Chapter 7

PATH AND TRAJECTORY PLANNING

In previous chapters we studied the geometry of robot arms, developing solutions for both the forward and inverse kinematics problems. The solutions to these problems depend only on the intrinsic geometry of the robot, and they do not reflect any constraints imposed by the workspace in which the robot operates. In particular, they do not take into account the possibility of collision between the robot and objects in the workspace or collision between the robot and the boundaries of its workspace (e.g., walls, floor, closed doors). In this chapter we address the problem of planning collision-free paths for robots. We will assume that the initial and final configurations of the robot are specified and that the problem is to find a collision-free path connecting these configurations.

The description of this problem is deceptively simple, yet the path planning problem is among the most difficult problems in computer science. For example, the computation time required by the best known complete[1] path planning algorithm grows exponentially with the number of internal degrees of freedom of the robot. For this reason, complete algorithms are used in practice only for simple robots with few degrees of freedom, such as mobile robots that translate in the plane. For robots that have more than a few degrees of freedom or are capable of rotational motion, path planning problems are often treated as search problems. The algorithms that are used for such problems are guided by heuristic strategies, and not guaranteed to find solutions for all problems. Nevertheless, they are quite effective in a wide

[1] An algorithm is said to be complete if it finds a solution whenever one exists, and signals failure in finite time when a solution does not exist.

Robot Modeling and Control, Second Edition.
Mark W. Spong, Seth Hutchinson, and M. Vidyasagar

range of practical applications, are fairly easy to implement, and require only moderate computation time for most problems.

Path planning provides a geometric description of robot motion, but it does not consider any dynamic aspects of the motion. For example, for a manipulator arm, what should be the joint velocities and accelerations while traversing the path? For a car-like robot, what should be the acceleration profile along the path? These kinds of questions are addressed by a trajectory planner. The trajectory planner computes a function $q(t)$ that completely specifies the desired motion of the robot as it traverses the path.

We begin in Section 7.1 by revisiting the notion of **configuration space** that was first introduced in Chapter 1. We give a brief description of the geometry of the configuration space and describe how obstacles in the workspace can be mapped into the configuration space. Then, in Section 7.2, we consider the problem of planning the motion of polygon-shaped robots that translate in a planar workspace that contains polygonal objects. While this is a very special case, it has wide applicability to problems that involve mobile robots, such as the one shown in Figure 1.3. In Section 7.3 we present a first method that can be applied to general path planning problems, the so-called **artificial potential field** method. Potential field methods explore the configuration space by following the gradient of a function that rewards progress toward the goal while penalizing collision with objects in the workspace. It is generally not possible to construct such a function without introducing local minima, at which gradient descent algorithms will terminate without finding a solution. **Randomization** was introduced as a method to deal with this problem. In Section 7.4, we describe two methods that randomly generate a set of sample configurations, and use these as vertices in a graph that represents a set of free paths in the configuration space. Finally, since each of these methods generates a sequence of configurations, we describe how polynomial splines can be used to generate smooth trajectories from a sequence of configurations in Section 7.5.

7.1 The Configuration Space

In Chapter 3 we saw that the forward kinematic map can be used to determine the position and orientation of the end-effector frame given the vector of joint variables. Furthermore, the A matrices can be used to infer the position and orientation of any link of the robot. Since each link of the robot is assumed to be a rigid body, the A matrices can be used to infer the position of any point on the robot, given the values of the joint variables. Likewise, for a mobile robotic platform, if we rigidly attach a coordinate frame to the

robot, we can infer the position of any point on the mobile platform once we know the position and orientation of the body-attached frame. In the path planning literature a complete specification of the location of every point on the robot is referred to as a **configuration**, and the set of all possible configurations is referred to as the **configuration space**. In this section, we formalize the notion of a configuration space, including how it can be represented mathematically, how the presence of objects in the workspace impose constraints on the set of valid configurations, and the representation of paths in the configuration space.

7.1.1 Representing the Configuration Space

We denote by $\mathcal{Q}$ a representation of the configuration space. For example, as we saw in Chapter 2, for any rigid object we can specify the location of every point on the object by rigidly attaching a coordinate frame to the object and then specifying the position and orientation of this frame. Thus, for a rigid object moving in the plane we could represent a configuration by the triple $q = (x, y, \theta)$, and the configuration space could be represented by $\mathcal{Q} = \mathbb{R}^2 \times SO(2)$. Alternatively, we could choose to represent the orientation of the object as a point on the unit circle, S^1. In this case, the configuration space would be represented by $\mathcal{Q} = \mathbb{R}^2 \times S^1$.

For manipulator arms, the vector of joint variables often provides a convenient representation of a configuration. For a one-link revolute arm the configuration space is merely the set of orientations of the link, and thus $\mathcal{Q} = S^1$, where S^1 represents the unit circle. We can locally parameterize $\mathcal{Q}$ by a single parameter, the joint angle θ_1, exactly as was done in Chapter 3 using the DH convention. For the two-link planar arm we have $\mathcal{Q} = S^1 \times S^1 = T^2$, in which T^2 represents the torus, and we can represent a configuration by $q = (\theta_1, \theta_2)$. For a Cartesian arm, we have $\mathcal{Q} = \mathbb{R}^3$ and we can represent a configuration by $q = (d_1, d_2, d_3)$.

For mobile robots, we typically treat the robot as a single rigid object, ignoring the motion of individual components such as rotating wheels or propellers. For ground vehicles, the configuration space is typically represented as $\mathcal{Q} = SE(2)$, or, in cases where the orientation of the vehicle is not relevant, as $\mathcal{Q} = \mathbb{R}^2$. The latter case is specifically addressed below, in Section 7.2. For robots that move in three-dimensional space, such as air vehicles or underwater robots, we typically define the configuration space as $\mathcal{Q} = SE(3)$.

7.1.2 Configuration Space Obstacles

A collision occurs when any point on the robot contacts either an object in the workspace or the boundary of the workspace (e.g., walls or the floor), both of which are regarded as obstacles for the purposes of planning collision-free paths. Self-collision is also possible, for example if the end effector attached to a manipulator arm comes into contact with the base link, but we will not consider the case of self-collision here. To describe collisions we introduce some additional notation. We define the robot's workspace as the Cartesian space in which the robot moves, denoted by $\mathcal{W}$. In general, we consider $\mathcal{W} = \mathbb{R}^2$ for mobile robots that move in the plane, and $\mathcal{W} = \mathbb{R}^3$ for robots that move in three-dimensional space (e.g., non-planar robot arms and mobile robots such as air vehicles). We denote the subset of the workspace that is occupied by the robot at configuration q by $\mathcal{A}(q)$, and by $\mathcal{O}_i$ the subset of the workspace occupied by the i^{th} obstacle. To plan a collision free path we must ensure that the robot never reaches a configuration q that causes it to make contact with an obstacle. The set of configurations for which the robot collides with an obstacle is referred to as the **configuration space obstacle** and it is defined by

$$\mathcal{QO} = \{q \in \mathcal{Q} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}$$

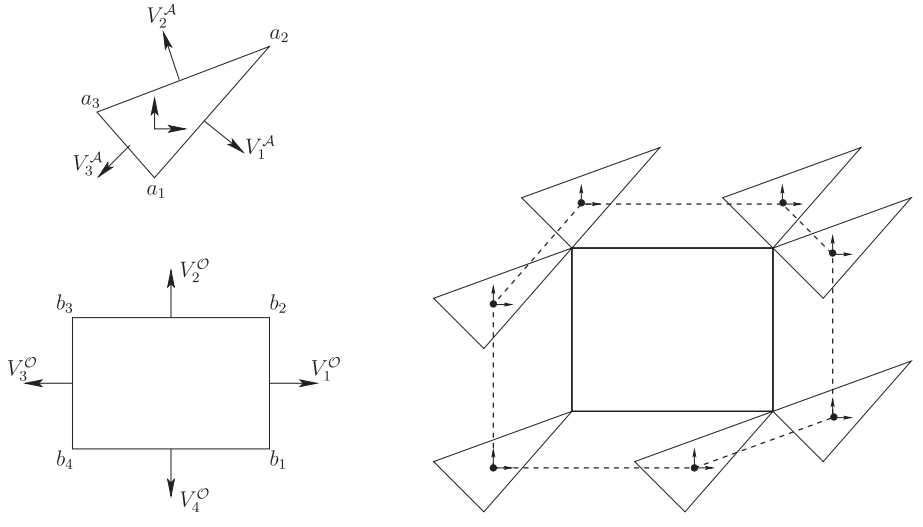
in which $\mathcal{O} = \bigcup \mathcal{O}_i$. The set of collision-free configurations, referred to as the free configuration space, is then simply the set difference

$$\mathcal{Q}_{\text{free}} = \mathcal{Q} \setminus \mathcal{QO}$$

Example 7.1 (A Rigid Body that Translates in the Plane)**.** *Consider a triangle-shaped mobile robot who possible motions include only translation in the plane as in Figure 7.1. For this case, the robot's configuration space is $\mathcal{Q} = \mathbb{R}^2$, so it is particularly easy to visualize both the configuration space and the configuration space obstacle region.*

If there is only one obstacle in the workspace and both the robot and the obstacle are convex polygons, it is a simple matter to compute the configuration space obstacle region $\mathcal{QO}$. Let $V_i^{\mathcal{A}}$ denote the vector that is normal to the i^{th} edge of the robot and $V_i^{\mathcal{O}}$ denote the vector that is normal to the i^{th} edge of the obstacle. Define a_i to be the vector from the origin of the robot's coordinate frame to the i^{th} vertex of the end effector, and b_j to be the vector from the origin of the world coordinate frame to the j^{th} vertex of the obstacle, as shown in Figure 7.1(a). The vertices of $\mathcal{QO}$ can be determined as follows.

- *For each pair $V_j^{\mathcal{O}}$ and $V_{j-1}^{\mathcal{O}}$, if $V_i^{\mathcal{A}}$ points between $-V_j^{\mathcal{O}}$ and $-V_{j-1}^{\mathcal{O}}$, then add to $\mathcal{QO}$ the vertices $b_j - a_i$ and $b_j - a_{i+1}$.*



(a) (b)

Figure 7.1: (a) The robot is a triangle-shaped rigid object in a two-dimensional workspace that contains a single rectangular obstacle. (b) The boundary of the configuration space obstacle $\mathcal{QO}$ (shown as a dashed line) can be obtained by computing the convex hull of the configurations at which the robot makes vertex-to-vertex contact with the single convex obstacle.

- *For each pair $V_i^{\mathcal{A}}$ and $V_{i-1}^{\mathcal{A}}$, if $V_j^{\mathcal{O}}$ points between $-V_i^{\mathcal{A}}$ and $-V_{i-1}^{\mathcal{A}}$, then add to $\mathcal{QO}$ the vertices $b_j - a_i$ and $b_{j+1} - a_i$.*

This is illustrated in Figure 7.1(b). Note that this algorithm essentially places the robot at all positions where vertex-to-vertex contact between robot and obstacle are possible. The origin of the robot's local coordinate frame at each such configuration defines a vertex of $\mathcal{QO}$. The polygon defined by these vertices is $\mathcal{QO}$.

If there are multiple convex obstacles $\mathcal{O}_i$, then the configuration space obstacle region is merely the union of the obstacle regions $\mathcal{QO}_i$ for the individual obstacles. For a nonconvex obstacle, the configuration space obstacle region can be computed by first decomposing the nonconvex obstacle into convex pieces $\mathcal{O}_i$ computing the configuration space obstacle region $\mathcal{QO}_i$ for each piece, and finally, computing the union of the $\mathcal{QO}_i$.

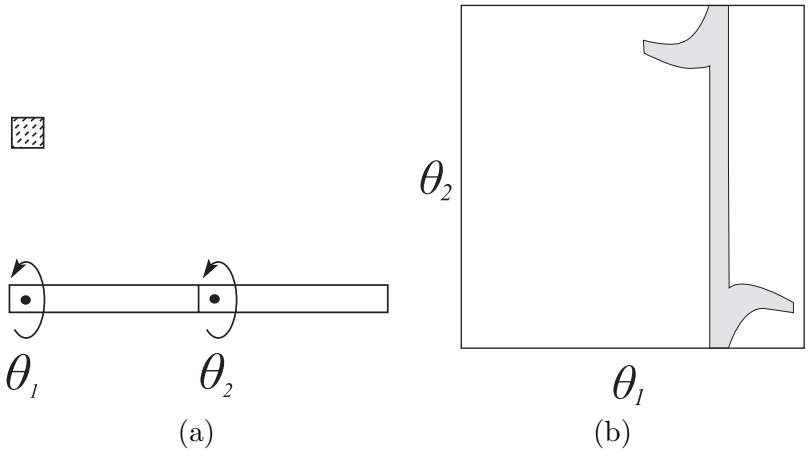


Figure 7.2: (a) The robot is a two-link planar arm and the workspace contains a single, small polygonal obstacle. (b) The corresponding configuration space obstacle region contains all configurations $q = (\theta_1, \theta_2)$ such that the arm at configuration q intersects the obstacle.

Example 7.2 (A Two-Link Planar Arm)**.** *The computation of $\mathcal{QO}$ is more difficult for robots with revolute joints. Consider a two-link planar arm in a workspace containing a single obstacle as shown in Figure 7.2(a). The configuration space obstacle region is illustrated in Figure 7.2(b).*

For values of θ_1 very near $\pi/2$, the first link of the arm collides with the obstacle. When the first link is near the obstacle (θ_1 near $\pi/2$), for some values of θ_2 the second link of the arm collides with the obstacle. The region $\mathcal{QO}$ shown in Figure 7.2(b) was computed using a discrete grid on the configuration space. For each cell in the grid, a collision test was performed, and the cell was shaded when a collision occurred. This is only an approximate representation of $\mathcal{QO}$; however, for robots with revolute joints, exact representations are very expensive to compute, and therefore such approximate representations are often used for robot arms with a few degrees of freedom.

Computing $\mathcal{QO}$ for the two-dimensional case of $\mathcal{Q} = \mathbb{R}^2$ and polygonal obstacles is straightforward, but, as can be seen from the two-link planar arm example, computing $\mathcal{QO}$ becomes difficult for even moderately complex configuration spaces. In the general case (for example, articulated arms or rigid bodies that can both translate and rotate), the problem of computing a representation of the configuration space obstacle region is intractable. One of the reasons for this complexity is that the size of the representation of the configuration space tends to grow exponentially with the number of

degrees of freedom. This is easy to understand intuitively by considering the number of n-dimensional unit cubes needed to fill a space of size k. For the one-dimensional case k unit intervals will cover the space. For the two-dimensional case k^2 squares are required. For the three-dimensional case k^3 cubes are required, and so on. Therefore, in this chapter we will develop methods that avoid the construction of an explicit representation of $\mathcal{QO}$ or of $\mathcal{Q}_{\text{free}}$.

7.1.3 Paths in the Configuration Space

The path planning problem is to find a path from an initial configuration q_{s} to a final configuration q_{f}, such that the robot does not collide with any obstacle as it traverses the path. More formally, a collision-free path from q_{s} to q_{f} is a continuous map, $\gamma : [0, 1] \to \mathcal{Q}_{\text{free}}$, with $\gamma(0) = q_{\text{s}}$ and $\gamma(1) = q_{\text{f}}$. We will develop path planning methods that compute a sequence of discrete configurations (set points) in the configuration space. In Section 7.5 we will show how smooth trajectories can be generated from such a sequence of set points.

7.2 Path Planning for $\mathcal{Q} = \mathbb{R}^2$

Before considering the general path planning problem, we first consider the special case when $\mathcal{Q} = \mathbb{R}^2$, and in particular, situations in which both the robot and obstacles can be represented as polygons in the plane, and in which the robot can move in any arbitrary direction (e.g., there are no constraints on the directions of motion such as would exist for car-like mobile robots, which cannot move in the direction perpendicular to the car's heading). Furthermore, we assume that the robot's workspace, and thus also its configuration space, is bounded. This is often an acceptable approximation for situations in which mobile robots navigate in workspaces such as warehouses, factory floors, office buildings, etc.

In this case, as we have seen in Example 7.1, it is a simple matter to construct an explicit representation of the configuration space obstacle region (and thus, of the free configuration space). In this section, we present three algorithms that can be used to construct collision-paths, given as input an explicit, polygonal representation of the configuration space obstacle regions. All three algorithms build graph data structures to represent the set of possible paths, and thus, once these representations have been constructed, the path planning problem is reduced to a graph search problem, for which many algorithms exist.

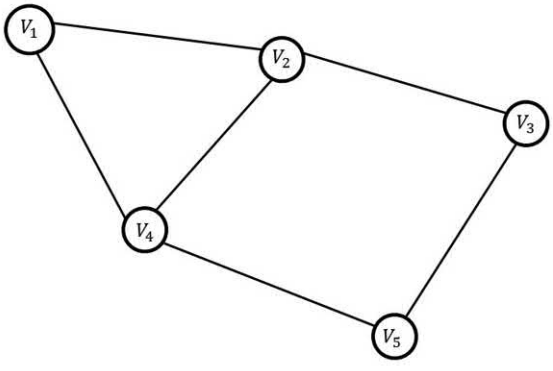


Figure 7.3: A graph with five vertices and six edges.

A graph is defined as $G = (V, E)$, in which V is a set of vertices, and E is a set of edges, each of which corresponds to a pair of vertices. An example is shown in Figure 7.3, in which the graph has vertices and edges

$$\begin{aligned} V &= \{v_1, v_2, ...v_5\} \\ E &= \{(v_1, v_2), (v_2, v_3), (v_2, v_4), (v_1, v_4), (v_4, v_5), (v_3, v_5)\} \end{aligned}$$

Vertices are said to be adjacent if they are connected by an edge. A path from vertex v_i to vertex v_j is a sequence of adjacent vertices, beginning at v_i and ending at v_j, or equivalently, the set of edges defined by the pairwise adjacent vertices in this path. For example, the edges $(v_1, v_4), (v_4, v_2), (v_2, v_3), (v_3, v_5)$ define a path from v_1 to v_5.

7.2.1 The Visibility Graph

A visibility graph is a graph whose vertices can "see" each other, that is, edges in the graph correspond to paths that do not intersect the interior of any obstacle; the 'line of sight" between adjacent vertices is unoccluded. For path planning in a polygonal configuration space, the set of vertices V includes (a) the start and goal configurations q_{s} and q_{f}, and (b) every vertex of the configuration space obstacles. The set of edges, E, includes all edges (v_i, v_j) such that the line segment joining v_i to v_j lies entirely in the free configuration space, or such that (v_i, v_j) corresponds to an edge of a polygonal configuration space obstacle. Figure 7.4(a) shows a configuration space containing polygonal obstacle regions, and Figure 7.4(b) shows the corresponding visibility graph.

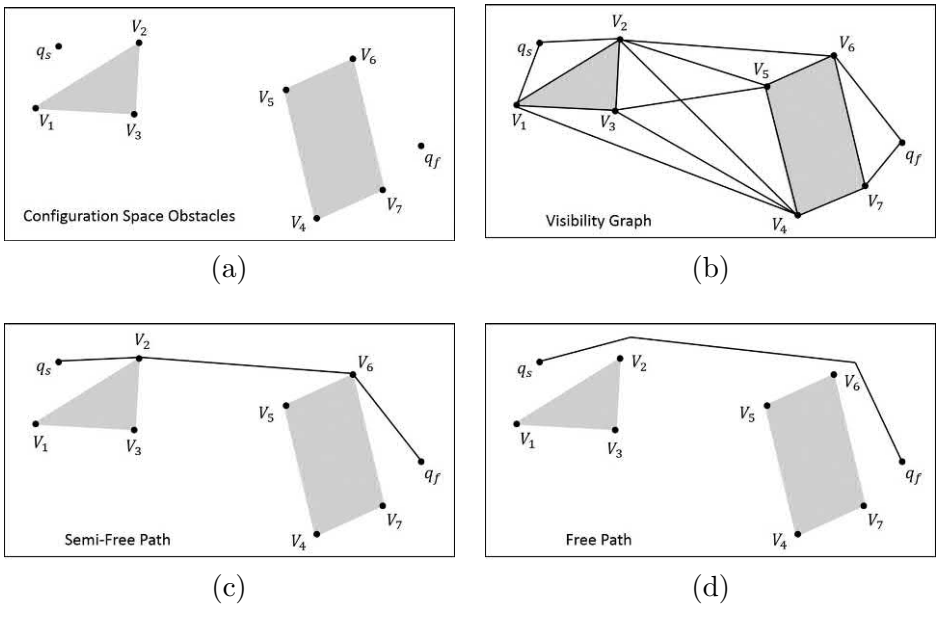


Figure 7.4: This figure illustrates the construction of a free path using the visibility graph. (a) An environment that contains polygonal obstacles, with start and goal configurations q_{s} and q_{f}. (b) The visibility graph. (c) A semi-free path from q_{s} to q_{f}. (d) A free path from q_{s} to q_{f}, obtained by a small perturbation of the path vertices that contact obstacle vertices for the semi-free path.

Any path in the visibility graph corresponds to a semi-free path in the configuration space, where by **semi-free** we mean that the path lies in the closure of the free configuration space (i.e., either in the free configuration space, or in the boundary of the configuration space obstacle region). However, it is always possible to deform a semi-free path into a **free** path by a small perturbation of the semi-free path. As an example, Figure 7.4(c) shows a semi-free path from q_{s} to q_{f} (note that the path contacts the boundary of the configuration space obstacles at the obstacle vertices v_2 and v_6), and Figure 7.4(d) shows a corresponding free path.

It can be shown (Problem 7–7) that the visibility graph contains the shortest semi-free paths for any q_{s} and q_{f}, provided these q_{s} and q_{f} are included in the construction of the visibility graph. It should be noted that such shortest paths may not be desirable in practice, since they bring the robot arbitrarily near to obstacles in the workspace, which could easily lead to collision if there is any uncertainty in the robot's location or in the map of the environment. For this reason, it is often preferable to plan paths that

maximize the clearance between the robot and any obstacles. The so-called **generalized Voronoi diagram**, discussed next, can be used to construct such paths.

7.2.2 The Generalized Voronoi Diagram

Consider a set of discrete points in the plane, $P = \{p_1, \ldots, p_n\}$. For each point p_i, we define its Voronoi cell to be the set of points in the plane that are closer to p_i than to any other $p_j \in P$,

$$\text{Vor}(p_i) = \{x \in \mathbb{R}^2 \mid \|x - p_i\| \leq \|x - p_j\|, \text{ for all } j \neq i\}$$

Two Voronoi regions are adjacent if $\text{Vor}(p_i) \cap \text{Vor}(p_j) \neq \emptyset$, in which case the intersection is a straight-line segment, referred to as a Voronoi edge, defined as

$$E_{ij} = \{x \in \mathbb{R}^2 \mid \|x - p_i\| = \|x - p_j\| \leq \|x - p_k\|, \text{for all } k \neq i, j\}$$

The Voronoi edge E_{ij} comprises the set of points in the plane that are minimally equidistant to the points p_i and p_j.

If we consider the points p_i as obstacles, then a collision-free path is merely a path that does not include any of the p_i. Define the **clearance** of a path γ to be the minimum distance between the path and any point $p \in P$, i.e.,

$$\rho(\gamma) = \min_{t \in [0,1]} \min_{p \in P} \|\gamma(t) - p\|$$

If q_s and q_f lie in Voronoi edges, then the maximum clearance path γ^* connecting q_s and q_f is contained completely in the set of Voronoi edges.

All of these concepts can be generalized from the case of discrete points to the case of polygons in the plane. A polygon consists of a set of vertices that are connected by edges. Together, these vertices and edges are sometimes referred to as the **features** that define the polygon. Thus, a polygon feature f is either a vertex, v or an edge, $e = (v, v')$. Now, let P be the set of features corresponding to the polygonal configuration space obstacles. The distance from a point x to a feature f of a polygon is defined as

$$d(x, f) = \begin{cases} \|x - v\| & : \quad f = v \\ \min_{\alpha \in [0,1]} \|x - (v - \alpha(v - v'))\| & : \quad f = e = (v, v') \end{cases}$$

in which $\|x - v\|$ is the Euclidean distance between the point x and the vertex v, and the expression $\min_{\alpha \in [0,1]} \|x - (v - \alpha(v - v'))\|$ gives the minimum distance from the point x to the edge $e = (v, v')$.

We define the Voronoi cell of feature f to be the set of points in the plane that are closer to f than to any other feature $f' \in P$,

$$\text{Vor}(f) = \{x \in \mathbb{R}^2 \mid d(x, f) \leq d(x, f'), \text{ for } f \neq f' \in P\}$$

We can also generalize the concept of Voronoi edges to be the set of points minimally equidistant to two features,

$$E_{ij} = \{x \in \mathbb{R}^2 \mid d(x, f_i) = d(x, f_j) \leq d(x, f_k), \text{for all } k \neq i, j\}$$

Since the set of features includes only points and line segments, the set of Voronoi edges will include only line segments that are minimally equidistant to two vertices or two edges, and sections of parabolas that are minimally equidistant to a vertex and an edge. We define the **generalized Voronoi diagram** as the graph $G = (V, E)$ whose edges are these Voronoi edges, and whose vertices are points at which multiple Voronoi edges intersect.

For the case of polygonal obstacles, we define the clearance of a path γ to be the minimum distance between the path and any feature $f \in P$, i.e.,

$$\rho(\gamma) = \min_{t \in [0,1]} \min_{f \in P} d(\gamma(t), f)$$

and, as above, if q_s and q_f lie in Voronoi edges, then the maximum clearance path γ^* connecting q_s and q_f is contained completely in the set of Voronoi edges.

For the case when q_s and q_f do not lie in Voronoi edges, it is necessary to also construct collision-free paths from each of q_s and q_f to Voronoi edges. This is actually quite simple to do. If, for example, q_s lies in the Voronoi region for a particular feature, f, a straight-line path from q_s that follows the gradient $\nabla d(q_\text{s}, f)$ will arrive to a Voronoi edge before reaching any other feature f' (Problem 7–8). For the case of an edge feature, this gradient is given by

$$\nabla d(q_\text{s}, f) = \frac{q_\text{s} - q^*}{\|q_\text{s} - q^*\|}$$

in which q^* is the closest point in the edge to q_s.

The path planning problem can now be solved as follows:

1. Find the straight-line path from q_s to the nearest Voronoi edge. Let ${q_\text{s}}'$ denote the point at which this path intersects the Voronoi edge.

2. Find the straight-line path from q_f to the nearest Voronoi edge. Let ${q_\text{f}}'$ denote the point at which this path intersects the Voronoi edge.

3. Find a path in the generalized Voronoi diagram from ${q_\text{s}}'$ to ${q_\text{f}}'$.

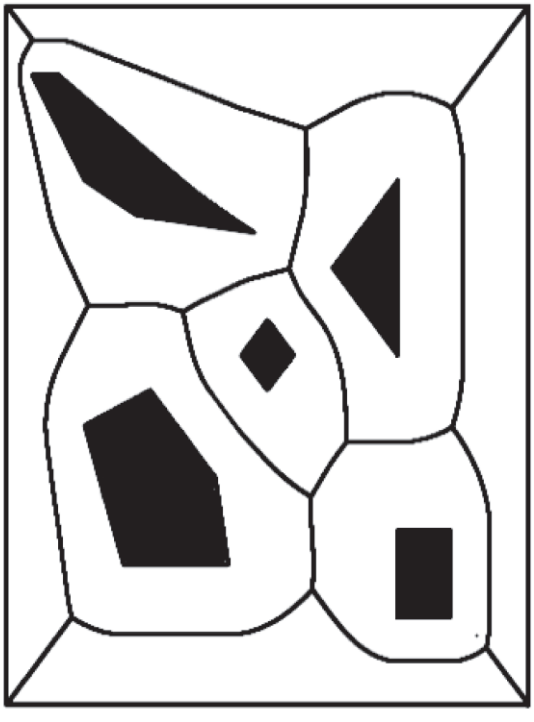


Figure 7.5: A polygonal configuration space containing five obstacles, and its generalized Voronoi diagram.

Efficient algorithms exist for computing the exact generalized Voronoi diagram, but in practice numerical, grid-based algorithms are often used, since (a) they are much easier to implement (b) the error induced by resorting to a discrete grid is typically small relative to the clearance achieved by paths that lie in the generalized Voronoi diagram, and (c) grid-based representations are feasible for two-dimensional space. Figure 7.5 shows a polygonal configuration space and the corresponding generalized Voronoi diagram.

7.2.3 Trapezoidal Decompositions

The visibility graph and the generalized Voronoi diagram are both graphs whose edges correspond directly to paths in the configuration space. For the visibility graph, edges correspond to portions of specific semi-free paths, while for the generalized Voronoi diagram, each Voronoi edge corresponds to a portion of a specific free path. Neither of these representations explicitly captures any information about the geometry of the free configuration space.

In contrast, a **spatial decomposition** of the free configuration space is a collection of regions $\{R_1, \ldots, R_m\}$ such that $\bigcup R_i = \mathcal{Q}_{\text{free}}$ and $\text{int}(R_i) \cap \text{int}(R_j) = \emptyset$ for all $i \neq j$, where $\text{int}(R)$ denotes the interior of region R. The regions R_i explicitly define the geometry of the free configuration space. A **convex spatial decomposition** has the additional property that each R_i is convex. Convex regions have the appealing property that for any $q_i, q_j \in R$, the line segment connecting them lies completely within R, that is, $q_i - \alpha(q_i - q_j) \in R$, for all $\alpha \in [0, 1]$. Therefore, constructing a free path between two configurations in the same convex subset of $\mathcal{Q}_{\text{free}}$ is trivial. A **trapezoidal decomposition** is a special case from the class convex spatial decomposition that applies to the case of polygons in the plane.

Figure 7.6(a) shows trapezoidal decomposition for the free configuration space for the case of polygonal obstacles. Each region in the decomposition is a trapezoid consisting of polygon edges and vertical line segments incident upon vertices of the polygons[2] A popular algorithm for constructing a trapezoidal decomposition proceeds by sweeping a vertical line across the configuration space, stopping at each vertex of the configuration space obstacle region, and making appropriate updates to the decomposition. Figure 7.6(b) shows one step in this process. At this stop of the sweep line, ℓ, regions R_3 and R_4 are "closed" and region R_5 is "opened."

The **connectivity graph** for a trapezoidal decomposition encodes adjacency relationships between the regions of the decomposition. The vertices of the connectivity graph correspond to the regions, and an edge exists between R_i and R_j if the two regions are adjacent (i.e., if $R_i \cap R_j \neq \emptyset$). Figure 7.6(c) shows the connectivity graph for the trapezoidal decomposition shown in Figure 7.6(a). For a given q_{s} and q_{f}, the path planning problem can be solved as follows.

1. Determine the region R_i that contains the initial configuration q_{s}.
2. Determine the region R_j that contains the final configuration q_{f}.
3. Find a path in the connectivity graph from R_i to R_j.
4. Construct a piecewise linear path from q_{s} and q_{f} that passes successively through the cells found in Step 3, crossing from one region to the next at the midpoint of the boundary of the two regions.

[2]Note that triangles are considered as trapezoids that have one side of length zero, and rectangles are special cases of trapezoids for which opposite sides are parallel.

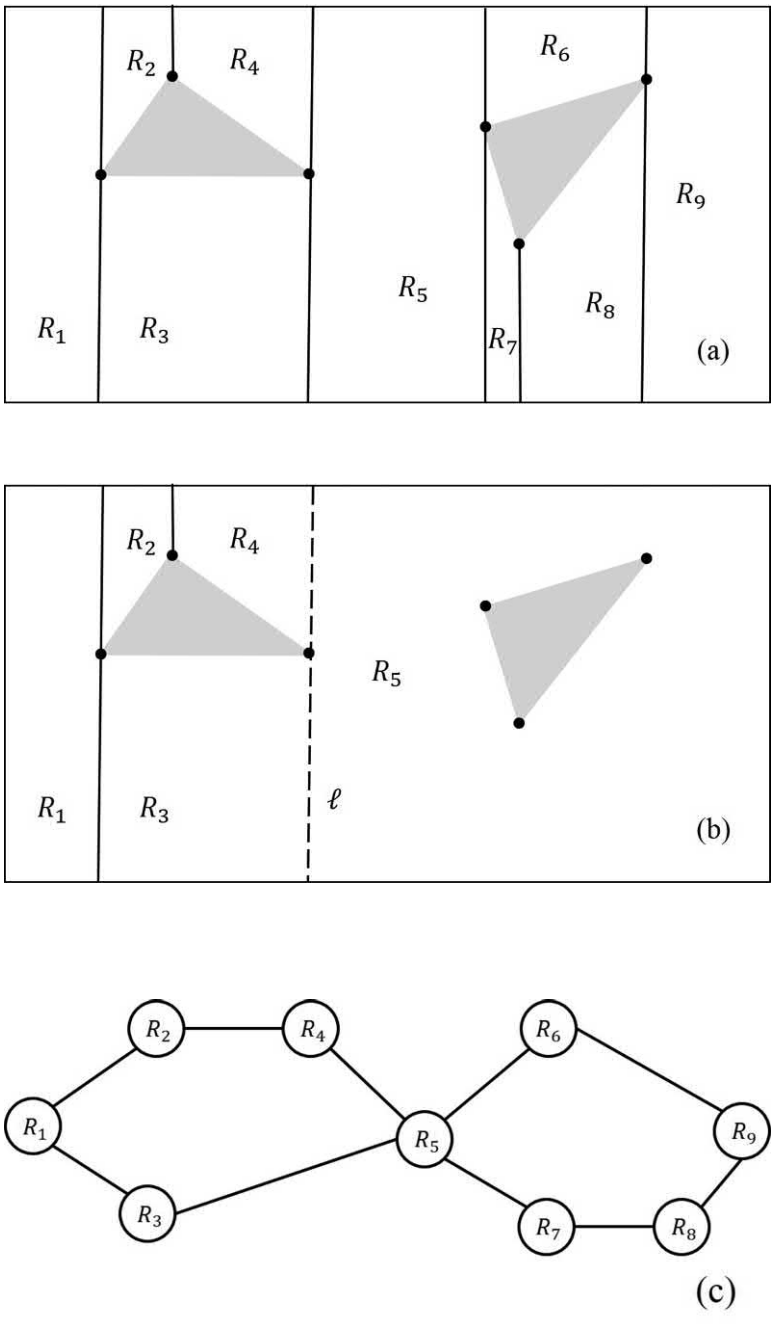


Figure 7.6: A trapezoidal decomposition for the free configuration space for the case of polygonal obstacles.

7.3 Artificial Potential Fields

The methods described in Section 7.2 are applicable only for very simple configuration spaces. For more complex configuration spaces (e.g., $SE(2)$ for a car-like robot, $SE(3)$ for an air vehicle, or the n-torus for an n-link arm) it is typically not feasible to build an explicit representation of $\mathcal{QO}$ or of $\mathcal{Q}_{\text{free}}$. An alternative is to develop a search algorithm that incrementally explores $\mathcal{Q}_{\text{free}}$ while searching for a path. One popular strategy for exploring $\mathcal{Q}_{\text{free}}$ uses an **artificial potential field** to guide the search.

The basic idea behind the potential field approach is to treat the robot as a point particle in the configuration space under the influence of an artificial potential field U. The field U is constructed so that the robot is attracted to the final configuration q_{f} while being repelled from the boundaries of $\mathcal{QO}$. If possible, U is constructed so that there is a single global minimum of U at q_{f} and there are no local minima. Unfortunately, it is typically difficult or even impossible to construct such a field.

In most potential field path planners, the field U is constructed as an additive field consisting of one component that attracts the robot to q_{f} and a second component that repels the robot from the boundary of $\mathcal{QO}$

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

Given this formulation path planning can be treated as an optimization problem, namely the problem of finding the global minimum of U starting from initial configuration q_{s}, and gradient descent methods are often used to find a solution. In physics, a conservative force field can be written as the negative gradient of a potential function. Thus, we may interpret gradient descent as being analogous to a particle moving under the influence of the force $F = -\nabla U$. Below we frequently refer to attractive and repulsive forces in order to exploit this natural intuition when defining potential fields.

We develop the potential field method for path planning in two stages. First, in Section 7.3.1, we develop the method for the special case of $\mathcal{Q} = \mathbb{R}^n$. By doing so, we can easily define U in terms of Euclidean distances, without dealing with the problem of defining metrics on arbitrary configuration spaces. This allows a straightforward initial development of the concepts and algorithms. In Section 7.3.2 we expand the approach to arbitrary configuration spaces. Rather than explicitly defining potential fields on these configuration spaces, we will define a collection of so-called **workspace potential fields**, and then show how the gradient of the corresponding configuration space potential can be computed using the Jacobian of the forward kinematic map. For both cases, we describe specific gradient descent algorithms for path planning.

7.3.1 Artificial Potential Fields for $\mathcal{Q} = \mathbb{R}^n$

For the case of $\mathcal{Q} = \mathbb{R}^n$, we will define an attractive potential in terms of the Euclidean distance to the goal, and a repulsive potential in terms of the Euclidean distance to the nearest obstacle boundary. We then describe how gradient descent methods can be used to find a collision-free path. Because gradient descent often fails for situations in which the potential field has multiple local minima, we end with a discussion of how randomization can be used to escape local minima of the potential function.

The Attractive Field

To attract the robot to its goal configuration, we will define an attractive potential field U_{att}. At the goal configuration $q = q_{\text{f}}$. There are several criteria that the potential field U_{att} should satisfy. First, U_{att} should be monotonically increasing with the distance to the goal configuration. The simplest choice for such a field grows linearly with this distance, a so-called **conic well potential**

$$U_{\text{att}}(q) = \zeta ||q - q_{\text{f}}||$$

in which ζ is a parameter used to scale the effects of the attractive potential. The gradient of such a field has unit magnitude everywhere except the goal configuration, where it is zero. This can lead to stability problems since there is a discontinuity in the attractive force at the goal position.

The **parabolic well potential** given by

$$U_{\text{att}}(q) = \frac{1}{2}\zeta ||q - q_{\text{f}}||^2$$

is continuously differentiable and increases monotonically with distance to the goal configuration. The attractive force is equal to the negative gradient of U_{att}, which is given by (Problem 7–9)

$$F_{\text{att}}(q) = -\nabla U_{\text{att}}(q) = -\zeta(q - q_{\text{f}}) \tag{7.1}$$

For the parabolic well the attractive force is a vector directed toward q_{f} with magnitude linearly related to the distance to q from q_{f}.

While this force converges linearly to zero as q approaches q_{f}, which is a desirable property, it grows without bound as q moves away from q_{f}. If q_{s} is very far from q_{f}, this may produce an initial attractive force that is very large. For this reason we may choose to combine the quadratic and

conic potentials so that the conic potential is active when q is distant from q_{f}, and the quadratic potential is active when q is near q_{f}. Of course, it is necessary that the gradient be defined at the boundary between the conic and quadratic fields. Such a field can be defined by

$$U_{\mathrm{att}}(q) = \begin{cases} \frac{1}{2}\zeta||q - q_{\mathrm{f}}||^2 & : \quad ||q - q_{\mathrm{f}}|| \leq d \\ \\ d\zeta||q - q_{\mathrm{f}}|| - \frac{1}{2}\zeta d^2 & : \quad ||q - q_{\mathrm{f}}|| > d \end{cases}$$

in which d is the distance that defines the transition from conic to parabolic well. In this case the force is given by

$$F_{\mathrm{att}}(q) \quad = \quad \begin{cases} -\zeta(q - q_{\mathrm{f}}) & : \quad ||q - q_{\mathrm{f}}|| \leq d \\ \\ -d\zeta \dfrac{(q - q_{\mathrm{f}})}{||q - q_{\mathrm{f}}||} & : \quad ||q - q_{\mathrm{f}}|| > d \end{cases}$$

The gradient is well defined at the boundary of the two fields since at the boundary $d = ||q - q_{\mathrm{f}}||$ and the gradient of the quadratic potential is equal to the gradient of the conic potential $F_{\mathrm{att}}(q) = -\zeta(q - q_{\mathrm{f}})$.

The Repulsive Field

In order to prevent collisions between the robot and obstacles we will define a **repulsive potential field** that grows as the configuration approaches the boundary of $\mathcal{QO}$. There are several criteria that such a repulsive field should satisfy. It should repel the robot from obstacles, never allowing the robot to collide with an obstacle, and, when the robot is far away from an obstacle, that obstacle should exert little or no influence on the motion of the robot. One way to achieve this is to define a potential function whose value approaches infinity as the configuration approaches an obstacle boundary, and whose value decreases to zero at a specified distance from the obstacle boundary.

We define $\rho(q)$ to be the distance from configuration q to the boundary of $\mathcal{QO}$,

$$\rho(q) = \min_{q' \in \partial\mathcal{QO}} \|q - q'\|$$

in which $\partial\mathcal{QO}$ denotes the boundary of the configuration space obstacle region. We define ρ_0 to be the distance of influence of an obstacle. This means that an obstacle will not repel the robot if the distance from q to the obstacle is greater than ρ_0.

One potential function that meets the criteria described above is given by

$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & : \quad \rho(q) \leq \rho_0 \\ \\ 0 & : \quad \rho(q) > \rho_0 \end{cases}$$

in which η is a parameter used to scale the effects of the attractive potential. The repulsive force is equal to the negative gradient of U_{rep}. For $\rho(q) \leq \rho_0$, this force is given by (Problem 7–13)

$$F_{\text{rep}}(q) = \eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)\frac{1}{\rho^2(q)}\nabla\rho(q) \tag{7.2}$$

If $\mathcal{QO}$ is convex and b is the point on the boundary of $\mathcal{QO}$ that is closest to q, then $\rho(q) = ||q - b||$, and its gradient is

$$\nabla\rho(q) = \frac{q - b}{||q - b||}$$

that is, the unit vector directed from b toward q.

If the obstacle is not convex, then the distance function ρ will not necessarily be differentiable everywhere, which implies discontinuity in the force vector. Figure 7.7 illustrates such a case. Here the obstacle region is defined by two rectangular obstacles. For all configurations to the left of the dashed line the force vector points to the right, while for all configurations to the right of the dashed line the force vector points to the left. Thus, when q crosses the dashed line, a discontinuity in force occurs. There are various ways to deal with this problem. The simplest of these is merely to ensure that the regions of influence of distinct obstacles do not overlap.

Gradient Descent Planning

Gradient descent is a well-known approach for solving optimization problems. The idea is simple. Starting at the initial configuration, take a small step in the direction of the negative gradient (which is the direction that decreases the potential as quickly as possible). This gives a new configuration, and the process is repeated until the final configuration is reached. More formally, a gradient descent algorithm constructs a sequence[3] of configurations, $q^0, q^1, \ldots, q^m$ such that $q^0 = q_{\text{s}}$ and $q^m = q_{\text{f}}$. The configuration

[3]Note that q^i is used to denote the value of q at the i^{th} iteration (not the i^{th} component of the vector q).

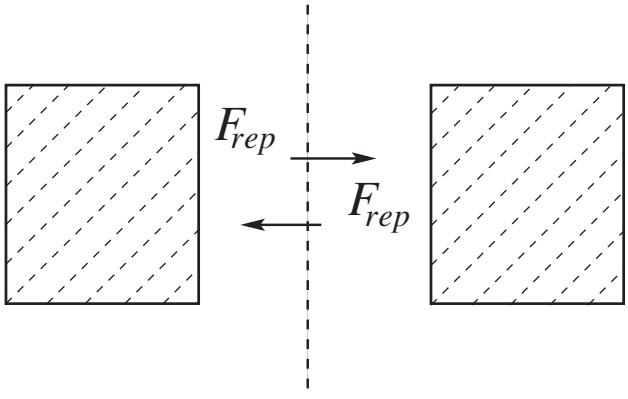


Figure 7.7: In this case the gradient of the repulsive potential given by Equation (7.2) is not continuous. In particular, the gradient changes discontinuously when q crosses the line midway between the two obstacles.

at step $i+1$ is given by

$$q^{i+1} = q^i - \alpha^i \nabla U(q^i) \tag{7.3}$$

The scalar parameter α^i scales the step size at the i^{th} iteration. Some variations of gradient descent replace the gradient $\nabla U(q^i)$ by a unit vector in the direction of the gradient; in this case α^i completely determines the step size at the i^{th} iteration. It is important that α^i be small enough that the robot is not allowed to "jump into" obstacles while being large enough that the algorithm does not require excessive computation time. In motion planning problems the choice for α^i is often made on an ad hoc or empirical basis, for example, based on the distance to the nearest obstacle or to the goal. A number of systematic methods for choosing α^i can be found in the optimization literature.

It is unlikely that we will ever exactly satisfy the condition $q^i = q_{\mathrm{f}}$ and for this reason gradient descent algorithms typically terminate when q^i is sufficiently near the goal configuration q_{f}, for example when $||q^i - q_{\mathrm{f}}|| < \epsilon$, where we choose ϵ to be a sufficiently small constant, based on the task requirements.

Escaping Local Minima

The problem that plagues all gradient descent algorithms is the possible existence of local minima in the potential field. For appropriate choice of α^i in (7.3), it can be shown that the gradient descent algorithm is guaranteed

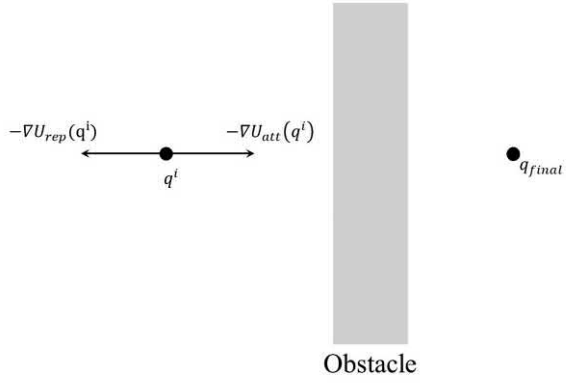


Figure 7.8: The configuration q^i is a local minimum in the potential field. At q^i the attractive force exactly cancels the repulsive force and the planner fails to make further progress.

to converge to a minimum in the field, but there is no guarantee that this minimum will be the global minimum. In our case this implies that there is no guarantee that this method will find a path to q_f. An simple example of this situation is shown in Figure 7.8.

This problem has long been known in the optimization community, where probabilistic methods such as simulated annealing have been developed to cope with it. Similarly, **randomized methods** have been developed to deal with this and other problems in robot motion planning. One method for escaping local minima combines gradient descent with randomization. This approach uses gradient descent until the planner finds itself stuck in a local minimum, and then uses a random walk to escape the local minimum. This requires solving two problems: determining when the planner is stuck in a local minimum and defining the random walk.

Typically, a heuristic is used to recognize a local minimum. For example, if several successive q^i lie within a small region of the configuration space, it is likely that there is a nearby local minimum. For example, if for some small positive ϵ_m we have $\|q^i - q^{i+1}\| < \epsilon_m$, $\|q^i - q^{i+2}\| < \epsilon_m$, and $\|q^i - q^{i+3}\| < \epsilon_m$ then assume q^i is near a local minimum, provided q^i is not sufficiently near the goal configuration.

Defining the random walk requires a bit more care. One approach is to simulate Brownian motion. The random walk consists of t random steps. A random step from $q = (q_1, \ldots, q_n)$ is obtained by randomly adding a small

fixed constant to each q_i,

$$q_{\text{random}-\text{step}} = (q_1 \pm v_1, \ldots, q_n \pm v_n)$$

with v_i a fixed small constant and the probability of adding $+v_i$ or $-v_i$ equal to 1/2 (that is, a uniform distribution). Without loss of generality, assume that $q = 0$. We can use probability theory to characterize the behavior of the random walk consisting of t random steps. In particular, if q^t is the configuration reached after t random steps, the probability density function for $q^t = (q_1, \ldots, q_n)$ is given by

$$p_i(q_i, t) \approx \frac{1}{v_i\sqrt{2\pi t}} \exp\left(-\frac{q_i^2}{2v_i^2 t}\right)$$

which is a zero mean Gaussian density function[4] with variance $v_i^2 t$. This is a result of the central limit theorem, which states that the probability density function for the sum of k independent, identically distributed random variables tends to a Gaussian density function as $k \to \infty$. The variance $v_i^2 t$ essentially determines the range of the random walk. If certain characteristics of local minima (for example, the size of the basin of attraction) are known in advance, these can be used to select the parameters v_i and t. Otherwise, they can be determined empirically or based on weak assumptions about the potential field.

7.3.2 Potential Fields for $\mathcal{Q} \neq \mathbb{R}^n$

For the case of $\mathcal{Q} \neq \mathbb{R}^n$, it is difficult to construct a potential field directly on the configuration space, and difficult to compute the gradient of such a field. The reasons for this include the difficulty of computing shortest distances to configuration space obstacles, the complex geometry of the configuration space itself, and the computational cost of computing an explicit representation of the boundary of the configuration space obstacle region. For this reason, when $\mathcal{Q} \neq \mathbb{R}^n$ we will define **workspace potential fields** directly on the workspace of the robot, and then map the gradients of these fields to a corresponding gradient for a configuration space potential function.

In particular, for an n-link arm, we will define a potential field for each of the origins of the n DH frames (excluding the fixed, frame 0). These

[4] A Gaussian density function is the classical bell shaped curve. The mean indicates the center of the curve (the peak of the bell) and the variance indicates the width of the bell. The probability density function (pdf) tells how likely it is that the variable q_i will lie in a certain interval. The higher the pdf values, the more likely that q_i will lie in the corresponding interval.

workspace potential fields will attract the origins of the DH frames to their goal locations while repelling them from obstacles. We will use these fields to define motions in the configuration space using the manipulator Jacobian matrix. A similar approach can be used for mobile robots. In this case, we define a set of **control points** on the robot that are sufficient to fully constrain its position, and define workspace potentials for each of these. For a mobile robot in the plane, two points are sufficient, while three (noncollinear) points are sufficient for free-flying robots.

The Attractive Field

To attract the robot to its goal configuration, we will define an attractive potential field $U_{\text{att},i}$ for o_i, the origin of the i^{th} DH frame. When all n origins reach their goal positions, the arm will have reached its goal configuration. As above, choices include the conic well and parabolic well potentials.

If we denote the position of the origin of the i^{th} DH frame by $o_i(q)$, then the conic well potential is given by

$$U_{\text{att},i}(q) = \zeta_i ||o_i(q) - o_i(q_{\text{f}})||$$

in which ζ_i is a parameter used to scale the effects of the attractive potential. The parabolic well potential is given by

$$U_{\text{att},i}(q) = \frac{1}{2}\zeta_i ||o_i(q) - o_i(q_{\text{f}})||^2$$

For the parabolic well, the workspace attractive force for o_i is equal to the negative gradient of $U_{\text{att},i}$, which is given by

$$F_{\text{att},i}(q) = -\nabla U_{\text{att},i}(q) = -\zeta(o_i(q) - o_i(q_{\text{f}}))$$

which is a vector directed toward $o_i(q_{\text{f}})$ with magnitude linearly related to the distance to $o_i(q)$ from $o_i(q_{\text{f}})$.

We can combine the conic and parabolic well potentials as we did in Section 7.3.1, which yields

$$U_{\text{att},i}(q) = \begin{cases} \frac{1}{2}\zeta_i ||o_i(q) - o_i(q_{\text{f}})||^2 & : \quad ||o_i(q) - o_i(q_{\text{f}})|| \leq d \\ \\ d\zeta_i ||o_i(q) - o_i(q_{\text{f}})|| - \frac{1}{2}\zeta_i d^2 & : \quad ||o_i(q) - o_i(q_{\text{f}})|| > d \end{cases}$$

in which d is the distance that defines the transition from conic to parabolic

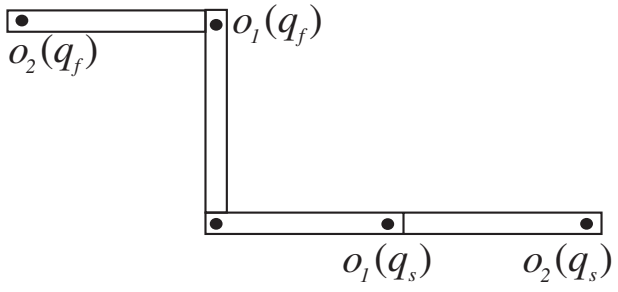


Figure 7.9: The initial configuration for the two-link arm is given by $\theta_1 = \theta_2 = 0$ and the final configuration is given by $\theta_1 = \theta_2 = \pi/2$. The origins for DH frames 1 and 2 are shown at both q_s and q_f.

well. In this case the workspace force for o_i is given by

$$F_{\mathrm{att},i}(q) = \begin{cases} -\zeta_i(o_i(q) - o_i(q_\mathrm{f})) & : \ ||o_i(q) - o_i(q_\mathrm{f})|| \leq d \\ -d\zeta_i \dfrac{(o_i(q) - o_i(q_\mathrm{f}))}{||o_i(q) - o_i(q_\mathrm{f})||} & : \ ||o_i(q) - o_i(q_\mathrm{f})|| > d \end{cases} \tag{7.4}$$

The gradient is well defined at the boundary of the two fields since at the boundary $d = ||o_i(q) - o_i(q_\mathrm{f})||$ and the gradient of the quadratic potential is equal to the gradient of the conic potential $F_{\mathrm{att},i}(q) = -\zeta_i(o_i(q) - o_i(q_\mathrm{f}))$.

Example 7.3 (Two-Link Planar Arm)**.** *Consider the two-link planar arm shown in Figure 7.9 with $a_1 = a_2 = 1$ and with initial and final configurations given by*

$$q_\mathrm{s} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \qquad q_\mathrm{f} = \begin{bmatrix} \frac{\pi}{2} \\ \frac{\pi}{2} \end{bmatrix}$$

Using the forward kinematic equations for this arm (see Example 3.3.1) we obtain

$$o_1(q_\mathrm{s}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad o_1(q_\mathrm{f}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad o_2(q_\mathrm{s}) = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \qquad o_2(q_\mathrm{f}) = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

Using these coordinates for the origins of the two DH frames at their initial and goal configurations, assuming that d is sufficiently large, we obtain the

attractive forces

$$\begin{aligned} F_{\text{att},1}(q_{\text{s}}) &= -\zeta_1(o_1(q_{\text{s}}) - o_1(q_{\text{f}})) = \zeta_1 \begin{bmatrix} -1 \\ 1 \end{bmatrix} \\ F_{\text{att},2}(q_{\text{s}}) &= -\zeta_2(o_2(q_{\text{s}}) - o_2(q_{\text{f}})) = \zeta_2 \begin{bmatrix} -3 \\ 1 \end{bmatrix} \end{aligned}$$

The Repulsive Field

To prevent collisions, we will define a workspace repulsive potential field for the origin of each DH frame (excluding frame 0). Note that by defining repulsive potentials only for the origins of the DH frames we cannot ensure that collisions never occur (for example, the middle portion of a long link might collide with an obstacle), but it is fairly easy to modify the method to prevent such collisions as we will see below. For now, we will deal only with the origins of the DH frames.

We define $\rho(o_i(q))$ to be the distance in the workspace from the origin of DH frame i to the nearest obstacle

$$\rho(o_i(q)) = \min_{x \in \partial\mathcal{O}} \|o_i(q) - x\|$$

Likewise, we now define ρ_0 to be the workspace distance of influence of an obstacle. This means that an obstacle will not repel o_i if the distance from o_i to the obstacle is greater than ρ_0.

Our workspace repulsive potential is now given by

$$U_{\text{rep},i}(q) = \begin{cases} \frac{1}{2}\eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right)^2 & : \quad \rho(o_i(q)) \leq \rho_0 \\ \\ 0 & : \quad \rho(o_i(q)) > \rho_0 \end{cases}$$

The workspace repulsive force is equal to the negative gradient of $U_{\text{rep},i}$. For $\rho(o_i(q)) \leq \rho_0$, this force is given by

$$F_{\text{rep},i}(q) = \eta_i \left(\frac{1}{\rho(o_i(q))} - \frac{1}{\rho_0} \right) \frac{1}{\rho^2(o_i(q))} \nabla\rho(o_i(q)) \tag{7.5}$$

in which the notation $\nabla\rho(o_i(q))$ indicates the gradient $\nabla\rho(x)$ evaluated at $x = o_i(q)$. If the obstacle region is convex and b is the point on the obstacle boundary that is closest to o_i, then $\rho(o_i(q)) = ||o_i(q) - b||$, and its gradient is

$$\nabla\rho(x) \Big|_{x=o_i(q)} = \frac{o_i(q) - b}{||o_i(q) - b||}$$

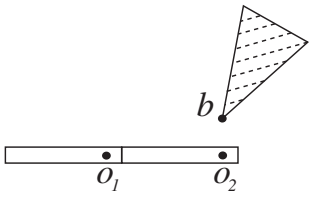


Figure 7.10: The obstacle shown repels o_2, but is outside the distance of influence for o_1. Therefore, it exerts no repulsive force on o_1.

that is, the unit vector directed from b toward $o_i(q)$.

Example 7.4 (Two-Link Planar Arm)**.** *Consider Example 7.3, with a single convex obstacle in the workspace as shown in Figure 7.10. Let* $\rho_0 = 1$*. This prevents the obstacle from repelling* o_1*, which is reasonable since link 1 can never contact the obstacle. The nearest obstacle point to* o_2 *is the vertex* b *of the polygonal obstacle. Suppose that* b *has the coordinates* $(2, 0.5)$*. Then the distance from* $o_2(q_s)$ *to* b *is* $\rho(o_2(q_s)) = 0.5$ *and* $\nabla\rho(o_2(q_s)) = [0, -1]^T$*. The repulsive force at the initial configuration for* o_2 *is then given by*

$$F_{\text{rep},2}(q_s) = \eta_2 \left(\frac{1}{0.5} - 1 \right) \frac{1}{0.25} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \eta_2 \begin{bmatrix} 0 \\ -4 \end{bmatrix}$$

This force has no effect on joint 1, but causes joint 2 to rotate slightly in the clockwise direction, moving link 2 away from the obstacle.

As mentioned above, defining repulsive fields only for the origins of the DH frames does not guarantee that the robot cannot collide with an obstacle. Figure 7.11 shows an example where this is the case. In this figure o_1 and o_2 are very far from the obstacle and therefore the repulsive influence may not be great enough to prevent link 2 from colliding with the obstacle. To cope with this problem we can use a set of **floating repulsive control points** $o_{float,i}$ typically one per link. The floating control points are defined as points on the boundary of a link that are closest to any workspace obstacle. Obviously the choice of the $o_{float,i}$ depends on the configuration q. For the case shown in Figure 7.11, $o_{float,2}$ would be located near the center of link 2, thus repelling the robot from the obstacle. The repulsive force acting on $o_{float,i}$ is defined in the same way as for the other control points using Equation (7.5).

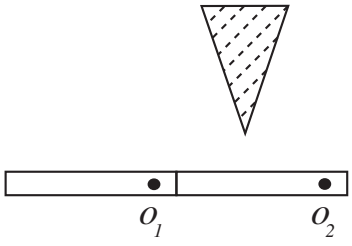


Figure 7.11: The repulsive forces exerted on the origins of the DH frames o_1 and o_2 may not be sufficient to prevent a collision between link 2 and the obstacle.

Mapping Workspace Forces to Joint Torques

We have shown how to construct potential fields in the robot's workspace that induce artificial forces on the origins o_i of the DH frames for the robot arm. In this section we describe how these artificial forces can be mapped to artificial joint torques.

As we derived in Chapter 4 using the principle of virtual work, if τ denotes the vector of joint torques induced by the workspace force F exerted at the end effector, then

$$J_v^T F = \tau$$

where J_v includes the top three rows of the manipulator Jacobian. We do not use the lower three rows, since we have considered only attractive and repulsive workspace forces, and not attractive and repulsive workspace torques. Note that for each o_i an appropriate Jacobian must be constructed, but this is straightforward given the techniques described in Chapter 4 and the A matrices for the arm. We denote the Jacobian for o_i by J_{o_i}.

Example 7.5 (Two-Link Planar Arm)**.** *Consider again the two-link arm of Example 7.3 with repulsive workspace forces as given in Example 7.4. The Jacobians that map joint velocities to linear velocities satisfy*

$$\dot{o}_i = J_{o_i}(q) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}$$

For the two-link arm the Jacobian matrix for o_2 is merely the Jacobian that we derived in Chapter 4, namely

$$J_{o_2}(q_1, q_2) = \begin{bmatrix} -s_1 - s_{12} & -s_{12} \\ c_1 + c_{12} & c_{12} \end{bmatrix}$$

The Jacobian matrix for o_1 is similar, but takes into account that motion of joint 2 does not affect the velocity of o_1. Thus

$$J_{o_1}(q_1, q_2) = \begin{bmatrix} -s_1 & 0 \\ c_1 & 0 \end{bmatrix}$$

At $q_s = (0, 0)$ we have

$$J_{o_1}^T(q_s) = \begin{bmatrix} -s_1 & c_1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

and

$$J_{o_2}^T(q_s) = \begin{bmatrix} -s_1 - s_{12} & c_1 + c_{12} \\ -s_{12} & c_{12} \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix}$$

Using these Jacobians, we can easily map the workspace attractive and repulsive forces to joint torques. If we let $\zeta_1 = \zeta_2 = \eta_2 = 1$ we obtain

$$\tau_{\text{att},1}(q_s) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\tau_{\text{att},2}(q_s) = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -3 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\tau_{\text{rep},2}(q_s) = \begin{bmatrix} 0 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ -4 \end{bmatrix} = \begin{bmatrix} -8 \\ -4 \end{bmatrix}$$

The total artificial joint torque acting on the arm is the sum of the artificial joint torques that result from all attractive and repulsive potentials

$$\tau(q) = \sum_i J_{o_i}^T(q) F_{\text{att},i}(q) + \sum_i J_{o_i}^T(q) F_{\text{rep},i}(q) \tag{7.6}$$

It is essential that we add the joint torques and not the workspace forces. In other words, we must use the Jacobians to transform forces to joint torques before we combine the effects of the potential fields. For example, Figure 7.12 shows a case in which two workspace forces F_1 and F_2, act on opposite corners of a rectangle. It is easy to see that $F_1 + F_2 = 0$, but that the combination of these forces produces a pure torque about the center of the rectangle.

Example 7.6 (Two-Link Planar Arm)**.** *Consider again the two-link planar arm of Example 7.3, with joint torques as determined in Example 7.5. In this*

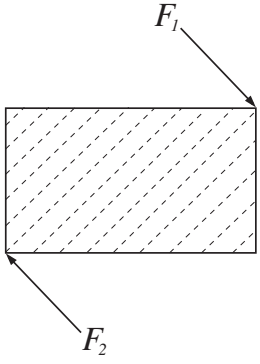


Figure 7.12: The two forces illustrated in the figure are vectors of equal magnitude in opposite directions. Vector addition of these two forces produces zero net force, but there is a net torque induced by these forces.

case the total joint torque induced by the attractive and repulsive workspace potential fields is given by

$$\begin{aligned}\tau(q_{\mathrm{s}}) &= \tau_{\mathrm{att},1}(q_{\mathrm{s}}) + \tau_{\mathrm{att},2}(q_{\mathrm{s}}) + \tau_{\mathrm{rep},2}(q_{\mathrm{s}}) \\ &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} -8 \\ -4 \end{bmatrix} = \begin{bmatrix} -5 \\ -3 \end{bmatrix}\end{aligned}$$

These joint torques have the effect of causing each joint to rotate in a clockwise direction, away from the goal, due to the close proximity of o_2 to the obstacle. By choosing a smaller value for η_2, this effect can be overcome.

Application to Mobile Robots

The methods described in this section can easily be extended to the case of mobile robots. To do so, we define a set of control points on the robot $\{o_i\}_{i=1\ldots m}$ such that $q = q_{\mathrm{f}}$ when $o_i(q) = o_i(q_{\mathrm{f}})$ for $i = 1, \ldots m$. We then proceed as above, treating these o_i in the same way that we treated DH frames. The Jacobian matrix used in this case is given by

$$J_{o_i}(q) = \left[\frac{\partial o_i}{\partial q}\right]$$

and an appropriate gradient for the configuration space potential function is given by

$$\tau(q) = \sum_i J_{o_i}^T(q) F_{\mathrm{att},i}(q) + \sum_i J_{o_i}^T(q) F_{\mathrm{rep},i}(q)$$

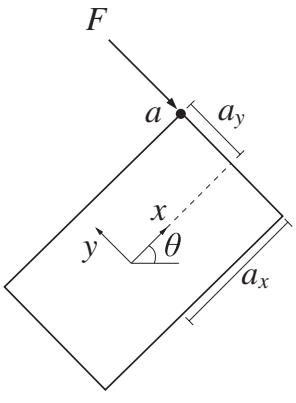


Figure 7.13: In this example, the robot is a polygon whose configuration can be represented as $q = (x, y, \theta)$, in which θ is the angle from the world x-axis to the x-axis of the robot's local frame. A force F is exerted on vertex a with local coordinates (a_x, a_y).

in which τ includes forces and moments applied with respect to the body-attached coordinate frame of the robot.

Example 7.7 (A Polygonal Robot in the Plane)**.** *Consider the polygonal robot shown in Figure 7.13. The vertex a has coordinates (a_x, a_y) in the robot's local coordinate frame. Therefore, if the robot's configuration is given by $q = (x, y, \theta)$, the forward kinematic map for vertex a (that is, the mapping from $q = (x, y, \theta)$ to the global coordinates of the vertex a) is given by*

$$a(x, y, \theta) = \begin{bmatrix} x + a_x \cos\theta - a_y \sin\theta \\ y + a_x \sin\theta + a_y \cos\theta \end{bmatrix}$$

The corresponding Jacobian matrix is given by

$$J_a(x, y, \theta) = \begin{bmatrix} 1 & 0 & -a_x \sin\theta - a_y \cos\theta \\ 0 & 1 & a_x \cos\theta - a_y \sin\theta \end{bmatrix}$$

Using the transpose of the Jacobian to map the workspace forces to generalized forces, we obtain

$$J_a^T(x, y, \theta) \begin{bmatrix} F_x \\ F_y \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ -F_x(a_x \sin\theta - a_y \cos\theta) + F_y(a_x \cos\theta - a_y \sin\theta) \end{bmatrix}$$

The bottom entry in this vector corresponds to the torque exerted about the origin of the robot frame.

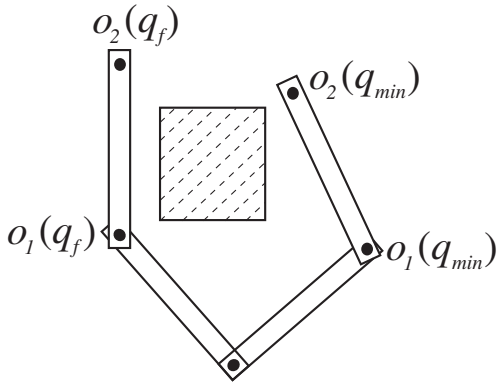


Figure 7.14: The configuration $q_{\min}$ is a local minimum in the potential field. At $q_{\min}$ the attractive force exactly cancels the repulsive force and the planner fails to make further progress.

Gradient Descent Planning

As above, the gradient descent algorithm constructs a sequence of configurations, $q^0, q^1, \ldots, q^m$ such that $q^0 = q_{\rm s}$ and $q^m = q_{\rm f}$, but in this case the k^{th} iteration is given by

$$q^{k+1} = q^k + \alpha^k \frac{\tau(q^k)}{||\tau(q^k)||}$$

in which τ is given by (7.6). The scalar α^k determines the step size at the k^{th} iteration, and the algorithm terminates when $||q^k - q_{\rm f}|| < \epsilon$, where we choose ϵ to be a sufficiently small constant, based on the task requirements.

The problem of local minima in the potential field also exists for this approach to defining workspace potentials and mapping workspace gradients to the configuration space. An example of this situation is shown in Figure 7.14.

There are a number of design choices that must be made when using this approach.

- The constant ζ_i in Equation (7.4) controls the relative influence of the attractive potential for control point o_i. It is not necessary that all of the ζ_i be set to the same value. Typically we assign a larger weight to one of the o_i than to the others, producing a "follow the leader" type of motion, in which the leader o_i is quickly attracted to its final position and the robot then reorients itself so that the other o_i reach their final positions.

- The constant η_i in Equation (7.5) controls the relative influence of the repulsive potential for o_i. As with the ζ_i it is not necessary that all of the η_i be set to the same value. In particular, we typically set the value of η_i to be much smaller for obstacles that are near the goal position of the robot (to avoid having these obstacles repel the robot from the goal).

- The constant ρ_0 in Equation (7.5) defines the distance of influence for obstacles. As with the η_i we can define a distinct value of ρ_0 for each obstacle. In particular, we do not want any obstacle's region of influence to include the goal position of any repulsive control point. We may also wish to assign distinct values of ρ_0 to the obstacles to avoid the possibility of overlapping regions of influence for distinct obstacles.

7.4 Sampling-Based Methods

The potential field approaches described above incrementally explore $\mathcal{Q}_{\text{free}}$ by generating a sequence of configurations $q^0, \ldots, q^m$ using a gradient descent approach. This exploration is inherently goal-driven (due to the attractive potential), and it is this bias that makes the approach susceptible to failure due to the presence of local minima in the potential field. As we have seen, applying a random walk can be an effective way to escape local minima, abandoning temporarily the goal-driven behavior in favor of a randomized strategy. Taken to an extreme, we could design a planner that completely abandons goal-driven search, instead relying completely on randomized strategies. This is the approach taken by **sampling-based planners**[5].

Sampling-based planners generate a sequence of configurations using a random sampling strategy. The simplest such strategy is merely to generate random samples from a uniform probability distribution on the configuration space. If two samples are sufficiently near to one another, planning a path between them can be accomplished using a simple, local planner. Iteratively applying this strategy produces a graph $G = (V, E)$ in which the vertex set V includes the randomly generated sample configurations, and edges correspond to local paths between sample configurations that lie in close proximity to one another. The graph G is referred to as a **configuration**

[5]While there are some sampling-based planners that use quasi-random, or even deterministic sampling strategies, the use of randomized approaches is far more prevalent, and here we consider only these approaches.

space roadmap.

In this section, we will describe two sampling-based planning algorithms. The first algorithm builds a **probabilistic roadmap** or **PRM**, which is a roadmap that attempts to uniformly cover the entire free configuration space. This approach is particularly useful when many planning problems are to be solved for a single workspace, so that the cost of building the roadmap can be amortized over many planning instances. The second algorithm builds a **rapidly-exploring random Tree** or **RRT**, which is a random tree whose root vertex corresponds to the initial configuration q_{s}. By using a clever sampling strategy to generate new vertices in the tree, this method is able to rapidly explore the free configuration space, and has proven to be effective for solving even difficult path planning problems.

7.4.1 Probabilistic Roadmaps (PRM)

In general, a configuration space roadmap is a one-dimensional network of curves that effectively represents $\mathcal{Q}_{\mathrm{free}}$. A roadmap is typically represented as a graph, in which edges correspond to curve segments, whose intersections correspond to vertices. When using roadmap methods, planning comprises three stages: (1) find a path from q_{s} to a configuration q_a in the roadmap, (2) find a path from q_{f} to a configuration q_b in the roadmap, (3) find a path in the roadmap from q_a to q_b. Steps (1) and (2) are typically much easier than finding a path from q_{s} to q_{f}. The visibility graph and generalized Voronoi diagram described in Section 7.2 are both examples of configuration space roadmaps (although for the visibility graph, both q_{s} and q_{f} are included in the roadmap by construction).

A **probabilistic roadmap**, or **PRM**, is a configuration space roadmap whose vertices correspond to randomly generated configurations, and whose edges correspond to collision-free paths between configurations. Constructing a PRM is a conceptually straightforward process. First, a set of random configurations is generated to serve as the vertices in the roadmap. Then, a simple, local path planner is used to generate paths that connect pairs of configurations. Finally, if the initial roadmap consists of multiple connected components[6], it is augmented during an enhancement phase, in which new vertices and edges are added in an attempt to connect disjoint components of the roadmap. To solve a path planning problem, the simple, local planner is used to add q_{s} and q_{f} to the roadmap, and the resulting roadmap is searched for a path from q_{s} to q_{f}. These four steps are illustrated in Figure

[6] A connected component is a maximal subgraph of the graph such that a path exists in the subgraph between any two vertices.

7.15. We now discuss these steps in more detail.

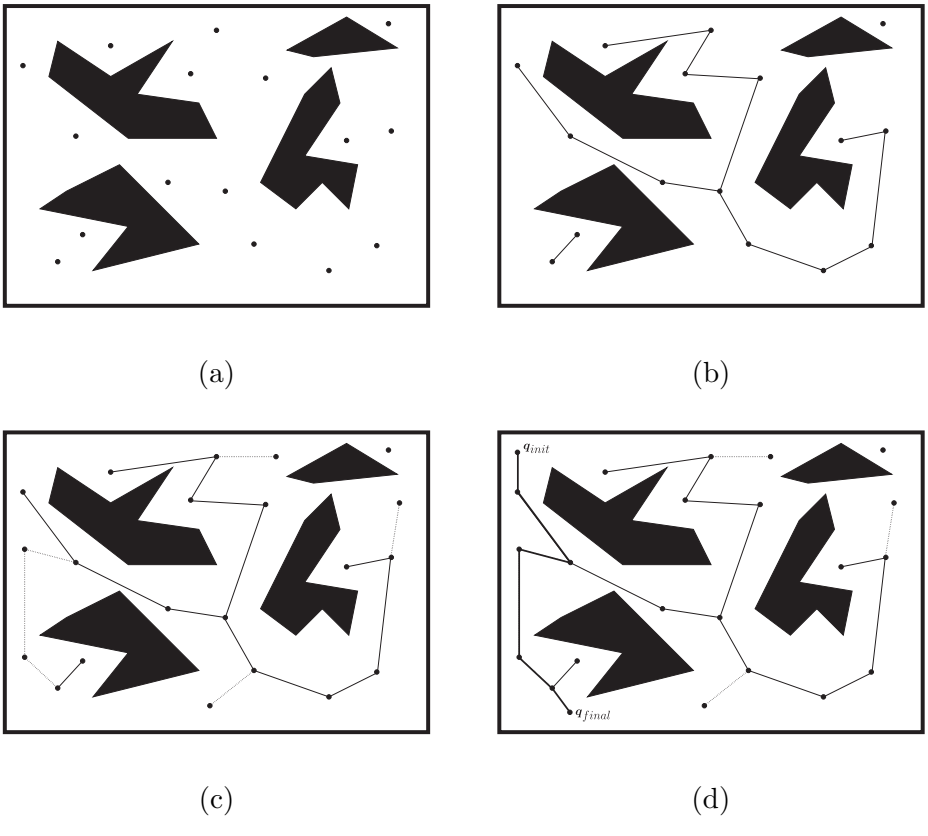


Figure 7.15: This figures illustrates the steps in the construction of a probabilistic roadmap for a two-dimensional configuration space containing polygonal obstacles. (a) First, a set of random samples is generated in the configuration space. Only collision-free samples are retained. (b) Each sample is connected to its nearest neighbors using a simple, straight-line path. If such a path causes a collision, the corresponding samples are not connected in the roadmap. (c) Since the initial roadmap contains multiple connected components, additional samples are generated and connected to the roadmap in an enhancement phase. (d) A path from q_{s} to q_{f} is found by connecting q_{s} and q_{f} to the roadmap and then searching this augmented roadmap for a path from q_{s} to q_{f}.

Sampling the Configuration Space

The simplest way to generate sample configurations is with uniform random sampling of the configuration space. Sample configurations that lie in $\mathcal{QO}$ are discarded. A simple collision checking algorithm can determine when this is the case. The disadvantage of this approach is that the number of samples it places in any particular region of $\mathcal{Q}_{\text{free}}$ is proportional to the volume of the region. Therefore, uniform sampling is unlikely to place samples in narrow passages of $\mathcal{Q}_{\text{free}}$. In the PRM literature, this is referred to as the **narrow passage problem**. It can be dealt with either by using more intelligent sampling schemes, or by using an enhancement phase during the construction of the PRM. In this section, we discuss the latter option.

Connecting Pairs of Configurations

Given a set of vertices that correspond to configurations, the next step in building the PRM is to determine which pairs of vertices should be connected by the local path planner. The typical approach is to attempt to connect each vertex to its k nearest neighbors, with k a parameter chosen by the user. Of course, to define the nearest neighbors, a distance function is required. Table 7.1 lists four distance functions that have been popular in the PRM literature. In this table, q and q' are the two configurations corresponding to different vertices in the roadmap, q_i refers to the value of the i^{th} coordinate of q, $\mathcal{A}$ is a set of reference points on the robot, and $\text{p}(q)$ refers to the workspace coordinates of reference point p at configuration q. Of these, the simplest, and perhaps most commonly used, is the 2-norm in configuration space.

Once pairs of neighboring vertices have been identified, a simple local planner is used to connect them. Often, a straight line in configuration space is used as the candidate plan, and thus, planning the path between two vertices is reduced to collision checking along a straight-line path in the configuration space. If a collision occurs on this path, it can be discarded, or a more sophisticated planner can be used to attempt to connect the vertices.

The simplest approach to collision detection along the straight-line path is to sample the path at a sufficiently fine discretization, and to check each sample for collision. This method works, provided the discretization is fine enough, but it is very inefficient. This is because many of the computations required to check for collision at one sample are repeated for the next sample (assuming that the robot has moved only a small amount between the two configurations). For this reason, incremental collision detection approaches have been developed. While these approaches are beyond the scope of this

2-norm in $\mathcal{Q}$:	$\|q' - q\| = \left[\sum_{i=1}^{n} (q'_i - q_i)^2\right]^{\frac{1}{2}}$
∞-norm in $\mathcal{Q}$:	$\max_n \|q'_i - q_i\|$
2-norm in workspace:	$\left[\sum_{\mathrm{p} \in \mathcal{A}} \|\mathrm{p}(q') - \mathrm{p}(q)\|^2\right]^{\frac{1}{2}}$
∞-norm in workspace:	$\max_{\mathrm{p} \in \mathcal{A}} \|\mathrm{p}(q') - \mathrm{p}(q)\|$

Table 7.1: Four commonly used distance functions.

text, a number of collision detection software packages are available in the public domain. Most developers of robot motion planners use one of these packages, rather than implementing their own collision detection routines.

Enhancement

After the initial PRM has been constructed, it is likely that it will consist of multiple connected components. Often these individual components lie in large regions of $\mathcal{Q}_{\text{free}}$ that are connected by narrow passages in $\mathcal{Q}_{\text{free}}$. The goal of the enhancement process is to connect as many of these disjoint components as possible.

One approach to enhancement is merely to attempt to connect pairs of vertices in two disjoint components, perhaps by using a more sophisticated planner such as described in Section 7.3. A common approach is to identify the largest connected component, and to attempt to connect the smaller components to it. The vertex in the smaller component that is closest to the larger component is typically chosen as the candidate for connection. A second approach is to choose a vertex randomly as a candidate for connection, and to bias the random choice based on the number of neighbors of the vertex; a vertex with fewer neighbors in the roadmap is more likely to be near a narrow passage, and should be a more likely candidate for connection.

Another approach to enhancement is to add more random vertices to the roadmap, in the hope of finding vertices that lie in or near the narrow passages. One approach is to identify vertices that have few neighbors, and

to generate sample configurations in regions around these vertices. The local planner is then used to attempt to connect these new configurations to the roadmap.

Path Smoothing

After the PRM has been generated, path planning amounts to connecting q_{s} and q_{f} to the roadmap using the local planner, and then performing path smoothing, since the resulting path will be composed of straight-line segments in the configuration space. The simplest path smoothing algorithm is to select two random points on the path and try to connect them with the local planner. This process is repeated until no significant progress is made.

7.4.2 Rapidly-Exploring Random Trees (RRTs)

In the classical PRM algorithm, the i^{th} sample configuration is obtained by sampling from a uniform probability distribution on $\mathcal{Q}$, independent of any previously generated samples[7]. As a result, it is possible, and even likely, that at any given iteration the newly generated sample configuration may be quite far from any existing vertices in the current roadmap. In such cases, the local planner is likely to fail to connect the new sample to any existing vertex, increasing the number of connected components in the roadmap. An alternative approach is to grow a single tree, starting from a vertex that corresponds to q_{s}, until some leaf vertex reaches the goal configuration. This is the approach embodied by **rapidly-exploring random trees** (**RRTs**).

The construction of an RRT is an iterative process in which a new vertex is added to an existing tree at each iteration. The process of adding a new vertex begins by generating a random sample, q_{sample}, from a uniform probability distribution on $\mathcal{Q}$; however, unlike the construction of PRMs, this vertex is not added to the existing tree. Instead, q_{sample} is used to determine how to "grow" the current tree. This is done by choosing the vertex q_{near} in the existing tree that is closest to q_{sample}, and taking a small step from q_{near} in the direction of q_{sample}. The configuration at which this step arrives, q_{new}, is added to the tree, and an edge is added from q_{near} to q_{new}. Figure 7.16 illustrates the process for a single iteration of the RRT construction algorithm.

While RRT-based algorithms have proven to be very effective in planning collision-free paths for robots, their true power lies in the method by which the new node q_{new} is generated and connected to the existing tree. For

[7] In the language of probability theory, the sample configurations correspond to independent random variables.

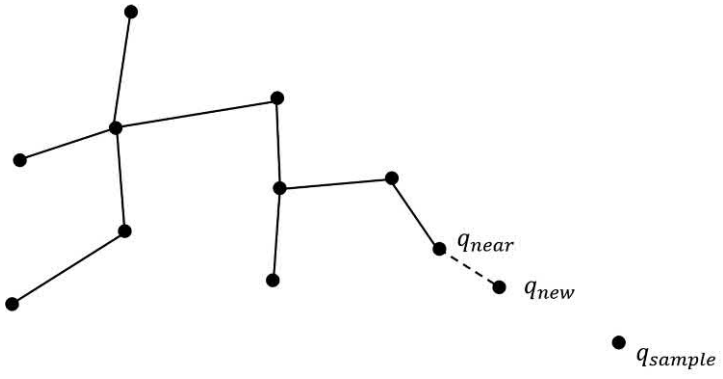


Figure 7.16: A new vertex is added to an existing tree by (i) generating a sample configuration generate q_{sample} from a uniform probability distribution on $\mathcal{Q}$, (ii) identifying q_{near}, the vertex in the current tree that is nearest to q_{sample}, and (iii) taking a small step from q_{near} toward q_{sample}.

PRMs, a new node is connected to the tree by solving a local path planning problem. For RRTs, the configuration q_{new} is chosen by "taking a step" toward q_{sample}. The simplest way to achieve this, of course, is to step along a straight-line path toward q_{sample}, but more general methods can be applied. Consider, for example, a robot whose system dynamics are described by

$$\dot{x} = f(x, u) \tag{7.7}$$

in which x denotes the state of the system and u denotes an input. If we define our RRT on the state space instead of the configuration space, we can generate the vertex x_{new} by integrating the dynamics (7.7) forward in time from the initial condition given by x_{near}

$$x_{\text{new}} = x_{\text{near}} + \int_0^{\Delta t} f(x, u)dt \tag{7.8}$$

Evaluating this integral requires knowing an appropriate control input u. Choosing u to ensure that x_{new} lies along the line connecting x_{near} to x_{sample} is a difficult problem, but luckily, ensuring that x_{new} lies exactly on this line is not essential to the efficacy of RRT algorithms. A popular way to determine x_{new} is to randomly select a set of candidate inputs, u^i, evaluate (7.8) for each of these, and retain the result that makes the best progress toward x_{sample}. Using this approach, RRTs have been applied to motion planning problems for car-like robots, unmanned air vehicles, autonomous underwater vehicles, spacecraft, satellites, and many others.

7.5 Trajectory Planning

In Section 7.1.3, we defined a path[8] from q_0 to q_f in configuration space as a continuous map, $\gamma : [0,1] \to \mathcal{Q}$, with $\gamma(0) = q_0$ and $\gamma(1) = q_\mathrm{f}$. A **trajectory** is a function of time $q(t)$ such that $q(t_0) = q_0$ and $q(t_f) = q_\mathrm{f}$. In this case, $t_f - t_0$ represents the amount of time taken to execute the trajectory. Since the trajectory is parameterized by time, we can compute velocities and accelerations along the trajectories by differentiation. If we think of the argument to γ as a time variable, then a path is a special case of a trajectory, one that will be executed in one unit of time. In other words, in this case γ gives a complete specification of the robot's trajectory, including the time derivatives (since one need only differentiate γ to obtain these).

As seen above, a path planning algorithm will not typically give the map γ; it will give only a sequence of points (called **via points**) along the path. This is also the case for other ways in which a path could be specified. In some cases, paths are specified by giving a sequence of end-effector poses, $T_6^0(k\Delta t)$. In this case, the inverse kinematic solution must be used to convert this to a sequence of joint configurations. A common way to specify paths for industrial robots is to physically lead the robot through the desired motion with a teach pendant, the so-called **teach and playback mode**. In some cases, this may be more efficient than deploying a path planning system, for example, in static environments when the same path will be executed many times. In this case, there is no need for calculation of the inverse kinematics; the desired motion is simply recorded as a set of joint angles (actually as a set of encoder values).

Below, we first consider **point-to-point** motion. In this case the task is to plan a trajectory from an initial configuration $q(t_0)$ to a final configuration $q(t_f)$. In some cases, there may be constraints on the trajectory (for example, if the robot must start and end with zero velocity). Nevertheless, it is easy to realize that there are infinitely many trajectories that will satisfy a finite number of constraints on the endpoints. It is common practice therefore to choose trajectories from a finitely parameterizable family, for example, polynomials of degree n, where n depends on the number of constraints to be satisfied. This is the approach that we will take in this text. Once we have seen how to construct trajectories between two configurations, it is straightforward to generalize the method to the case of trajectories specified by multiple via points.

[8] In Section 7.1.3 we used q_s to denote the initial configuration. In this section, we use q_0 to denote the first configuration in a sequence of two or more configurations that will be used to define a path.

7.5.1 Trajectories for Point-to-Point Motion

As described above, the problem is to find a trajectory that connects the initial and final configurations while satisfying other specified constraints at the endpoints, such as velocity and/or acceleration constraints. Without loss of generality, we will consider planning the trajectory for a single joint, since the trajectories for the remaining joints will be created independently and in exactly the same way. Thus, we will concern ourselves with the problem of determining $q(t)$, where $q(t)$ is a scalar joint variable.

We suppose that at time t_0 the joint variable satisfies

$$q(t_0) = q_0 \tag{7.9}$$
$$\dot{q}(t_0) = v_0 \tag{7.10}$$

and we wish to attain the values at t_f

$$q(t_f) = q_f \tag{7.11}$$
$$\dot{q}(t_f) = v_f \tag{7.12}$$

Figure 7.17 shows a suitable trajectory for this motion. In addition, we may wish to specify the constraints on initial and final accelerations. In this case we have two additional equations

$$\ddot{q}(t_0) = \alpha_0 \tag{7.13}$$
$$\ddot{q}(t_f) = \alpha_f \tag{7.14}$$

Below we will investigate several specific ways to compute trajectories using low-order polynomials. We begin with cubic polynomials, which allow specification of initial and final positions and velocities. We then describe quintic polynomial trajectories, which also allow the specification of the initial and final accelerations. After describing these two general polynomial trajectories, we describe trajectories that are pieced together from segments of constant acceleration.

Cubic Polynomial Trajectories

Consider first the case where we wish to generate a polynomial joint trajectory between two configurations, and that we wish to specify the start and end velocities for the trajectory. This gives four constraints that the trajectory must satisfy. Therefore, at a minimum we require a polynomial with four independent coefficients that can be chosen to satisfy these constraints. Thus, we consider a cubic trajectory of the form

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \tag{7.15}$$

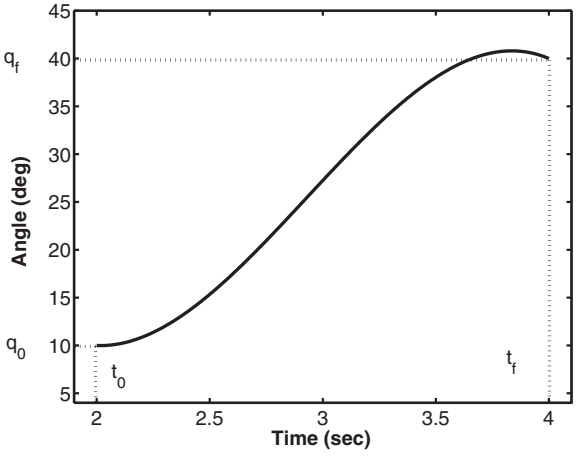


Figure 7.17: A typical joint space trajectory.

Then the desired velocity is given as

$$\dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \tag{7.16}$$

Combining Equations (7.15) and (7.16) with the four constraints yields four equations in four unknowns

$$\begin{aligned} q_0 &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 \\ v_0 &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 \\ q_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 \\ v_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2 \end{aligned}$$

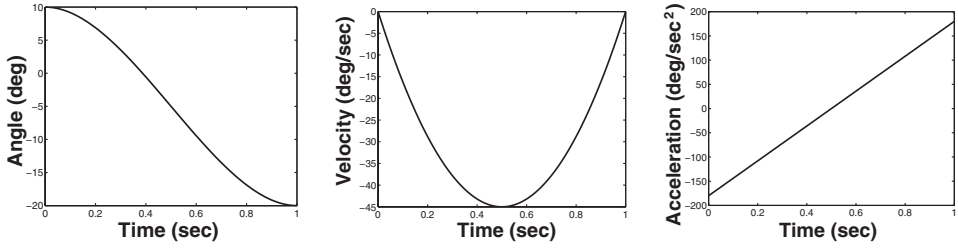
These four equations can be combined into a single matrix equation

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix} \tag{7.17}$$

It can be shown (Problem 7–19) that the determinant of the coefficient matrix in Equation (7.17) is equal to $(t_f - t_0)^4$ and, hence, Equation (7.17) always has a unique solution provided a nonzero time interval is allowed for the execution of the trajectory.

As an illustrative example, we may consider the special case that the initial and final velocities are zero. Suppose we take $t_0 = 0$ *and* $t_f = 1$ *sec,*

Example 7.8 (Cubic Polynomial Trajectory)**.**



(a) (b) (c)

Figure 7.18: (a) Cubic polynomial trajectory. (b) Velocity profile for cubic polynomial trajectory. (c) Acceleration profile for cubic polynomial trajectory.

with

$$v_0 = 0 \qquad v_f = 0$$

Thus, we want to move from the initial position q_0 to the final position q_f in 1 second, starting and ending with zero velocity. From Equation (7.17) we obtain

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ 0 \\ q_f \\ 0 \end{bmatrix}$$

This is then equivalent to the four equations

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= 0 \\ a_2 + a_3 &= q_f - q_0 \\ 2a_2 + 3a_3 &= 0 \end{aligned}$$

These latter two equations can be solved to yield

$$\begin{aligned} a_2 &= 3(q_f - q_0) \\ a_3 &= -2(q_f - q_0) \end{aligned}$$

The required cubic polynomial function is therefore

$$q(t) = q_0 + 3(q_f - q_0)t^2 - 2(q_f - q_0)t^3$$

The corresponding velocity and acceleration curves are given as

$$\begin{aligned}\dot{q}(t) &= 6(q_f - q_0)t - 6(q_f - q_0)t^2 \\ \ddot{q}(t) &= 6(q_f - q_0) - 12(q_f - q_0)t\end{aligned}$$

Figure 7.18 shows these trajectories with $q_0 = 10°$, $q_f = -20°$.

Quintic Polynomial Trajectories

As can be seen in Figure 7.18, a cubic trajectory gives continuous positions and velocities at the start and finish points times but discontinuities in the acceleration. The derivative of acceleration is called the **jerk**. A discontinuity in acceleration leads to an impulsive jerk, which may excite vibrational modes in the manipulator and reduce tracking accuracy. For this reason, one may wish to specify constraints on the acceleration as well as on the position and velocity. In this case, we have six constraints (one each for initial and final configurations, initial and final velocities, and initial and final accelerations). Therefore we require a fifth order polynomial

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{7.18}$$

Using Equations (7.9)–(7.14) and taking the appropriate number of derivatives we obtain the following equations,

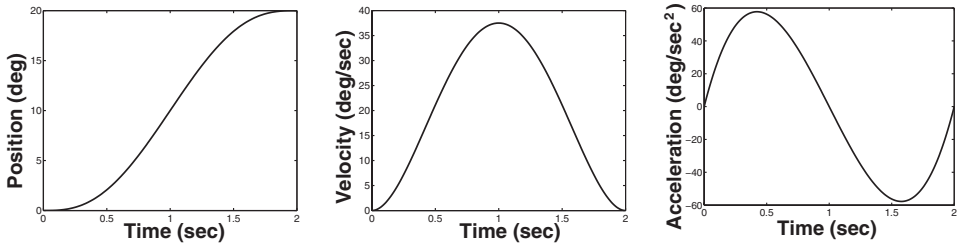
$$\begin{aligned}q_0 &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5 \\ v_0 &= a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4 \\ \alpha_0 &= 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3 \\ q_f &= a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5 \\ v_f &= a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4 \\ \alpha_f &= 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3\end{aligned}$$

which can be written as

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ \alpha_0 \\ q_f \\ v_f \\ \alpha_f \end{bmatrix} \tag{7.19}$$

Figure 7.19 shows a quintic polynomial trajectory with $q(0) = 0$, $q(2) = 20$ *with zero initial and final velocities and accelerations.*

Example 7.9 (Quintic Polynomial Trajectory)**.**



(a) (b) (c)

Figure 7.19: (a) Quintic polynomial trajectory, (b) its velocity profile, and (c) its acceleration profile.

Linear Segments with Parabolic Blends (LSPB)

Another way to generate suitable joint space trajectories is by using so-called **linear segments with parabolic blends** (**LSPB**). This type of trajectory has a **trapezoidal velocity profile** and is appropriate when a constant velocity is desired along a portion of the path. The LSPB trajectory is such that the velocity is initially "ramped up" to its desired value and then "ramped down" when it approaches the goal position. To achieve this we specify the desired trajectory in three parts. The first part from time t_0 to time t_b is a quadratic polynomial. This results in a linear "ramp" velocity. At time t_b, called the **blend time**, the trajectory switches to a linear function. This corresponds to a constant velocity. Finally, at $t_f - t_b$ the trajectory switches once again, this time to a quadratic polynomial so that the velocity is linear.

We choose the blend time t_b so that the position curve is symmetric as shown in Figure 7.20. For convenience suppose that $t_0 = 0$ and $\dot{q}(t_f) = 0 = \dot{q}(0)$. Then between times 0 and t_b we have

$$q(t) = a_0 + a_1 t + a_2 t^2$$

so that the velocity is

$$\dot{q}(t) \quad = \quad a_1 + 2a_2 t$$

The constraints $q_0 = 0$ and $\dot{q}(0) = 0$ imply that

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= 0 \end{aligned}$$

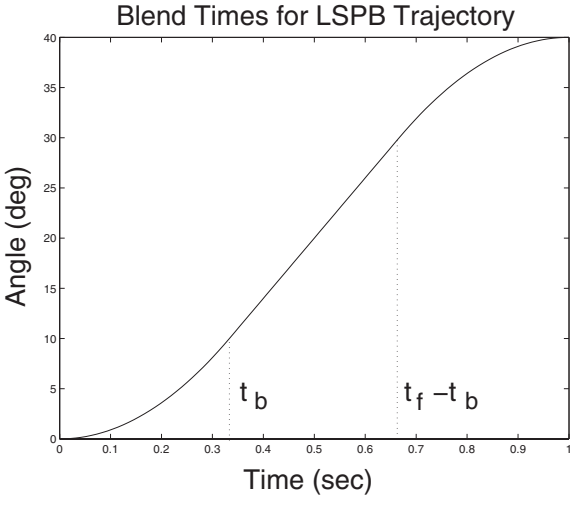


Figure 7.20: Blend times for LSPB trajectory.

At time t_b we want the velocity to equal a given constant, say V. Thus, we have

$$\dot{q}(t_b) = 2a_2 t_b = V$$

which implies that

$$a_2 = \frac{V}{2t_b}$$

Therefore the required trajectory between 0 and t_b is given as

$$\begin{aligned} q(t) &= q_0 + \frac{V}{2t_b}t^2 = q_0 + \frac{\alpha}{2}t^2 \\ \dot{q}(t) &= \frac{V}{t_b}t = \alpha t \\ \ddot{q} &= \frac{V}{t_b} = \alpha \end{aligned}$$

where α denotes the acceleration.

Now, between time t_b and $t_f - t_b$, the trajectory is a linear segment with velocity V

$$q(t) = q(t_b) + V(t - t_b)$$

Since, by symmetry,

$$q\left(\frac{t_f}{2}\right) = \frac{q_0 + q_f}{2}$$

we have

$$\frac{q_0 + q_f}{2} = q(t_b) + V(\frac{t_f}{2} - t_b)$$

which implies that

$$q(t_b) = \frac{q_0 + q_f}{2} - V(\frac{t_f}{2} - t_b)$$

Since the two segments must "blend" at time t_b, we require

$$q_0 + \frac{V}{2}t_b = \frac{q_0 + q_f - Vt_f}{2} + Vt_b$$

which, upon solving for the blend time t_b, gives

$$t_b = \frac{q_0 - q_f + Vt_f}{V} \tag{7.20}$$

Note that we have the constraint $0 < t_b \leq \frac{t_f}{2}$. This leads to the inequality

$$\frac{q_f - q_0}{V} < t_f \leq \frac{2(q_f - q_0)}{V}$$

To put it another way we have the inequality

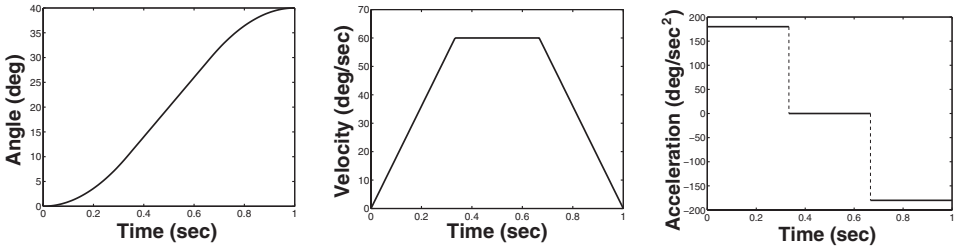
$$\frac{q_f - q_0}{t_f} < V \leq \frac{2(q_f - q_0)}{t_f}$$

Thus, the specified velocity must be between these limits or the motion is not possible.

The portion of the trajectory between $t_f - t_b$ and t_f is now found by symmetry considerations (Problem 7–23). The complete LSPB trajectory is given by

$$q(t) = \begin{cases} q_0 + \frac{\alpha}{2}t^2 & 0 \leq t \leq t_b \\ \frac{q_f + q_0 - Vt_f}{2} + Vt & t_b < t \leq t_f - t_b \\ q_f - \frac{\alpha t_f^2}{2} + \alpha t_f t - \frac{\alpha}{2}t^2 & t_f - t_b < t \leq t_f \end{cases} \tag{7.21}$$

Figure 7.21(a) shows such an LSPB trajectory, where the maximum velocity $V = 60$. In this case $t_b = \frac{1}{3}$. The velocity and acceleration curves are given in Figures 7.21(b) and 7.21(c), respectively.



(a) (b) (c)

Figure 7.21: (a) LSPB trajectory. (b) Velocity profile for LSPB trajectory. (c) Acceleration profile for LSPB trajectory.

Minimum-Time Trajectories

An important variation of the LSPB trajectory is obtained by leaving the final time t_f unspecified and seeking the "fastest" trajectory between q_0 and q_f with a given constant acceleration α, that is, the trajectory with the final time t_f a minimum. This is sometimes called a **bang-bang** trajectory since the optimal solution is achieved with the acceleration at its maximum value $+\alpha$ until an appropriate **switching time** t_s at which time it abruptly switches to its minimum value $-\alpha$ (maximum deceleration) from t_s to t_f.

Returning to our simple example in which we assume that the trajectory begins and ends at rest, that is, with zero initial and final velocities, symmetry considerations would suggest that the switching time t_s is just $\frac{t_f}{2}$. This is indeed the case. For nonzero initial and/or final velocities, the situation is more complicated and we will not discuss it here. If we let V_s denote the velocity at time t_s then we have $V_s = \alpha t_s$ and using Equation (7.20) with $t_b = t_s$ we obtain

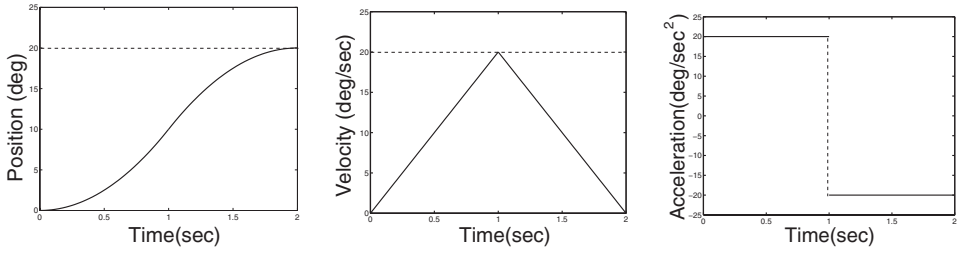
$$t_s = \frac{q_0 - q_f + V_s t_f}{V_s}$$

The symmetry condition $t_s = \frac{t_f}{2}$ implies that

$$V_s = \frac{q_f - q_0}{t_s}$$

and using the fact that $V_s = \alpha t_s$, we obtain

$$\frac{q_f - q_0}{t_s} = \alpha t_s$$



(a) (b) (c)

Figure 7.22: (a) Minimum-time trajectory. (b) Velocity profile for minimum-time trajectory. (c) Acceleration profile for minimum-time trajectory.

which implies that

$$t_s = \sqrt{\frac{q_f - q_0}{\alpha}}$$

Figure 7.22 shows the position, velocity, and acceleration for such a minimum-time trajectory.

7.5.2 Trajectories for Paths Specified by Via Points

Now that we have examined the problem of planning a trajectory between two configurations, we generalize our approach to the case of planning a trajectory that passes through a sequence of configurations, called **via points**. Consider the simple example of a path specified by three points, q_0, q_1, and q_2, such that the via points are reached at times t_0, t_1, and t_2, respectively. If in addition to these three constraints we impose constraints on the initial and final velocities and accelerations, we obtain the following set of constraints,

$$\begin{aligned} q(t_0) &= q_0 \ , \ \dot{q}(t_0) = v_0 \ , \ \ddot{q}(t_0) = \alpha_0 \\ q(t_1) &= q_1 \ , \ q(t_2) = q_2 \ , \ \dot{q}(t_2) = v_2 \ , \ \ddot{q}(t_2) = \alpha_2 \end{aligned}$$

which could be satisfied by generating a trajectory using the sixth-order polynomial

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 + a_6 t^6 \qquad (7.22)$$

One advantage to this approach is that, since $q(t)$ is continuously differentiable, we need not worry about discontinuities in either velocity or

acceleration at the via point, q_1. However, to determine the coefficients for this polynomial, we must solve a linear system of dimension seven. The clear disadvantage to this approach is that as the number of via points increases, the dimension of the corresponding linear system also increases, making the method intractable when many via points are used.

An alternative to using a single high-order polynomial for the entire trajectory is to use low-order polynomials for trajectory segments between adjacent via points. These polynomials are sometimes referred to as interpolating polynomials or blending polynomials. With this approach, we must take care that velocity and acceleration constraints are satisfied at the via points, where we switch from one polynomial to another.

For the first segment of the trajectory, suppose that the initial and final times are t_0 and t_f, respectively, and the constraints on initial and final velocities are given by

$$\begin{array}{lll} q(t_0) = q_0 & ; & q(t_f) = q_1 \\ \dot{q}(t_0) = v_0 & ; & \dot{q}(t_f) = v_1 \end{array} \tag{7.23}$$

the required cubic polynomial for this segment of the trajectory can be computed from

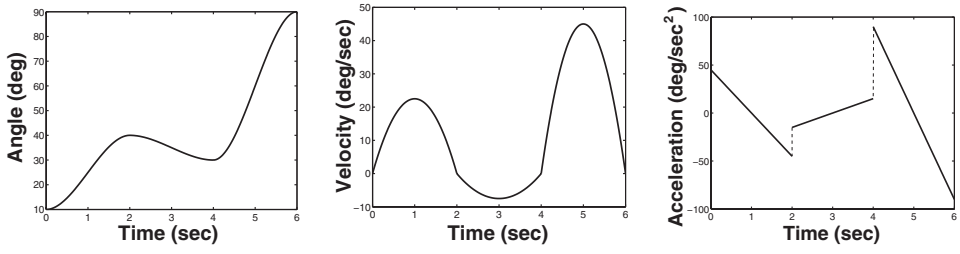
$$q(t_0) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t_0)^3 \tag{7.24}$$

where

$$\begin{aligned} a_0 &= q_0 \\ a_1 &= v_0 \\ a_2 &= \frac{3(q_1 - q_0) - (2v_0 + v_1)(t_f - t_0)}{(t_f - t_0)^2} \\ a_3 &= \frac{2(q_0 - q_1) + (v_0 + v_1)(t_f - t_0)}{(t_f - t_0)^3} \end{aligned}$$

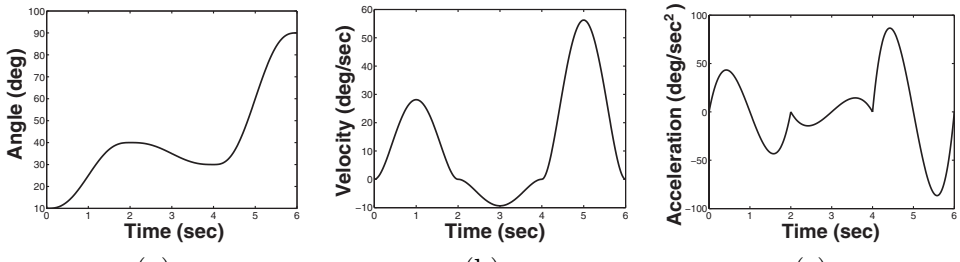
A sequence of moves can be planned using the above formula by using the end conditions q_f, v_f of the i^{th} move as initial conditions for the subsequent move.

Figure 7.23 shows a 6-second move, computed in three parts using Equation (7.24), where the trajectory begins at 10° and is required to reach 40° at 2 seconds, 30° at 4 seconds, and 90° at 6 seconds, with zero velocity at 0, 2, 4, and 6 seconds.



(a) (b) (c)

Figure 7.23: (a) Cubic spline trajectory made from three cubic polynomials. (b) Velocity profile for multiple cubic polynomial trajectory. (c) Acceleration profile for multiple cubic polynomial trajectory.



(a) (b) (c)

Figure 7.24: (a) Trajectory with multiple quintic segments. (b) Velocity profile for multiple quintic segments. (c) Acceleration profile for multiple quintic segments.

Figure 7.24 shows the same 6-second trajectory as above with the added constraints that the accelerations should be zero at the blend times.

7.6 Chapter Summary

In this chapter we studied methods for generating collision-free trajectories for robot manipulators. We divided the problem into two parts, first computing a collision-free path then by using interpolating polynomials to convert these paths into continuous trajectories.

We began by giving a more in-depth introduction to the concept of **configuration space**, including a catalog of configuration spaces for common robots and an explanation of how obstacles in the workspace map to configuration space obstacle regions. We described an algorithm to compute the configuration space obstacle region for the special case of a polygonal robot

that translates in the plane (i.e., $\mathcal{Q} = \mathbb{R}^2$) in an environment populated by polygonal obstacles. For this special case, we introduced three planning algorithms that exploit three unique graph-based representations of the free configuration space: the visibility graph, the generalized Voronoi diagram, and the trapezoidal decomposition. These approaches are often applicable for problems that involve ground-based mobile robots.

For more complex robots, for example, robots that have high dimensional configuration spaces, we introduced algorithms that incrementally search the configuration space for a free path. The first such algorithm operates by constructing an artificial potential field whose negative gradient acts as a kind of artificial force that pushes the robot away from obstacles, while guiding it toward the goal configuration. We first developed this method for the case of $\mathcal{Q} = \mathbb{R}^n$, and then extended it to the more general case of non-Euclidean configuration spaces by using the relationship $\tau = J^T F$. In both cases, gradient descent was used as the method to guide the incremental exploration of the configuration space, and thus, for both cases, there are potential problems arising from the existence of multiple local minima in the field. We briefly described how random walks could be used to escape these local minima, and used this notion of randomization to motivate the development of sampling-based planning algorithms.

Sampling-based planners construct a roadmap in the configuration space using a random sampling scheme. We described two approaches: the probabilistic roadmap (PRM), and the rapidly-exploring random tree (RRT). For the PRM, a set of random samples is generated, and each of these samples is connected to a set of its nearest neighbors using a simple local motion planner (often a simple straight-line planner in the configuration space). Once the roadmap has been constructed, planning amounts to connecting the initial and goal configurations to the roadmap (again, using the simple, local planner), then searching the roadmap for a connecting path. In the case of RRTs, a random tree is grown from the start configuration by iteratively generating a random sample in the configuration space, then taking a small step toward this sample from its nearest neighbor in the current tree. Both of these methods have proven effective for a large variety of path planning problems.

Finally, we showed that, given a sequence of set points, a trajectory can be constructed using a low-order polynomial defined in terms of initial and final conditions for joint variables and their derivatives. We described cubic and quintic trajectories, along with trajectories that are pieced together from segments of constant acceleration (including minimum time, or bang-bang trajectories).

Problems

7–1 Describe the configuration space for a mobile robot that can translate and rotate in the plane.

7–2 Describe the configuration space for the three-link manipulator shown in Figure 3.12.

7–3 Describe the configuration space for the two-link manipulator shown in Figure 3.13.

7–4 Describe the configuration space for the two-link manipulator shown in Figure 3.14.

7–5 Describe the configuration space for the three-link manipulator shown in Figure 3.15.

7–6 Describe the configuration space for a six-link anthropomorphic arm equipped with a spherical wrist.

7–7 Show that the visibility graph includes all shortest semi-free paths from q_s to q_f. Note, this is equivalent to showing that a necessary condition for a path to be a shortest semi-free path is that it be included in the visibility graph.

7–8 Suppose q_s lies in the Voronoi region for a particular feature, f. Show that a straight-line path from q_s that follows the gradient $\nabla d(q_s, f)$ will arrive to a Voronoi edge before reaching any other feature f'.

7–9 Verify Equation (7.1).

7–10 Derive the equations needed to compute the shortest distance from a point p to the line segment in the plane with vertices a_1 and a_2.

7–11 Derive the equations needed to compute the shortest distance from a point p to the polygon in the plane with vertices a_i, $i = 1, \ldots, n$.

7–12 Derive the equations needed to compute the shortest distance from a point p to the polygon in three dimensions with vertices a_i, $i = 1, \ldots, n$.

7–13 Verify Equation (7.5).

7–14 Consider a simple polygonal robot with four vertices, such that at $q = (0, 0, 0)$ the vertices are located at $a_1(0) = (0, 0)$, $a_2(0) = (1, 0)$,

$a_3(0) = (1, 1)$, and $a_4(0) = (0, 1)$. If two point obstacles are located at $o_1 = (3, 3)$ and $o_2 = (-3, -3)$, determine the artificial workspace and configuration space forces that act on the robot.

7–15 Write a computer program to implement the path planner described in Section 7.3.2 for a three-link planar arm moving among polygonal obstacles.

7–16 Write a simple computer program to perform collision checking for the case of a polygonal robot moving in the plane among polygonal obstacles. Your program should accept a configuration q as input, and should return a value that indicates whether q is a collision-free configuration.

7–17 Give a procedure for generating random samples of orientations in $SO(n)$ given that you have access to a random number generator that can generate samples from the uniform distribution on the unit interval. Your samples need not be uniformly distributed on $SO(n)$.

7–18 Write a computer program to implement the PRM planner described in Section 7.4.1 for a three-link planar arm moving among polygonal obstacles.

7–19 Show by direct calculation that the determinant of the coefficient matrix in Equation (7.17) is $(t_f - t_0)^4$.

7–20 Suppose we wish a manipulator to start from an initial configuration at time t_0 and track a conveyor. Discuss the steps needed in planning a suitable trajectory for this problem.

7–21 Suppose we desire a joint space trajectory $\dot{q}_i^d(t)$ for the i^{th} joint (assumed to be revolute) that begins at rest at position q_0 at time t_0 and reaches position q_1 in 2 seconds with a final velocity of 1 radian/sec. Compute a cubic polynomial satisfying these constraints. Sketch the trajectory as a function of time.

7–22 Compute a LSPB trajectory to satisfy the same requirements as in Problem 7–21. Sketch the resulting position, velocity, and acceleration profiles.

7–23 Fill in the details of the computation of the LSPB trajectory. In other words, compute the portion of the trajectory between times $t_f - t_b$ and t_f and verify Equations (7.21).

7–24 Write a Matlab m-file, lspb.m, to generate an LSPB trajectory, given appropriate initial data.

7–25 Rewrite the Matlab m-files, cubic.m, quintic.m, and lspb.m to turn them into Matlab functions. Document them appropriately.

Notes and References

The earliest work on robot planning was done in the late sixties and early seventies in a few university-based Artificial Intelligence (AI) labs [40], [45], and [129]. This research dealt with high level planning using symbolic reasoning that was much in vogue at the time in the AI community. Geometry was not often explicitly considered in early robot planners, in part because it was not clear how to represent geometric constraints in a computationally feasible manner. The configuration space and its application to path planning were introduced in [97]. This was the first rigorous, formal treatment of the geometric path planning problem, and it initiated a surge in path planning research.

The earliest work in geometric path planning developed methods to construct volumetric representations of the free configuration space. These included exact methods [147], and approximate methods [20], [73], and [97]. In the former case, the best known algorithms have exponential complexity and require exact descriptions of both the robot and its environment, while in the latter case, the size of the representation of configuration space grows exponentially in the dimension of the configuration space. The best known algorithm for the path planning problem, giving an upper bound on the amount of computation time required to solve the problem, appeared in [21]. That real robots rarely have an exact description of the environment, and a drive for faster planning systems led to the development of potential fields approaches [77], and [79].

By the early nineties, a great deal of research had been done on the geometric path planning problem, and this work is nicely summarized in the textbook [87]. This textbook helped to generate a renewed interest in the path planning problem, and it provided a common framework in which to analyze and express path planning algorithms.

In the early nineties, randomization was introduced in the robot planning community [10], originally to circumvent the problems with local minima in potential fields. Early randomized motion planners proved effective for a large range of problems, but sometimes required extensive computation time for some robots in certain environments [75]. This limitation, together

with the idea that a robot will operate in the same environment for a long period of time led to the development of the probabilistic roadmap planners [74, 132, 75]. Rapidly-exploring random trees were introduced in [90, 91].

Comprehensive reviews of motion planning research, including sensor-based approaches, can be found in [24, 88, 89].

Much work has been done in the area of collision detection in recent years [96], [115], [177], and [178]. This work is primarily focused on finding efficient, incremental methods for detecting collisions between objects when one or both are moving. A number of public domain collision detection software packages are currently available on the Internet.