4 *Potential Functions*

HAVING SEEN the difficulty of explicitly representing the configuration space, an alternative is to develop search algorithms that incrementally "explore" free space while searching for a path. Already, we have seen that the Bug algorithms maneuver through free space without constructing the configuration space, but the Bug algorithms are limited to simple two-dimensional configuration spaces. Therefore, this chapter introduces navigation planners that apply to a richer class of robots and produce a greater variety of paths than the Bug methods, i.e., they apply to a more general class of configuration spaces, including those that are multidimensional and non-Euclidean.

A *potential function* is a differentiable real-valued function $U : \mathbb{R}^m \to \mathbb{R}$. The value of a potential function can be viewed as energy and hence the gradient of the potential is force. The *gradient* is a vector $\nabla U(q) = DU(q)^T = [\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)]^T$ which points in the direction that locally maximally increases U. See appendix C.5 for a more rigorous definition of the gradient. We use the gradient to define a *vector field,* which assigns a vector to each point on a manifold. A gradient vector field, as its name suggests, assigns the gradient of some function to each point. When U is energy, the gradient vector field has the property that work done along any closed path is zero.

The potential function approach directs a robot as if it were a particle moving in a gradient vector field. Gradients can be intuitively viewed as forces acting on a positively charged particle robot which is attracted to the negatively charged goal. Obstacles also have a positive charge which forms a repulsive force directing the robot away from obstacles. The combination of repulsive and attractive forces hopefully directs the robot from the start location to the goal location while avoiding obstacles (figure 4.1).

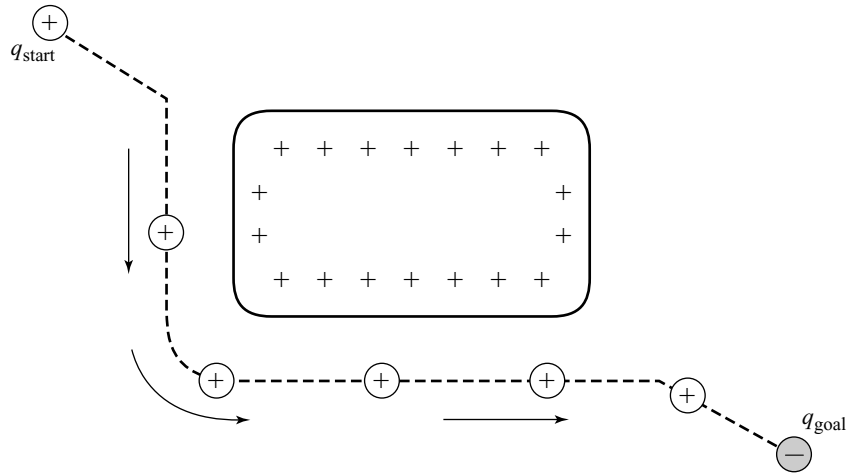


Figure 4.1 The negative charge attracts the robot and the positive charge repels it, resulting in a path, denoted by the dashed line, around the obstacle and to the goal.

Note that in this chapter, we mainly deal with first-order systems (i.e., we ignore dynamics), so we view the gradients as velocity vectors instead of force vectors. Potential functions can be viewed as a landscape where the robots move from a "high-value" state to a "low-value" state. The robot follows a path "downhill" by following the negated gradient of the potential function. Following such a path is called *gradient descent,* i.e.,

$$\dot{c}(t) = -\nabla U(c(t)).$$

The robot terminates motion when it reaches a point where the gradient vanishes, i.e., it has reached a q^* where $\nabla U(q^*) = 0$. Such a point q^* is called a *critical point* of U. The point q^* is either a maximum, minimum, or a saddle point (figure 4.2). One can look at the second derivative to determine the type of critical point. For real-valued functions, this second derivative is the *Hessian* matrix

$$\begin{bmatrix} \frac{\partial^2 U}{\partial q_1^2} & \cdots & \frac{\partial^2 U}{\partial q_1 \partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 U}{\partial q_1 \partial q_n} & \cdots & \frac{\partial^2 U}{\partial q_n^2} \end{bmatrix}.$$

When the Hessian is nonsingular at q^*, the critical point at q^* is *non-degenerate,* implying that the critical point is isolated [173]. When the Hessian is positive-definite, the critical point is a local minimum; when the Hessian is negative-definite, then

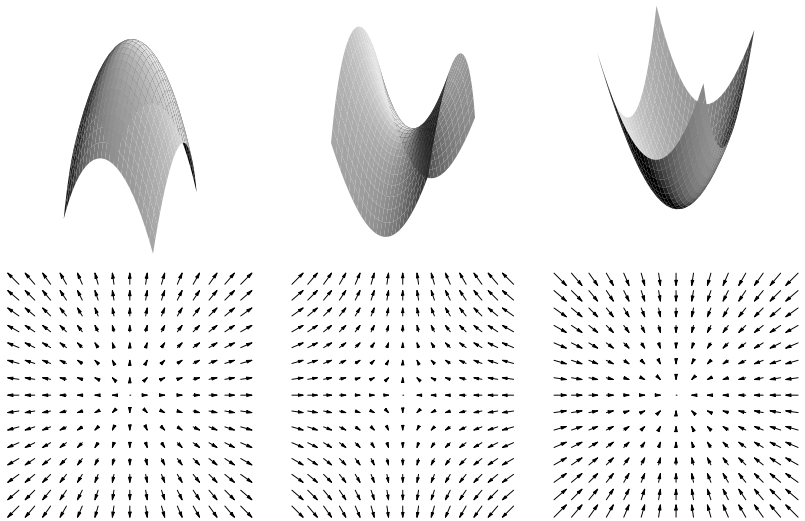


Figure 4.2 Different types of critical points: (Top) Graphs of functions. (Bottom) Gradients of functions.

the critical point is a local maximum. Generally, we consider potential functions whose Hessians are nonsingular, i.e., those which only have isolated critical points. This also means that the potential function is never flat.

For gradient descent methods, we do not have to compute the Hessian because the robot generically terminates its motion at a local minimum, not at a local maximum or a saddle point. Since gradient descent decreases U, the robot cannot arrive at a local maximum, unless of course the robot starts at a maximum. Since we assume that the function is never flat, the set of maxima contains just isolated points, and the likelihood of starting at one is practically zero. However, even if the robot starts at a maximum, any perturbation of the robot position frees the robot, allowing the gradient vector field to induce motion onto the robot. Arriving at a saddle point is also unlikely, because they are unstable as well. Local minima, on the other hand, are stable because after any perturbation from a minimum, gradient descent returns the robot to the minimum. Therefore, the only critical point where the robot can generically terminate is a local minimum. Hopefully this is where the goal is located. See figure 4.3 for an example of a configuration space with its corresponding potential function, along with its energy surface landscape and gradient vector field.

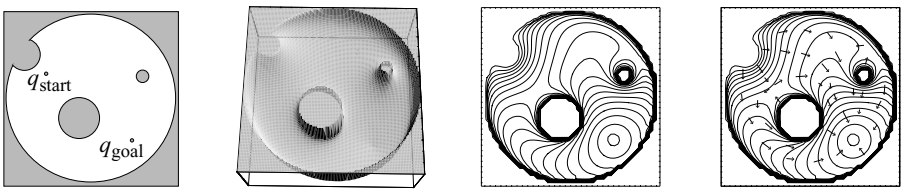


Figure 4.3 (a) A configuration space with three circular obstacles bounded by a circle. (b) Potential function energy surface. (c) Contour plot for energy surface. (d) Gradient vectors for potential function.

There are many potential functions other than the attractive/repulsive potential. Many of these potential functions are efficient to compute and can be computed online [234]. Unfortunately, they all suffer one problem—the existence of local minima not corresponding to the goal. This problem means that potential functions may lead the robot to a point which is not the goal; in other words, many potential functions do not lead to complete path planners. Two classes of approaches address this problem: the first class augments the potential field with a search-based planner, and the second defines a potential function with one local minimum, called a *navigation function* [239]. Although complete (or resolution complete), both methods require full knowledge of the configuration space prior to the planning event.

Finally, unless otherwise stated, the algorithms presented in this chapter apply to spaces of arbitrary dimension, even though the figures are drawn in two dimensions. Also, we include some discussion of implementation on a mobile base operating in the plane (i.e., a point in a two-dimensional Euclidean configuration space).

4.1 Additive Attractive/Repulsive Potential

The simplest potential function in $\mathcal{Q}_{\text{free}}$ is the attractive/repulsive potential. The intuition behind the attractive/repulsive potential is straightforward: the goal attracts the robot while the obstacles repel it. The sum of these effects draws the robot to the goal while deflecting it from obstacles. The potential function can be constructed as the sum of attractive and repulsive potentials

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

The Attractive Potential

There are several criteria that the potential field U_{att} should satisfy. First, U_{att} should be monotonically increasing with distance from q_{goal}. The simplest choice is the *conic potential,* measuring a scaled distance to the goal, i.e., $U(q) = \zeta d(q, q_{\text{goal}})$. The ζ is a parameter used to scale the effect of the attractive potential. The attractive gradient is $\nabla U(q) = \frac{\zeta}{d(q,q_{\text{goal}})}(q - q_{\text{goal}})$. The gradient vector points away from the goal with magnitude ζ at all points of the configuration space except the goal, where it is undefined. Starting from any point other than the goal, by following the negated gradient, a path is traced toward the goal.

When numerically implementing this method, gradient descent may have "chattering" problems since there is a discontinuity in the attractive gradient at the origin. For this reason, we would prefer a potential function that is continuously differentiable, such that the magnitude of the attractive gradient decreases as the robot approaches q_{goal}. The simplest such potential function is one that grows quadratically with the distance to q_{goal}, e.g.,

$$U_{\text{att}}(q) = \frac{1}{2}\zeta d^2(q, q_{\text{goal}}),$$

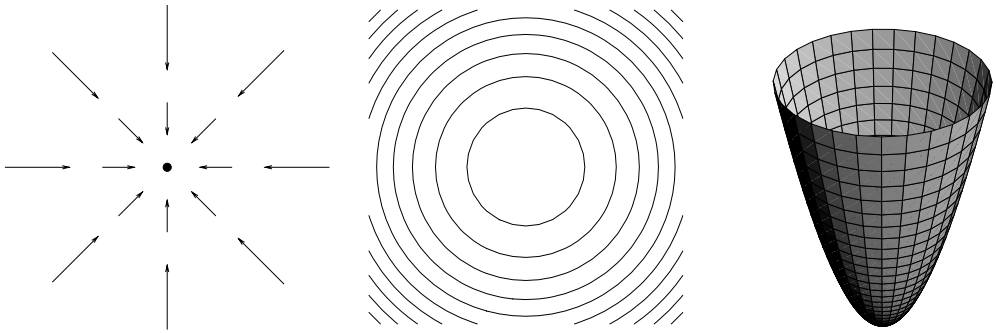
with the gradient

$$\begin{aligned} \nabla U_{\text{att}}(q) &= \nabla \left(\frac{1}{2}\zeta d^2(q, q_{\text{goal}}) \right), \\ &= \frac{1}{2}\zeta \nabla d^2(q, q_{\text{goal}}), \\ &= \zeta(q - q_{\text{goal}}), \end{aligned} \tag{4.1}$$

which is a vector based at q, points away from q_{goal}, and has a magnitude proportional to the distance from q to q_{goal}. The farther away q is from q_{goal}, the bigger the magnitude of the vector. In other words, when the robot is far away from the goal, the robot quickly approaches it; when the robot is close to the goal, the robot slowly approaches it. This feature is useful for mobile robots because it reduces "overshoot" of the goal (resulting from step quantization).

In figure 4.4(a), the goal is in the center and the gradient vectors for various points are drawn. Figure 4.4(b) contains a contour plot for U_{att}; each solid circle corresponds to a set of points q where $U_{\text{att}}(q)$ is constant. Finally, figure 4.4(c) plots the graph of the attractive potential.

Note that while the gradient $\nabla U_{\text{att}}(q)$ converges linearly to zero as q approaches q_{goal} (which is a desirable property), it grows without bound as q moves away from q_{goal}. If q_{start} is far from q_{goal}, this may produce a desired velocity that is too large. For this reason, we may choose to combine the quadratic and conic potentials so that the



(a) (b) (c)

Figure 4.4 (a) Attractive gradient vector field. (b) Attractive potential isocontours. (c) Graph of the attractive potential.

conic potential attracts the robot when it is very distant from q_{goal} and the quadratic potential attracts the robot when it is near q_{goal}. Of course it is necessary that the gradient be defined at the boundary between the conic and quadratic portions. Such a field can be defined by

$$U_{\text{att}}(q) = \begin{cases} \frac{1}{2}\zeta d^2(q, q_{\text{goal}}), & d(q, q_{\text{goal}}) \leq d^*_{\text{goal}}, \\ d^*_{\text{goal}}\zeta d(q, q_{\text{goal}}) - \frac{1}{2}\zeta (d^*_{\text{goal}})^2, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}. \end{cases} \tag{4.2}$$

and in this case we have

$$\nabla U_{\text{att}}(q) = \begin{cases} \zeta(q - q_{\text{goal}}), & d(q, q_{\text{goal}}) \leq d^*_{\text{goal}}, \\ \frac{d^*_{\text{goal}}\zeta(q - q_{\text{goal}})}{d(q, q_{\text{goal}})}, & d(q, q_{\text{goal}}) > d^*_{\text{goal}}, \end{cases} \tag{4.3}$$

where d^*_{goal} is the threshold distance from the goal where the planner switches between conic and quadratic potentials. The gradient is well defined at the boundary of the two fields since at the boundary where $d(q, q_{\text{goal}}) = d^*_{\text{goal}}$, the gradient of the quadratic potential is equal to the gradient of the conic potential, $\nabla U_{\text{att}}(q) = \zeta(q - q_{\text{goal}})$.

The Repulsive Potential

A repulsive potential keeps the robot away from an obstacle. The strength of the repulsive force depends upon the robot's proximity to the an obstacle. The closer the robot is to an obstacle, the stronger the repulsive force should be. Therefore, the

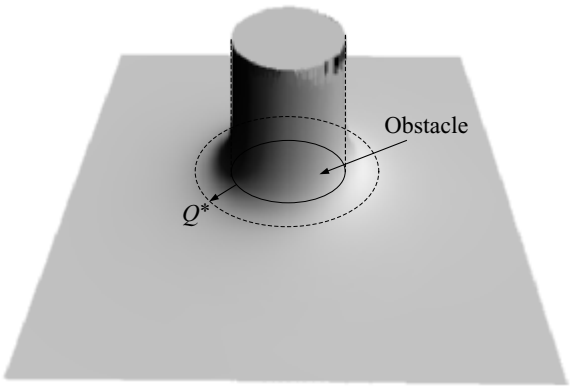


Figure 4.5 The repulsive gradient operates only in a domain near the obstacle.

repulsive potential is usually defined in terms of distance to the closest obstacle $D(q)$, i.e.,

$$U_{\text{rep}}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{D(q)} - \frac{1}{Q^*}\right)^2, & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases} \tag{4.4}$$

whose gradient is

$$\nabla U_{\text{rep}}(q) = \begin{cases} \eta\left(\frac{1}{Q^*} - \frac{1}{D(q)}\right)\frac{1}{D^2(q)}\nabla D(q), & D(q) \leq Q^*, \\ 0, & D(q) > Q^*, \end{cases} \tag{4.5}$$

where the $Q^* \in \mathbb{R}$ factor allows the robot to ignore obstacles sufficiently far away from it and the η can be viewed as a gain on the repulsive gradient. These scalars are usually determined by trial and error. (See figure 4.5.)

When numerically implementing this solution, a path may form that oscillates around points that are two-way equidistant from obstacles, i.e., points where D is nonsmooth. To avoid these oscillations, instead of defining the repulsive potential function in terms of distance to the *closest* obstacle, the repulsive potential function is redefined in terms of distances to *individual* obstacles where $d_i(q)$ is the distance to obstacle $\mathcal{QO}_i$, i.e.,

$$d_i(q) = \min_{c \in \mathcal{QO}_i} d(q, c). \tag{4.6}$$

Note that the min operator returns the smallest $d(q, c)$ for all points c in $\mathcal{QO}_i$.

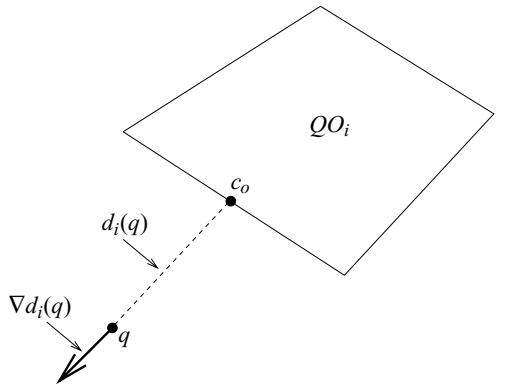


Figure 4.6 The distance between x and $\mathcal{QO}_i$ is the distance to the closest point on $\mathcal{QO}_i$. The gradient is a unit vector pointing away from the nearest point.

It can be shown for convex obstacles $\mathcal{QO}_i$ where c is the closest point to x that the gradient of $d_i(q)$ is

$$\nabla d_i(q) = \frac{q - c}{d(q, c)}. \tag{4.7}$$

The vector $\nabla d_i(q)$ describes the direction that maximally increases the distance to $\mathcal{QO}_i$ from q (figure 4.6).

Now, each obstacle has its own potential function,

$$U_{\text{rep}_i}(q) = \begin{cases} \frac{1}{2}\eta \left(\frac{1}{d_i(q)} - \frac{1}{Q_i^*} \right)^2, & \text{if } d_i(q) \leq Q_i^*, \\ 0, & \text{if } d_i(q) > Q_i^*, \end{cases}$$

where Q_i^* defines the size of the domain of influence for obstacle $\mathcal{QO}_i$. Then $U_{\text{rep}}(q) = \sum_{i=1}^{n} U_{\text{rep}_i}(q)$. Assuming that there are only convex obstacles or nonconvex ones can be decomposed into convex pieces, oscillations do not occur because the planner does not have radical changes in the closest point anymore.

4.2 Gradient Descent

Gradient descent is a well-known approach to optimization problems. The idea is simple. Starting at the initial configuration, take a small step in the direction opposite the gradient. This gives a new configuration, and the process is repeated until the gradient is zero. More formally, we can define a gradient descent algorithm (algorithm 4).

Algorithm 4 Gradient Descent

Input: A means to compute the gradient $\nabla U(q)$ at a point q

Output: A sequence of points $\{q(0), q(1), \ldots, q(i)\}$

1: $q(0) = q_{\text{start}}$
2: $i = 0$
3: **while** $\nabla U(q(i)) \neq 0$ **do**
4: $\quad q(i+1) = q(i) + \alpha(i)\nabla U(q(i))$
5: $\quad i = i + 1$
6: **end while**

In algorithm 4, the notation $q(i)$ is used to denote the value of q at the ith iteration and the final path consists of the sequence of iterates $\{q(0), q(1), \ldots, q(i)\}$. The value of the scalar $\alpha(i)$ determines the step size at the i iteration. It is important that $\alpha(i)$ 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 $\alpha(i)$ is often made on an *ad hoc* or empirical basis, perhaps based on the distance to the nearest obstacle or to the goal. A number of systematic methods for choosing $\alpha(i)$ can be found in the optimization literature [45]. Finally, it is unlikely that we will ever exactly satisfy the condition $\nabla U(q(i)) = 0$. For this reason, this condition is often replaced with the more forgiving condition $\|\nabla U(q(i))\| < \epsilon$, in which ϵ is chosen to be sufficiently small, based on the task requirements.

4.3 Computing Distance for Implementation in the Plane

In this section, we discuss some implementation issues in constructing the attractive/repulsive potential function. The attractive potential function is rather straightforward if the robot knows its current location and the goal location. The challenge lies in computing the repulsive function because it requires calculation of distance to obstacles. Therefore, in this section, we discuss two different methods to compute distance, and hence the repulsive potential function. The first method deals with sensor-based implementation on a mobile robot and borrows ideas from chapter 2 in inferring distance information from sensors. The second method assumes the configuration space has been discretized into a grid of pixels and computes distance on the grid.

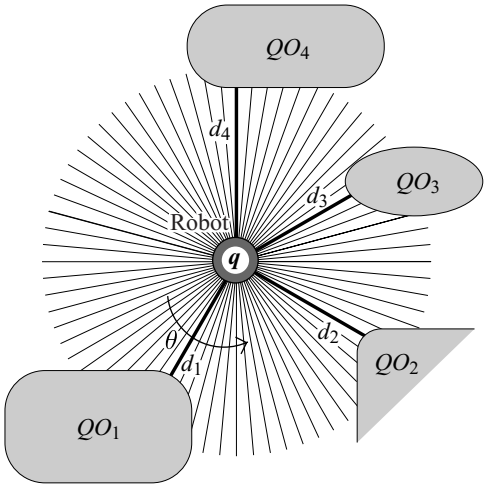


Figure 4.7 Local minima of rays determine the distance to nearby obstacles.

4.3.1 Mobile Robot Implementation

Thus far, the discussion has been general to any configuration space where we can define distance. Now let's consider some issues in implementing these potential functions on a planar mobile robot equipped with range sensors radially distributed around its circumference. These range sensors approximate a value of the raw distance function ρ defined in chapter 2. Whereas $D(q)$ corresponds to the global minimum of the raw distance function ρ, a $d_i(q)$ corresponds to a local minimum with respect to θ of $\rho(q, \theta)$ (figure 4.7). For example, any sensor in the sonar array whose value is less than that of both its left and right neighbors is a local minimum. Such sensors face the closest points on their corresponding obstacles. Therefore, these sensors point in the direction that maximally brings the robot closest to the obstacles, i.e., $-\nabla d_i(q)$. The distance gradient points in the opposite direction. An obstacle distance function may be incorrect if the obstacle is partially occluded by another.

4.3.2 Brushfire Algorithm: A Method to Compute Distance on a Grid

In this subsection, we explain a method for computing distance from a map representation called a *grid,* which is a two-dimensional array of square elements called *pixels.* A pixel has a value of zero if it is completely free of obstacles and one if it is completely or even partially occupied by an obstacle.

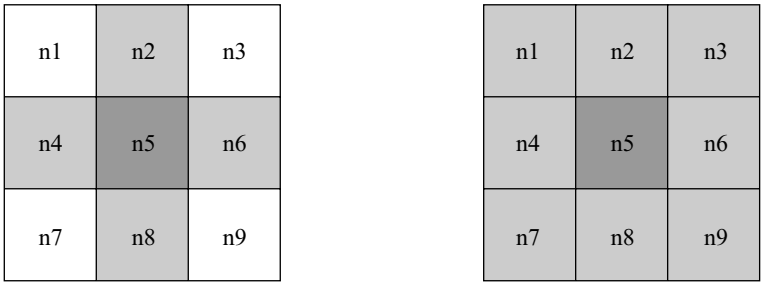


Figure 4.8 Four-point vs. eight-point connectivity.

The user or planner has a choice in determining the neighboring relationships of pixels in a grid. When only the north, south, east, and west pixels are considered neighbors, the grid has *four-point connectivity*. When the grid also includes the diagonals as neighbors, then it has *eight-point connectivity* (figure 4.8). Four-point connectivity has the advantage in that it respects the Manhattan distance function (the L^1 metric) because it measures distance as if one were driving in city blocks in midtown Manhattan.

The *brushfire algorithm* uses a grid to approximate distance, and hence the repulsive function. The input to the algorithm is a grid of pixels where the free-space pixels have a value of zero and the obstacles have a value of one. The output is a grid of pixels whose corresponding values each measure distance to the nearest obstacle. These values can then be used to compute a repulsive potential function, as well as its gradient.

In the first step of the brushfire algorithm, all zero-valued pixels neighboring one-valued pixels are labeled with a two. The algorithm can use four-point or eight-point connectivity to determine adjacency. Next, all zero-valued pixels adjacent to two's are labeled with a three. This procedure repeats, i.e., all zero-valued pixels adjacent to an i are labeled with an $i + 1$, as if a brushfire is growing from each of the obstacles until the fire consumes all of the free-space pixels. The procedure terminates when all pixels have an assigned value (figure 4.9).

The brushfire method produces a map of distance values to the nearest obstacle. The gradient of distance at a pixel is determined by finding a neighbor with the lowest pixel value. The gradient is then a vector which points to this neighbor. Note that this vector points in either one of four or one of eight possible directions. If there are multiple neighbors with the same lowest value, simply pick one to define the gradient. Just as the grid is an approximation of the workspace, so is the computed gradient an approximation of the actual distance gradient.

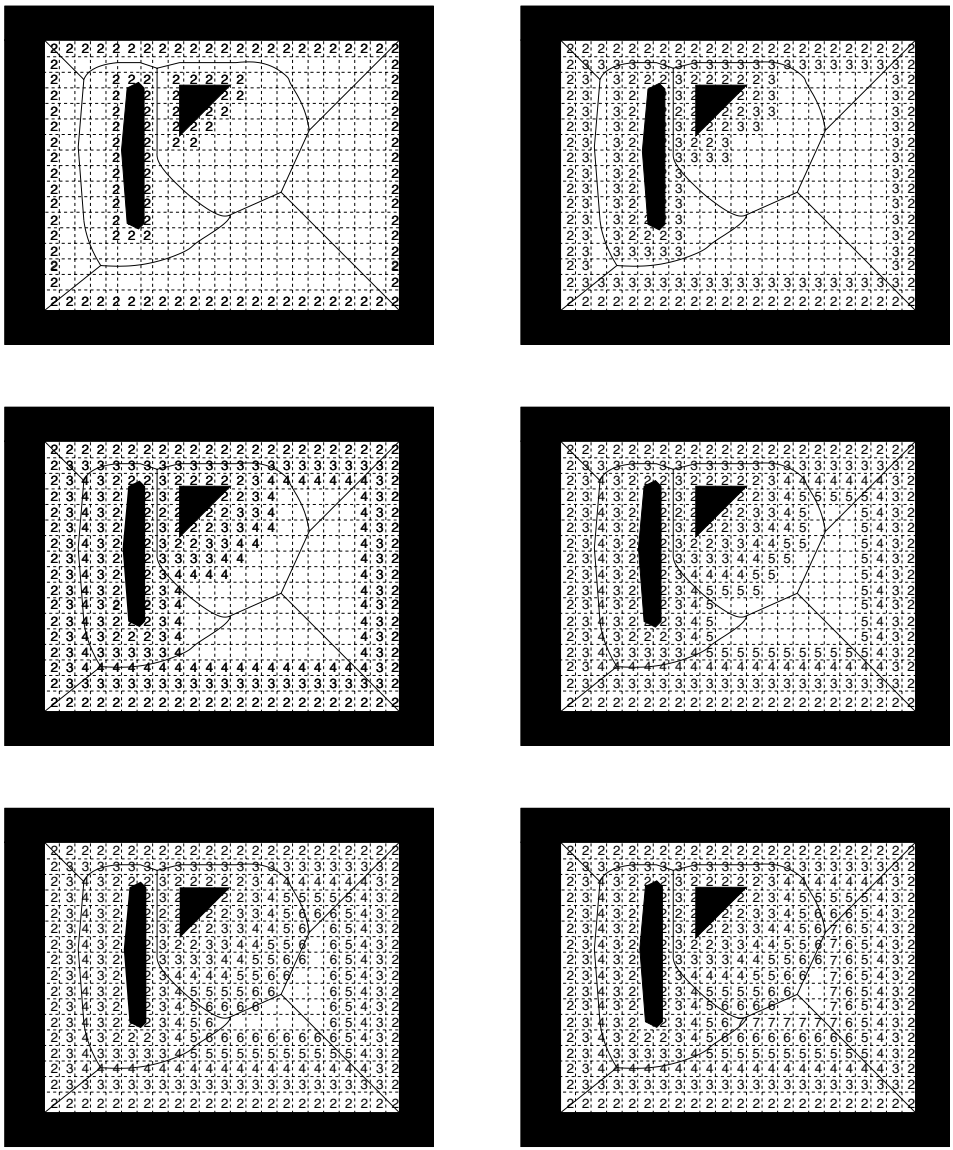


Figure 4.9 Propagation of the brushfire algorithm with eight-point connectivity. The solid lines pass through pixels where fronts collide.

With distance and gradient to the nearest obstacle inhand, a planner can compute the repulsive function. The attractive potential function can be computed analytically and together with the repulsive function, a planner can invoke the additive attractive/repulsive function described in section 4.1.

It is worth noting that the method described here generalizes into higher dimensions where pixels then become volume elements. For example, in three-dimensions, four-point connectivity generalizes to six-point connectivity and eight-point connectivity generalizes to twenty-six-point connectivity. So, when assigning incremental pixel values to neighboring pixels in higher dimensions, the algorithm choses the appropriate adjacency relationship and then iterates through as described above. Although possible, it would become computationally intractable to compute the brushfire in higher dimensions.

4.4 Local Minima Problem

The problem that plagues all gradient descent algorithms is the possible existence of local minima in the potential field. For appropriate choice of $\alpha(i)$, it can be shown that the gradient descent algorithm is generically guaranteed to converge to a minimum in the field, but there is no guarantee that this minimum will be the global minimum. This means that there is no guarantee that gradient descent will find a path to q_{goal}. In figure 4.10, the robot is initially attracted to the goal as it approaches the horseshoe-shaped obstacle. The goal continues to attract the robot, but the bottom arm of the obstacle deflects the robot upward until the top arm of the horseshoe begins to influence

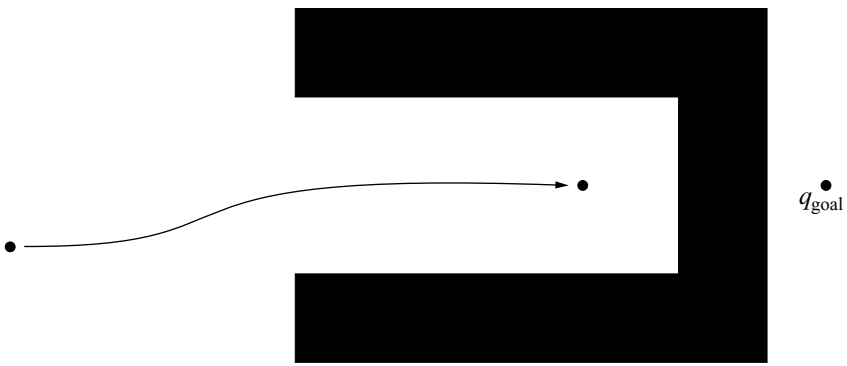


Figure 4.10 Local minimum inside the concavity. The robot moves into the concavity until the repulsive gradient balances out the attractive gradient.

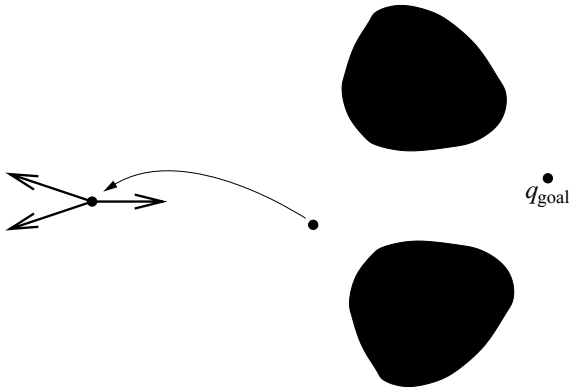


Figure 4.11 Local minimum without concave obstacles. The robot moves away from the two convex obstacles until it reaches a point where the gradient vanishes; at this point, the sum of the attractive gradient and the repulsive gradient is zero.

the robot. At this point, the effect of the top and bottom arms keeps the robot halfway between them and the goal continues to attract the robot. The robot reaches a point where the effect of the obstacle's base counteracts the attraction of the goal. In other words, the robot has reached a q^* where $\nabla U(q^*) = 0$ and q^* is *not* the goal. Note, this problem is not limited to concave obstacles as can be seen in figure 4.11. Local minima present a significant drawback to the attractive/repulsive approach, and thus the attractive/repulsive technique is not complete.

Barraquand and Latombe [37] developed search techniques other than gradient descent to overcome the problem of local minima present when planning with potential functions. Their planner, the Randomized Path Planner (RPP) [37], used a variety of potential functions some of which were simplified expressions of the potentials presented in this chapter. RPP followed the negative gradient of the specified potential function and when stuck at a local minimum, it initiated a series of random walks. Often the random walks allowed RPP to escape the local minimum and in that case, the negative gradient to the goal was followed again.

4.5 Wave-Front Planner

The wave-front planner [38, 208] affords the simplest solution to the local minima problem, but can only be implemented in spaces that are represented as grids. For the sake of discussion, consider a two-dimensional space. Initially, the planner starts with the standard binary grid of zeros corresponding to free space and ones to obstacles. The

planner also knows the pixel locations of the start and goal. The goal pixel is labeled with a two. In the first step, all zero-valued pixels neighboring the goal are labeled with a three. Next, all zero-valued pixels adjacent to threes are labeled with four. This procedure essentially grows a wave front from the goal where at each iteration, all pixels on the wave front have the same path length, measured with respect to the grid, to the goal. This procedure terminates when the wave front reaches the pixel that contains the robot start location.

The planner then determines a path via gradient descent on the grid starting from the start. Essentially, the planner determines the path one pixel at a time. Assume that the value of the start pixel is 33. The next pixel in the path is any neighboring pixel whose value is 32. There could be multiple choices; simply pick any one of the choices. The next pixel is then one whose value is 31. Boundedness of the free space (and hence the discretization) and continuity of the distance function ensure that construction of the wave front guarantees that there will always be a neighboring pixel whose value is one less than that of the current pixel and that this procedure forms a path in the grid to the goal, i.e., to the pixel whose value is two.

Figure 4.12 contains six panels that demonstrate various stages of the wave-front propagation using four-point connectivity. Note that all points on the wave front have the same Manhattan distance to the goal. In the lower-left panel, note how the wave-front seemingly collides on itself. We will see later that the point of initial collision corresponds to a saddle point of the function that measures distance to the goal. This point then propagates away from the start as well. The trace of this propagation corresponds to a set of points that have two choices for shortest paths back to the goal, either going around the top of the triangle or below it.

The wave-front planner essentially forms a potential function on the grid which has one local minimum and thus is resolution complete. The planner also determines the shortest path, but at the cost of coming dangerously close to obstacles. The major drawback of this method is that the planner has to search the entire space for a path.

Finally, just like the brushfire method, the wave-front planner generalizes into higher dimensions as well. Consider the three-dimensional case first. Just as a pixel has four edges, a voxel (a three-dimensional pixel) has six faces. Therefore, the analogy to four-point connectivity with pixels is six-point connectivity with voxels. For a voxel with value i, if we assume six-point connectivity, then we assign $i + 1$ to the surrounding six voxels that share a face with the current voxel. Likewise, if we assume twenty-six-point connectivity (analogous to eight-point connectivity), then we assign $i + 1$ to all surrounding voxels that share a face, edge or vertex. It should be noted, however, implementation of the wavefront planner in higher dimensions becomes computationally intractable.

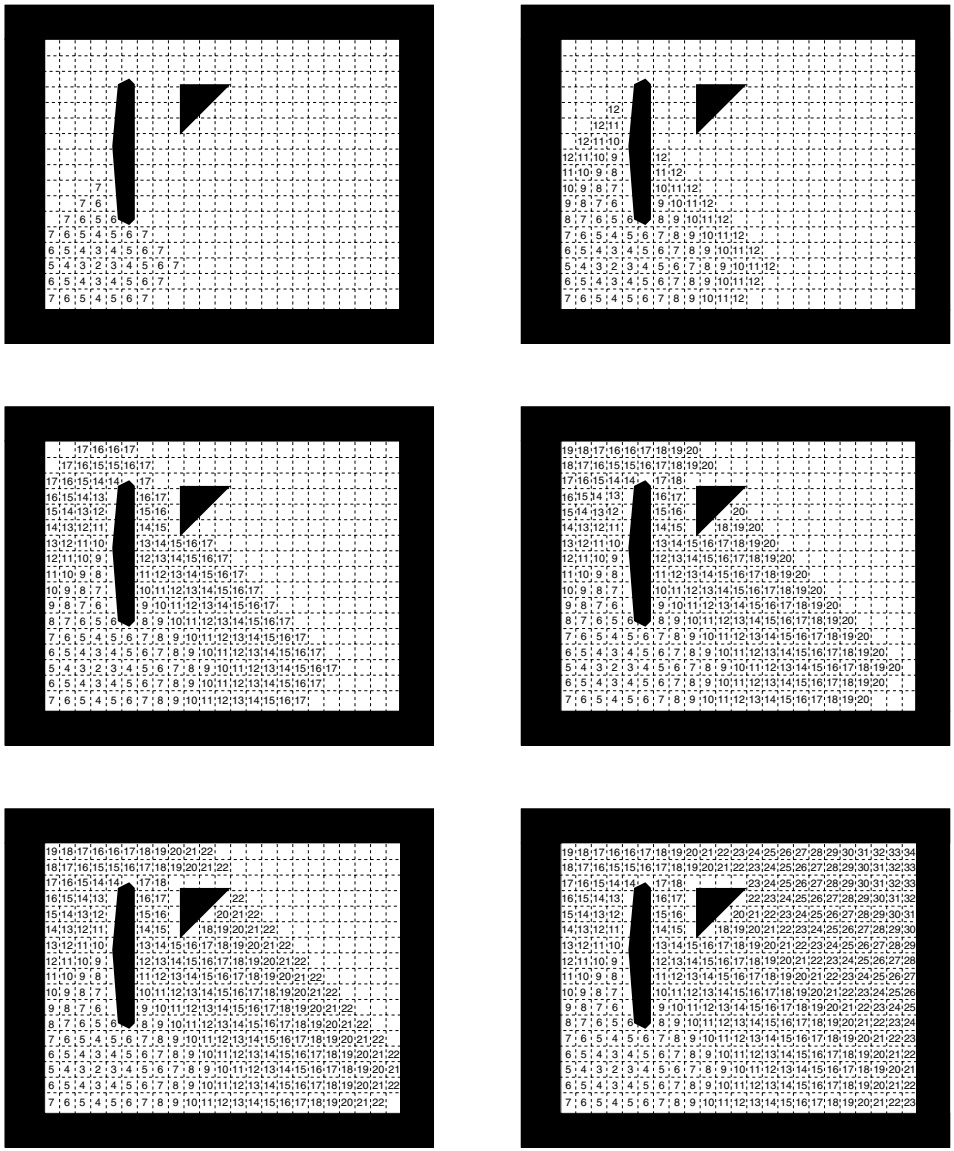


Figure 4.12 Propagation of the wave front using four-point connectivity (assume the start is in the upper-right corner and the goal is the origin of the wave front).

4.6 Navigation Potential Functions

Thus far, we have seen in chapter 2 that the Bug algorithms are complete sensor-based planners that work in unknown spaces, but are limited to planar configuration spaces. Then, at the beginning of this chapter, we have seen that the attractive/repulsive potential function approach applies to a general class of configuration spaces, but suffers from local minima problems, and hence is not complete. The wave-front planner addresses the local minima problem, but requires time and storage exponential in the dimension of the space. In this section, we introduce a new potential that is a function of distance to the obstacles, has only one minimum and applies to a limited class of configuration spaces with dimension two, three, and more. Such potential functions are called *navigation functions,* formally defined in [239, 364].

DEFINITION 4.6.1 *A function* $\varphi : \mathcal{Q}_{\text{free}} \rightarrow [0, 1]$ *is called a* navigation function *if it*

- *is smooth (or at least* C^k *for* $k \geq 2$*),*
- *has a unique minimum at* q_{goal} *in the connected component of the free space that contains* q_{goal}*,*
- *is uniformly maximal on the boundary of the free space, and*
- *is Morse.*

A *Morse function* is one whose critical points are all non-degenerate. This means that critical points are isolated, and if a Morse function is used for gradient descent, any random perturbation will destabilize saddles and maxima. The navigation function approach represents obstacles as $\mathcal{QO}_i = \{q \mid \beta_i(q) \leq 0\}$; in other words, $\beta_i(q)$ is negative in the interior of $\mathcal{QO}_i$, zero on the boundary of $\mathcal{QO}_i$, and positive in the exterior of $\mathcal{QO}_i$.

4.6.1 Sphere-Space

This approach initially assumes that the configuration space is bounded by a sphere centered at q_0 and has n $\dim(\mathcal{Q}_{\text{free}})$-dimensional spherical obstacles centered at $q_1, \ldots q_n$. The obstacle distance functions are easy to define as

$$\beta_0(q) = -d^2(q, q_0) + r_0^2,$$
$$\beta_i(q) = d^2(q, q_i) - r_i^2,$$

where r_i is the radius of the sphere. Note that $\beta_i(q)$ increases continuously as q moves away from the obstacle. Instead of considering the distance to the closest obstacle or

the distance to each individual obstacle, we consider

$$\beta(q) = \prod_{i=0}^{n} \beta_i(q). \tag{4.8}$$

Note that $\beta(q)$ is zero on the boundary of any obstacle, and positive at all points in the interior of the free space. This presumes that the obstacles are disjoint.

This approach uses β to form a repulsive-like function. The attractive portion of the navigation function is a power of distance to the goal, i.e.,

$$\gamma_\kappa(q) = (d(q, q_{\text{goal}}))^{2\kappa}, \tag{4.9}$$

where γ_κ has zero value at the goal and continuously increases as q moves away from the goal. The function $\frac{\gamma_\kappa}{\beta}(q)$ is equal to zero only at the goal, and it goes to infinity as q approaches the boundary of *any* obstacle. More importantly, for a large enough κ, the function $\frac{\gamma_\kappa}{\beta}(q)$ has a unique minimum. This is true because as κ increases, the term $\partial\gamma_\kappa/\partial q$ dominates $\partial\beta/\partial q$, meaning that the gradient of $\frac{\gamma_\kappa}{\beta}$ points toward the goal. Essentially, increasing κ has the effect of making $\frac{\gamma_\kappa}{\beta}$ take the form of a steep bowl centered at the goal. Increasing κ also causes other critical points to gravitate toward the obstacles, as the range of repulsive influence of the obstacles becomes small relative to the overwhelming influence of the attractive field.

Near an obstacle, only that obstacle has a significant effect on the value of $\frac{\gamma_\kappa}{\beta}$. Therefore, the only opportunity for a local minimum to appear is along a radial line between the obstacle and the goal. On this line near the boundary of an obstacle, the Hessian of $\frac{\gamma_\kappa}{\beta}$ cannot be positive definite because $\frac{\gamma_\kappa}{\beta}$ is quickly decreasing in value moving from the obstacle to the goal. Therefore there cannot be any local minimum for large κ, except at the goal [239].

So $\frac{\gamma_\kappa}{\beta}$ has a unique minimum, but unfortunately it can have arbitrarily large values, making it difficut to compute. Therefore, we introduce the analytical switch, which is defined as

$$\sigma_\lambda(x) = \frac{x}{\lambda + x}, \quad \lambda > 0. \tag{4.10}$$

Since $\sigma_\lambda(x)$ is zero at $x = 0$, converges to one as x approaches ∞, and is continuous (figure 4.13), we can use $\sigma_\lambda(x)$ to bound the value of the function $\frac{\gamma_\kappa}{\beta}$, i.e.,

$$s(q, \lambda) = \left(\sigma_\lambda \circ \frac{\gamma_\kappa}{\beta}\right)(q) = \left(\frac{\gamma_\kappa}{\lambda\beta + \gamma_\kappa}\right)(q). \tag{4.11}$$

The function $s(q, \lambda)$ has a zero value at the goal, unitary value on the boundary of any obstacle, and varies continuously in the free space. It has a unique minimum for a large enough κ. However, it is still not necessarily a Morse function because it may have degenerate critical points. So, we introduce another function that essentially

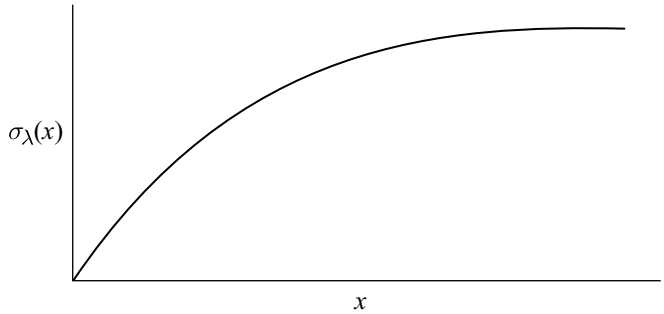


Figure 4.13 Analytic switch function which is used to map the range of a function to the unit interval.

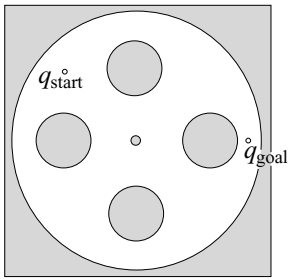


Figure 4.14 Configuration space bounded by a circle with five circle obstacles.

sharpens $s(q, \lambda)$ so its critical points become nondegenerate, i.e., so that $s(q, \lambda)$ can become a Morse function. This sharpening function is

$$\xi_\kappa(x) = x^{\frac{1}{\kappa}}. \tag{4.12}$$

For $\lambda = 1$, the resulting navigation function on a sphere-world is then

$$\varphi(q) = \left(\xi_\kappa \circ \sigma_1 \circ \frac{\gamma_\kappa}{\beta}\right)(q) = \frac{d^2(q, q_{\text{goal}})}{[(d(q, q_{\text{goal}}))^{2\kappa} + \beta(q)]^{1/\kappa}}, \tag{4.13}$$

which is guaranteed to have a single minimum at q_{goal} for a sufficiently large κ [239]. Consider the configuration space in figure 4.14. The effect of increasing κ can be seen in figure 4.15, which plots the contour lines for φ as κ increases. For $\kappa = 3$, φ has three local minima, one of which is the global minimum. For $\kappa = 4$ and 6, the local minima become more apparent because it is easier to see the contour lines (actually loops) that encircle the local minima. For $\kappa = 7$ and 8, the "bad" minima are there

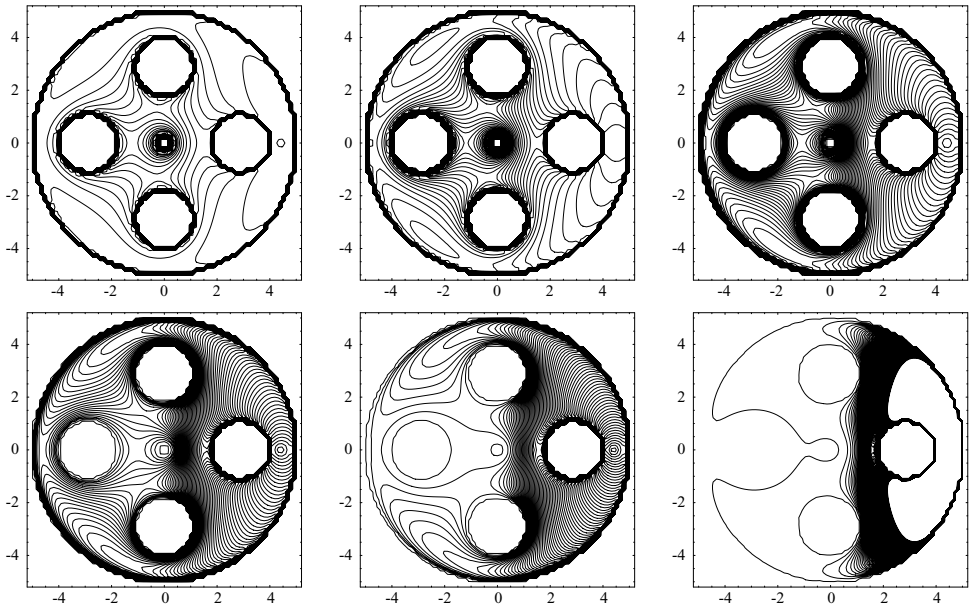


Figure 4.15 Navigation function for a sphere-space with five obstacles for $\kappa = 3$, $\kappa = 4$, $\kappa = 6$, $\kappa = 7$, $\kappa = 8$, and $\kappa = 10$.

but hard to see. Eventually, the "bad" minima morph into saddle points, which are unstable. For $\kappa = 10$, φ has a unique minimum. Therefore, gradient descent will direct the robot to the goal.

We can see the effect of the potential function steepening, critical points gravitating toward the goal, and local minima turning into saddles, in figure 4.16. Unfortunately, this steepening effect has an adverse consequence. The drawback to this particular navigation function is that it is flat near the goal and far away from the goal, but has sharp transitions in between (figure 4.16). This makes implementation of a gradient descent approach quite difficult because of numerical errors.

4.6.2 Star-Space

The result of sphere-spaces is just the first step toward a more general planner. A sphere-space can serve as a "model space" for any configuration space that is diffeomorphic to the sphere-space. Once we have a navigation function for the model space, to find a navigation function for the diffeomorphic configuration space, we need only find the diffeomorphism relating the two spaces.

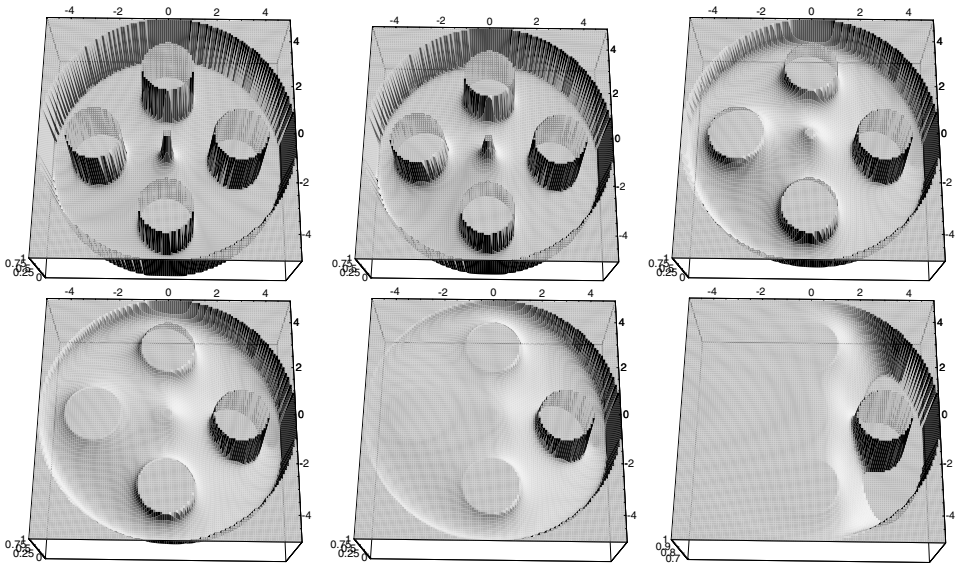


Figure 4.16 Navigation function for a sphere-space with five obstacles for $\kappa = 3$, $\kappa = 4$, $\kappa = 6$, $\kappa = 7$, $\kappa = 8$, and $\kappa = 10$.

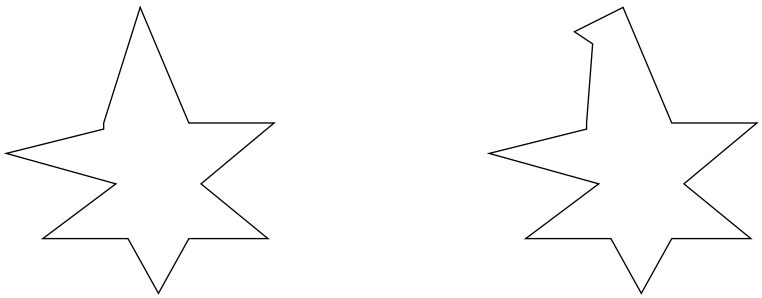


Figure 4.17 (Left) Star-shaped set. (Right) Not a star-shaped set.

In this subsection we consider *star-spaces* consisting of a *star-shaped* configuration space populated by star-shaped obstacles. A star-shaped set S is a set where there exists at least one point that is within line of sight of all other points in the set, i.e.,

$$\exists x \text{ such that } \forall y \in S, \qquad tx + (1-t)y \in S \quad \forall t \in [0,1].$$

See figure 4.17. All convex sets are star-shaped, but the converse is not true.

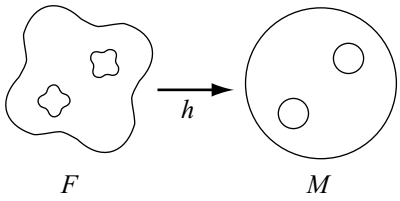


Figure 4.18 The diffeomorphism h maps the star-space F to the sphere-space M.

The approach is to map a configuration space populated by star-shaped obstacles into a space populated by sphere-shaped obstacles. It can be shown [364] that for two free configuration spaces M and F, if $\varphi : M \to [0, 1]$ is a navigation function on M and there exists a mapping $h : F \to M$ which is a diffeomorphism, i.e., it is smooth, bijective, and has a smooth inverse, then $\phi = \varphi \circ h$ is a navigation function on F (see figure 4.18). This diffeormorphism ensures that there is a one-to-one correspondence between critical points. We will use this property to define navigation functions in star-spaces using results from sphere-spaces.

The h mapping between the star- and sphere-spaces will be constructed using a translated scaling map

$$T_i(q) = \nu_i(q)(q - q_i) + p_i, \tag{4.14}$$

where

$$\nu_i(q) = (1 + \beta_i(q))^{1/2} \frac{r_i}{d(q, q_i)}, \tag{4.15}$$

where q_i is the center of the star-shaped set, and p_i and r_i are, respectively, the center and radius of the spherical obstacle. Here $\beta_i(q)$ defines a star-shaped set such that $\beta_i(q)$ is negative in the interior, zero on the boundary, and positive in the exterior.

Note that if q is in the boundary of the star-shaped obstacle, then $(1 + \beta_i(q)) = 1$, and thus $T_i(q) = r_i \frac{q - q_i}{d(q, q_i)} + p_i$. In other words, $T_i(q)$ maps points on the boundary of the star-shaped set to a sphere.

For the star-shaped obstacle $\mathcal{QO}_i$, we define the analytical switch

$$s_i(q, \lambda) = \left(\sigma_\lambda \circ \frac{\gamma_\kappa \bar{\beta}_i}{\beta_i} \right)(q) = \left(\frac{\gamma_\kappa \bar{\beta}_i}{\gamma_\kappa \bar{\beta}_i + \lambda \beta_i} \right)(q), \tag{4.16}$$

where

$$\bar{\beta}_i = \prod_{j=0, j \neq i}^{n} \beta_j, \tag{4.17}$$

i.e., $\bar{\beta}_i$ is zero on the boundary of the obstacles except the "current" obstacle $\mathcal{QO}_i$. Note that $s_i(q, \lambda)$ is one on the boundary of $\mathcal{QO}_i$, but is zero at the goal and on the boundary of all other obstacles except $\mathcal{QO}_i$.

We define a similar switch for the goal which is one at the goal and zero on the boundary of the free space, i.e.,

$$s_{q_{\text{goal}}}(q, \lambda) = 1 - \sum_{i=0}^{M} s_i. \tag{4.18}$$

Now, using the above switches and a translated scaling map, we can define a mapping between star-space and sphere-space as

$$h_\lambda(q) = s_{q_{\text{goal}}}(q, \lambda) T_{q_{\text{goal}}}(q) + \sum_{i=0}^{M} s_i(q, \lambda) T_i(q), \tag{4.19}$$

where $T_{q_{\text{goal}}}(q) = q$ is just the identity map, used for notational consistency.

Note that $h_\lambda(q)$ is exactly $T_i(q)$ on the boundary of the $\mathcal{QO}_i$ because s_i is one on the boundary of $\mathcal{QO}_i$ and for all $j \neq i$, s_j is zero on the boundary of $\mathcal{QO}_i$ (here we include $s_{q_{\text{goal}}}$ as one of the s_j's). In other words, for each i, $h_\lambda(q)$ is T_i on the boundary of obstacle $\mathcal{QO}_i$, which maps the boundary of a star to a sphere. Moreover, $h_\lambda(q)$ is continuous and thus $h_\lambda(q)$ maps the entire star-space to a sphere-space. It can be shown that for a suitable λ, $h_\lambda(q)$ is smooth, bijective, and has a smooth inverse, i.e., is a diffeomorphism [239]. Therefore, since we have a navigation function on a sphere-space, we also have a navigation function on the star-space.

4.7 Potential Functions in Non-Euclidean Spaces

Putting the issue of local minima aside for a moment, another major challenge for implementing potential functions is constructing the configuration space in the first place. This is especially challenging when the configuration space is non-Euclidean and multidimensional. In order to deal with this difficulty, we will define a potential function in the workspace, which is Euclidean, and then lift it to the configuration space. Here, we compute a gradient in the configuration space as a function of gradients in the workspace. To do so, instead of thinking of gradients as velocity vectors, we will now think of them as forces. We then establish a relationship between a workspace force and a configuration space force. Then we apply this relationship to a single rigid-body robot, i.e., we show how to derive a configuration space force using workspace forces acting a rigid-body robot. Finally, we apply this relationship to a multibody robot. We focus the discussion in this section on the attractive/repulsive potentials from section 4.1.

4.7.1 Relationship between Forces in the Workspace and Configuration Space

Since the workspace is a subset of a low-dimensional space (either $\mathbb{R}^2$ or $\mathbb{R}^3$), it is much easier to implement and evaluate potential functions over the workspace than over the configuration space. Now, we will treat the gradient in the workspace as forces. Naturally, workspace potential functions give rise to workspace forces, but ultimately, we will need forces in the configuration space to determine the path for the robot.

Let x and q be coordinate vectors representing a point in the workspace and the configuration of the robot, respectively, where the coordinates x and q are related by the forward kinematics (chapter 3) $x = \phi(q)$. Let f and u denote generalized forces in the workspace and the configuration space, respectively. To represent a force f acting at a point $x = \phi(q)$ in the workspace as a generalized force u acting in the robot's configuration space, we use the principle of virtual work, which essentially says that work (or power) is a coordinate-independent quantity. This means that power measured in workspace coordinates must be equal to power measured in configuration space coordinates. In the workspace, the power done by a force f is the familiar $f^T \dot{x}$. In the configuration space, power is given by $u^T \dot{q}$. From chapter 3, section 3.8, we know that $\dot{x} = J\dot{q}$, where $J = \partial\phi/\partial q$ is the Jacobian of the forward kinematic map. Therefore, the mapping from workspace forces to configuration space forces is given by

$$\begin{aligned} f^T J\dot{q} &= u^T \dot{q} \\ f^T J &= u^T \\ J^T f &= u. \end{aligned}$$

EXAMPLE 4.7.1 (A Force Acting on a Vertex of a Polygonal Robot) *Consider the polygonal robot shown in figure 4.19. The vertex a has coordinates* $[a_x, a_y]^T$ *in the robot's local coordinate frame. Therefore, if the robot's coordinate frame is located at* $[x, y]^T$ *with orientation* θ*, the forward kinematic map for vertex a (i.e., the mapping from* $q = [x, y, \theta]^T$ *to the global coordinates of the vertex a) is given by*

$$\phi(q) = \begin{bmatrix} x + a_x \cos\theta - a_y \sin\theta \\ y + a_x \sin\theta + a_y \cos\theta \end{bmatrix}. \tag{4.20}$$

The corresponding Jacobian matrix is given by

$$J(q) = \frac{\partial\phi}{\partial q}(q) = \begin{bmatrix} 1 & 0 & -a_x \sin\theta - a_y \cos\theta \\ 0 & 1 & a_x \cos\theta - a_y \sin\theta \end{bmatrix}. \tag{4.21}$$

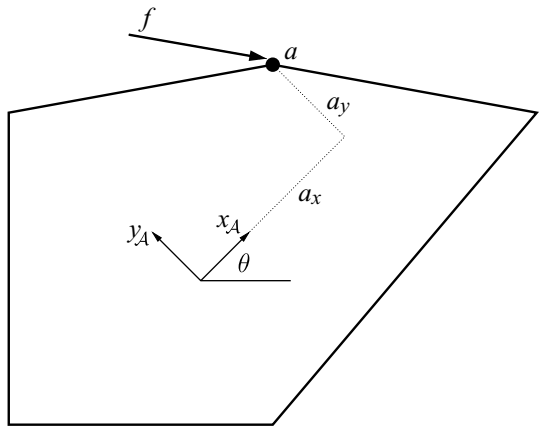


Figure 4.19 The robot $\mathcal{A}$, with coordinate frame oriented at angle θ from the world frame, and vertex a with local coordinates (a_x, a_y).

Therefore, the configuration space force is given by

$$\begin{bmatrix} u_x \\ u_y \\ u_\theta \end{bmatrix} = J^T(q) \begin{bmatrix} f_x \\ f_y \end{bmatrix} \tag{4.22}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -a_x \sin\theta - a_y \cos\theta & a_x \cos\theta - a_y \sin\theta \end{bmatrix} \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

$$= \begin{bmatrix} f_x \\ f_y \\ -f_x(a_x \sin\theta + a_y \cos\theta) + f_y(a_x \cos\theta - a_y \sin\theta) \end{bmatrix} \tag{4.23}$$

and u_θ corresponds to the torque exerted about the origin of the robot frame. Our result for u_θ can be verified by the familiar torque equation $\tau = r \times f$, where r is the vector from the robot's origin to the point of application of f, and $\tau = u_\theta$.

4.7.2 Potential Functions for Rigid-Body Robots

As before, our goal in defining potential functions is to construct a potential function that repels the robot from obstacles, with a global minimum that corresponds to q_{goal}. In the configuration space, this task was conceptually simple because the robot was represented by a single point, which we treated as a point mass under the influence of

a potential field. In the workspace, things are not so simple; the robot has finite area in the plane and volume in three dimensions. Evaluating the effect of a potential field on the robot would involve computing an integral over the area/volume defined by the robot, and this can be quite complex (both mathematically and computationally). An alternative approach is to select a subset of points on the robot, called control points, and to define a workspace potential for each of these points. Evaluating the effect of the potential field on a single point is no different from the evaluations required in section 4.1. We then use the relationship established in the previous subsection to convert the individual workspace forces to configuration space forces. We then add them to get the total configuration space force. As a result, we have approximately "lifted" the total workspace forces on the robot to a generalized force in the configuration space.

We need to pick control points $\{r_i\}$ on the robot. The minimum number of control points depends upon the number of degrees of freedom of the robot. It is the number of points required to "pin down" the robot. For example, for a rigid-body robot in the plane, we can fix the position of the robot by fixing the position of two of its points. For each r_j, the attractive potential is

$$U_{\text{att},j}(q) = \begin{cases} \frac{1}{2}\zeta_i d^2(r_j(q), r_j(q_{\text{goal}})), & d(r_j(q), r_j(q_{\text{goal}})) \le d^*_{\text{goal}} \\ d^*_{\text{goal}}\zeta_j d(r_j(q), r_j(q_{\text{goal}})) - \frac{1}{2}\zeta_j d^{*2}_{\text{goal}}, & d(r_j(q), r_j(q_{\text{goal}})) > d^*_{\text{goal}}. \end{cases}$$

With this potential function, the workspace force for attractive control point r_i is defined by

$$\nabla U_{\text{att},j}(q) = \begin{cases} \zeta_i(r_j(q) - r_j(q_{\text{goal}})), & d(r_j(q), r_j(q_{\text{goal}})) \le d^*_{\text{goal}}, \\ \frac{d^*_{\text{goal}}\zeta_j(r_j(q) - r_j(q_{\text{goal}}))}{d(r_j(q), r_j(q_{\text{goal}}))}, & d(r_j(q), r_j(q_{\text{goal}})) > d^*_{\text{goal}}. \end{cases}$$

For the workspace repulsive potential fields, we use the same control points $\{r_j\}$, and define the repulsive potential for r_j as

$$U_{\text{rep}i,j}(q) = \begin{cases} \frac{1}{2}\eta_j \left(\frac{1}{d_i(r_j(q))} - \frac{1}{Q^*_i}\right)^2, & d_i(r_j(q)) \le Q^*_i, \\ 0, & d_i(r_j(q)) > Q^*_i, \end{cases} \tag{4.24}$$

where $d_i(r_j(q))$ is the shortest distance between the control point r_j and obstacle $\mathcal{WO}_i$, and Q^*_i is the workspace distance of influence for obstacles. The gradient of each $U_{\text{rep}i,j}$ corresponds to a workspace force,

$$\nabla U_{\text{rep}i,j}(q) = \begin{cases} \eta_j \left(\frac{1}{Q^*_i} - \frac{1}{d_i(r_j(q))}\right) \frac{1}{d_i^2(r_j(q))} \nabla d_i(r_j(q)), & d_i(r_j(q)) \le Q^*_i, \\ 0, & d_i(r_j(q)) > Q^*_i. \end{cases} \tag{4.25}$$

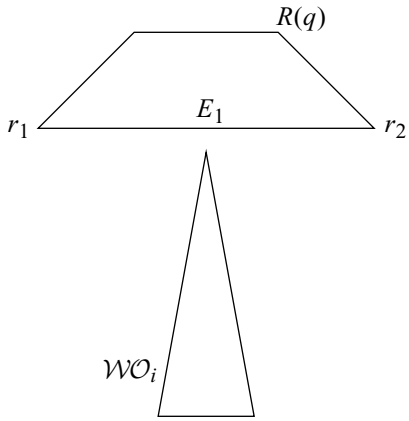


Figure 4.20 The repulsive forces exerted on the robot vertices r_1 and r_2 may not be sufficient to prevent a collision between edge E_1 and the obstacle.

Often the vertices of the robot are used as the repulsive control points, but it is important to note that this selection of repulsive control points does not guarantee that the robot cannot collide with an obstacle. Figure 4.20 shows an example where this is the case. In this figure, the repulsive control points r_1 and r_2 are very far from the obstacle $\mathcal{WO}_i$, and therefore the repulsive influence may not be great enough to prevent the robot edge E_1 from colliding with the obstacle. For this reason, we could add a *floating* repulsive control point, r_{float}. The floating control point is defined as that point on the boundary of the robot that is closest to any workspace obstacle. Obviously the choice of r_{float} depends on the configuration q. For the example shown in figure 4.20, r_{float} would be located at the center of edge E_1, thus repelling the robot from the obstacle. The repulsive force acting on r_{float} is defined in the same way as for the other control points, using (4.25).

The total configuration space force acting on the robot is the sum of the configuration space forces that result from all attractive and repulsive control points, i.e.,

$$\begin{aligned} u(q) &= \sum_j u_{\text{att}\,j} + \sum_{ij} u_{\text{rep}ij} \\ &= \sum_j J_j^T(q)\nabla u_{\text{att}ij}(q) + \sum_i \sum_j J_j^T(q)\nabla u_{\text{rep}ij} \end{aligned} \tag{4.26}$$

in which $J_j(q)$ is the Jacobian matrix for control point r_j. It is essential that the addition of forces be done in the configuration space and *not* in the workspace.

Path-Planning Algorithm

Having defined a configuration space force, which we will again treat as a velocity, we can use the same gradient descent method for this case as in section 4.1. As before, there are a number of design choices that must be made.

ζ_j controls the relative influence of the attractive potential for control point r_j. It is not necessary that all of the ζ_i be set to the same value. We might choose to weight one of the control points more heavily than the others, producing a "follow the leader" type of motion, in which the leader control point is quickly attracted to its final position, and the robot then reorients itself so that the other attractive control points reach their final positions.

η_j controls the relative influence of the repulsive potential for control point r_j. As with the ζ_i it is not necessary that all of the η_j be set to the same value.

Q_i^* We can define a distinct Q_i^* 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 Q_i^*'s to the obstacles to avoid the possibility of overlapping regions of influence for distinct obstacles.

4.7.3 Path Planning for Articulated Bodies

It is straightforward to extend the methods of the previous subsection to the case of articulated manipulators. Attractive control points are defined on the end effector and repulsive control points are placed on the links. It may be a good idea to use at least one floating control point for each link of the robot, since each link is a rigid body and we would like to prevent the links from colliding with obstacles. For each control point, a Jacobian matrix is computed (see chapter 3, section 3.8). These Jacobians map workspace forces to generalized configuration space forces (joint torques for revolute joints, joint forces for prismatic joints). With these exceptions, the formalism of section 4.7.2 can be directly applied to the path-planning problem for articulated arms (of course the implementation may be a bit more difficult, since the Jacobians may be a bit more difficult to construct, and computing distances to polyhedrons in three dimensions is a bit more involved than computing distances to polygons in the plane). Naturally, this method will be plagued with local minima.

Problems

1. Prove that $d_i(x)$ is a local minimum of $\rho(x, s)$ with respect to s. Show that $D(x)$ can be defined in terms of $d_i(x)$.
2. Does the wave-front planner in a discrete grid yield the shortest distance? (If so, in what metric?)
3. Write a program that determines a path between two points in a planar grid using the wave-front planner. Input from a file a set of obstacles in a known workspace. This file should be a list of vertices and the program should automatically convert the polygonal representation to a binary grid of pixels. Input from the keyboard a start and goal location and write a program to display a meaningful output that a path is indeed determined. Use either four-point or eight-point connectivity.
4. Write a program that determines a path for a planar convex robot that can orient from a start to a final configuration using the wave-front planner. Input from a file a robot and from a separate file a set of obstacles in a known workspace. Input a start and goal configuration from the keyboard. Hand in meaningful output where the robot *must* orient to get from start to goal. Hand in meaningful output where a path is not found.
5. The two-link manipulator in figure 4.21 has no joint limits. Use the wavefront planner to draw the shortest path between the start and goal configurations.
6. Implement, either in simulation or on a mobile robot, a sensor-based attractive/repulsive potential function.

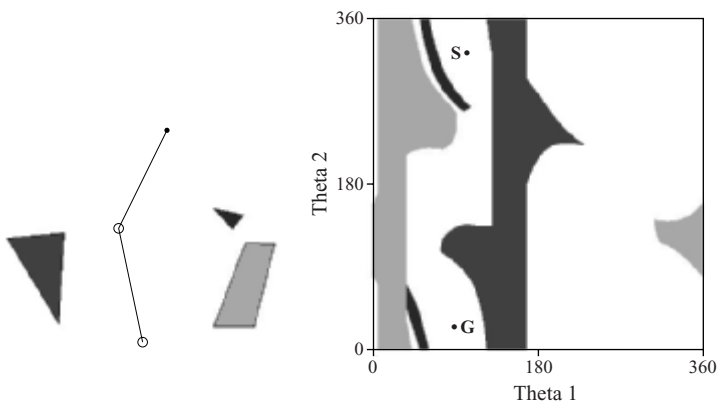


Figure 4.21 (Left) The initial configuration of a two-link manipulator in a polygonal workspace. (Right) The configuration space of the two-link manipulator with a start and goal configuration labeled S and G respectively.

7. Implement the attractive/repulsive potential function for a point robot in a configuration space with the following obstacles
   (a) polygons
   (b) polygons and circles
   (c) polyhedrons
   (d) polyhedrons and spheres
   (e) polyhedrons, spheres, and cylinders.
8. Adapt the attractive/repulsive potential function method to handle moving obstacles.
9. Explain why the paths resulting from the Bug2 algorithm and the navigation potential function look similar.