

# 2

## *Bug Algorithms*

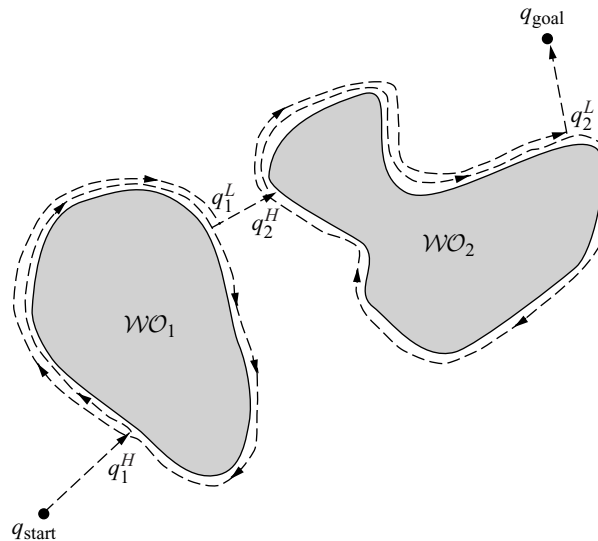
EVEN A simple planner can present interesting and difficult issues. The Bug1 and Bug2 algorithms [301] are among the earliest and simplest sensor-based planners with provable guarantees. These algorithms assume the robot is a point operating in the plane with a contact sensor or a zero range sensor to detect obstacles. When the robot has a finite range (nonzero range) sensor, then the Tangent Bug algorithm [217] is a Bug derivative that can use that sensor information to find shorter paths to the goal. The Bug and Bug-like algorithms are straightforward to implement; moreover, a simple analysis shows that their success is guaranteed, when possible. These algorithms require two behaviors: move on a straight line and follow a boundary. To handle boundary-following, we introduce a curve-tracing technique based on the implicit function theorem at the end of this chapter. This technique is general to following any path, but we focus on following a boundary at a fixed distance.

### **2.1 Bug1 and Bug2**

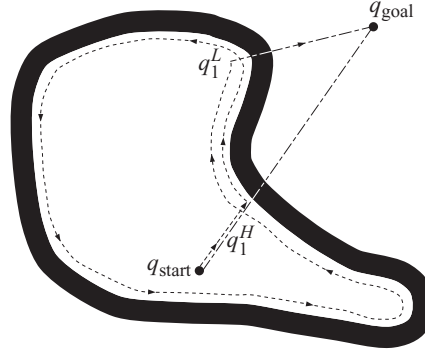
Perhaps the most straight forward path planning approach is to move toward the goal, unless an obstacle is encountered, in which case, circumnavigate the obstacle until motion toward the goal is once again allowable. Essentially, the Bug1 algorithm formalizes the “common sense” idea of moving toward the goal and going around obstacles. The robot is assumed to be a point with perfect positioning (no positioning error) with a contact sensor that can detect an obstacle boundary if the point robot “touches” it. The robot can also measure the distance  $d(x, y)$  between any two points  $x$

and  $y$ . Finally, assume that the workspace is *bounded*. Let  $B_r(x)$  denote a ball of radius  $r$  centered on  $x$ , i.e.,  $B_r(x) = \{y \in \mathbb{R}^2 \mid d(x, y) < r\}$ . The fact that the workspace is bounded implies that for all  $x \in \mathcal{W}$ , there exists an  $r < \infty$  such that  $\mathcal{W} \subset B_r(x)$ .

The start and goal are labeled  $q_{\text{start}}$  and  $q_{\text{goal}}$ , respectively. Let  $q_0^L = q_{\text{start}}$  and the  $m$ -line be the line segment that connects  $q_i^L$  to  $q_{\text{goal}}$ . Initially,  $i = 0$ . The Bug1 algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves along the  $m$ -line toward  $q_{\text{goal}}$  until it either encounters the goal or an obstacle. If the robot encounters an obstacle, let  $q_1^H$  be the point where the robot first encounters an obstacle and call this point a *hit point*. The robot then circumnavigates the obstacle until it returns to  $q_1^H$ . Then, the robot determines the closest point to the goal on the perimeter of the obstacle and traverses to this point. This point is called a *leave point* and is labeled  $q_1^L$ . From  $q_1^L$ , the robot heads straight toward the goal again, i.e., it reinvokes the motion-to-goal behavior. If the line that connects  $q_1^L$  and the goal intersects the current obstacle, then there is no path to the goal; note that this intersection would occur immediately “after” leaving  $q_1^L$ . Otherwise, the index  $i$  is incremented and this procedure is then repeated for  $q_i^L$  and  $q_i^H$  until the goal is reached or the planner determines that the robot cannot reach the goal (figures 2.1, 2.2). Finally, if the line to the goal “grazes” an obstacle, the robot need not invoke a boundary following behavior, but rather continues onward toward the goal. See algorithm 1 for a description of the Bug1 approach.



**Figure 2.1** The Bug1 algorithm successfully finds the goal.



**Figure 2.2** The Bug1 algorithm reports the goal is unreachable.

---

**Algorithm 1** Bug1 Algorithm

---

**Input:** A point robot with a tactile sensor

**Output:** A path to the  $q_{\text{goal}}$  or a conclusion no such path exists

---

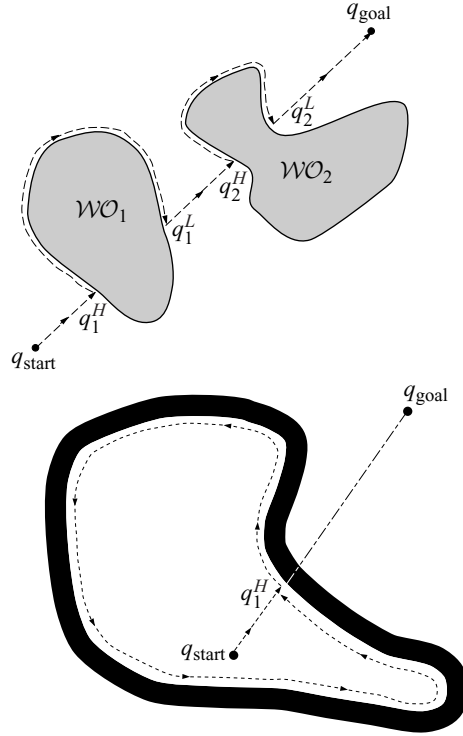
```

1: while Forever do
2:   repeat
3:     From  $q_{i-1}^L$ , move toward  $q_{\text{goal}}$ .
4:   until  $q_{\text{goal}}$  is reached or an obstacle is encountered at  $q_i^H$ .
5:   if Goal is reached then
6:     Exit.
7:   end if
8:   repeat
9:     Follow the obstacle boundary.
10:  until  $q_{\text{goal}}$  is reached or  $q_i^H$  is re-encountered.
11:  Determine the point  $q_i^L$  on the perimeter that has the shortest distance to the goal.
12:  Go to  $q_i^L$ .
13:  if the robot were to move toward the goal then
14:    Conclude  $q_{\text{goal}}$  is not reachable and exit.
15:  end if
16: end while

```

---

Like its Bug1 sibling, the Bug2 algorithm exhibits two behaviors: motion-to-goal and boundary-following. During motion-to-goal, the robot moves toward the goal on the  $m$ -line; however, in Bug2 the  $m$ -line connects  $q_{\text{start}}$  and  $q_{\text{goal}}$ , and thus remains fixed. The boundary-following behavior is invoked if the robot encounters an obstacle,



**Figure 2.3** (Top) The Bug2 algorithm finds a path to the goal. (Bottom) The Bug2 algorithm reports failure.

but this behavior is different from that of Bug1. For Bug2, the robot circumnavigates the obstacle until it reaches a new point on the  $m$ -line closer to the goal than the initial point of contact with the obstacle. At this time, the robot proceeds toward the goal, repeating this process if it encounters an object. If the robot re-encounters the original departure point from the  $m$ -line, then there is no path to the goal (figures 2.3, 2.4). Let  $x \in \mathcal{W}_{\text{free}} \subset \mathbb{R}^2$  be the current position of the robot,  $i = 1$ , and  $q_0^L$  be the start location. See algorithm 2 for a description of the Bug2 approach.

At first glance, it seems that Bug2 is a more effective algorithm than Bug1 because the robot does not have to entirely circumnavigate the obstacles; however, this is not always the case. This can be seen by comparing the lengths of the paths found by the two algorithms. For Bug1, when the  $i$ th obstacle is encountered, the robot completely circumnavigates the boundary, and then returns to the leave point. In the worst case, the robot must traverse half the perimeter,  $p_i$ , of the obstacle to reach this leave point.

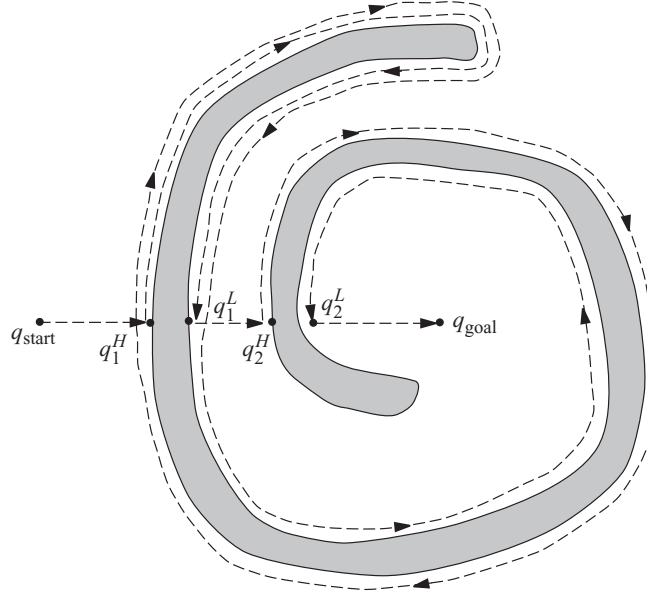


Figure 2.4 Bug2 Algorithm.

Moreover, in the worst case, the robot encounters all  $n$  obstacles. If there are no obstacles, the robot must traverse a distance of length  $d(q_{\text{start}}, q_{\text{goal}})$ . Thus, we obtain

$$(2.1) \quad L_{\text{Bug1}} \leq d(q_{\text{start}}, q_{\text{goal}}) + 1.5 \sum_{i=1}^n p_i.$$

For Bug2, the path length is a bit more complicated. Suppose that the line through  $q_{\text{start}}$  and  $q_{\text{goal}}$  intersects the  $i$ th obstacle  $n_i$  times. Then, there are at most  $n_i$  leave points for this obstacle, since the robot may only leave the obstacle when it returns to a point on this line. It is easy to see that half of these intersection points are not valid leave points because they lie on the “wrong side” of the obstacle, i.e., moving toward the goal would cause a collision. In the worst case, the robot will traverse nearly the entire perimeter of the obstacle for each leave point. Thus, we obtain

$$(2.2) \quad L_{\text{Bug2}} \leq d(q_{\text{start}}, q_{\text{goal}}) + \frac{1}{2} \sum_{i=1}^n n_i p_i.$$

Naturally, (2.2) is an upper-bound because the summation is over all of the obstacles as opposed to over the set of obstacles that are encountered by the robot.

---

**Algorithm 2** Bug2 Algorithm
 

---

**Input:** A point robot with a tactile sensor

**Output:** A path to  $q_{\text{goal}}$  or a conclusion no such path exists
 

---

```

1: while True do
2:   repeat
3:     From  $q_{i-1}^L$ , move toward  $q_{\text{goal}}$  along  $m$ -line.
4:   until
      $q_{\text{goal}}$  is reached or
     an obstacle is encountered at hit point  $q_i^H$ .
5:   Turn left (or right).
6:   repeat
7:     Follow boundary
8:   until
      $q_{\text{goal}}$  is reached or
      $q_i^H$  is re-encountered or
      $m$ -line is re-encountered at a point  $m$  such that
      $m \neq q_i^H$  (robot did not reach the hit point),
      $d(m, q_{\text{goal}}) < d(m, q_i^H)$  (robot is closer), and
     If robot moves toward goal, it would not hit the obstacle
9:   Let  $q_{i+1}^L = m$ 
10:  Increment  $i$ 
17: end while

```

---

A casual examination of (2.1) and (2.2) shows that  $L_{\text{Bug2}}$  can be arbitrarily longer than  $L_{\text{Bug1}}$ . This can be achieved by constructing an obstacle whose boundary has many intersections with the  $m$ -line. Thus, as the “complexity” of the obstacle increases, it becomes increasingly likely that Bug1 could outperform Bug2 (figure 2.4).

In fact, Bug1 and Bug2 illustrate two basic approaches to search problems. For each obstacle that it encounters, Bug1 performs an *exhaustive search* to find the optimal leave point. This requires that Bug1 traverse the entire perimeter of the obstacle, but having done so, it is certain to have found the optimal leave point. In contrast, Bug2 uses an *opportunistic* approach. When Bug2 finds a leave point that is better than any it has seen before, it commits to that leave point. Such an algorithm is also called *greedy*, since it opts for the first promising option that is found. When the obstacles are simple, the greedy approach of Bug2 gives a quick payoff, but when the obstacles are complex, the more conservative approach of Bug1 often yields better performance.

## 2.2 Tangent Bug

*Tangent Bug* [216] serves as an improvement to the Bug2 algorithm in that it determines a shorter path to the goal using a range sensor with a 360 degree infinite orientation resolution. Sometimes orientation is called *azimuth*. We model this range sensor with the *raw distance function*  $\rho : \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$ . Consider a point robot situated at  $x \in \mathbb{R}^2$  with rays radially emanating from it. For each  $\theta \in S^1$ , the value  $\rho(x, \theta)$  is the distance to the closest obstacle along the ray from  $x$  at an angle  $\theta$ . More formally,

$$(2.3) \quad \rho(x, \theta) = \min_{\lambda \in [0, \infty]} d(x, x + \lambda[\cos \theta, \sin \theta]^T),$$

such that  $x + \lambda[\cos \theta, \sin \theta]^T \in \bigcup_i \mathcal{W}\mathcal{O}_i$ .

Note that there are infinitely many  $\theta \in S^1$  and hence the infinite resolution. This assumption is approximated with a finite number of range sensors situated along the circumference of a circular mobile robot which we have modeled as a point.

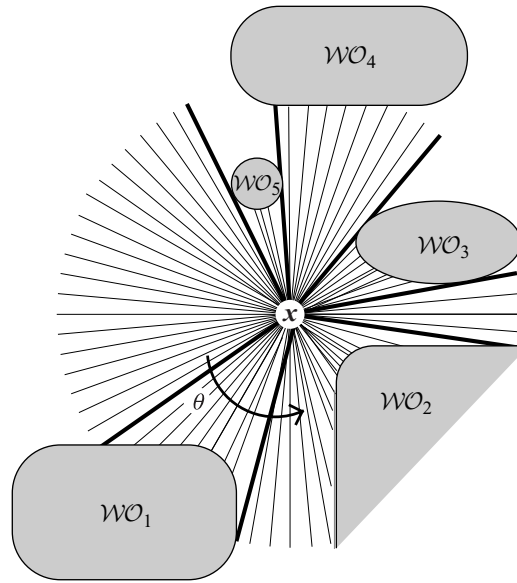
Since real sensors have limited range, we define the *saturated raw distance function*, denoted  $\rho_R : \mathbb{R}^2 \times S^1 \rightarrow \mathbb{R}$ , which takes on the same values as  $\rho$  when the obstacle is within sensing range, and has a value of infinity when the ray lengths are greater than the sensing range,  $R$ , meaning that the obstacles are outside the sensing range. More formally,

$$\rho_R(x, \theta) = \begin{cases} \rho(x, \theta), & \text{if } \rho(x, \theta) < R \\ \infty, & \text{otherwise.} \end{cases}$$

The Tangent Bug planner assumes that the robot can detect discontinuities in  $\rho_R$  as depicted in figure 2.5. For a fixed  $x \in \mathbb{R}^2$ , an *interval of continuity* is defined to be a connected set of points  $x + \rho(x, \theta)[\cos \theta, \sin \theta]^T$  on the boundary of the free space where  $\rho_R(x, \theta)$  is finite and continuous with respect to  $\theta$ .

The endpoints of these intervals occur where  $\rho_R(x, \theta)$  loses continuity, either as a result of one obstacle blocking another or the sensor reaching its range limit. The endpoints are denoted  $O_i$ . Figure 2.6 contains an example where  $\rho_R$  loses continuity. The points  $O_1, O_2, O_3, O_5, O_6, O_7$ , and  $O_8$  correspond to losses of continuity associated with obstacles blocking other portions of  $\mathcal{W}_{\text{free}}$ ; note the rays are tangent to the obstacles here. The point  $O_4$  is a discontinuity because the obstacle boundary falls out of range of the sensor. The sets of points on the boundary of the free space between  $O_1$  and  $O_2$ ,  $O_3$  and  $O_4$ ,  $O_5$  and  $O_6$ ,  $O_7$  and  $O_8$  are the intervals of continuity.

Just like the other Bugs, Tangent Bug iterates between two behaviors: motion-to-goal and boundary-following. However, these behaviors are different than in the Bug1 and Bug2 approaches. Although motion-to-goal directs the robot to the goal,



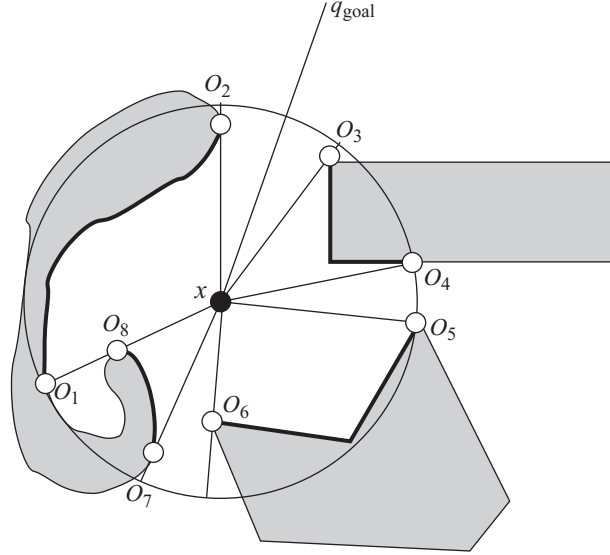
**Figure 2.5** The thin lines are values of the raw distance function,  $\rho_R(x, \theta)$ , for a fixed  $x \in \mathbb{R}^2$ , and the thick lines indicate discontinuities, which arise either because an obstacle occludes another or the sensing range is reached. Note that the segments terminating in free space represent infinitely long rays.

this behavior may have a phase where the robot follows the boundary. Likewise, the boundary-following behavior may have a phase where the robot does not follow the boundary.

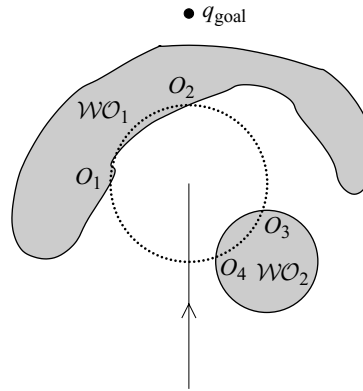
The robot initially invokes the motion-to-goal behavior, which itself has two parts. First, the robot moves in a straight line toward the goal until it senses an obstacle  $R$  units away and directly between it and the goal. This means that a line segment connecting the robot and goal must intersect an interval of continuity. For example, in figure 2.7,  $\mathcal{WO}_2$  is within sensing range, but does not block the goal, but  $\mathcal{WO}_1$  does. When the robot initially senses an obstacle, the circle of radius  $R$  becomes tangent to the obstacle. Immediately after, this tangent point splits into two  $O_i$ 's, which are the endpoints of the interval. If the obstacle is in front of the robot, then this interval intersects the segment connecting the robot and the goal.

The robot then moves toward the  $O_i$  that maximally decreases a heuristic distance to the goal. An example of a heuristic distance is the sum  $d(x, O_i) + d(O_i, q_{\text{goal}})$ . (The heuristic distance can be more complicated when factoring in available information

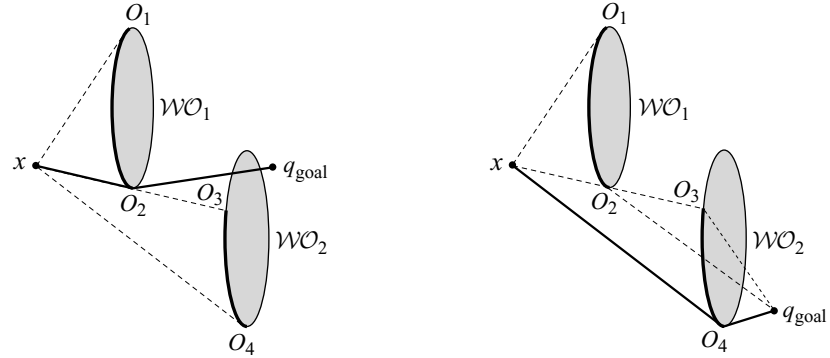




**Figure 2.6** The points of discontinuity of  $\rho_R(x, \theta)$  correspond to points  $O_i$  on the obstacles. The thick solid curves represent connected components of the range of  $\rho_R(x, \theta)$ , i.e., the intervals of continuity. In this example, the robot, to the best of its sensing range, believes there is a straight-line path to the goal.



**Figure 2.7** The vertical represents the path of the robot and the dotted circle its sensing range. Currently, the robot is located at the “top” of the line segment. The points  $O_i$  represent the points of discontinuity of the saturated raw distance function. Note that the robot passes by  $W O_2$ .



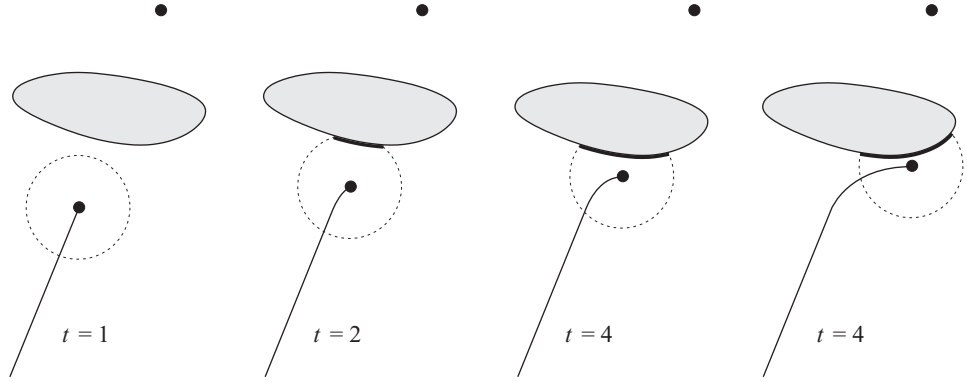
**Figure 2.8** (Left) The planner selects  $O_2$  as a subgoal for the robot. (Right) The planner selects  $O_4$  as a subgoal for the robot. Note the line segment between  $O_4$  and  $q_{\text{goal}}$  cuts through the obstacle.

with regard to the obstacles.) In figure 2.8 (left), the robot sees  $\mathcal{WC}_1$  and drives to  $O_2$  because  $i = 2$  minimizes  $d(x, O_i) + d(O_i, q_{\text{goal}})$ . When the robot is located at  $x$ , it cannot know that  $\mathcal{WC}_2$  blocks the path from  $O_2$  to the goal. In figure 2.8 (right), when the robot is located at  $x$  but the goal is different, it has enough sensor information to conclude that  $\mathcal{WC}_2$  indeed blocks a path from  $O_2$  to the goal, and therefore drives toward  $O_4$ . So, even though driving toward  $O_2$  may initially minimize  $d(x, O_i) + d(O_i, q_{\text{goal}})$  more than driving toward  $O_4$ , the planner effectively assigns an infinite cost to  $d(O_2, q_{\text{goal}})$  because it has enough information to conclude that any path through  $O_2$  will be suboptimal.

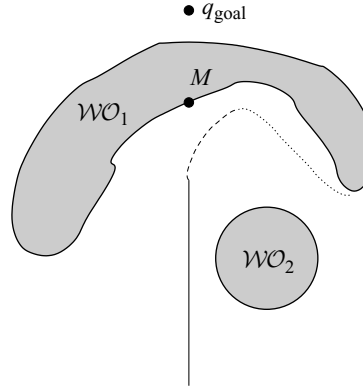
The set  $\{O_i\}$  is continuously updated as the robot moves toward a particular  $O_i$ , which can be seen in figure 2.9. When  $t = 1$ , the robot has not sensed the obstacle, hence the robot moves toward the goal. When  $t = 2$ , the robot initially senses the obstacle, depicted by a thick solid curve. The robot continues to move toward the goal, but off to the side of the obstacle heading toward the discontinuity in  $\rho$ . For  $t = 3$  and  $t = 4$ , the robot senses more of the obstacle and continues to decrease distance toward the goal while hugging the boundary.

The robot undergoes the motion-to-goal behavior until it can no longer decrease the heuristic distance to the goal. Put differently, it finds a point that is like a local minimum of  $d(\cdot, O_i) + d(O_i, q_{\text{goal}})$  restricted to the path that motion-to-goal dictates.

When the robot switches to boundary-following, it finds the point  $M$  on the sensed portion of the obstacle which has the shortest distance on the obstacle to the goal. Note that if the sensor range is zero, then  $M$  is the same as the hit point from the Bug1 and Bug2 algorithms. This sensed obstacle is also called the *followed obstacle*. We make a distinction between the followed obstacle and the *blocking obstacle*. Let  $x$  be



**Figure 2.9** Demonstration of motion-to-goal behavior for a robot with a finite sensor range moving toward a goal which is “above” the light gray obstacle.



**Figure 2.10** The workspace is the same as in figure 2.7. The solid and dashed segments represent the path generated by motion-to-goal and the dotted path represents the boundary-following path. Note that  $M$  is the “local minimum” point.

the current position of the robot. The blocking obstacle is the closest obstacle within sensor range that intersects the segment  $(1 - \lambda)x + \lambda q_{\text{goal}} \quad \forall \lambda \in [0, 1]$ . Initially, the blocking obstacle and the followed obstacle are the same.

Now the robot moves in the same direction as if it were in the motion-to-goal behavior. It continuously moves toward the  $O_i$  on the followed obstacle in the chosen direction (figure 2.10). While undergoing this motion, the planner also updates two values:  $d_{\text{followed}}$  and  $d_{\text{reach}}$ . The value  $d_{\text{followed}}$  is the shortest distance between the boundary which had been sensed and the goal. Let  $\Lambda$  be all of the points within

line of sight of  $x$  with range  $R$  that are on the followed obstacle  $\mathcal{WO}_f$ , i.e.,  $\Lambda = \{y \in \partial\mathcal{WO}_f : \lambda x + (1 - \lambda)y \in \mathcal{Q}_{\text{free}} \ \forall \lambda \in [0, 1]\}$ . The value  $d_{\text{reach}}$  is the distance between the goal and the closest point on the followed obstacle that is within line of sight of the robot, i.e.,

$$d_{\text{reach}} = \min_{c \in \Lambda} d(q_{\text{goal}}, c).$$

When  $d_{\text{reach}} < d_{\text{followed}}$ , the robot terminates the boundary-following behavior.

Let  $T$  be the point where a circle centered at  $x$  of radius  $R$  intersects the segment that connects  $x$  and  $q_{\text{goal}}$ . This is the point on the periphery of the sensing range that is closest to the goal when the robot is located at  $x$ . Starting with  $x = q_{\text{start}}$  and  $d_{\text{leave}} = d(q_{\text{start}}, q_{\text{goal}})$ , see algorithm 3.

---

**Algorithm 3** Tangent Bug Algorithm

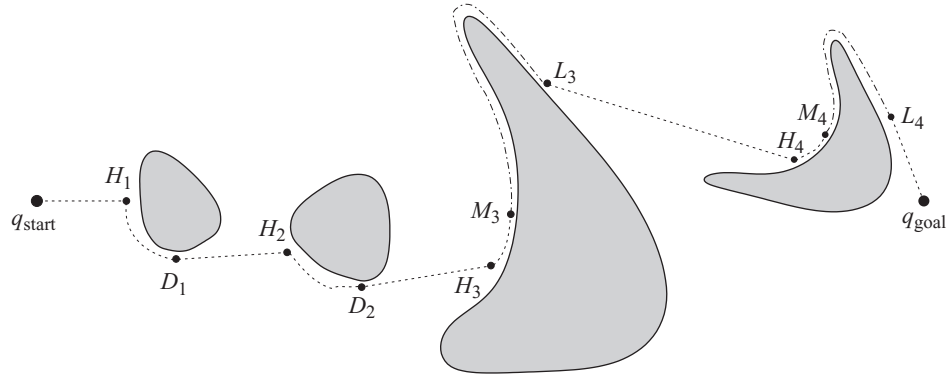
---

**Input:** A point robot with a range sensor

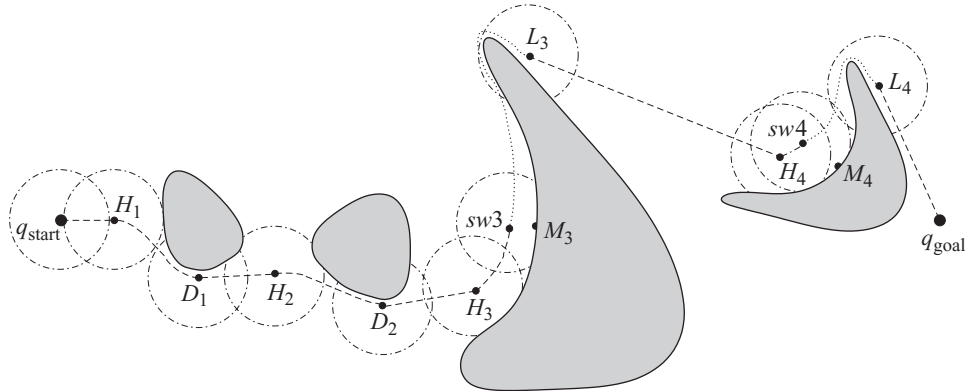
**Output:** A path to the  $q_{\text{goal}}$  or a conclusion no such path exists

---

- 1: **while** True **do**
  - 2:   **repeat**
  - 3:     Continuously move toward the point  $n \in \{T, O_i\}$  which minimizes  $d(x, n) + d(n, q_{\text{goal}})$
  - 4:   **until**
    - the goal is encountered **or**
    - The direction that minimizes  $d(x, n) + d(n, q_{\text{goal}})$  begins to increase  $d(x, q_{\text{goal}})$ , i.e., the robot detects a “local minimum” of  $d(\cdot, q_{\text{goal}})$ .
  - 5:   Chose a boundary following direction which continues in the same direction as the most recent motion-to-goal direction.
  - 6:   **repeat**
  - 7:     Continuously update  $d_{\text{reach}}$ ,  $d_{\text{followed}}$ , and  $\{O_i\}$ .
  - 8:     Continuously moves toward  $n \in \{O_i\}$  that is in the chosen boundary direction.
  - 9:   **until**
    - The goal is reached.
    - The robot completes a cycle around the obstacle in which case the goal cannot be achieved.
    - $d_{\text{reach}} < d_{\text{followed}}$
  - 10: **end while**
-

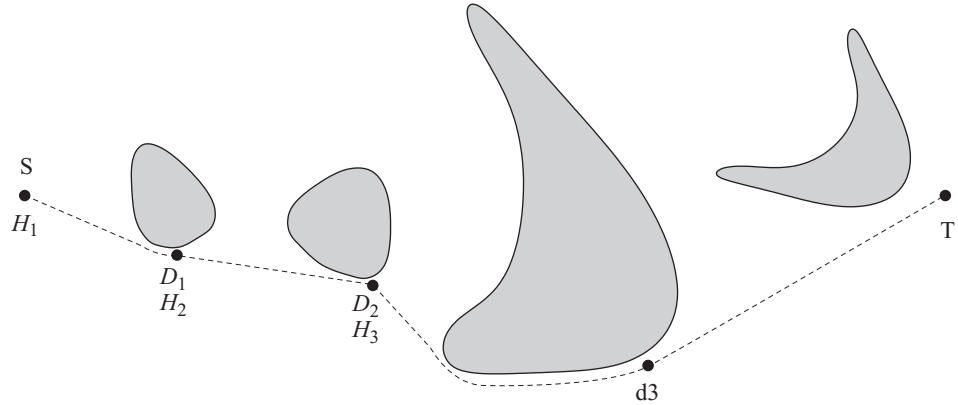


**Figure 2.11** The path generated by Tangent Bug with zero sensor range. The dashed lines correspond to the motion-to-goal behavior and the dotted lines correspond to boundary-following.



**Figure 2.12** Path generated by Tangent Bug with finite sensor range. The dashed lines correspond to the motion-to-goal behavior and the dotted lines correspond to boundary-following. The dashed-dotted circles correspond to the sensor range of the robot.

See figures 2.11, 2.12 for example runs. Figure 2.11 contains a path for a robot with zero sensor range. Here the robot invokes a motion-to-goal behavior until it encounters the first obstacle at hit point  $H_1$ . Unlike Bug1 and Bug2, encountering a hit point does not change the behavior mode for the robot. The robot continues with the motion-to-goal behavior by turning right and following the boundary of the first obstacle. The robot turned right because that direction minimized its heuristic distance to the goal. The robot departs this boundary at a *depart point*  $D_1$ . The robot



**Figure 2.13** Path generated by Tangent Bug with infinite sensor range. The dashed-lines correspond to the motion-to-goal behavior and there is no boundary-following.

continues with the motion-to-goal behavior, maneuvering around a second obstacle, until it encounters the third obstacle at  $H_3$ . The robot turns left and continues to invoke the motion-to-goal behavior until it reaches  $M_3$ , a minimum point. Now, the planner invokes the boundary-following behavior until the robot reaches  $L_3$ . Note that since we have zero sensing range,  $d_{\text{reach}}$  is the distance between the robot and the goal. The procedure continues until the robot reaches the goal. Only at  $M_i$  and  $L_i$  does the robot switch between behaviors. Figures 2.12, 2.13 contain examples where the robot has a finite and infinite sensing ranges, respectively.

## 2.3 Implementation

Essentially, the bug algorithms have two behaviors: drive toward a point and follow an obstacle. The first behavior is simply a form of gradient descent of  $d(\cdot, n)$  where  $n$  is either  $q_{\text{goal}}$  or an  $O_i$ . The second behavior, boundary-following, presents a challenge because the obstacle boundary is not known *a priori*. Therefore, the robot planner must rely on sensor information to determine the path. However, we must concede that the full path to the goal will not be determined from one sensor reading: the sensing range of the robot may be limited and the robot may not be able to “see” the entire world from one vantage point. So, the robot planner has to be incremental. We must determine first what information the robot requires and then where the robot should move to acquire more information. This is indeed the challenge of sensor-based planning. Ideally, we would like this approach to be reactive with sensory information

feeding into a simple algorithm that outputs translational and rotational velocity for the robot.

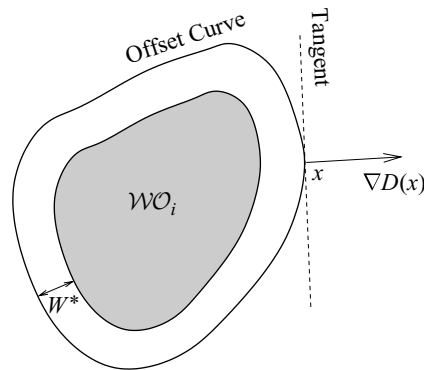
There are three questions: What information does the robot require to circumnavigate the obstacle? How does the robot infer this information from its sensor data? How does the robot use this information to determine (locally) a path?

### 2.3.1 What Information: The Tangent Line

If the obstacle were flat, such as a long wall in a corridor, then following the obstacle is trivial: simply move parallel to the obstacle. This is readily implemented using a sensing system that can determine the obstacle's surface normal  $n(x)$ , and hence a direction parallel to its surface. However, the world is not necessarily populated with flat obstacles; many have nonzero curvature. The robot can follow a path that is consistently orthogonal to the surface normal; this direction can be written as  $n(x)^\perp$  and the resulting path satisfies  $\dot{c}(t) = v$  where  $v$  is a basis vector in  $(n(c(t)))^\perp$ . The sign of  $v$  is based on the “previous” direction of  $\dot{c}$ .

Consistently determining the surface normal can be quite challenging and therefore for implementation, we can assume that obstacles are “locally flat.” This means the sensing system determines the surface normal, the robot moves orthogonal to this normal for a short distance, and then the process repeats. In a sense, the robot determines the sequence of short straight-line segments to follow, based on sensor information.

This flat line, loosely speaking, is the tangent (figure 2.14). It is a linear approximation of the curve at the point where the tangent intersects the curve. The tangent



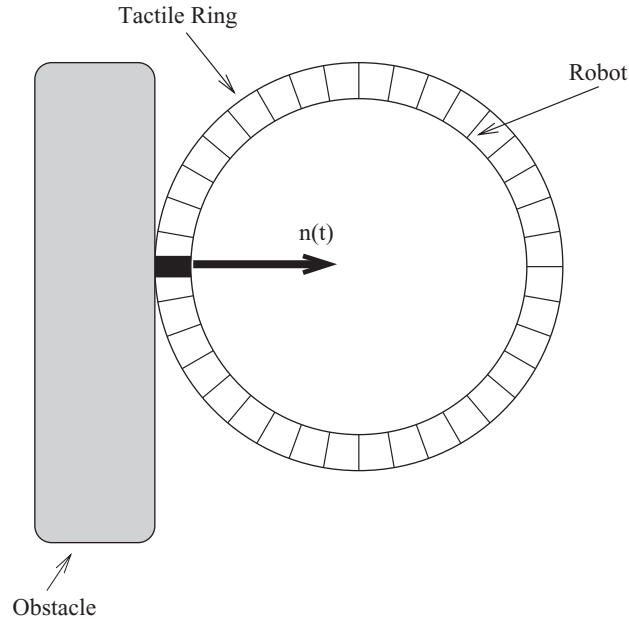
**Figure 2.14** The solid curve is the offset curve. The dashed line represents the tangent to the offset curve at  $x$ .

can also be viewed as a first-order approximation to the function that describes the curve. Let  $c : [0, 1] \rightarrow \mathcal{W}_{\text{free}}$  be the function that defines a path. Let  $x = c(s_0)$  for a  $s_0 \in [0, 1]$ . The tangent at  $x$  is  $\frac{dc}{ds} \big|_{s=s_0}$ . The tangent space can be viewed as a line whose basis vector is  $\frac{dc}{ds} \big|_{s=s_0}$ , i.e.,  $\left\{ \alpha \frac{dc}{ds} \big|_{s=s_0} \mid \alpha \in \mathbb{R} \right\}$ .

### 2.3.2 How to Infer Information with Sensors: Distance and Gradient

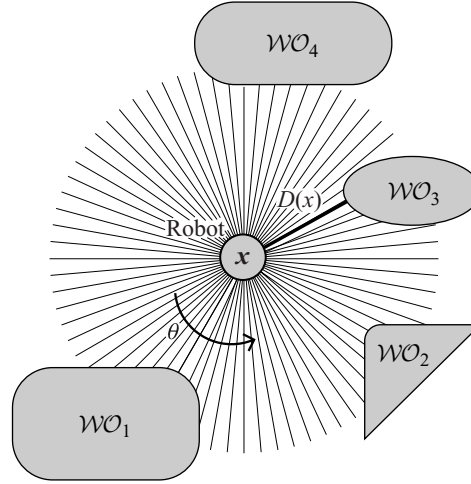
The next step is to infer the tangent from sensor data. Instead of thinking of the robot as a point in the plane, let's think of it as a circular base which has a fine array of tactile sensors radially distributed along its circumference (figure 2.15). When the robot contacts an obstacle, the direction from the contacted sensor to the robot's center approximates the surface normal. With this information, the robot can determine a sequence of tangents to follow the obstacle.

Unfortunately, using a tactile sensor to prescribe a path requires the robot to collide with obstacles, which endangers the obstacles and the robot. Instead, the robot should follow a path at a safe distance  $\mathcal{W}^* \in \mathbb{R}$  from the nearest obstacle. Such a path is called



**Figure 2.15** A fine-resolution tactile sensor.





**Figure 2.16** The global minimum of the rays determines the distance to the closest obstacle; the gradient points in a direction away from the obstacle along the ray.

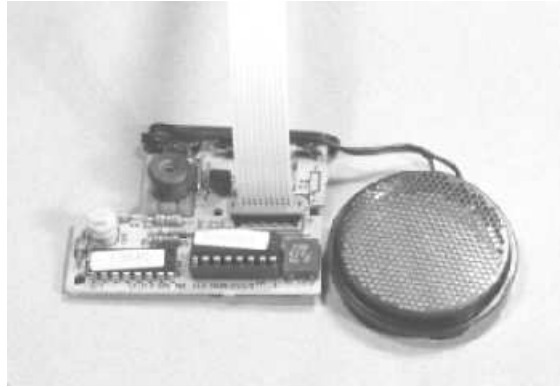
an *offset curve* [381]. Let  $D(x)$  be the distance from  $x$  to the closest obstacle, i.e.,

$$(2.4) \quad D(x) = \min_{c \in \bigcup_i \mathcal{WO}_i} d(x, c).$$

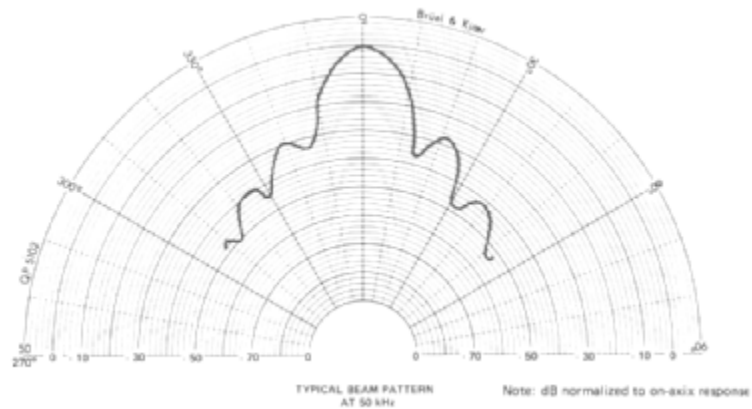
To measure this distance with a mobile robot equipped with an onboard range sensing ring, we use the raw distance function again. However, instead of looking for discontinuities, we look for the global minimum. In other words,  $D(x) = \min_s \rho(x, s)$  (figure 2.16).

We will need to use the gradient of distance. In general, the gradient is a vector that points in the direction that maximally increases the value of a function. See appendix C.5 for more details. Typically, the  $i$ th component of the gradient vector is the partial derivative of the function with respect to its  $i$ th coordinate. In the plane,  $\nabla D(x) = [\frac{\partial D(x)}{\partial x_1} \quad \frac{\partial D(x)}{\partial x_2}]^T$  which points in the direction that increases distance the most. Finally, the gradient is the unit direction associated with the smallest value of the raw distance function. Since the raw distance function seemingly approximates a sensing system with individual range sensing elements radially distributed around the perimeter of the robot, an algorithm defined in terms of  $D$  can often be implemented using realistic sensors.

There are many choices for range sensors; here, we investigate the use of ultrasonic sensors (figure 2.17), which are commonly found on mobile robots. Conventional ultrasonic sensors measure distance using time of flight. When the speed of sound

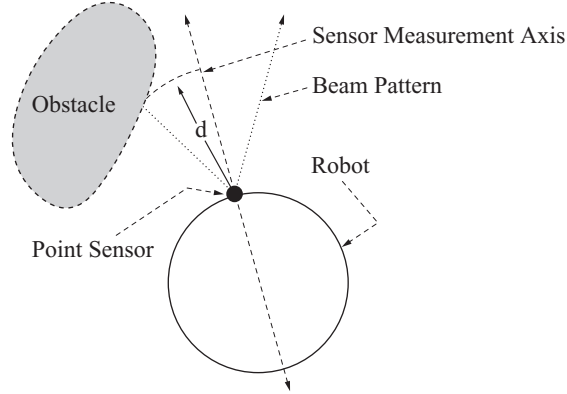


**Figure 2.17** The disk on the right is the standard Polaroid ultrasonic transducer found on many mobile robots; the circuitry on the left drives the transducer.



**Figure 2.18** Beam pattern for the Polaroid transducer.

in air is constant, the time that the ultrasound requires to leave the transducer, strike an obstacle, and return is proportional to the distance to the point of reflection on the obstacle [113]. This obstacle, however, can be located anywhere along the angular spread of the sonar sensor's beam pattern (figure 2.18). Therefore, the distance information that sonars provide is fairly accurate in depth, but not in azimuth. The beam pattern can be approximated with a cone (figure 2.19). For the commonly used Polaroid transducer, the arcbase is 22.5 degrees. When the reading of the sensor is  $d$ , the point of reflection can be anywhere along the arc base of length  $\frac{2\pi d 22.5}{360}$ .



**Figure 2.19** Centerline model.

Initially, assume that the echo originates from the center of the sonar cone. We acknowledge that this is a naive model, and we term this the *centerline model* (figure 2.19). The ultrasonic sensor with the smallest reading approximates the global minimum of the raw distance function, and hence  $D(x)$ . The direction that this sensor is facing approximates the negated gradient  $-\nabla D(x)$  because this sensor faces the closest obstacle. The tangent is then the line orthogonal to the direction associated with the smallest sensor reading.

### 2.3.3 How to Process Sensor Information: Continuation Methods

The tangent to the offset curve is  $(\nabla D(x))^\perp$ , the line orthogonal to  $\nabla D(x)$  (figure 2.14). The vector  $\nabla D(x)$  points in the direction that maximally increases distance; likewise, the vector  $-\nabla D(x)$  points in the direction that maximally decreases distance; they both point along the same line, but in opposite directions. Therefore, the vector  $(\nabla D(x))^\perp$  points in the direction that locally maintains distance; it is perpendicular to both  $\nabla D(x)$  and  $-\nabla D(x)$ . This would be the tangent of the offset curve which maintains distance to the nearby obstacle.

Another way to see why  $(\nabla D(x))^\perp$  is the tangent is to look at the definition of the offset curve. For a safety distance  $\mathcal{W}^*$ , we can define the offset curve implicitly as the set of points where  $G(x) = D(x) - \mathcal{W}^*$  maps to zero. The set of nonzero points (or vectors) that map to zero is called the *null space* of a map. For a curve implicitly defined by  $G$ , the tangent space at a point  $x$  is the null space of  $DG(x)$ , the Jacobian of  $G$  [410]. In general, the  $i, j$ th component of the Jacobian matrix is the partial derivative of the  $i$ th component function with respect to the  $j$ th coordinate and thus the Jacobian

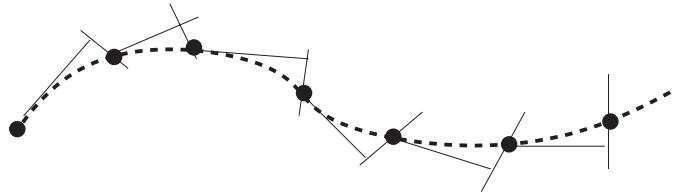
is a mapping between tangent spaces. Since in this case,  $G$  is a real-valued function ( $i = 1$ ), the Jacobian is just a row vector  $DD(x)$ . Here, we are reusing the symbol  $D$ . The reader is forced to use context to determine if  $D$  means distance or differential.

In Euclidean spaces, the  $i$ th component of a single-row Jacobian equals the  $i$ th component of the gradient and thus  $\nabla D(x) = (DD(x))^T$ . Therefore, since the tangent space is the null space of  $DD(x)$ , the tangent for boundary-following in the plane is the line orthogonal to  $\nabla D(x)$ , i.e.,  $(\nabla D(x))^\perp$ , and can be derived from sensor information.

Using distance information, the robot can determine the tangent direction to the offset curve. If the obstacles are flat, then the offset curve is also flat, and simply following the tangent is sufficient to follow the boundary of an unknown obstacle. Consider, instead, an obstacle with curvature. We can, however, assume that the obstacle is locally flat. The robot can then move along the tangent for a short distance, but since the obstacle has curvature, the robot will not follow the offset curve, i.e., it will “fall off” of the offset curve. To reaccess the offset curve, the robot moves either toward or away from the obstacle until it reaches the safety distance  $\mathcal{W}^*$ . In doing so, the robot is moving along a line defined by  $\nabla D(x)$ , which can be derived from sensor information.

Essentially, the robot is performing a numerical procedure of prediction and correction. The robot uses the tangent to locally predict the shape of the offset curve and then invokes a correction procedure once the tangent approximation is not valid. Note that the robot does not explicitly trace the path but instead “hovers” around it, resulting in a sampling of the path, not the path itself (figure 2.20).

A numerical tracing procedure can be posed as one which traces the roots of the expression  $G(x) = 0$ , where in this case  $G(x) = D(x) - \mathcal{W}^*$ . Numerical curve-tracing techniques rest on the *implicit function theorem* [9, 232, 307] which locally defines a curve that is implicitly defined by a map  $G : Y \times \mathbb{R} \rightarrow Y$ . Specifically, the roots of  $G$  locally define a curve parameterized by  $\lambda \in \mathbb{R}$ . See appendix D for a formal definition.



**Figure 2.20** The dashed line is the actual path, but the robot follows the thin black lines, predicting and correcting along the path. The black circles are samples along the path.

For boundary following at a safety distance  $\mathcal{W}^*$ , the function  $G(y, \lambda) = D(y, \lambda) - \mathcal{W}^*$  implicitly defines the offset curve. Note that the  $\lambda$ -coordinate corresponds to a tangent direction and the  $y$ -coordinates to the line or hyperplane orthogonal to the tangent. Let  $Y$  denote this hyperplane and  $D_Y G$  be the matrix formed by taking the derivative of  $G(x) = D(x) - \mathcal{W}^* = 0$  with respect to the  $y$ -coordinates. It takes the form  $D_Y G(x) = D_Y D(x)$  where  $D_Y$  denotes the differential with respect to the  $y$ -coordinates. If  $D_Y G(y, \lambda)$  is surjective at  $x = (\lambda, y)^T$ , then the implicit function theorem states that the roots of  $G(y, \lambda)$  locally define a curve that follows the boundary at a distance  $\mathcal{W}^*$  as  $\lambda$  is varied, i.e.,  $y(\lambda)$ .

By numerically tracing the roots of  $G$ , we can locally construct a path. While there are a number of curve-tracing techniques [232], let us consider an adaptation of a common predictor-corrector scheme. Assume that the robot is located at a point  $x$  which is a fixed distance  $\mathcal{W}^*$  away from the boundary. The robot takes a “small” step,  $\Delta\lambda$ , in the  $\lambda$ -direction (i.e., the tangent to the local path). In general, this *prediction step* takes the robot off the offset path. Next, a *correction method* is used to bring the robot back onto the offset path. If  $\Delta\lambda$  is small, then the local path will intersect a *correcting plane*, which is a plane orthogonal to the  $\lambda$ -direction at a distance  $\Delta\lambda$  away from the origin.

The correction step finds the location where the offset path intersects the correcting plane and is an application of the Newton convergence theorem [232]. See appendix D.2 for a more formal definition of this theorem. The Newton convergence theorem also requires that  $D_Y G(y, \lambda)$  be full rank at every  $(y, \lambda)$  in a neighborhood of the offset path. This is true because for  $G(x) = D(x) - \mathcal{W}^*$ ,  $[0 \ D_Y G(y, \lambda)]^T = DG(y, \lambda)$ . Since  $DG(y, \lambda)$  is full rank, so must be  $D_Y G(y, \lambda)$  on the offset curve. Since the set of nonsingular matrices is an open set, we know there is a neighborhood around each  $(y, \lambda)$  in the offset path where  $DG(y, \lambda)$  is full rank and hence we can use the iterative Newton method to implement the corrector step. If  $y^h$  and  $\lambda^h$  are the  $h$ th estimates of  $y$  and  $\lambda$ , the  $h + 1$ st iteration is defined as

$$(2.5) \quad y^{h+1} = y^h - (D_Y G)^{-1} G(y^h, \lambda^h),$$

where  $D_Y G$  is evaluated at  $(y^h, \lambda^h)$ . Note that since we are working in a Euclidean space, we can determine  $D_Y G$  solely from distance gradient, and hence, sensor information.

## Problems

1. Prove that  $D(x)$  is the global minimum of  $\rho(x, s)$  with respect to  $s$ .
2. What are the tradeoffs between the Bug1 and Bug2 algorithms?

3. Extend the Bug1 and Bug2 algorithms to a two-link manipulator.
4. What is the difference between the Tangent Bug algorithm with zero range detector and Bug2? Draw examples.
5. What are the differences between the path in figure 2.11 and the paths that Bug1 and Bug2 would have generated?
6. The Bug algorithms also assume the planner knows the location of the goal and the robot has perfect positioning. Redesign one of the Bug algorithms to relax the assumption of perfect positioning. Feel free to introduce a new type of “reasonable” sensor (not a high-resolution Global Positioning System).
7. In the Bug1 algorithm, prove or disprove that the robot does not encounter any obstacle that does not intersect the disk of radius  $d(q_{\text{start}}, q_{\text{goal}})$  centered at  $q_{\text{goal}}$ .
8. What assumptions do the Bug1, Bug2, and Tangent Bug algorithms make on robot localization, both in position and orientation?
9. Prove the completeness of the Tangent Bug algorithm.
10. Adapt the Tangent Bug algorithm so that it has a limited field of view sensor, i.e., it does not have a 360 degree field of view range sensor.
11. Write out  $D_Y G$  for boundary following in the planar case.
12. Let  $G_1(x) = D(x) + 1$  and let  $G_2(x) = D(x) + 2$ . Why are their Jacobians the same?
13. Let  $G(x, y) = y^3 + y - x^2$ . Write out a  $y$  as a function of  $x$  in an interval about the origin for the curve defined by  $G(x, y) = 0$ .