13 Robot Motion Planning

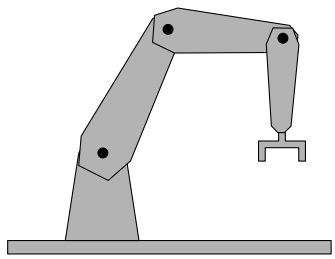
Getting Where You Want to Be

One of the ultimate goals in robotics is to design autonomous robots: robots that you can tell *what* to do without having to say *how* to do it. Among other things, this means a robot has to be able to plan its own motion.

To be able to plan a motion, a robot must have some knowledge about the environment in which it is moving. For example, a mobile robot moving around in a factory must know where obstacles are located. Some of this information—where walls and machines are located—can be provided by a floor plan. For other information the robot will have to rely on its sensors. It should be able to detect obstacles that are not on the floor plan—people, for instance. Using the information about the environment, the robot has to move to its goal position without colliding with any of the obstacles.

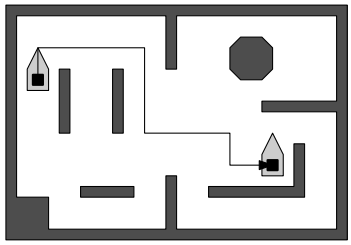
This *motion planning problem* has to be solved whenever any kind of robot wants to move in physical space. The description above assumed that we have an autonomous robot moving around in some factory environment. That kind of robot is still quite rare compared to the robot arms that are now widely employed in industry.

A robot arm, or *articulated robot*, consists of a number of links, connected by joints. Normally, one end of the arm—its *base*—is firmly connected to the ground, while the other end carries a *hand* or some kind of tool. The number of links varies between three to six or even more. The joints are usually of two types, the *revolute joint* type that allows the links to rotate around the joint, much like an elbow, or the *prismatic joint* type that allows one of the links to slide in and out. Robot arms are mostly used to assemble or manipulate parts of an object, or to perform tasks like welding or spraying. To do this, they must be able to move from one position to another, without colliding with the environment, the object they are operating on, or—an interesting complication—with themselves.



In this chapter we introduce some of the basic notions and techniques used in motion planning. The general motion planning problem is quite difficult, and we shall make some simplifying assumptions.

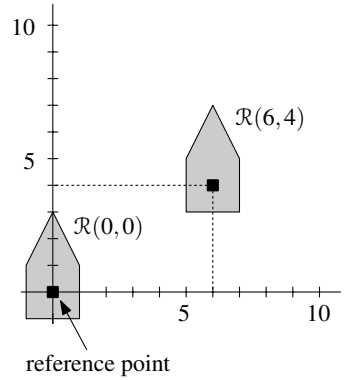
The most drastic simplification is that we will look at a 2-dimensional motion planning problem. The environment will be a planar region with polygonal



obstacles, and the robot itself will also be polygonal. We also assume that the environment is static—there are no people walking in the way of our robot—and known to the robot. The restriction to planar robots is not as severe as it appears at first sight: for a robot moving around on a work floor, a floor plan showing the location of walls, tables, machines, and so on, is often sufficient to plan a motion.

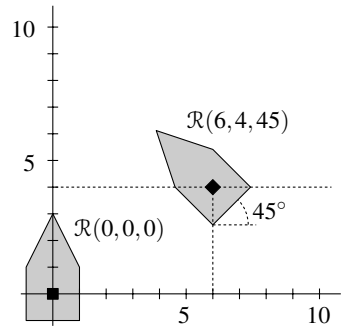
The types of motions a robot can execute depend on its mechanics. Some robots can move in any direction, while others are constrained in their motions. *Car-like robots*, for instance, cannot move sideways—otherwise parallel parking would be less challenging. In addition, they often have a certain minimum turning radius. The geometry of the motions of car-like robots is quite complicated, so we will restrict ourselves to robots that can move in arbitrary directions. In fact, we will mainly look at robots that can translate only; at the end of the chapter we'll briefly consider the case of robots that can also change their orientation by rotation.

13.1 Work Space and Configuration Space



Let $\mathcal{R}$ be a robot moving around in a 2-dimensional environment, or *work space*, consisting of a set $S = \{\mathcal{P}_1, \ldots, \mathcal{P}_t\}$ of obstacles. We assume that $\mathcal{R}$ is a simple polygon. A *placement*, or *configuration*, of the robot can now be specified by a translation vector. We denote the robot translated over a vector (x,y) by $\mathcal{R}(x,y)$. For instance, if the robot is the polygon with vertices $(1,-1)$, $(1,1)$, $(0,3)$, $(-1,1)$, and $(-1,-1)$, then the vertices of $\mathcal{R}(6,4)$ are $(7,3)$, $(7,5)$, $(6,7)$, $(5,5)$, and $(5,3)$. With this notation, a robot can be specified by listing the vertices of $\mathcal{R}(0,0)$.

An alternative way to view this is in terms of a *reference point*. This is most intuitive if the origin $(0,0)$ lies in the interior of $\mathcal{R}(0,0)$. By definition, this point is then called the reference point of the robot. We can specify a placement of $\mathcal{R}$ by simply stating the coordinates of the reference point if the robot is in the given placement. Thus $\mathcal{R}(x,y)$ specifies that the robot is placed with its reference point at (x,y). In general, the reference point does not have to be inside the robot; it can also be a point outside the robot, which we might imagine to be attached to the robot by an invisible stick. By definition, this point is at the origin for $\mathcal{R}(0,0)$.



Now suppose the robot can change its orientation by rotation, say around its reference point. We then need an extra parameter, ϕ, to specify the orientation of the robot. We let $\mathcal{R}(x,y,\phi)$ denote the robot with its reference point at (x,y) and rotated counterclockwise through an angle ϕ. So what is specified initially is $\mathcal{R}(0,0,0)$.

In general, a placement of a robot is specified by a number of parameters that corresponds to the number of *degrees of freedom (DOF)* of the robot. This number is two for planar robots that can only translate, and it is three for planar robots that can rotate as well as translate. The number of parameters we need

for a robot in 3-dimensional space is higher, of course: a translating robot in $\mathbb{R}^3$ has three degrees of freedom, and a robot that is free to translate and rotate in $\mathbb{R}^3$ has six degrees of freedom.

The parameter space of a robot $\mathcal{R}$ is usually called its *configuration space*. It is denoted by $\mathcal{C}(\mathcal{R})$. A point p in this configuration space corresponds to a certain placement $\mathcal{R}(p)$ of the robot in the work space.

In the example of a translating and rotating robot in the plane the configuration space is 3-dimensional. A point (x, y, ϕ) in this space corresponds to the placement $\mathcal{R}(x, y, \phi)$ in the work space. The configuration space is not the Euclidean 3-dimensional space; it is the space $\mathbb{R}^2 \times [0 : 360)$. Because rotations over zero and 360 degrees are equivalent, the configuration space of a rotating robot has a special topology, which is like a cylinder.

The configuration space of a translating robot in the plane is the 2-dimensional Euclidean plane, and therefore identical to the work space. Still, it is useful to distinguish the two notions: the work space is the space where the robot actually moves around—the real world, so to speak—and the configuration space is the parameter space of the robot. A polygonal robot in the work space is represented by a point in configuration space, and any point in configuration space corresponds to some placement of an actual robot in work space.

We now have a way to specify a placement of the robot, namely by specifying values for the parameters determining the placement or, in other words, by specifying a point in configuration space. But clearly not all points in configuration space are possible; points corresponding to placements where the robot intersects one of the obstacles in S are forbidden. We call the part of the configuration space consisting of these points the *forbidden configuration space*, or *forbidden space* for short. It is denoted by $\mathcal{C}_{\text{forb}}(\mathcal{R}, S)$. The rest of the configuration space, which consists of the points corresponding to *free placements*—placements where the robot does not intersect any obstacle—is called the *free configuration space*, or *free space*, and it is denoted by $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$.

A path for the robot maps to a curve in the configuration space, and vice versa: every placement along the path simply maps to the corresponding point in configuration space. A collision-free path maps to a curve in the free space. Figure 13.1 illustrates this for a translating planar robot. On the left the work space is shown, with a collision-free path from the initial position to the goal position of the robot. On the right the configuration space is shown, with the grey area indicating the forbidden part of it. The unshaded area in between the grey area is the free space. For clarity, the obstacles are still shown in the configuration space, although they have no meaning there. The curve corresponding to the collision-free path is also shown.

We have seen how to map placements of the robot to points in the configuration space, and paths of the robot to curves in that space. Can we also map obstacles to configuration space? The answer is yes: an obstacle $\mathcal{P}$ is mapped to the set of points p in configuration space such that $\mathcal{R}(p)$ intersects $\mathcal{P}$. The resulting set is called the *configuration-space obstacle*, or $\mathcal{C}$-*obstacle* for short, of $\mathcal{P}$.

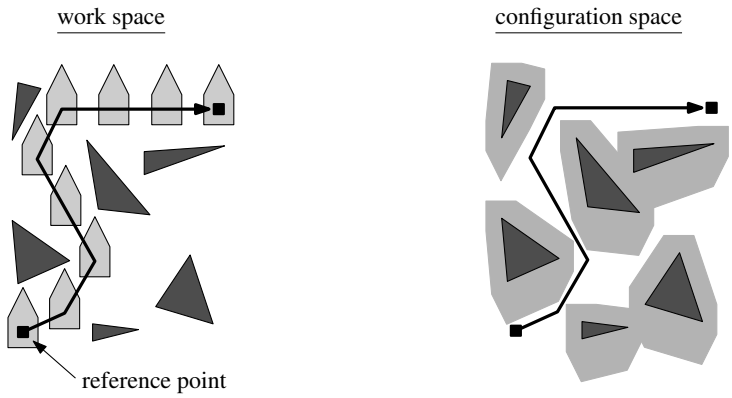
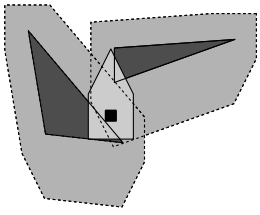


Figure 13.1
A path in the work space and the corresponding curve in the configuration space

$\mathcal{C}$-obstacles may overlap even when the obstacles in the work space are disjoint. This happens when there are placements of the robot where it intersects more than one obstacle at the same time.



There is one subtle issue that we have ignored so far: does the robot collide with an obstacle when it touches that obstacle? In other words, do we define the obstacles to be topologically open or closed sets? In the remainder we will choose the first option: obstacles are open sets, so that the robot is allowed to touch them. This is of little importance in this chapter, but it will become useful in Chapter 15. In practice a movement where the robot passes very close to an obstacle cannot be considered safe because of possible errors in robot control. Such movements can be avoided by slightly enlarging all the obstacles before the computation of a path.

13.2 A Point Robot

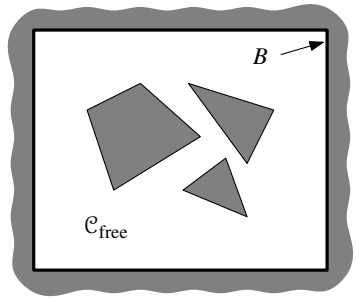
Before we try to plan the motion of a polygonal robot in the plane, let's have a look at point robots. Given the mapping from work space to configuration space that we saw in the previous section this is not such a strange idea. Furthermore, it's always good to start with a simple case. As before, we denote the robot by $\mathcal{R}$ and we denote the obstacles by $\mathcal{P}_1, \ldots, \mathcal{P}_t$. The obstacles are polygons with disjoint interiors, whose total number of vertices is denoted by n. For a point robot, the work space and the configuration space are identical. (That is to say, if we make the natural assumption that its reference point is the point robot itself. Otherwise the configuration space is a translated copy of the work space.)

Rather than finding a path from a particular start position to a particular goal position we will construct a data structure storing a representation of the free space. This data structure can then be used to compute a path between any two given start and goal positions. Such an approach is useful if the work space of the robot does not change and many paths have to be computed.

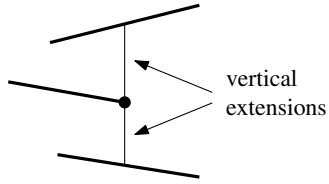
To simplify the description we restrict the motion of the robot to a large bounding box B that contains the set of polygons. In other words, we add one extra

infinitely large obstacle, which is the area outside B. The free configuration space $\mathcal{C}_{\text{free}}$ now consists of the part of B not covered by any obstacle:

$$\mathcal{C}_{\text{free}} = B \setminus \bigcup_{i=1}^{t} \mathcal{P}_i.$$



The free space is a possibly disconnected region, which may have holes. Our goal is to compute a representation of the free space that allows us to find a path for any start and goal position. We will use the *trapezoidal map* for this. Recall from Chapter 6 that the trapezoidal map of a set of non-intersecting line segments inside a bounding box is obtained by drawing two vertical extensions from every segment endpoint, one going upward until a segment (or the bounding box) is hit, and one going downward until a segment (or the bounding box) is hit. In Chapter 6 we developed a randomized algorithm, TRAPEZOIDALMAP, that computes the trapezoidal map of a set of n segments in $O(n \log n)$ expected time. The following algorithm, which computes a representation of the free space, uses this algorithm as a subroutine.



Algorithm COMPUTEFREESPACE(S)
Input. A set S of disjoint polygons.
Output. A trapezoidal map of $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ for a point robot $\mathcal{R}$.
1. Let E be the set of edges of the polygons in S.
2. Compute the trapezoidal map $\mathcal{T}(E)$ with algorithm TRAPEZOIDALMAP described in Chapter 6.
3. Remove the trapezoids that lie inside one of the polygons from $\mathcal{T}(E)$ and return the resulting subdivision.

The algorithm is illustrated in Figure 13.2. Part (a) of the figure shows the trapezoidal map of the obstacle edges inside the bounding box; this is what is computed in line 2 of the algorithm. Part (b) shows the map after the trapezoids inside the obstacles have been removed in line 3.

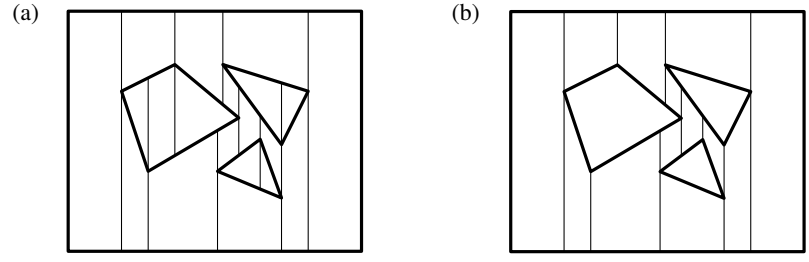


Figure 13.2
Computing a trapezoidal map of the free space

There is one detail left: how do we find the trapezoids inside the obstacles, which have to be removed? This is not so difficult, because after running TRAPEZOIDALMAP we know for each trapezoid the edge that bounds it from the top, and we known to which obstacle that edge belongs. To decide whether or not to remove the trapezoid, it suffices to check whether the edge bounds the obstacle from above or from below. The latter test takes only constant time,

because the edges of the obstacles are listed in order along the boundary so that the obstacle lies to a specific, known side of the edges.

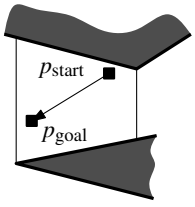
The expected time taken by TRAPEZOIDALMAP is $O(n \log n)$, so we get the following result.

Lemma 13.1 *A trapezoidal map of the free configuration space for a point robot moving among a set of disjoint polygonal obstacles with n edges in total can be computed by a randomized algorithm in $O(n \log n)$ expected time.*

In what follows, we will denote the trapezoidal map of the free space by $\mathcal{T}(\mathcal{C}_{\text{free}})$.

How do we use $\mathcal{T}(\mathcal{C}_{\text{free}})$ to find a path from a start position p_{start} to a goal position p_{goal}?

If p_{start} and p_{goal} are in the same trapezoid of the map, this is easy: the robot can simply move to its goal in a straight line.



If the start and goal position are in different trapezoids, however, then things are not so easy. In this case the path will cross a number of trapezoids and it may have to make turns in some of them. To guide the motion across trapezoids we construct a *road map* through the free space. The road map is a graph $\mathcal{G}_{\text{road}}$, which is embedded in the plane. More precisely, it is embedded in the free space. Except for an initial and final portion, paths will always follow the road map. Notice that any two neighboring trapezoids share a vertical edge that is a vertical extension of a segment endpoint. This leads us to define the road map as follows. We place one node in the center of each trapezoid, and we place one node in the middle of each vertical extension. There is an arc between two

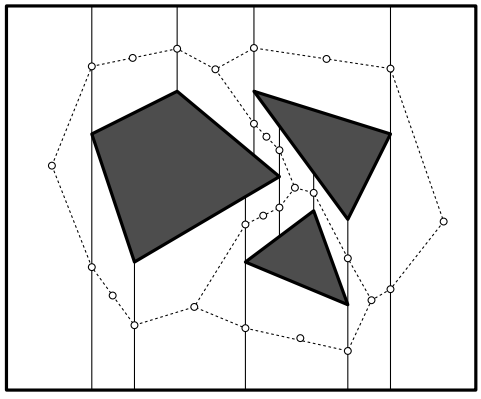


Figure 13.3
A road map

nodes if and only if one node is in the center of a trapezoid and the other node is on the boundary of that same trapezoid. The arcs are embedded in the plane as straight line segments, so following an arc in the road map corresponds to a straight-line motion of the robot. Figure 13.3 illustrates this. The road map $\mathcal{G}_{\text{road}}$ can be constructed in $O(n)$ time by traversing the doubly-connected edge list of $\mathcal{T}(\mathcal{C}_{\text{free}})$. Using the arcs in the road map we can go from the node in the center of one trapezoid to the node in the center of a neighboring trapezoid via the node on their common boundary.

We can use the road map, together with the trapezoidal map, to plan a motion from a start to a goal position. To this end we first determine the trapezoids Δ_{start} and Δ_{goal} containing these points. If they are the same trapezoid, then we move from p_{start} to p_{goal} in a straight line. Otherwise, let v_{start} and v_{goal} be the nodes of $\mathcal{G}_{\text{road}}$ that have been placed in the center of these trapezoids. The path from p_{start} to p_{goal} that we will construct now consists of three parts: the first part is a straight-line motion from p_{start} to v_{start}, the second part is a path from v_{start} to v_{goal} along the arcs of the road map, and the final part is a straight-line motion from v_{goal} to p_{goal}. Figure 13.4 illustrates this.

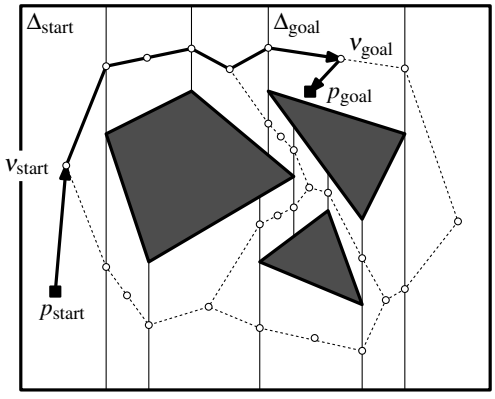


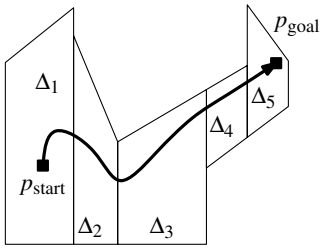
Figure 13.4
A path computed from the road map of Figure 13.3

The following algorithm summarizes how a path is found.

Algorithm COMPUTEPATH($\mathcal{T}(\mathcal{C}_{\text{free}}), \mathcal{G}_{\text{road}}, p_{\text{start}}, p_{\text{goal}}$)
Input. The trapezoidal map $\mathcal{T}(\mathcal{C}_{\text{free}})$ of the free space, the road map $\mathcal{G}_{\text{road}}$, a start position p_{start}, and goal position p_{goal}.
Output. A path from p_{start} to p_{goal} if it exists. If a path does not exist, this fact is reported.

1. Find the trapezoid Δ_{start} containing p_{start} and the trapezoid Δ_{goal} containing p_{goal}.
2. **if** Δ_{start} or Δ_{goal} does not exist
3. **then** Report that the start or goal position is in the forbidden space.
4. **else** Let v_{start} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{start}.
5. Let v_{goal} be the node of $\mathcal{G}_{\text{road}}$ in the center of Δ_{goal}.
6. Compute a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal} using breadth-first search.
7. **if** there is no such path
8. **then** Report that there is no path from p_{start} to p_{goal}.
9. **else** Report the path consisting of a straight-line motion from p_{start} to v_{start}, the path found in $\mathcal{G}_{\text{road}}$, and a straight-line motion from v_{goal} to p_{goal}.

Before we analyze the time complexity of algorithm, let's think about its correctness. Are the paths we report always collision-free, and do we always find a collision-free path if one exists?



The first question is easy to answer: any path we report must be collision-free, since it consists of segments inside trapezoids and all trapezoids are in the free space.

To answer the second question, suppose that there is a collision-free path from p_{start} to p_{goal}. Obviously p_{start} and p_{goal} must lie in one of the trapezoids covering the free space, so it remains to show that there is a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal}. The path from p_{start} to p_{goal} must cross a sequence of trapezoids. Denote the sequence of trapezoids by $\Delta_1, \Delta_2, \ldots, \Delta_k$. By definition, $\Delta_1 = \Delta_{\text{start}}$ and $\Delta_k = \Delta_{\text{goal}}$. Let v_i be the node of $\mathcal{G}_{\text{road}}$ that is in the center of Δ_i. If the path goes from Δ_i to Δ_{i+1}, then Δ_i and Δ_{i+1} must be neighbors, so they share a vertical extension. But $\mathcal{G}_{\text{road}}$ is constructed such that the nodes of such trapezoids are connected through the node on their common boundary. Hence, there is a path (consisting of two arcs) in $\mathcal{G}_{\text{road}}$ from v_i to v_{i+1}. This means that there is a path from v_1 to v_k as well. It follows that the breadth-first search in $\mathcal{G}_{\text{road}}$ will find some (possibly different) path from v_{start} to v_{goal}.

We now analyze the time the algorithm takes.

Finding the trapezoids containing the start and goal can be done in $O(\log n)$ using the point location structure of Chapter 6. Alternatively, we can simply check all trapezoids in linear time; we shall see that the rest of the algorithm takes linear time anyway, so this does not increase the time bound asymptotically.

The breadth-first search takes linear time in the size of the graph $\mathcal{G}_{\text{road}}$. This graph has one node per trapezoid plus one node per vertical extension. Both the number of vertical extensions and the number of trapezoids are linear in the total number of vertices of the obstacles. The number of arcs in the graph is linear as well, because it is planar. Hence, the breadth-first search takes $O(n)$ time.

The time to report the path is bounded by the maximum number of arcs on a path in $\mathcal{G}_{\text{road}}$, which is $O(n)$.

We get the following theorem.

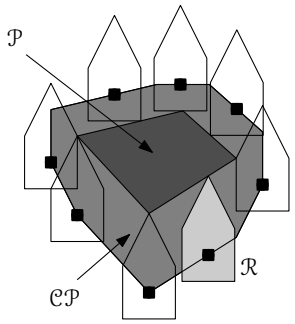
Theorem 13.2 *Let $\mathcal{R}$ be a point robot moving among a set S of polygonal obstacles with n edges in total. We can preprocess S in $O(n \log n)$ expected time, such that between any start and goal position a collision-free path for $\mathcal{R}$ can be computed in $O(n)$ time, if it exists.*

The path computed by the algorithm of this section is collision-free, but we can give no guarantee that the path does not make large detours. In Chapter 15 we will develop an algorithm that actually computes the shortest possible path. That algorithm, however, will be slower by an order of magnitude.

13.3 Minkowski Sums

In the previous section we solved the motion planning problem for a point robot; we computed a trapezoidal map of its free space and used that map to

plan its motions. The same approach can be used if the robot is a polygon. There is one difference that makes dealing with a polygonal robot more difficult: the configuration-space obstacles are no longer the same as the obstacles in work space. Therefore we start by studying the free configuration space of a translating polygonal robot. In the next section we will then describe how to compute it, so that we can use it to plan the motion of the robot.



We assume that the robot $\mathcal{R}$ is convex, and for the moment we also assume that the obstacles are convex. Recall that we use $\mathcal{R}(x,y)$ to denote the placement of $\mathcal{R}$ with its reference point at (x,y). The configuration-space obstacle, or $\mathcal{C}$-obstacle, of an obstacle $\mathcal{P}$ and the robot $\mathcal{R}$ is defined as the set of points in configuration space such that the corresponding placement of $\mathcal{R}$ intersects $\mathcal{P}$. So if we denote the $\mathcal{C}$-obstacle of $\mathcal{P}$ by $\mathcal{CP}$, then we have

$$\mathcal{CP} := \{(x,y) \ : \ \mathcal{R}(x,y) \cap \mathcal{P} \neq \emptyset\}.$$

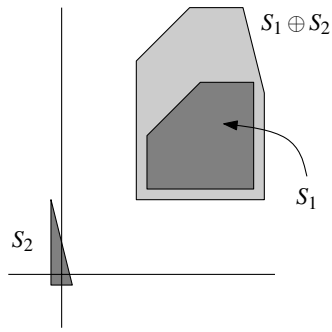
You can visualize the shape of $\mathcal{CP}$ by sliding $\mathcal{R}$ along the boundary of $\mathcal{P}$; the curve traced by the reference point of $\mathcal{R}$ is the boundary of $\mathcal{CP}$.

We can describe this in a different way using the notion of *Minkowski sums*. The Minkowski sum of two sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$, denoted by $S_1 \oplus S_2$, is defined as

$$S_1 \oplus S_2 := \{p+q \ : \ p \in S_1, q \in S_2\},$$

where $p+q$ denotes the vector sum of the vectors p and q, that is, if $p = (p_x, p_y)$ and $q = (q_x, q_y)$ then we have

$$p+q := (p_x + q_x, p_y + q_y).$$



Because a polygon is a planar set the definition of Minkowski sums also applies to them.

To be able to express the $\mathcal{C}$-obstacles as Minkowski sums, we need one more piece of notation. For a point $p = (p_x, p_y)$ we define $-p := (-p_x, -p_y)$, and for a set S we define $-S := \{-p \ : \ p \in S\}$. In other words, we get $-S$ by reflecting S about the origin. We now have the following theorem.

Theorem 13.3 *Let $\mathcal{R}$ be a planar, translating robot and let $\mathcal{P}$ be an obstacle. Then the $\mathcal{C}$-obstacle of $\mathcal{P}$ is $\mathcal{P} \oplus (-\mathcal{R}(0,0))$.*

Proof. We have to prove that $\mathcal{R}(x,y)$ intersects $\mathcal{P}$ if and only if we have that $(x,y) \in \mathcal{P} \oplus (-\mathcal{R}(0,0))$.

First, suppose that $\mathcal{R}(x,y)$ intersects $\mathcal{P}$, and let $q = (q_x, q_y)$ be a point in the intersection. It follows from $q \in \mathcal{R}(x,y)$ that we have $(q_x - x, q_y - y) \in \mathcal{R}(0,0)$ or, equivalently, that $(-q_x + x, -q_y + y) \in -\mathcal{R}(0,0)$. Because we also have $q \in \mathcal{P}$, this implies that $(x,y) \in \mathcal{P} \oplus (-\mathcal{R}(0,0))$.

Conversely, let $(x,y) \in \mathcal{P} \oplus (-\mathcal{R}(0,0))$. Then there are points $(r_x, r_y) \in \mathcal{R}(0,0)$ and $(p_x, p_y) \in \mathcal{P}$ such that $(x,y) = (p_x - r_x, p_y - r_y)$ or, in other words, such that $p_x = r_x + x$ and $p_y = r_y + y$, which implies that $\mathcal{R}(x,y)$ intersects $\mathcal{P}$. □

So for a planar translating robot $\mathcal{R}$ the $\mathcal{C}$-obstacles are the Minkowski sums of the obstacles and $-\mathcal{R}(0,0)$. (Sometimes $\mathcal{P} \oplus (-\mathcal{R}(0,0))$ is referred to as the *Minkowski difference* of $\mathcal{P}$ and $\mathcal{R}(0,0)$. Since Minkowski differences are defined differently in the mathematics literature we shall avoid this term.)

In the remainder of this section we will derive some useful properties of Minkowski sums and develop an algorithm to compute them.

We start with a simple observation about extreme points on Minkowski sums.

Observation 13.4 *Let $\mathcal{P}$ and $\mathcal{R}$ be two objects in the plane, and let $\mathcal{CP} := \mathcal{P} \oplus \mathcal{R}$. An extreme point in direction $\vec{d}$ on $\mathcal{CP}$ is the sum of extreme points in direction $\vec{d}$ on $\mathcal{P}$ and $\mathcal{R}$.*

Figure 13.5 illustrates the observation. Using this observation we now prove that the Minkowski sum of two convex polygons has linear complexity.

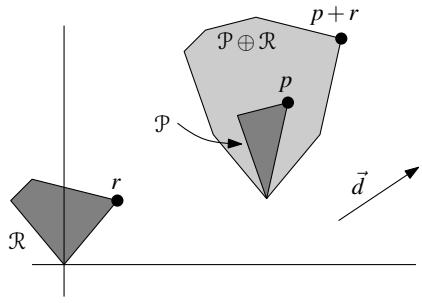
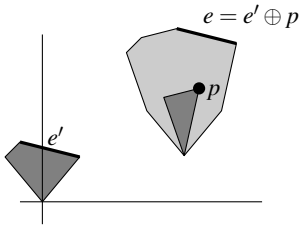


Figure 13.5
An extreme point on a Minkowski sum is the sum of extreme points

Theorem 13.5 *Let $\mathcal{P}$ and $\mathcal{R}$ be two convex polygons with n and m edges, respectively. Then the Minkowski sum $\mathcal{P} \oplus \mathcal{R}$ is a convex polygon with at most $n+m$ edges.*

Proof. The convexity of the Minkowski sum of two convex sets follows directly from the definition.

To see that the complexity of the Minkowski sum is linear, consider an edge e of $\mathcal{P} \oplus \mathcal{R}$. This edge is extreme in the direction of its outer normal. Hence, it must be generated by points on $\mathcal{P}$ and $\mathcal{R}$ that are extreme in the same direction. Moreover, at least one of $\mathcal{P}$ and $\mathcal{R}$ must have an edge that is extreme in that direction. We charge e to this edge. This way each edge is charged at most once, so the total number of edges is at most $n+m$. (If $\mathcal{P}$ and $\mathcal{R}$ don't have parallel edges, then the number of edges of the Minkowski sum is exactly $n+m$.) □



So the Minkowski sum of two convex polygons is convex and has linear complexity. But there is more: the boundaries of two Minkowski sums can only intersect in a very special manner. To make this precise, we need one more piece of terminology.

Consider two planar objects o_1 and o_2, each bounded by a simple closed curve. Intuitively, the pair o_1, o_2 is called a *pair of pseudodiscs* if their boundaries ∂o_1 and ∂o_2 intersect in at most two points. Figure 13.6 illustrates this. In degenerate situations—when the boundaries have a 1-dimensional overlap

for instance—this definition is not quite general enough. Therefore we formally define o_1, o_2 to be a pair of pseudodiscs if the following condition holds: $\partial o_1 \cap \text{int}(o_2)$ is connected and $\partial o_2 \cap \text{int}(o_1)$ is connected. (Here $\text{int}(o)$ denotes the interior of an object o.) A collection of objects, each bounded by a simple closed curve, is called a *collection of pseudodiscs* if every pair of objects in the collection is a pair of pseudodiscs. Examples of collections of pseudodiscs are collections of discs, and collections of axis-parallel squares. Note that the pseudodisc property is about the way in which (the boundaries of) two objects can interact—it does not make sense to say of a single object that it is a pseudodisc.

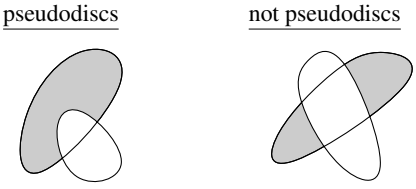


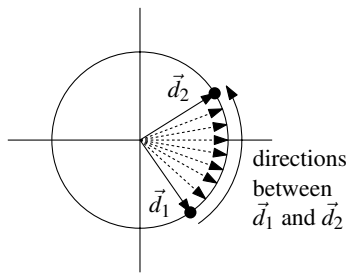
Figure 13.6
The pseudodisc property

Now consider two polygons $\mathcal{P}$ and $\mathcal{P}'$. We say that an intersection point $p \in \partial\mathcal{P} \cap \partial\mathcal{P}'$ is a *boundary crossing* if $\partial\mathcal{P}$ crosses from the interior of $\mathcal{P}'$ to the exterior of $\mathcal{P}'$ at p. Polygonal pseudodiscs satisfy the following important property:

Observation 13.6 *A pair of polygonal pseudodiscs $\mathcal{P}, \mathcal{P}'$ defines at most two boundary crossings.*

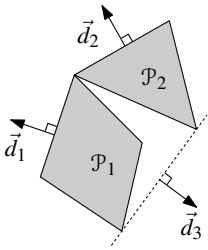
Below we will prove that a collection of Minkowski sums forms a collection of pseudodiscs. But first we need one more observation about directions and extreme points on pairs of convex polygons with disjoint interiors. We say that one polygon is more extreme in a direction $\vec{d}$ than another polygon if its extreme points lie further in that direction than the extreme points of the other polygon. For instance, a polygon is more extreme in the positive x-direction if its rightmost points lie to the right of the rightmost points of the other polygon.

We will look at extreme points for various directions. To this end we model the set of all directions by the unit circle centered at the origin: a point p on the unit circle represents the direction given by the vector from the origin to p. The range from a direction $\vec{d}_1$ to a direction $\vec{d}_2$ is defined as the directions corresponding to points in the counterclockwise circle segment from the point representing $\vec{d}_1$ to the point representing $\vec{d}_2$. Note that the range from $\vec{d}_1$ to $\vec{d}_2$ is not the same as the range from $\vec{d}_2$ to $\vec{d}_1$. The following observation is illustrated in Figure 13.7.



Observation 13.7 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be convex polygons with disjoint interiors. Let $\vec{d}_1$ and $\vec{d}_2$ be directions in which $\mathcal{P}_1$ is more extreme than $\mathcal{P}_2$. Then either $\mathcal{P}_1$ is more extreme than $\mathcal{P}_2$ in all directions in the range from $\vec{d}_1$ to $\vec{d}_2$, or it is more extreme in all directions in the range from $\vec{d}_2$ to $\vec{d}_1$.*

We are now ready to prove that Minkowski sums are pseudodiscs.



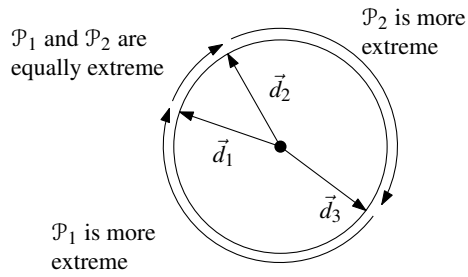
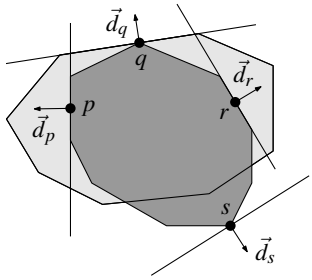


Figure 13.7
One convex polygon is more extreme than another for a connected range of directions

Theorem 13.8 *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two convex polygons with disjoint interiors, and let $\mathcal{R}$ be another convex polygon. Then the two Minkowski sums $\mathcal{P}_1 \oplus \mathcal{R}$ and $\mathcal{P}_2 \oplus \mathcal{R}$ are pseudodiscs.*



Proof. Define $\mathcal{CP}_1 := \mathcal{P}_1 \oplus \mathcal{R}$ and $\mathcal{CP}_2 := \mathcal{P}_2 \oplus \mathcal{R}$. By symmetry, it suffices to show that $\partial\mathcal{CP}_1 \cap \text{int}(\mathcal{CP}_2)$ is connected.

Suppose $\partial\mathcal{CP}_1 \cap \text{int}(\mathcal{CP}_2)$ is not connected. Then there are four alternating directions $\vec{d}_p, \vec{d}_q, \vec{d}_r, \vec{d}_s$—these directions are outward normals of points $p, q, r, s \in \partial\mathcal{CP}_1$ that occur in the given order along $\partial\mathcal{CP}_1$ with $p, r \in \text{int}(\mathcal{CP}_2)$ and $q, s \notin \text{int}(\mathcal{CP}_2)$—-such that $\mathcal{CP}_2$ is more extreme than $\mathcal{CP}_1$ in directions $\vec{d}_p$ and $\vec{d}_r$ while this is not the case for $\vec{d}_q$ and $\vec{d}_s$. From Observation 13.4 it follows that $\mathcal{P}_1$ is more extreme than $\mathcal{P}_2$ in directions $\vec{d}_p$ and $\vec{d}_r$ and not more extreme in directions $\vec{d}_q$ and $\vec{d}_s$. But this contradicts Observation 13.7. □

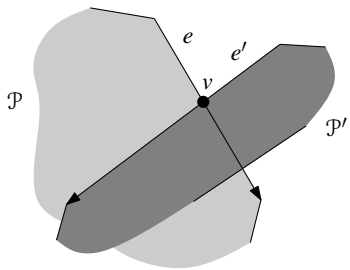
This result is useful in combination with the following theorem.

Theorem 13.9 *Let S be a collection of convex polygonal pseudodiscs with n edges in total. Then the complexity of their union is at most $2n$.*

Proof. We prove the bound by charging every vertex of the union to a pseudodisc vertex in such a way that any pseudodisc vertex is charged at most two times. This leads to a bound of $2n$ on the complexity of the union.

The charging is done as follows. There are two types of vertices in the union boundary: pseudodisc vertices and intersection points of the interiors of two pseudodisc edges.

Vertices of the former type are simply charged to themselves.



Now consider a union vertex v that is the intersection of an edge e of a pseudodisc $\mathcal{P} \in S$ and an edge e' of a pseudodisc $\mathcal{P}' \in S$. Then $e \cap e'$ is a boundary crossing. Hence, by Observation 13.6 either e does not have another boundary crossing with $\partial\mathcal{P}'$, or e' does not have another boundary crossing with $\partial\mathcal{P}$ (or both). Assume without loss of generality that e does not have another crossing with $\partial\mathcal{P}'$. Starting at $e \cap e'$, follow e into the interior of $\mathcal{P}'$. Because e does not cross $\partial\mathcal{P}'$ a second time, we must reach an endpoint of e before we reach the exterior of $\mathcal{P}'$. We charge v to this endpoint of e.

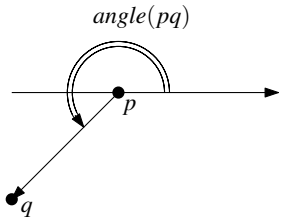
We claim that if we do the charging in this way, then every pseudodisc vertex v of any pseudodisc $\mathcal{P}$ is charged at most twice.

First consider the case that v does not lie in the interior or on the boundary of any other pseudodisc $\mathcal{P}'$. Then obviously v is a union vertex, and it is charged only by itself. Next consider the case where v lies in the interior of some

pseudodisc $\mathcal{P}'$. This means that v lies in the interior of the union. Now follow the two edges of $\mathcal{P}$ incident to v until the union boundary is reached at a crossing with some other edge; these two crossings, if they exist, are the only ones that get charged to v. Finally, if v lies on the boundary of some other pseudodisc $\mathcal{P}'$, then v may be charged by itself and (similar to the case where v is in a pseudodisc interior) by the first two crossings along its two incident edges. It can only be charged from both its incident edges, however, when the incident edges go from v into the interior of the union of the pseudodiscs. In that case v is not a union vertex and it will not be charged by itself. Hence, in all cases v is charged at most twice. □

The proof of this theorem depends heavily on the pseudodiscs being polygonal, but the theorem itself generalizes to arbitrary pseudodiscs: the complexity of the union of any set of pseudodiscs is linear in the total complexity of the pseudodiscs. This implies for instance that the union of n discs in the plane has $O(n)$ complexity. In fact, the following stronger result holds: the number of boundary intersections that show up as union vertices is linear in the number of pseudodiscs. This more general theorem is a lot more difficult to prove.

Before we return to our motion planning application, we give an algorithm to compute the Minkowski sum of two convex polygons $\mathcal{P}$ and $\mathcal{R}$. A very simple algorithm is the following. For each pair v, w of vertices, with $v \in \mathcal{P}$ and $w \in \mathcal{R}$, compute $v + w$. Next, compute the convex hull of all these sums. Unfortunately this algorithm inefficient when the polygons have many vertices, because it looks at every pair of vertices. Below we give an alternative algorithm, which is as easy to implement. It only looks at pairs of vertices that are extreme in the same direction—this is allowed because of Observation 13.4—which makes it run in linear time. In the algorithm we use the notation $angle(pq)$ to denote the angle that the vector $\vec{pq}$ makes with the positive x-axis.



Algorithm MINKOWSKISUM($\mathcal{P}, \mathcal{R}$)
Input. A convex polygon $\mathcal{P}$ with vertices $v_1, \ldots, v_n$, and a convex polygon $\mathcal{R}$ with vertices $w_1, \ldots, w_m$. The lists of vertices are assumed to be in counterclockwise order, with v_1 and w_1 being the vertices with smallest y-coordinate (and smallest x-coordinate in case of ties).
Output. The Minkowski sum $\mathcal{P} \oplus \mathcal{R}$.

```
1.  i ← 1; j ← 1
2.  v_{n+1} ← v_1; v_{n+2} ← v_2; w_{m+1} ← w_1; w_{m+2} ← w_2
3.  repeat
4.      Add v_i + w_j as a vertex to P ⊕ R.
5.      if angle(v_i v_{i+1}) < angle(w_j w_{j+1})
6.        then i ← (i + 1)
7.        else if angle(v_i v_{i+1}) > angle(w_j w_{j+1})
8.                then j ← (j + 1)
9.                else i ← (i + 1); j ← (j + 1)
10. until i = n + 1 and j = m + 1
```

MINKOWSKISUM runs in linear time, because at each execution of the **repeat** loop either i or j is incremented and—as is not difficult to prove—they will

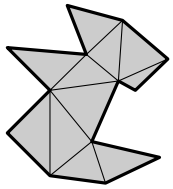
not be incremented after reaching the values $n+1$ and $m+1$. The fact that the correct pairs of vertices are taken is similar to the proof of Theorem 13.5; one just has to observe that any vertex of the Minkowski sum is the sum of two original vertices that are extreme in a common direction, and argue that the angle test ensures that all extreme pairs are found.

We conclude with the following theorem:

Theorem 13.10 *The Minkowski sum of two convex polygons with n and m vertices, respectively, can be computed in $O(n+m)$ time.*

What happens if one or both of the polygons are not convex? This question is not so hard to answer if we realize that the following equality holds for any sets S_1, S_2, and S_3:

$$S_1 \oplus (S_2 \cup S_3) = (S_1 \oplus S_2) \cup (S_1 \oplus S_3).$$



Now consider the Minkowski sum of a non-convex polygon $\mathcal{P}$ and a convex polygon $\mathcal{R}$ with n and m vertices respectively. What is the complexity of $\mathcal{P} \oplus \mathcal{R}$? We know from Chapter 3 that the polygon $\mathcal{P}$ can be triangulated into $n-2$ triangles $t_1, \ldots, t_{n-2}$, where n is its number of vertices. From the equality above we can conclude that

$$\mathcal{P} \oplus \mathcal{R} = \bigcup_{i=1}^{n-2} t_i \oplus \mathcal{R}.$$

Since t_i is a triangle and $\mathcal{R}$ a convex polygon with m vertices, we know that $t_i \oplus \mathcal{R}$ is a convex polygon with at most $m+3$ vertices. Moreover, the triangles have disjoint interiors, so the collection of Minkowski sums is a collection of pseudodiscs. Hence, the complexity of their union is linear in the sum of their complexities. This implies that the complexity of $\mathcal{P} \oplus \mathcal{R}$ is $O(nm)$.

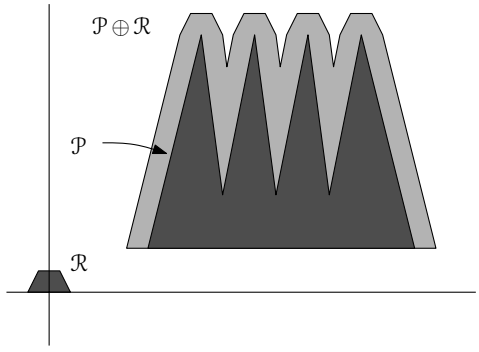


Figure 13.8
The Minkowski sum of a non-convex and a convex polygon

This upper bound on the complexity of a non-convex and a convex polygon is tight in the worst case. To see this, consider a polygon $\mathcal{P}$ with $\lfloor n/2 \rfloor$ spikes pointing upward, and a much smaller polygon $\mathcal{R}$ that is the top half of a regular $(2m-2)$-gon. The Minkowski sum of these polygons will also have $\lfloor n/2 \rfloor$ spikes, each of which has m vertices at its top. Figure 13.8 illustrates the construction.

To bound the complexity of the Minkowski sum of two non-convex polygons $\mathcal{P}$ and $\mathcal{R}$, we triangulate both polygons. We get a collection of $n-2$ triangles t_i, and a collection of $m-2$ triangles u_j. The Minkowski sum of $\mathcal{P}$ and $\mathcal{R}$ is now the union of the Minkowski sums of the pairs t_i, u_j. Each sum $t_i \oplus u_j$ has constant complexity. Hence, $\mathcal{P} \oplus \mathcal{R}$ is the union of $(n-2)(m-2)$ constant-complexity polygons. This implies that the total complexity of $\mathcal{P} \oplus \mathcal{R}$ is $O(n^2m^2)$. Again, this bound is tight in the worst case: there are non-convex polygons whose Minkowski sum really has $\Theta(n^2m^2)$ complexity. Figure 13.9 illustrates this.

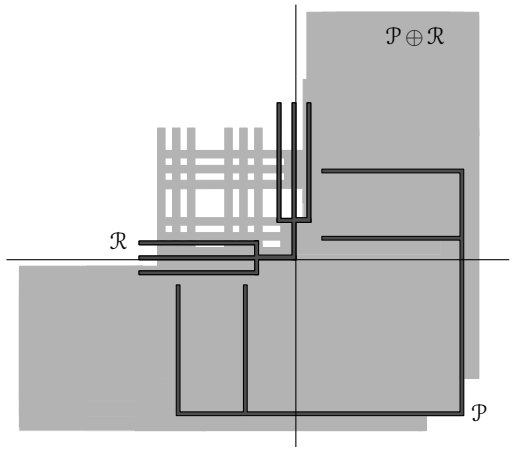


Figure 13.9
The Minkowski sum of two non-convex polygons

The following theorem summarizes the results on the complexity of Minkowski sums. For completeness the complexity in the case of two convex polygons is given as well.

Theorem 13.11 *Let $\mathcal{P}$ and $\mathcal{R}$ be polygons with n and m vertices, respectively. The complexity of the Minkowski sum $\mathcal{P} \oplus \mathcal{R}$ is bounded as follows:*
(i) it is $O(n+m)$ if both polygons are convex;
(ii) it is $O(nm)$ if one of the polygons is convex and one is non-convex;
(iii) it is $O(n^2m^2)$ if both polygons are non-convex.
These bounds are tight in the worst case.

Computing Minkowski sums of non-convex polygons is not very difficult: triangulate both polygons, compute the Minkowski sum of each pair of triangles, and take their union. This approach is basically the same as the approach described in the next section for computing the forbidden space of a translating robot, so we omit the details here.

13.4 Translational Motion Planning

It is time to return to the planar motion planning problem. Recall that our robot $\mathcal{R}$ can only translate and that the obstacles are disjoint polygons. Early in the

previous section we have shown that the $\mathcal{C}$-obstacle corresponding to an obstacle $\mathcal{P}_i$ is the Minkowski sum $\mathcal{P}_i \oplus (-\mathcal{R})$. Moreover, we have seen that Minkowski sums of convex polygons are pseudodiscs. We use this to prove our first major result on the motion planning problem, which states that the complexity of the free space of a translating planar robot is linear.

Theorem 13.12 *Let $\mathcal{R}$ be a convex robot of constant complexity, translating among a set S of non-intersecting polygonal obstacles with a total of n edges. Then the complexity of the free configuration space $\mathcal{C}_{\text{free}}(\mathcal{R}, S)$ is $O(n)$.*

Proof. First we triangulate each obstacle polygon. We get a set of $O(n)$ triangular, and hence convex, obstacles with disjoint interiors. The free configuration space is the complement of the union of the $\mathcal{C}$-obstacles of these triangles. Because the robot has constant complexity, the $\mathcal{C}$-obstacles have constant complexity, and according to Theorem 13.8 they form a set of pseudodiscs. Theorem 13.9 now implies that the union has linear complexity. □

It remains to find an algorithm to construct the free space. Rather than computing the free space $\mathcal{C}_{\text{free}}$, we shall compute the forbidden space $\mathcal{C}_{\text{forb}}$; the free space is simply its complement.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_n$ denote the triangles that we get when we triangulate the obstacles. We want to compute

$$\mathcal{C}_{\text{forb}} \;=\; \bigcup_{i=1}^{n} \mathcal{C}\mathcal{P}_i \;=\; \bigcup_{i=1}^{n} \mathcal{P}_i \oplus (-\mathcal{R}(0,0)).$$

In Section 13.3 we saw how to compute the individual Minkowski sums $\mathcal{C}\mathcal{P}_i$. To compute their union we use a simple divide-and-conquer approach.

Algorithm FORBIDDENSPACE($\mathcal{C}\mathcal{P}_1, \ldots, \mathcal{C}\mathcal{P}_n$)
Input. A collection of $\mathcal{C}$-obstacles.
Output. The forbidden space $\mathcal{C}_{\text{forb}} = \bigcup_{i=1}^{n} \mathcal{C}\mathcal{P}_i$.
1. **if** $n = 1$
2. **then return** $\mathcal{C}\mathcal{P}_1$
3. **else** $\mathcal{C}^1_{\text{forb}} \leftarrow$FORBIDDENSPACE($\mathcal{P}_1, \ldots, \mathcal{P}_{\lceil n/2 \rceil}$)
4. $\mathcal{C}^2_{\text{forb}} \leftarrow$FORBIDDENSPACE($\mathcal{P}_{\lceil n/2 \rceil + 1}, \ldots, \mathcal{P}_n$)
5. Compute $\mathcal{C}_{\text{forb}} = \mathcal{C}^1_{\text{forb}} \cup \mathcal{C}^2_{\text{forb}}$.
6. **return** $\mathcal{C}_{\text{forb}}$

The heart of this algorithm is the subroutine to compute the union of two planar regions, which we need to perform the merge step (line 5). If we represent these regions by doubly-connected edge lists, this can be done by the overlay algorithm described in Chapter 2.

The following lemma summarizes the result.

Lemma 13.13 *The free configuration space $\mathcal{C}_{\text{free}}$ of a convex robot of constant complexity translating among a set of polygons with n edges in total can be computed in $O(n \log^2 n)$ time.*

Proof. In Chapter 3 we saw that a polygon with m vertices can be triangulated in $O(m \log m)$ time. (In fact, it can even be done in $O(m)$ time with a very complicated algorithm, as stated in the notes and comments of Chapter 3.) Hence, if m_i denotes the complexity of obstacle $\mathcal{P}_i$, then triangulating all the obstacles takes time proportional to

$$\sum_{i=1}^{t} m_i \log m_i \;\leqslant\; \sum_{i=1}^{t} m_i \log n \;=\; n \log n.$$

Computing the $\mathcal{C}$-obstacles of each of the resulting triangles takes linear time in total. It remains to bound the time that FORBIDDENSPACE needs to compute the union of the $\mathcal{C}$-obstacles.

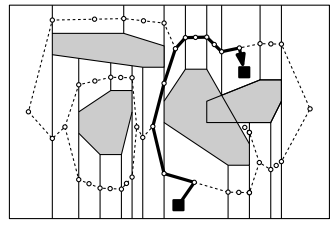
Using the results from Chapter 2, the merge step (line 5) can be done in $O((n_1 + n_2 + k)\log(n_1 + n_2))$ where n_1, n_2, and k denote the complexity of $\mathcal{C}^1_{\text{forb}}$, $\mathcal{C}^2_{\text{forb}}$, and $\mathcal{C}^1_{\text{forb}} \cup \mathcal{C}^2_{\text{forb}}$. Theorem 13.12 states that the complexity of the free space—and, hence, of the forbidden space—is linear in the sum of the complexities of the obstacles. In our case this means that n_1, n_2, and k are all $O(n)$, so the time for the merge step is $O(n \log n)$. We get the following recurrence for $T(n)$, the time the algorithm needs when applied to a set of n constant-complexity $\mathcal{C}$-obstacles:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n).$$

The solution of this recurrence is $O(n \log^2 n)$. □

The result of this theorem is not the best possible—see the notes and comments of this chapter.

Now that we have computed the free space, we can continue in exactly the same way as in Section 13.2: we compute a trapezoidal map of the free space, together with a roadmap. Given a start and a goal placement of the robot $\mathcal{R}$, we find a path as follows. First, we map the start and goal placement to points in the configuration space. Then we compute a path between these two points through the free space using the trapezoidal map and the road map, as described in Section 13.2. Finally, we map the path back to a path for $\mathcal{R}$ in the work space.



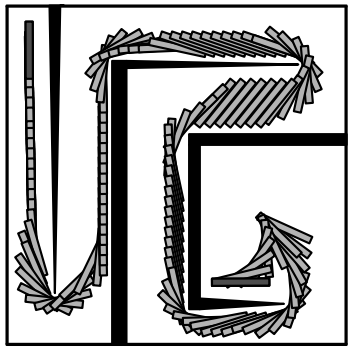
The next theorem summarizes the result of our efforts.

Theorem 13.14 *Let $\mathcal{R}$ be a convex robot of constant complexity translating among a set S of disjoint polygonal obstacles with n edges in total. We can preprocess S in $O(n \log^2 n)$ expected time, such that between any start and goal position a collision-free path for $\mathcal{R}$, if it exists, can be computed in $O(n)$ time.*

13.5* Motion Planning with Rotations

In the previous sections the robot was only allowed to translate. When the robot is circular this does not limit its possible motion. On the other hand, when it is long and skinny, translational motion is often not enough: it may have to change

its orientation to be able to pass through a narrow passage or to go around a corner. In this section we sketch a method to plan motion for robots that can rotate as well as translate.



Let $\mathcal{R}$ be a convex polygonal robot that can translate and rotate in a planar work space that contains a set $\mathcal{P}_1, \ldots, \mathcal{P}_t$ of disjoint polygonal obstacles. The robot $\mathcal{R}$ has three degrees of freedom: two translational and one rotational degree of freedom. Hence, we can specify a placement for $\mathcal{R}$ by three parameters: the x- and y-coordinate of a reference point of $\mathcal{R}$, and an angle ϕ that specifies its orientation. As in Section 13.1, we use $\mathcal{R}(x,y,\phi)$ to denote the robot placed with its reference point at (x,y) and rotated over an angle ϕ.

The configuration space that we get is the 3-dimensional space $\mathbb{R}^2 \times [0:360)$, with a topology where points $(x,y,0)$ and $(x,y,360)$ are identified. Recall that $\mathcal{CP}_i$, the $\mathcal{C}$-obstacle of an obstacle $\mathcal{P}_i$, is defined as follows:

$$\mathcal{CP}_i := \{(x,y,\phi) \in \mathbb{R}^2 \times [0:360) \ : \ \mathcal{R}(x,y,\phi) \cap \mathcal{P}_i \neq \emptyset\}.$$

What do these $\mathcal{C}$-obstacles look like? This question is difficult to answer directly, but we can get an idea by looking at cross-sections with planes of constant ϕ. In such a plane the rotation angle is fixed, so we are dealing with a purely translational problem. Hence, we know the shape of the cross-section: it is a Minkowski sum. More precisely, the cross-section of $\mathcal{CP}_i$ with the plane $h : \phi = \phi_0$ is equal to $\mathcal{P}_i \oplus \mathcal{R}(0,0,\phi_0)$. (More precisely, it is a copy of the

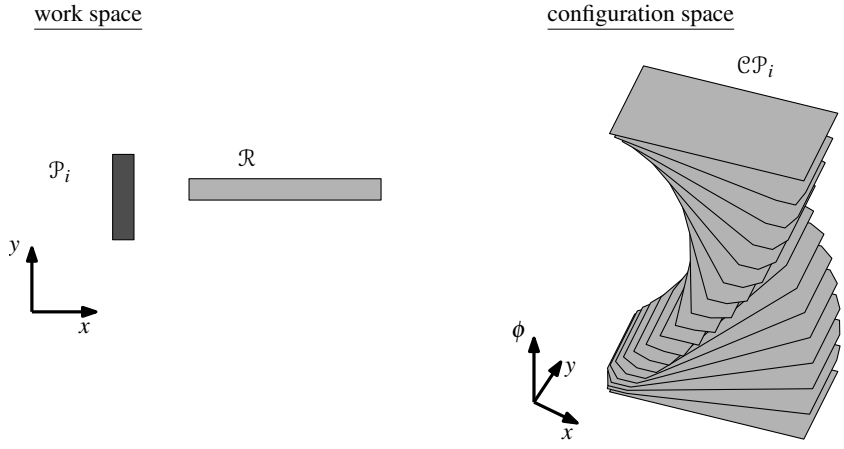


Figure 13.10
The $\mathcal{C}$-obstacle of a rotating and translating robot

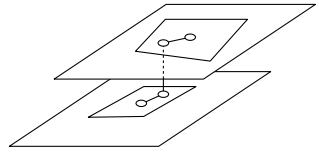
Minkowski sum placed at height ϕ_0.) Now imagine sweeping a horizontal plane upwards through configuration space, starting at $\phi = 0$ and ending at $\phi = 360$. At any time during the sweep the cross-section of the plane with $\mathcal{CP}_i$ is a Minkowski sum. The shape of the Minkowski sum changes continuously: at $\phi = \phi_0$ the cross-section is $\mathcal{P}_i \oplus \mathcal{R}(0,0,\phi_0)$, and at $\phi = \phi_0 + \varepsilon$ the cross-section is $\mathcal{P}_i \oplus \mathcal{R}(0,0,\phi_0 + \varepsilon)$. This means that $\mathcal{CP}_i$ looks like a twisted pillar, as in Figure 13.10. The edges and facets of this twisted pillar, except for the top facet and bottom facet, are curved.

So we know more or less what $\mathcal{C}$-obstacles look like. The free space is the complement of the union of these $\mathcal{C}$-obstacles. Due to the nasty shape of the $\mathcal{C}$-obstacles, the free space is rather complicated: its boundary is no longer polygonal, but curved. Moreover, the combinatorial complexity of the free space can be quadratic for a convex robot, and even cubic for a non-convex robot. Nevertheless, we can solve the motion planning problem using the same approach we took before: compute a decomposition of the free space into simple cells, and construct a road map to guide the motions between neighboring cells. Given a start and goal placement of the robot we then find a path as follows. We map these placements to points in configuration space, find the cells that contain the points, and construct a path consisting of three parts: a path from the start position to the node of the road map in the center of the start cell, a path along the road map to the node in the center of the goal cell, and a path inside the goal cell to the final destination. It remains to map the path in configuration space back to a motion in the work space.

Because of the complex shape of the $\mathcal{C}$-obstacles, it is difficult to compute a suitable cell decomposition, especially when it comes to an actual implementation. Therefore we shall describe a different, simpler approach. As we shall see, however, this approach has its drawbacks as well. Our approach is based on the same observation we used to study the shape of the $\mathcal{C}$-obstacles, namely that the motion planning problem reduces to a purely translational problem if we restrict the attention to a horizontal cross-section of the configuration space. We will call such a cross-section a *slice*. The idea is to compute a finite number of slices. A path for the robot now consists of two types of motion: motions within a slice—these are purely translational—and motions from one slice to the next or previous one—these will be purely rotational.

Let's formalize this. Let z denote the number of slices we take. For every integer i with $0 \leqslant i \leqslant z-1$, let $\phi_i = i \times (360/z)$. We compute a slice of the free space for each ϕ_i. Since within the slice we are dealing with a purely translational problem for the robot $\mathcal{R}(0,0,\phi_i)$, we can compute the slice using the methods of the previous section. This will give us the trapezoidal map $\mathcal{T}_i$ of the free space within the slice. For each $\mathcal{T}_i$ we compute a road map $\mathcal{G}_i$. These road maps are used to plan the motions within a slice, as in Section 13.2.

It remains to connect consecutive slices. More precisely, we connect every pair of roadmaps $\mathcal{G}_i$, $\mathcal{G}_{i+1}$ to obtain a roadmap $\mathcal{G}_{\text{road}}$ of the entire configuration space. This is done as follows. We take the trapezoidal maps of each pair of consecutive slices, and compute their overlay with the algorithm of Chapter 2. (Strictly speaking, we should say that we compute the overlay of the projections of $\mathcal{T}_i$ and $\mathcal{T}_{i+1}$ onto the plane $h : \phi = 0$.) This tells us all the pairs Δ_1, Δ_2 with $\Delta_1 \in \mathcal{T}_i$ and $\Delta_2 \in \mathcal{T}_{i+1}$ such that Δ_1 intersects Δ_2. Let $(x,y,0)$ be a point in $\Delta_1 \cap \Delta_2$. We then add an extra node to $\mathcal{G}_{\text{road}}$ at (x,y,ϕ_i) and at (x,y,ϕ_{i+1}), which we connect by an arc. Moving from one slice to the other along this arc corresponds to a rotation from ϕ_i to ϕ_{i+1}, or back. Furthermore, the node at (x,y,ϕ_i) is connected to the node at the center of Δ_1, and the node at (x,y,ϕ_{i+1}) is connected to the node at the center of Δ_2. These connections stay within a slice, so they correspond to purely translational motions. We connect $\mathcal{G}_{z-1}$ and



$\mathcal{G}_0$ in the same way. Note that paths in the graph $\mathcal{G}_{\text{road}}$ correspond to paths of the robot that are composed of purely translational motion (when we move along an arc connecting nodes in the same slice) and purely rotational motion (when we move along an arc connecting nodes in different slices).

Once we have constructed this road map, we can use it to plan a motion for $\mathcal{R}$ from any start placement $\mathcal{R}(x_{\text{start}}, y_{\text{start}}, \phi_{\text{start}})$ to any goal placement $\mathcal{R}(x_{\text{goal}}, y_{\text{goal}}, \phi_{\text{goal}})$. To do this, we first determine the slices closest to the start and goal placement by rounding the orientations ϕ_{start} and ϕ_{goal} to the nearest orientation ϕ_i for which we constructed a slice. Within those slices we determine the trapezoids Δ_{start} and Δ_{goal} containing the start and goal position. If one of these trapezoids does not exist because the start or goal position lies in the forbidden space within the slice, then we report that we cannot compute a path. Otherwise, let v_{start} and v_{goal} be the nodes of the road map that have been placed in their center. We try to find a path in $\mathcal{G}_{\text{road}}$ from v_{start} to v_{goal} using breadth-first search. If there is no path in the graph, we report that we cannot compute a motion. Otherwise we report a motion consisting of five parts: a purely rotational motion from the start position to the nearest slice, a purely translational motion within that slice to the node v_{start}, a motion that corresponds to the path from v_{start} to v_{goal} in $\mathcal{G}_{\text{road}}$, a purely translational motion from v_{goal} to the final position within that slice (which is the slice nearest to the goal position), and finally a purely rotational motion to the real goal position.

This method is a generalization of the method we used for translating motions, but is has a major problem: it is not always correct. Sometimes it may erroneously report that a path does not exist. For instance, the start position can be in the free space, whereas the start position within the nearest slice is not. In this case we report that there is no path, which need not be true. Even worse is that the paths we report need not be collision-free. The translational motions within a slice are okay, because we solved the problem within a slice exactly, but the rotational motions from one slice to the next may cause problems: the placements within the two slices are collision-free, but halfway the robot could collide with an obstacle. Both problems are less likely to occur when we increase the number of slices, but we can never be certain of the correctness of the result. This is especially bothersome for the second problem: we definitely don't want our possibly very expensive robot to have a collision.

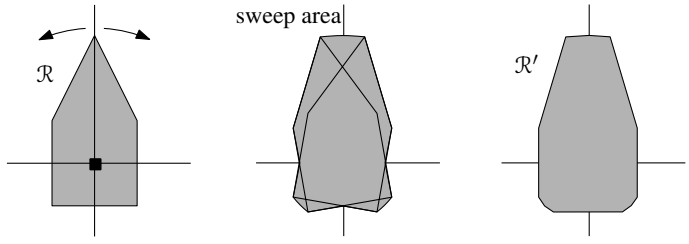


Figure 13.11
Enlarging the robot

Therefore we use the following trick. We make the robot slightly larger, and use the method described above on the enlarged robot $\mathcal{R}'$. This is done in such a way that although $\mathcal{R}'$ can collide during rotations, the original robot $\mathcal{R}$

cannot. To achieve this, the robot is enlarged as follows. Rotate $\mathcal{R}$ clockwise and counterclockwise over an angle of $(180/z)^\circ$. During this rotation $\mathcal{R}$ sweeps a part of the plane. We use for the enlarged robot $\mathcal{R}'$ a convex polygon that contains the sweep area—see Figure 13.11. We now compute the trapezoidal maps and the road map for $\mathcal{R}'$ instead of $\mathcal{R}$. It is not difficult to prove that $\mathcal{R}$ cannot collide with an obstacle during a purely rotational motion between two adjacent slices, even though $\mathcal{R}'$ can. By enlarging the robot we have introduced another way to incorrectly decide that there is no path. Again, this becomes less likely when the number of slices increases. So with a large number of slices, the method probably performs reasonably well in practical situations.

13.6 Notes and Comments

The motion-planning problem has received a lot of attention over the years, both from people working in computational geometry and from people working in robotics, and this chapter only scratches the surface of all the research. A much more extensive treatment of the problem is given by Latombe [243]. Sharir [352] gives a survey of the theoretical results that have been obtained. Nevertheless, the concepts we have introduced—configuration space, decomposition of the free space, road maps that transform the geometric problem into a graph searching problem—underly the majority of approaches that have been proposed.

These concepts date back to the work of Lozano-Pérez [258, 259, 260]. An important difference between his method and the method of this chapter is that he used an approximate decomposition of the free space. The approach of Section 13.2, which uses an exact decomposition of the free space of a planar translating robot into trapezoids, is based on more recent work by Kedem et al. [231, 232]. An improved algorithm, which runs in $O(n \log n)$ time, was given by Bhattacharya and Zorbas [68].

A very general method that is based on finding an exact cell decomposition of the free space was given by Schwartz and Sharir [341]. It is based on a decomposition method of Collins [136]. Unfortunately, this method takes time doubly exponential in the dimension of the configuration space. This can be improved using a decomposition method of Chazelle et al. [102].

In this chapter we have seen that the cell decomposition approach leads to an $O(n \log^2 n)$ algorithm when applied to a convex robot translating in the plane. The bottleneck in the algorithm was the computation of the union of a collection of Minkowski sums. Using a randomized incremental algorithm, instead of a divide-and-conquer algorithm, this step can be done in $O(n \log n)$ time [58, 280]. Our approach was based on the fact that the Minkowski sums of convex disjoint polygons with some fixed other convex polygon form a set of pseudodiscs, and that the union complexity of these pseudodiscs is $O(n)$. The latter statement is even true for non-polygonal pseudodiscs: there are at most $6n - 12$ breakpoints (that is, boundary intersections of two pseudodiscs) on the boundary of the union of n pseudodiscs [231]. Here a set of objects is defined

to be a set of pseudodiscs if the boundaries of any pair of them intersect in at most two points. Extensions of this result, and many other results on the union complexity of certain families of objects are discussed in a recent survey by Agarwal et al. [3].

The translational motion planning problem in 3-dimensional space can be solved in $O(n^2 \log^3 n)$ time [22].

The approach we sketched for robots that can translate and rotate is approximate: it isn't guaranteed to find a path if it exists. It is possible to find an exact solution by computing an exact cell decomposition of the free space in $O(n^3)$ time [33]. For a convex robot, the running time can be reduced to $O(n^2 \log^2 n)$ [233].

The free space of a robot may consist of a number of disconnected components. Of course, the motions of the robot are confined to the component where it starts; to go to another component it would have to pass through the forbidden space. Hence, it is sufficient to compute only a *single cell* in the free space, instead of the entire free space. Usually the worst-case complexity of a single cell is one order of magnitude lower than the complexity of the entire free space. This can be used to speed up the asymptotic running time of the motion planning algorithms. The book by Agarwal and Sharir [353] and the thesis by Halperin [205] discuss single cells and their connection to motion planning at length.

The theoretical complexity of the motion planning is exponential in the number of degrees of freedom of the robot, which makes the problem appear intractable for high DOF robots. Under some mild restrictions on the shape of the robot and the obstacles—which are likely to be satisfied in practical situations—one can show that the complexity of the free space is only linear [364, 365].

Cell decomposition methods are not the only exact methods for motion planning. Another approach is the so-called *retraction method*. Here a road map is constructed directly, without decomposing the free space. Furthermore, a retraction function is defined, which maps any point in the free space to a point on the road map. Once this has been done, paths can be found by retracting both start and goal to the road map and next following a path along the road map. Different types of road maps and retraction functions have been proposed. A nice road map is the Voronoi diagram, because it stays as far away from the obstacles as possible. If the robot is a disc, then we can use the normal Voronoi diagram—this was already discussed in Section 7.3. Otherwise one has to use a different distance function in the definition of the Voronoi diagram, which depends on the shape of the robot. Still, such a diagram can often be computed in $O(n \log n)$ time [255, 296], leading to another $O(n \log n)$ time algorithm for translational motion planning. A very general road map method has been proposed by Canny [80]. It can solve almost any motion planning problem in time $O(n^d \log n)$, where d is the dimension of the configuration space, that is, the number of degrees of freedom of the robot. Unfortunately, the method is very complicated, and it has the disadvantage that most of the time the robot moves in contact with an obstacle. This is often not the preferred type of motion.

In this chapter we have concentrated mainly on exact motion planning. There are also a number of heuristic approaches.

For instance, one can use *approximate cell decompositions* [74, 258, 259, 398] instead of exact ones. These are often based on quadtrees.

Another heuristic is the *potential field method* [39, 235, 378]. Here one defines a potential field on the configuration space by making the goal position attract the robot and making the obstacles repel it. The robot then moves in the direction dictated by the potential field. The problem with this approach is that the robot can get stuck in local minima of the potential field. Various techniques have been proposed to escape the minima.

Another heuristic which has recently become popular is the probabilistic road map method [230, 310]. It computes a number of random placements for the robot, which are connected in some way to form a road map of the free space. The road map can then be used to plan paths between any given start and goal placement.

Minkowski sums play an important role not only in motion planning, but also in other problems. An example is the problem of placing one polygon inside another [119]; this can be useful if one wants to cut out some shape from a piece of fabric. For some basic results on properties of Minkowski sums and their computation we refer to [43, 197].

In this chapter we concentrated on finding some path for the robot, but we didn't try to find a short path. This is the topic of Chapter 15.

Finally, we note that we allowed paths where the robot touches an obstacle. Such paths are sometimes called "semi-free" [243, 340]. Paths that do not touch any obstacle are then called "free". It is useful to be aware of these terms when studying the motion planning literature.

13.7 Exercises

13.1 Let $\mathcal{R}$ be a robotic arm with a fixed base and seven links. The last joint of $\mathcal{R}$ is a prismatic joint, the other ones are revolute joints. Give a set of parameters that determines a placement of $\mathcal{R}$. What is the dimension of the configuration space resulting from your choice of parameters?

13.2 In the road map $\mathcal{G}_{\text{road}}$ that was constructed on the trapezoidal decomposition of the free space we added a node in the center of each trapezoid and on each vertical wall. It is possible to avoid the nodes in the center of each trapezoid. Show how the graph can be changed such that only nodes on the vertical walls are required. (Avoid an increase in the number of edges in the graph.) Explain how to adapt the query algorithm.

13.3 Prove that the shape of $\mathcal{CP}_i$ is independent of the choice of the reference point in the robot $\mathcal{R}$.

13.4 Draw the Minkowski sum $\mathcal{P}_1 \oplus \mathcal{P}_2$ for the case where

a. both $\mathcal{P}_1$ and $\mathcal{P}_2$ are unit discs;
b. both $\mathcal{P}_1$ and $\mathcal{P}_2$ are unit squares;
c. $\mathcal{P}_1$ is a unit disc and $\mathcal{P}_2$ is a unit square;
d. $\mathcal{P}_1$ is a unit square and $\mathcal{P}_2$ is a triangle with vertices $(0,0),(1,0),(0,1)$.

13.5 Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two convex polygons. Let S_1 be the collection of vertices of $\mathcal{P}_1$ and S_2 be the collection of vertices of $\mathcal{P}_2$. Prove that

$$\mathcal{P}_1 \oplus \mathcal{P}_2 = \mathit{ConvexHull}(S_1 \oplus S_2).$$

13.6 Prove Observation 13.4.

13.7 In Theorem 13.9 we gave an $O(n)$ bound on the complexity of the union of a set of polygonal pseudodiscs with n vertices in total. We are interested in the precise bound.

a. Assume that the union boundary contains m original vertices of the polygons. Show that the complexity of the union boundary is at most $2n-m$. Use this to prove an upper bound of $2n-3$ on the complexity of the union boundary.
b. Prove a lower bound of $2n-6$ by constructing an example that has this complexity.