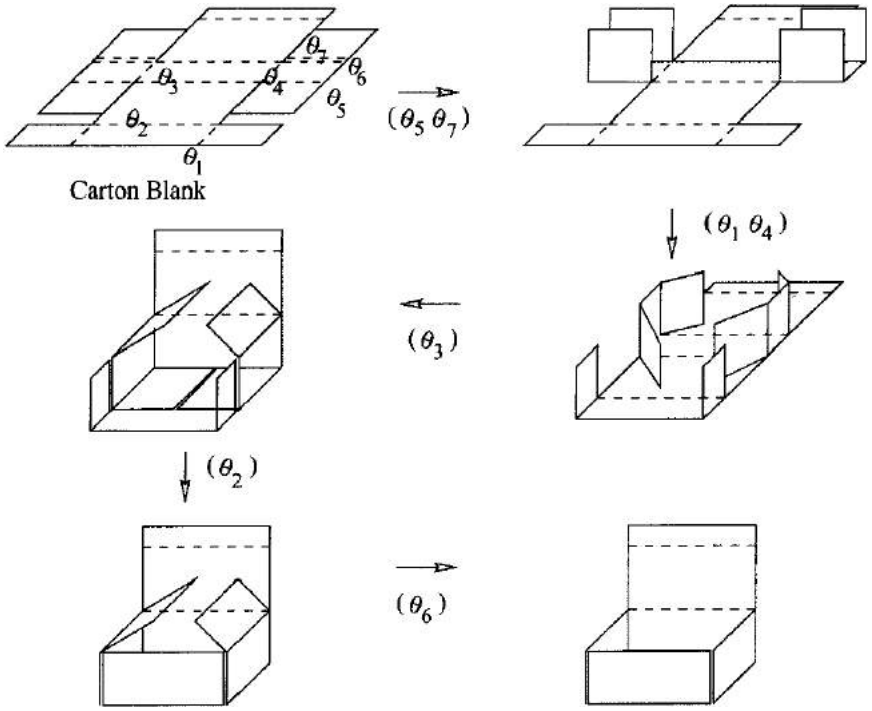


Figure 7.30: A folding sequence that was computed using the algorithm in [661].

initial configuration (at which the box is completely unfolded).

Simplifying knots A *knot* is a closed curve that does not intersect itself, is embedded in $\mathbb{R}^3$, and cannot be untangled to produce a simple loop (such as a circular path). If the knot is allowed to intersect itself, then any knot can be untangled; therefore, a careful definition of what it means to untangle a knot is needed. For a closed curve, $\tau : [0,1] \rightarrow \mathbb{R}^3$, embedded in $\mathbb{R}^3$ (it cannot intersect itself), let the set $\mathbb{R}^3 \setminus \tau([0,1])$ of points not reached by the curve be called the *ambient space* of τ. In knot theory, an *ambient isotopy* between two closed curves, τ_1 and τ_2, embedded in $\mathbb{R}^3$ is a homeomorphism between their ambient spaces. Intuitively, this means that τ_1 can be warped into τ_2 without allowing any self-intersections. Therefore, determining whether two loops are equivalent seems closely related to motion planning. Such equivalence gives rise to groups that characterize the space of knots and are closely related to the fundamental group described in Section 4.1.3. For more information on knot theory, see [8, 451, 511].

A motion planning approach was developed in [571] to determine whether a closed curve is equivalent to the *unknot*, which is completely untangled. This can be expressed as a curve that maps onto $\mathbb{S}^1$, embedded in $\mathbb{R}^3$. The algorithm takes as input a knot expressed as a circular chain of line segments embedded in $\mathbb{R}^3$. In this case, the unknot can be expressed as a triangle in $\mathbb{R}^3$. One of the most challenging examples solved by the planner is shown in Figure 7.31. The planner is sampling-based and shares many similarities with the RDT algorithm of Section 5.5 and the Ariadne's clew and expansive space planners described in Section 5.4.4. Since the task is not to produce a collision-free path, there are several unique aspects in comparison to motion planning. An energy function is defined over the collection of segments to try to guide the search toward simpler configurations. There are two kinds of local operations that are made by the planner: 1) Try to move a vertex toward a selected subgoal in the ambient space. This is obtained by using random sampling to grow a search tree. 2) Try to delete a vertex, and connect the neighboring vertices by a straight line. If no collision occurs along the intermediate configurations, then the knot has been simplified. The algorithm terminates when it is unable to further simplify the knot.

Drug design A sampling-based motion planning approach to pharmaceutical drug design is taken in [601]. The development of a drug is a long, incremental process, typically requiring years of research and experimentation. The goal is to find a relatively small molecule called a *ligand*, typically comprising a few dozen atoms, that docks with a receptor cavity in a specific protein [615]; Figure 1.14 (Section 1.2) illustrated this. Examples of drug molecules were also given in Figure 1.14. Protein-ligand docking can stimulate or inhibit some biological activity, ultimately leading to the desired pharmacological effect. The problem of finding suitable ligands is complicated due to both energy considerations and the flexibility of the ligand. In addition to satisfying structural considerations, factors such as synthetic accessibility, drug pharmacology and toxicology greatly complicate and

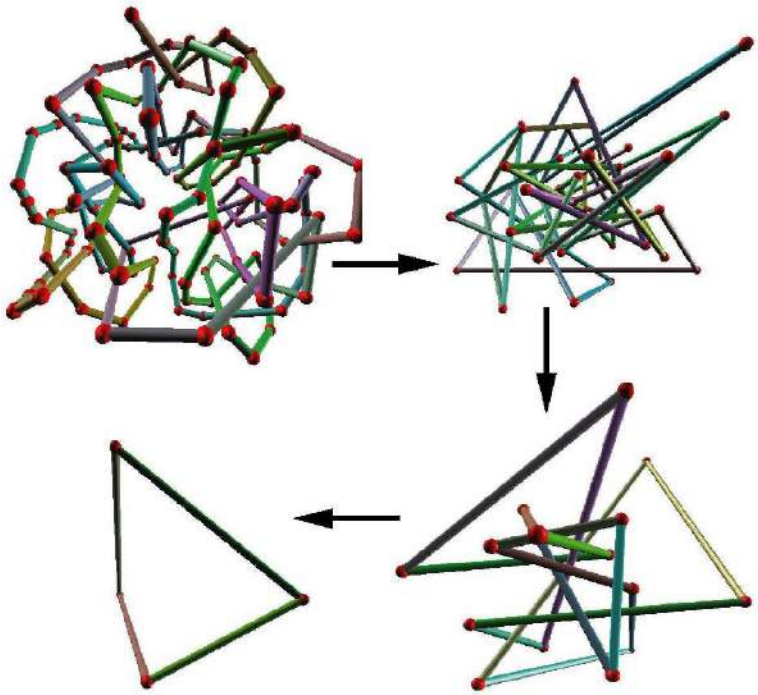


Figure 7.31: The planner in [571] untangles the famous Ochiai unknot benchmark in a few minutes on a standard PC.

lengthen the search for the most effective drug molecules.

One popular model used by chemists in the context of drug design is a *pharmacophore*, which serves as a template for the desired ligand [229, 339, 383, 860]. The pharmacophore is expressed as a set of *features* that an effective ligand should possess and a set of *spatial constraints* among the features. Examples of features are specific atoms, centers of benzene rings, positive or negative charges, hydrophobic or hydrophilic centers, and hydrogen bond donors or acceptors. Features generally require that parts of the molecule must remain fixed in $\mathbb{R}^3$, which induces kinematic closure constraints. These features are developed by chemists to encapsulate the assumption that ligand binding is due primarily to the interaction of some features of the ligand to "complementary" features of the receptor. The interacting features are included in the pharmacophore, which is a template for screening candidate drugs, and the rest of the ligand atoms merely provide a scaffold for holding the pharmacophore features in their spatial positions. Figure 7.32 illustrates the pharmacophore concept.

Candidate drug molecules (ligands), such as the ones shown in Figure 1.14, can be modeled as a tree of bodies as shown in Figure 7.33. Some bonds can rotate, yielding revolute joints in the model; other bonds must remain fixed. The drug design problem amounts to searching the space of configurations (called *conformations*) to try to find a low-energy configuration that also places certain atoms

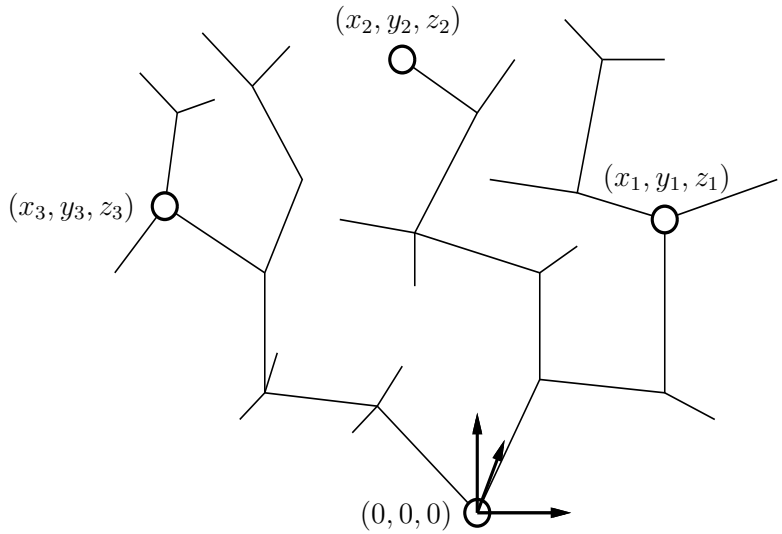


Figure 7.32: A pharmacophore is a model used by chemists to simplify the interaction process between a ligand (candidate drug molecule) and a protein. It often amounts to holding certain features of the molecule fixed in $\mathbb{R}^3$. In this example, the positions of three atoms must be fixed relative to the body frame of an arbitrarily designated root atom. It is assumed that these features interact with some complementary features in the cavity of the protein.

in specified locations in $\mathbb{R}^3$. This additional constraint arises from the pharmacophore and causes the planning to occur on $\mathcal{C}_{clo}$ from Section 7.4 because the pharmacophores can be expressed as closure constraints.

An energy function serves a purpose similar to that of a collision detector. The evaluation of a ligand for drug design requires determining whether it can achieve low-energy conformations that satisfy the pharmacophore constraints. Thus, the task is different from standard motion planning in that there is no predetermined goal configuration. One of the greatest difficulties is that the energy functions are extremely complicated, nonlinear, and empirical. Here is typical example (used in [601]):

$$\begin{aligned} e(q) = \quad & \textstyle\sum_{bonds} \frac{1}{2} K_b (R - R')^2 + \sum_{ang} \frac{1}{2} K_a (\alpha - \alpha')^2 + \\ & \textstyle\sum_{torsions} K_d [1 + \cos(p\theta - \theta')] + \\ & \textstyle\sum_{i,j} \left\{ 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \frac{c_i c_j}{\epsilon r_{ij}} \right\}. \end{aligned} \tag{7.25}$$

The energy accounts for torsion-angle deformations, van der Waals potential, and Coulomb potentials. In (7.25), the first sum is taken over all bonds, the second over all bond angles, the third over all rotatable bonds, and the last is taken over all pairs of atoms. The variables are the force constants, K_b, K_a, and K_d; the dielectric constant, ϵ; a periodicity constant, p; the Lennard-Jones radii, σ_{ij}; well depth, ϵ_{ij}; partial charge, c_i; measured bond length, R; equilibrium bond length,

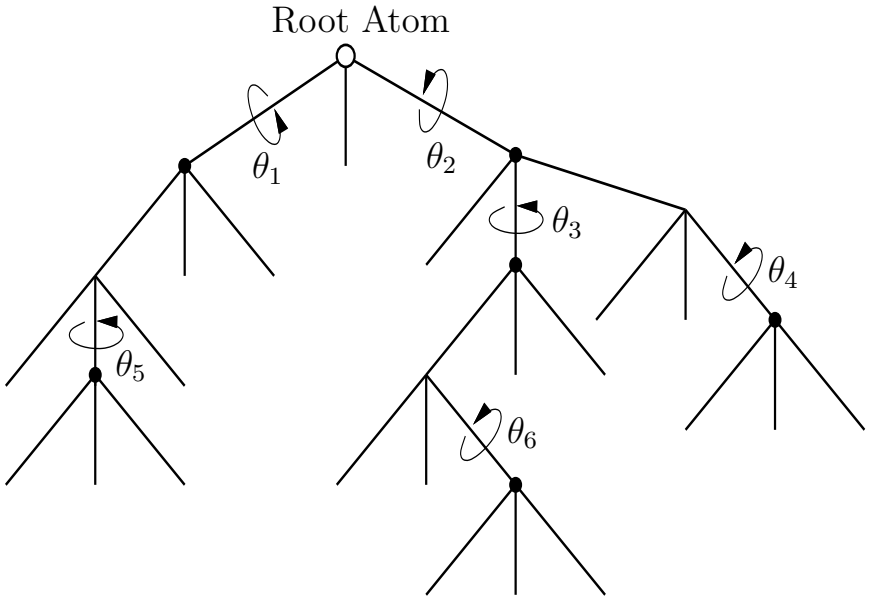


Figure 7.33: The modeling of a flexible molecule is similar to that of a robot. One atom is designated as the root, and the remaining bodies are arranged in a tree. If there are cyclic chains in the molecules, then constraints as described in Section 4.4 must be enforced. Typically, only some bonds are capable of rotation, whereas others must remain rigid.

R'; measured bond angle, α; equilibrium bond angle, α'; measured torsional angle, θ; equilibrium torsional angle, θ'; and distance between atom centers, r_{ij}. Although the energy expression is very complicated, it only depends on the configuration variables; all others are constants that are estimated in advance.

Protein folding In computational biology, the problem of protein folding shares many similarities with drug design in that the molecules have rotatable bonds and energy functions are used to express good configurations. The problems are much more complicated, however, because the protein molecules are generally much larger than drug molecules. Instead of a dozen degrees of freedom, which is typical for a drug molecule, proteins have hundreds or thousands of degrees of freedom. When proteins appear in nature, they are usually in a folded, low-energy configuration. The *structure problem* involves determining precisely how the protein is folded so that its biological activity can be completely understood. In some studies, biologists are even interested in the pathway that a protein takes to arrive in its folded state [24, 25]. This leads directly to an extension of motion planning that involves arriving at a goal state in which the molecule is folded. In [24, 25], sampling-based planning algorithms were applied to compute folding pathways for proteins. The protein starts in an unfolded configuration and must arrive in a specified folded configuration without violating energy constraints along the way. Figure 7.34 shows an example from [25]. That work also draws interesting

connections between protein folding and box folding, which was covered previously.

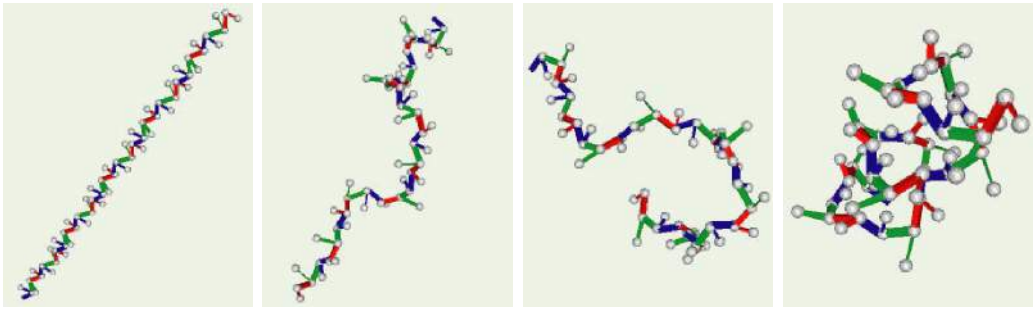


Figure 7.34: Protein folding for a polypeptide, computed by a sampling-based roadmap planning algorithm [24]

7.6 Coverage Planning

Imagine automating the motion of a lawnmower for an estate that has many obstacles, such as a house, trees, garage, and a complicated property boundary. What are the best zig-zag motions for the lawnmower? Can the amount of redundant traversals be minimized? Can the number of times the lawnmower needs to be stopped and rotated be minimized? This is one example of *coverage planning*, which is motivated by applications such as lawn mowing, automated farming, painting, vacuum cleaning, and mine sweeping. A survey of this area appears in [217]. Even for a region in $\mathcal{W} = \mathbb{R}^2$, finding an optimal-length solution to coverage planning is NP-hard, by reduction to the closely related Traveling Salesman Problem [36, 709]. Therefore, we are willing to tolerate approximate or even heuristic solutions to the general coverage problem, even in $\mathbb{R}^2$.

Boustrophedon decomposition One approach to the coverage problem is to decompose $\mathcal{C}_{free}$ into cells and perform boustrophedon (from the Greek "ox turning") motions in each cell as shown in Figure 7.35 [222]. It is assumed that the robot is a point in $\mathcal{W} = \mathbb{R}^2$, but it carries a *tool* of thickness ϵ that hangs evenly over the sides of the robot. This enables it to paint or mow part of $\mathcal{C}_{free}$ up to distance $\epsilon/2$ from either side of the robot as it moves forward. Such motions are often used in printers to reduce the number of carriage returns.

If $\mathcal{C}_{obs}$ is polygonal, a reasonable decomposition can be obtained by adapting the vertical decomposition method of Section 6.2.2. In that algorithm, critical events were defined for several cases, some of which are not relevant for the boustrophedon motions. The only events that need to be handled are shown in Figure 7.36a [216]. This produces a decomposition that has fewer cells, as shown in Figure 7.36b. Even though the cells are nonconvex, they can always be sliced nicely into vertical strips, which makes them suitable for boustrophedon motions. The original vertical decomposition could also be used, but the extra cell boundaries

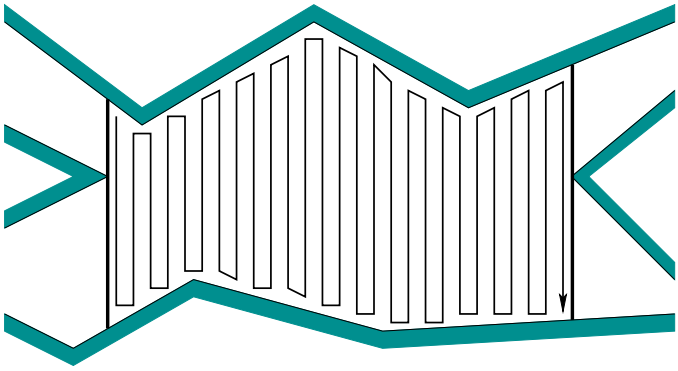


Figure 7.35: An example of the ox plowing motions.

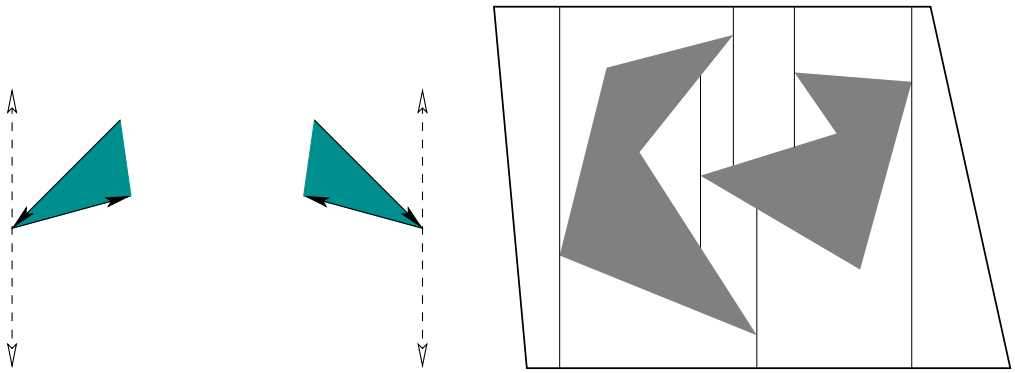


Figure 7.36: (a) Only the first case from Figure 6.2 is needed: extend upward and downward. All other cases are neglected. (b) The resulting decomposition is shown, which has fewer cells than that of the vertical decomposition in Figure 6.3.

would cause unnecessary repositioning of the robot. A similar method, which furthermore optimizes the number of robot turns, is presented in [468].

Spanning tree covering An interesting approximate method was developed by Gabriely and Rimon; it places a tiling of squares inside of $\mathcal{C}_{free}$ and computes the spanning tree of the resulting connectivity graph [372, 373]. Suppose again that $\mathcal{C}_{free}$ is polygonal. Consider the example shown in Figure 7.37a. The first step is to tile the interior of $\mathcal{C}_{free}$ with squares, as shown in Figure 7.37b. Each square should be of width ϵ, for some constant $\epsilon > 0$. Next, construct a roadmap $\mathcal{G}$ by placing a vertex in the center of each square and by defining an edge that connects the centers of each pair of adjacent cubes. The next step is to compute a *spanning tree* of $\mathcal{G}$. This is a connected subgraph that has no cycles and touches every vertex of $\mathcal{G}$; it can be computed in $O(n)$ time, if $\mathcal{G}$ has n edges [683]. There are many possible spanning trees, and a criterion can be defined and optimized to induce preferences. One possible spanning tree is shown Figure 7.37c.

Once the spanning tree is made, the robot path is obtained by starting at a

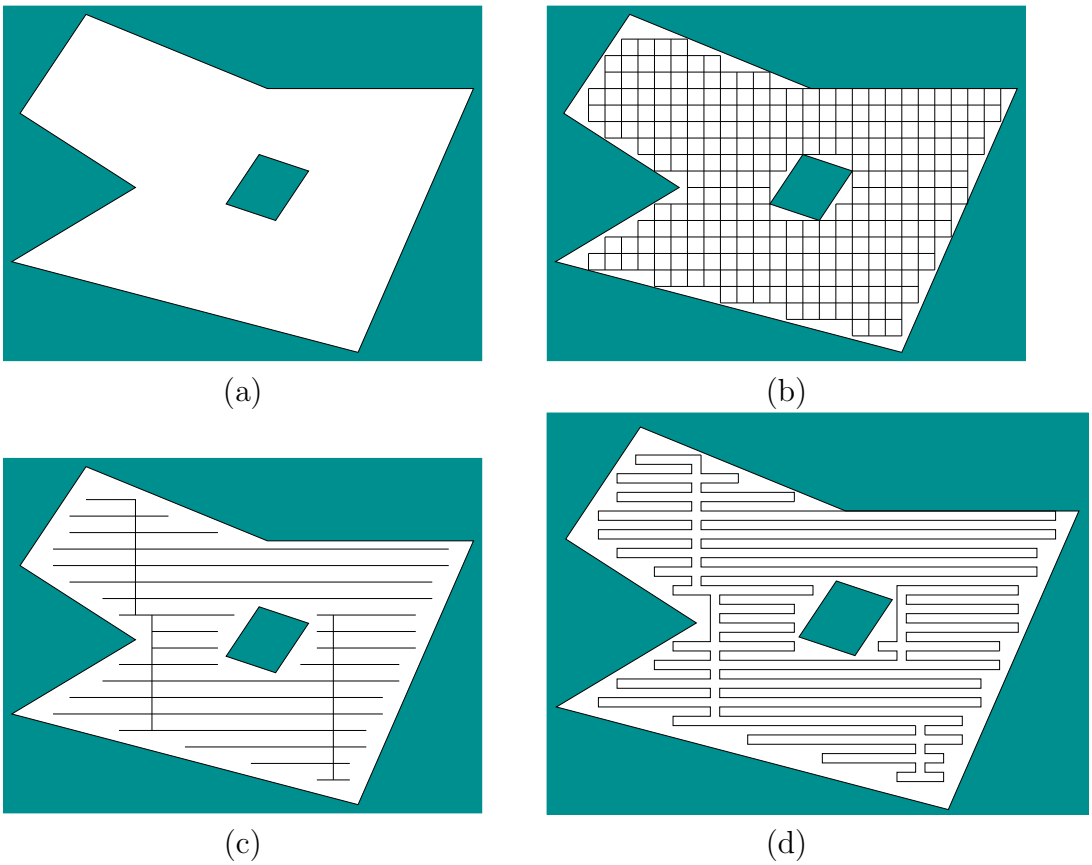


Figure 7.37: (a) An example used for spanning tree covering. (b) The first step is to tile the interior with squares. (c) The spanning tree of a roadmap formed from grid adjacencies. (d) The resulting coverage path.

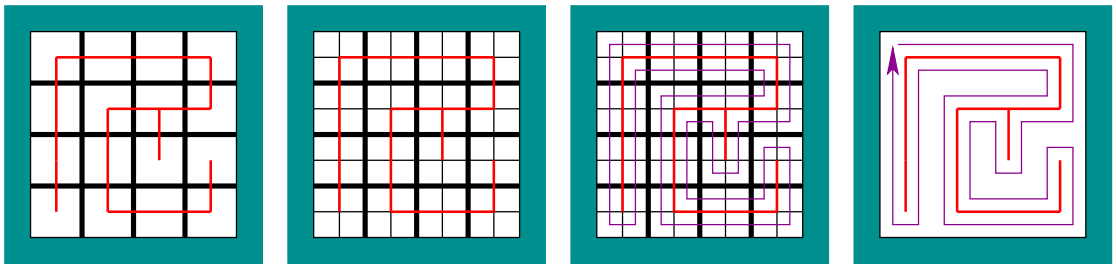


Figure 7.38: A circular path is made by doubling the resolution and following the perimeter of the spanning tree.

point near the spanning tree and following along its perimeter. This path can be precisely specified as shown in Figure 7.38. Double the resolution of the tiling, and form the corresponding roadmap. Part of the roadmap corresponds to the spanning tree, but also included is a loop path that surrounds the spanning tree. This path visits the centers of the new squares. The resulting path for the example of Figure 7.37a is shown in Figure 7.37d. In general, the method yields an optimal route, once the approximation is given. A bound on the uncovered area due to approximation is given in [372]. Versions of the method that do not require an initial map are also given in [372, 373]; this involves reasoning about information spaces, which are covered in Chapter 11.

7.7 Optimal Motion Planning

This section can be considered transitional in many ways. The main concern so far with motion planning has been *feasibility* as opposed to *optimality*. This placed the focus on finding *any* solution, rather than further requiring that a solution be optimal. In later parts of the book, especially as uncertainty is introduced, optimality will receive more attention. Even the most basic forms of decision theory (the topic of Chapter 9) center on making optimal choices. The requirement of optimality in very general settings usually requires an exhaustive search over the state space, which amounts to computing continuous cost-to-go functions. Once such functions are known, a feedback plan is obtained, which is much more powerful than having only a path. Thus, optimality also appears frequently in the design of feedback plans because it sometimes comes at no additional cost. This will become clearer in Chapter 8. The quest for optimal solutions also raises interesting issues about how to approximate a continuous problem as a discrete problem. The interplay between time discretization and space discretization becomes very important in relating continuous and discrete planning problems.

7.7.1 Optimality for One Robot

Euclidean shortest paths One of the most straightforward notions of optimality is the Euclidean shortest path in $\mathbb{R}^2$ or $\mathbb{R}^3$. Suppose that $\mathcal{A}$ is a rigid body that translates only in either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$, which contains an obstacle region $\mathcal{O} \subset \mathcal{W}$. Recall that, ordinarily, $\mathcal{C}_{free}$ is an open set, which means that any path, $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$, can be shortened. Therefore, shortest paths for motion planning must be defined on the closure $\mathrm{cl}(\mathcal{C}_{free})$, which allows the robot to make contact with the obstacles; however, their interiors must not intersect.

For the case in which $\mathcal{C}_{obs}$ is a polygonal region, the shortest-path roadmap method of Section 6.2.4 has already been given. This can be considered as a kind of multiple-query approach because the roadmap completely captures the structure needed to construct the shortest path for any query. It is possible to make a single-query algorithm using the *continuous Dijkstra paradigm* [443, 708]. This method propagates a *wavefront* from q_I and keeps track of critical events

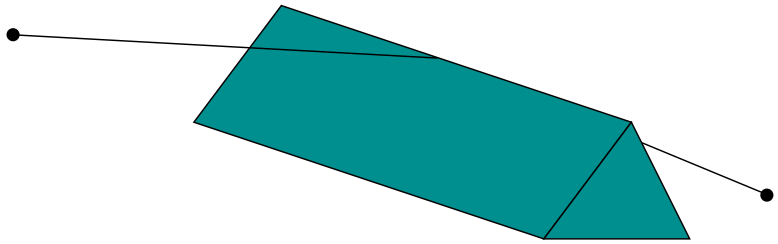


Figure 7.39: For a polyhedral environment, the shortest paths do not have to cross vertices. Therefore, the shortest-path roadmap method from Section 6.2.4 does not extend to three dimensions.

in maintaining the wavefront. As events occur, the wavefront becomes composed of *wavelets*, which are arcs of circles centered on obstacle vertices. The possible events that can occur are 1) a wavelet disappears, 2) a wavelet collides with an obstacle vertex, 3) a wavelet collides with another wavelet, or 4) a wavelet collides with a point in the interior of an obstacle edge. The method can be made to run in time $O(n \lg n)$ and uses $O(n \lg n)$ space. A roadmap is constructed that uses $O(n)$ space. See Section 8.4.3 for a related method.

Such elegant methods leave the impression that finding shortest paths is not very difficult, but unfortunately they do not generalize nicely to $\mathbb{R}^3$ and a polyhedral $\mathcal{C}_{obs}$. Figure 7.39 shows a simple example in which the shortest path does not have to cross a vertex of $\mathcal{C}_{obs}$. It may cross anywhere in the interior of an edge; therefore, it is not clear where to draw the bitangent lines that would form the shortest-path roadmap. The lower bounds for this problem are also discouraging. It was shown in [172] that the 3D shortest-path problem in a polyhedral environment is NP-hard. Most of the difficulty arises because of the precision required to represent 3D shortest paths. Therefore, efficient polynomial-time approximation algorithms exist [215, 763].

General optimality criteria It is difficult to even define optimality for more general C-spaces. What does it mean to have a shortest path in $SE(2)$ or $SE(3)$? Consider the case of a planar, rigid robot that can translate and rotate. One path could minimize the amount of rotation whereas another tries to minimize the amount of translation. Without more information, there is no clear preference. Ulam's distance is one possibility, which is to minimize the distance traveled by k fixed points [474]. In Chapter 13, differential models will be introduced, which lead to meaningful definitions of optimality. For example, the shortest paths for a slow-moving car are shown in Section 15.3; these require a precise specification of the constraints on the motion of the car (it is more costly to move a car sideways than forward).

This section formulates some optimal motion planning problems, to provide a smooth transition into the later concepts. Up until now, actions were used in Chapter 2 for discrete planning problems, but they were successfully avoided for basic motion planning by directly describing paths that map into $\mathcal{C}_{free}$. It will be

convenient to use them once again. Recall that they were convenient for defining costs and optimal planning in Section 2.3.

To avoid for now the complications of differential equations, consider making an approximate model of motion planning in which every path must be composed of a sequence of shortest-path segments in $\mathcal{C}_{free}$. Most often these are line segments; however, for the case of $SO(3)$, circular arcs obtained by spherical linear interpolation may be preferable. Consider extending Formulation 2.3 from Section 2.3.2 to the problem of motion planning.

Let the C-space $\mathcal{C}$ be embedded in $\mathbb{R}^m$ (i.e., $\mathcal{C} \subset \mathbb{R}^m$). An action will be defined shortly as an m-dimensional vector. Given a scaling constant ϵ and a configuration q, an action u produces a new configuration, $q' = q + \epsilon u$. This can be considered as a *configuration transition equation*, $q' = f(q, u)$. The path segment represented by the action u is the shortest path (usually a line segment) between q and q'. Following Section 2.3, let π_K denote a *K-step plan*, which is a sequence (u_1, u_2, …, u_K) of K actions. Note that if π_K and q_I are given, then a sequence of states, q_1, q_2, …, q_{K+1}, can be derived using f. Initially, $q_1 = q_I$, and each following state is obtained by $q_{k+1} = f(q_k, u_k)$. From this a path, $\tau : [0, 1] \rightarrow \mathcal{C}$, can be derived.

An approximate optimal planning problem is formalized as follows:

Formulation 7.4 (Approximate Optimal Motion Planning)

1. The following components are defined the same as in Formulation 4.1: $\mathcal{W}$, $\mathcal{O}$, $\mathcal{A}$, $\mathcal{C}$, $\mathcal{C}_{obs}$, $\mathcal{C}_{free}$, and q_I. It is assumed that $\mathcal{C}$ is an n-dimensional manifold.

2. For each $q \in \mathcal{C}$, a possibly infinite *action space*, $U(q)$. Each $u \in U$ is an n-dimensional vector.

3. A positive constant $\epsilon > 0$ called the *step size*.

4. A set of *stages*, each denoted by k, which begins at $k = 1$ and continues indefinitely. Each stage is indicated by a subscript, to obtain q_k and u_k.

5. A *configuration transition function* $f(q, u) = q + \epsilon u$ in which $q + \epsilon u$ is computed by vector addition on $\mathbb{R}^m$.

6. Instead of a goal state, a goal region X_G is defined.

7. Let L denote a real-valued cost functional, which is applied to a K-step plan, π_K. This means that the sequence $(u_1, \ldots, u_K)$ of actions and the sequence $(q_1, \ldots, q_{K+1})$ of configurations may appear in an expression of L. Let $F = K + 1$. The *cost functional* is

$$L(\pi_K) = \sum_{k=1}^{K} l(q_k, u_k) + l_F(q_F). \tag{7.26}$$

   The final term $l_F(q_F)$ is outside of the sum and is defined as $l_F(q_F) = 0$ if $q_F \in X_G$ and $l_F(q_F) = \infty$ otherwise. As in Formulation 2.3, K is not necessarily a constant.

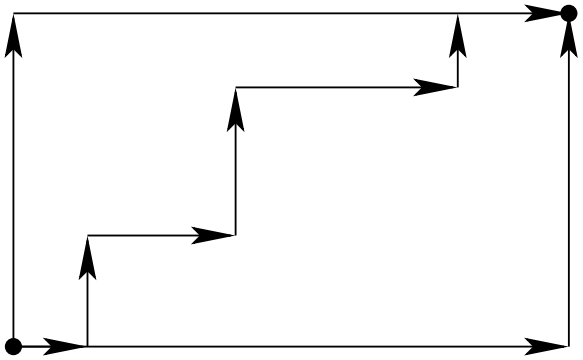


Figure 7.40: Under the Manhattan (L_1) motion model, all monotonic paths that follow the grid directions have equivalent length.

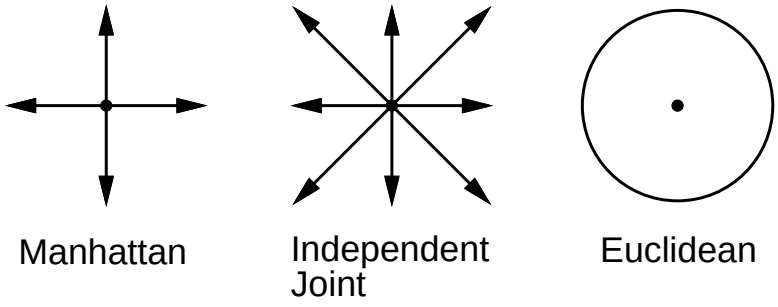


Figure 7.41: Depictions of the action sets, $U(q)$, for Examples 7.4, 7.5, and 7.6.

8. Each $U(q)$ contains the special *termination action* u_T, which behaves the same way as in Formulation 2.3. If u_T is applied to q_k at stage k, then the action is repeatedly applied forever, the configuration remains in q_k forever, and no more cost accumulates.

The task is to compute a sequence of actions that optimizes (7.26). Formulation 7.4 can be used to define a variety of optimal planning problems. The parameter ϵ can be considered as the resolution of the approximation. In many formulations it can be interpreted as a *time step*, $\epsilon = \Delta t$; however, note that no explicit time reference is necessary because the problem only requires constructing a path through $\mathcal{C}_{free}$. As ϵ approaches zero, the formulation approaches an exact optimal planning problem. To properly express the exact problem, differential equations are needed. This is deferred until Part IV.

Example 7.4 (Manhattan Motion Model) Suppose that in addition to u_T, the action set $U(q)$ contains $2n$ vectors in which only one component is nonzero and must take the value 1 or -1. For example, if $\mathcal{C} = \mathbb{R}^2$, then

$$U(q) = \{(1, 0), (-1, 0), (0, -1), (0, 1), u_T\}. \tag{7.27}$$

When used in the configuration transition equation, this set of actions produces "up," "down," "left," and "right" motions and a "terminate" command. This

produces a topological graph according to the 1-neighborhood model, (5.37), which was given in Section 5.4.2. The action set for this example and the following two examples are shown in Figure 7.41 for comparison. The cost term $l(q_k, u_k)$ is defined to be 1 for all $q_k \in \mathcal{C}_{free}$ and u_k. If $q_k \in \mathcal{C}_{obs}$, then $l(q_k, u_k) = \infty$. Note that the set of configurations reachable by these actions lies on a grid, in which the spacing between 1-neighbors is ϵ. This corresponds to a convenient special case in which time discretization (implemented by ϵ) leads to a regular space discretization. Consider Figure 7.40. It is impossible to take a shorter path along a diagonal because the actions do not allow it. Therefore, all monotonic paths along the grid produce the same costs.

Optimal paths can be obtained by simply applying the dynamic programming algorithms of Chapter 2. This example provides a nice unification of concepts from Section 2.2, which introduced grid search, and Section 5.4.2, which explained how to adapt search methods to motion planning. In the current setting, only algorithms that produce optimal solutions on the corresponding graph are acceptable.

This form of optimization might not seem relevant because it does not represent the Euclidean shortest-path problem for $\mathbb{R}^2$. The next model adds more actions, and does correspond to an important class of optimization problems in robotics. ■

Example 7.5 (Independent-Joint Motion Model) Now suppose that $U(q)$ includes u_T and the set of all 3^n vectors for which every element is either -1, 0, or 1. Under this model, a path can be taken along any diagonal. This still does not change the fact that all reachable configurations lie on a grid. Therefore, the standard grid algorithms can be applied once again. The difference is that now there are $3^n - 1$ edges emanating from every vertex, as opposed to $2n$ in Example 7.4. This model is appropriate for robots that are constructed from a collection of links attached by revolute joints. If each joint is operated independently, then it makes sense that each joint could be moved either forward, backward, or held stationary. This corresponds exactly to the actions. However, this model cannot nicely approximate Euclidean shortest paths; this motivates the next example. ■

Example 7.6 (Euclidean Motion Model) To approximate Euclidean shortest paths, let $U(q) = \mathbb{S}^{n-1} \cup \{u_T\}$, in which $\mathbb{S}^{n-1}$ is the m-dimensional unit sphere centered at the origin of $\mathbb{R}^n$. This means that in k stages, any piecewise-linear path in which each segment has length ϵ can be formed by a sequence of inputs. Therefore, the set of reachable states is no longer confined to a grid. Consider taking $\epsilon = 1$, and pick any point, such as $(\pi, \pi) \in \mathbb{R}^2$. How close can you come to this point? It turns out that the set of points reachable with this model is dense in $\mathbb{R}^n$ if obstacles are neglected. This means that we can come arbitrarily close to any point in $\mathbb{R}^n$. Therefore, a finite grid cannot be used to represent the problem. Approximate solutions can still be obtained by numerically computing an optimal cost-to-go function over $\mathcal{C}$. This approach is presented in Section 8.5.2.

One additional issue for this problem is the precision defined for the goal. If the goal region is very small relative to ϵ, then complicated paths may have to be selected to arrive precisely at the goal. ■

Example 7.7 (Weighted-Region Problem) In outdoor and planetary navigation applications, it does not make sense to define obstacles in the crisp way that has been used so far. For each patch of terrain, it is more convenient to associate a cost that indicates the estimated difficulty of its traversal. This is sometimes considered as a "grayscale" model of obstacles. The model can be easily captured in the cost term $l(q_k, u_k)$. The action spaces can be borrowed from Examples 7.4 or 7.5. Stentz's algorithm [913], which is introduced in Section 12.3.2, generates optimal navigation plans for this problem, even assuming that the terrain is initially unknown. Theoretical bounds for optimal weighted-region planning problems are given in [709, 710]. An approximation algorithm appears in [820]. ■

7.7.2 Multiple-Robot Optimality

Suppose that there are two robots as shown in Figure 7.42. There is just enough room to enable the robots to translate along the corridors. Each would like to arrive at the bottom, as indicated by arrows; however, only one can pass at a time through the horizontal corridor. Suppose that at any instant each robot can either be *on* or *off*. When it is *on*, it moves at its maximum speed, and when it is *off*, it is stopped.[4] Now suppose that each robot would like to reach its goal as quickly as possible. This means each would like to minimize the total amount of time that it is *off*. In this example, there appears to be only two sensible choices: 1) $\mathcal{A}_1$ stays *on* and moves straight to its goal while $\mathcal{A}_2$ is *off* just long enough to let $\mathcal{A}_1$ pass, and then moves to its goal. 2) The opposite situation occurs, in which $\mathcal{A}_2$ stays *on* and $\mathcal{A}_1$ must wait. Note that when a robot waits, there are multiple locations at which it can wait and still yield the same time to reach the goal. The only important information is how long the robot was *off*.

Thus, the two interesting plans are that either $\mathcal{A}_2$ is *off* for some amount of time, $t_{off} > 0$, or $\mathcal{A}_1$ is *off* for time t_{off}. Consider a vector of costs of the form (L_1, L_2), in which each component represents the cost for each robot. The costs of the plans could be measured in terms of time wasted by waiting. This yields $(0, t_{off})$ and $(t_{off}, 0)$ for the cost vectors associated with the two plans (we could equivalently define cost to be the total time traveled by each robot; the time on is the same for both robots and can be subtracted from each for this simple example). The two plans are better than or equivalent to any others. Plans with this property are called *Pareto optimal* (or *nondominated*). For example, if $\mathcal{A}_2$ waits 1 second too long for $\mathcal{A}_1$ to pass, then the resulting costs are $(0, t_{off} + 1)$, which is clearly

[4]This model allows infinite acceleration. Imagine that the speeds are slow enough to allow this approximation. If this is still not satisfactory, then jump ahead to Chapter 13.

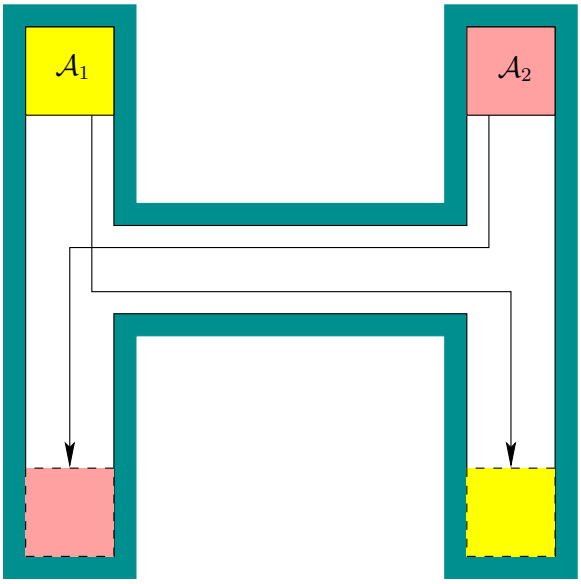


Figure 7.42: There are two Pareto-optimal coordination plans for this problem, depending on which robot has to wait.

worse than $(0, t_{off})$. The resulting plan is not Pareto optimal. More details on Pareto optimality appear in Section 9.1.1.

Another way to solve the problem is to scalarize the costs by mapping them to a single value. For example, we could find plans that optimize the average wasted time. In this case, one of the two best plans would be obtained, yielding t_{off} average wasted time. However, no information is retained about which robot had to make the sacrifice. Scalarizing the costs usually imposes some kind of artificial preference or prioritization among the robots. Ultimately, only one plan can be chosen, which might make it seem inappropriate to maintain multiple solutions. However, finding and presenting the alternative Pareto-optimal solutions could provide valuable information if, for example, these robots are involved in a complicated application that involves many other time-dependent processes. Presenting the Pareto-optimal solutions is equivalent to discarding all of the worse plans and showing the best alternatives. In some applications, priorities between robots may change, and if a scheduler of robots has access to the Pareto-optimal solutions, it is easy to change priorities by switching between Pareto-optimal plans without having to generate new plans each time.

Now the Pareto-optimality concept will be made more precise and general. Suppose there are m robots, $\mathcal{A}^1$, ..., $\mathcal{A}^m$. Let γ refer to a motion plan that gives the paths and timing functions for all robots. For each $\mathcal{A}^i$, let L_i denote its cost functional, which yields a value $L_i(\gamma) \in [0, \infty]$ for a given plan, γ. An m-dimensional vector, $L(\gamma)$, is defined as

$$L(\gamma) = (L_1(\gamma), L_2(\gamma), \ldots, L_m(\gamma)). \tag{7.28}$$

Two plans, γ and γ', are called *equivalent* if $L(\gamma) = L(\gamma')$. A plan γ is said

to *dominate* a plan γ' if they are not equivalent and $L_i(\gamma) \leq L_i(\gamma')$ for all i such that $1 \leq i \leq m$. A plan is called *Pareto optimal* if it is not dominated by any others. Since many Pareto-optimal plans may be equivalent, the task is to determine one representative from each equivalence class. This will be called finding the *unique* Pareto-optimal plans. For the example in Figure 7.42, there are two unique Pareto-optimal plans, which were already given.

Scalarization For the motion planning problem, a Pareto-optimal solution is also optimal for a scalar cost functional that is constructed as a linear combination of the individual costs. Let $\alpha_1, \ldots, \alpha_m$ be positive real constants, and let

$$l(\gamma) = \sum_{i=1}^{m} \alpha_i L_i(\gamma). \tag{7.29}$$

It is easy to show that any plan that is optimal with respect to (7.29) is also a Pareto-optimal solution [606]. If a Pareto optimal solution is generated in this way, however, there is no easy way to determine what alternatives exist.

Computing Pareto-optimal plans Since optimization for one robot is already very difficult, it may not be surprising that computing Pareto-optimal plans is even harder. For some problems, it is even possible that a continuum of Pareto-optimal solutions exist (see Example 9.3), which is very discouraging. Fortunately, for the problem of coordinating robots on topological graphs, as considered in Section 7.2.2, there is only a finite number of solutions [386]. A grid-based algorithm, which is based on dynamic programming and computes all unique Pareto-optimal coordination plans, is presented in [606]. For the special case of two polygonal robots moving on a tree of piecewise-linear paths, a complete algorithm is presented in [212].

Further Reading

This chapter covered some of the most direct extensions of the basic motion planning problem. Extensions that involve uncertainties are covered throughout Part III, and the introduction of differential constraints to motion planning is the main focus of Part IV. Numerous other extensions can be found by searching through robotics research publications or the Internet.

The treatment of time-varying motion planning in Section 7.1 assumes that all motions are predictable. Most of the coverage is based on early work [153, 506, 818, 819]; other related work includes [367, 368, 532, 812, 875, 905]. To introduce uncertainties into this scenario, see Chapter 10. The logic-based representations of Section 2.4 have been extended to *temporal logics* to allow time-varying aspects of discrete planning problems (see Part IV of [382]).

For more on multiple-robot motion planning, see [15, 34, 41, 319, 320, 345, 364, 408, 606, 780, 886]. Closely related is the problem of planning for modular reconfigurable robots [180, 209, 385, 552, 990]. In both contexts, nonpositive curvature (NPC) is an

important condition that greatly simplifies the structure of optimal paths [139, 385, 386]. For points moving on a topological graph, the topology of $\mathcal{C}_{free}$ is described in [5]. Over the last few years there has also been a strong interest in the coordination of a team or swarm of robots [165, 228, 274, 300, 305, 336, 361, 665].

The complexity of assembly planning is studied in [398, 515, 733, 972]. The problem is generally NP-hard; however, for some special cases, polynomial-time algorithms have been developed [9, 429, 973, 974]. Other works include [186, 427, 453, 458, 542].

Hybrid systems have attracted widespread interest over the past decade. Most of this work considers how to design control laws for piecewise-smooth systems [137, 634]. Early sources of hybrid control literature appear in [409]. The manipulation planning framework of Section 7.3.2 is based on [16, 17, 166]. The manipulation planning framework presented in this chapter ignores grasping issues. For analyses and algorithms for grasping, see [256, 487, 681, 781, 799, 800, 826, 827, 920]. Manipulation on a microscopic scale is considered in [126].

To read beyond Section 7.4 on sampling-based planning for closed kinematic chains, see [244, 246, 432, 979]. A complete planner for some closed chains is presented in [697]. For related work on inverse kinematics, see [309, 693]. The power of redundant degrees of freedom in robot systems was shown in [158].

Section 7.5 is a synthesis of several applications. The application of motion planning techniques to problems in computational biology is a booming area; see [24, 25, 33, 245, 514, 589, 601, 654, 1001] for some representative papers. The knot-planning coverage is based on [572]. The box-folding presentation is based on [661]. A robotic system and planning technique for creating origami is presented in [65].

The coverage planning methods presented in Section 7.6 are based on [222] and [372, 373]. A survey of coverage planning appears in [217]. Other references include [6, 7, 164, 374, 444, 468, 976]. For discrete environments, approximation algorithms for the problem of optimally visiting all states in a goal set (the *orienteering problem*) are presented and analyzed in [115, 188].

Beyond two dimensions, optimal motion planning is extremely difficult. See Section 8.5.2 for dynamic programming-based approximations. See [215, 763] for approximate shortest paths in $\mathbb{R}^3$. The weighted region problem is considered in [709, 710]. Pareto-optimal coordination is considered in [212, 386, 606].

Exercises

1. Consider the obstacle region, (7.1), in the state space for time-varying motion planning.

   (a) To ensure that X_{obs} is polyhedral, what kind of paths should be allowed? Show how the model primitives H_i that define $\mathcal{O}$ are expressed in general, using t as a parameter.

   (b) Repeat the exercise, but for ensuring that X_{obs} is semi-algebraic.

2. Propose a way to adapt the sampling-based roadmap algorithm of Section 5.6 to solve the problem of time-varying motion planning with bounded speed.

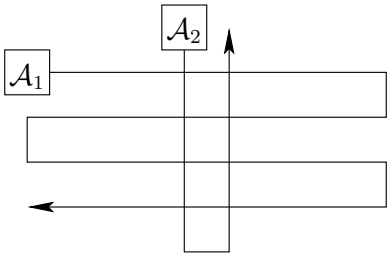


Figure 7.43: Two translating robots moving along piecewise-linear paths.

3. Develop an efficient algorithm for computing the obstacle region for two translating polygonal robots that each follow a linear path.

4. Sketch the coordination space for the two robots moving along the fixed paths shown in Figure 7.43.

5. Suppose there are two robots, and each moves on its own roadmap of three paths. The paths in each roadmap are arranged end-to-end in a triangle.

   (a) Characterize the fixed-roadmap coordination space that results, including a description of its topology.

   (b) Now suppose there are n robots, each on a triangular roadmap, and characterize the fixed-roadmap coordination space.

6. Consider the state space obtained as the Cartesian product of the C-spaces of n identical robots. Suppose that each robot is labeled with a unique integer. Show that X can be partitioned nicely into $n!$ regions in which X_{obs} appears identical and the only difference is the labels (which indicate the particular robots that are in collision).

7. Suppose there are two robots, and each moves on its own roadmap of three paths. The paths in one roadmap are arranged end-to-end in a triangle, and the paths in the other are arranged as a Y. Characterize the fixed-roadmap coordination space that results, including a description of its topology.

8. Design an efficient algorithm that takes as input a graph representation of the connectivity of a linkage and computes an active-passive decomposition. Assume that all links are revolute. The algorithm should work for either 2D or 3D linkages (the dimension is also an input). Determine the asymptotic running time of your algorithm.

9. Consider the problem of coordinating the motion of two robots that move along precomputed paths but in the presence of predictable moving obstacles. Develop a planning algorithm for this problem.

10. Consider a manipulator in $\mathcal{W} = \mathbb{R}^2$ made of four links connected in a chain by revolute joints. There is unit distance between the joints, and the first joint is attached at $(0, 0)$ in $\mathcal{W} = \mathbb{R}^2$. Suppose that the end of the last link, which is position $(1, 0)$ in its body frame, is held at $(0, 2) \in \mathcal{W}$.

(a) Use kinematics expressions to express the closure constraints for a configuration $q \in \mathcal{C}$.

(b) Convert the closure constraints into polynomial form.

(c) Use differentiation to determine the constraints on the allowable velocities that maintain closure at a configuration $q \in \mathcal{C}$.

Implementations

11. Implement the vertical decomposition algorithm to solve the path-tuning problem, as shown in Figure 7.5.

12. Use grid-based sampling and a search algorithm to compute collision-free motions of three robots moving along predetermined paths.

13. Under the conditions of Exercise 12, compute Pareto-optimal coordination strategies that optimize the time (number of stages) that each robot takes to reach its goal. Design a wavefront propagation algorithm that keeps track of the complete (ignoring equivalent strategies) set of minimal Pareto-optimal coordination strategies at each reached state. Avoid storing entire plans at each discretized state.

14. To gain an appreciation of the difficulties of planning for closed kinematic chains, try motion planning for a point on a torus among obstacles using only the implicit torus constraint given by (6.40). To simplify collision detection, the obstacles can be a collection of balls in $\mathbb{R}^3$ that intersect the torus. Adapt a sampling-based planning technique, such as the bidirectional RRT, to traverse the torus and solve planning problems.

15. Implement the spanning-tree coverage planning algorithm of Section 7.6.

16. Develop an RRT-based planning algorithm that causes the robot to chase an unpredictable moving target in a planar environment that contains obstacles. The algorithm should run quickly enough so that replanning can occur during execution. The robot should execute the first part of the most recently computed path while simultaneously computing a better plan for the next time increment.

17. Modify Exercise 16 so that the robot assumes the target follows a predictable, constant-velocity trajectory until some deviation is observed.

18. Show how to handle unexpected obstacles by using a fast enough planning algorithm. For simplicity, suppose the robot is a point moving in a polygonal obstacle region. The robot first computes a path and then starts to execute it. If the obstacle region changes, then a new path is computed from the robot's current position. Use vertical decomposition or another algorithm of your choice (provided it is fast enough). The user should be able to interactively place or move obstacles during plan execution.

19. Use the manipulation planning framework of Section 7.3.2 to develop an algorithm that solves the famous Towers of Hanoi problem by a robot that carries the rings [166]. For simplicity, suppose a polygonal robot moves polygonal parts in $\mathcal{W} = \mathbb{R}^2$ and rotation is not allowed. Make three pegs, and initially place all parts on one peg, sorted from largest to smallest. The goal is to move all of the parts to another peg while preserving the sorting.

20. Use grid-based approximation to solve optimal planning problems for a point robot in the plane. Experiment with using different neighborhoods and metrics. Characterize the combinations under which good and bad approximations are obtained.