

Bug Algorithms

文档作者：李拥祺

注：

- 本文大部分内容来源于这本书：

Principles of Robot Motion: Theory, Algorithms, and Implementations[M]. MIT Press, 2005.

- 文中尽量给出引用的图片与资料的出处，但是难免有一些疏忽，如有侵权请及时告知。
 - 文章内容是作者本人在学习过程中的一些笔记，参考网上资料，加上了一些个人理解，由于水平有限，无法避免出现错误，欢迎大家讨论。
-

0. 概述

- Bug这个词有 虫 的含义，顾名思义，这个算法的总体思想就是采用类似昆虫的运动决策方法：**没遇到障碍物，我就走直线；遇到障碍物，我就沿着障碍物走。**
 - 优点：1) 无需全局地图和障碍物形状，只需要通过传感器获取局部信息；2) 计算简单；3) 具备完备性；
 - 缺点：1) 在比较复杂的环境中计算效率不高；
 - Bug算法大多数情况下被当作是一种避障算法，属于**reactive planner**，你可以把起始点到目标点的直线连接当作全局路径来理解；
 - 总的来说，只要是符合 没有障碍物走直线，碰到障碍物绕行 思想的这类算法，都可以归纳于 **Bug-like** 算法，如 **VisBug**，**DistBug**，**I-Bug**；
 - 本文只介绍 **Bug1**、**Bug2**、**Tangent Bug** 三种。
 - 如果大家嫌看书麻烦，可以直接参考资料3，这个是一个介绍 Bug 算法的课件。
-

1. Bug1

1.1 Robot

- 看成一个**位置准确**的点；
- 具有感知障碍物边界的能力；
- 具有测量任意两点距离的能力；
- 假设工作空间是有边界；

1.2 算法伪代码

Algorithm 1 Bug1 Algorithm**Input:** A point robot with a tactile sensor**Output:** A path to the q_{goal} or a conclusion no such path exists

```

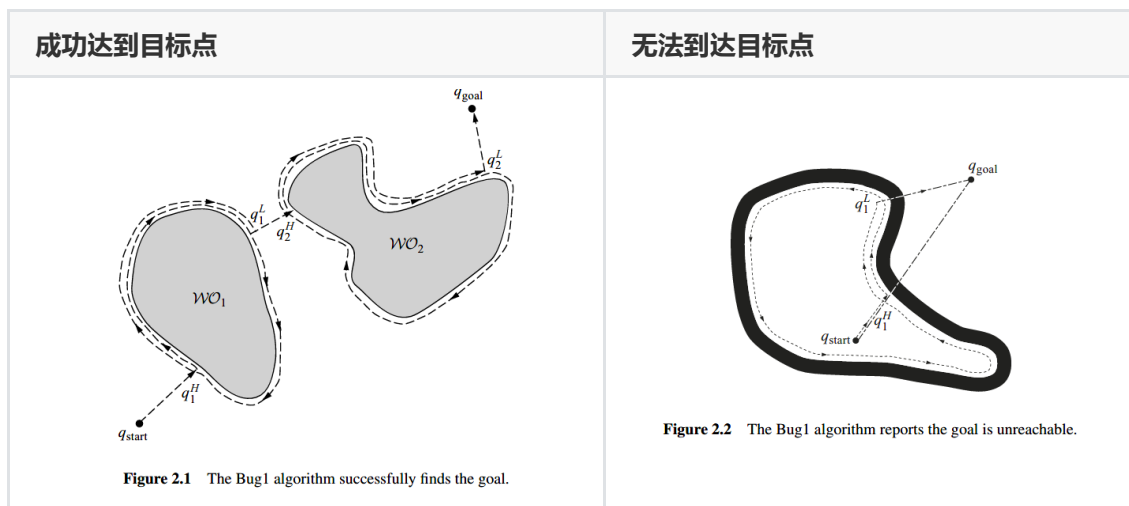
1: while Forever do
2:   repeat
3:     From  $q_{i-1}^L$ , move toward  $q_{goal}$ .
4:   until  $q_{goal}$  is reached or an obstacle is encountered at  $q_i^H$ .
5:   if Goal is reached then
6:     Exit.
7:   end if
8:   repeat
9:     Follow the obstacle boundary.
10:  until  $q_{goal}$  is reached 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  $q_{goal}$  is not reachable and exit.
15:  end if
16: end while

```

- 算法总体上来说比较容易理解：首先，规划一条从起点 q_{start} 到目标点 q_{goal} 的直线，并且沿着该直线前进直到检测到障碍物（这里将检测到障碍物的点或者碰到障碍物的点称为 **hit point**，记为 q_i^H ， i 为这种类型的点的编号顺序）；然后沿着障碍物环绕一圈，计算得到离目标点最近的点（称为 **leave point**，记为 q_i^L ），前进到该点，规划一条从这个点到 q_{goal} 的直线，重复前面的操作即可。
- 如何结束呢？
 - 到达目标点 q_{goal} ；
 - 发现无法到达目标点；
- 如何确认自己无法到达目标点？

判断从 q_i^L 到 q_{goal} 的直线是否与现在的障碍物相交

 - 书上给出的判断依据是：从 q_i^L 离开之后**立即马上**再次碰到/检测到现有障碍物；
 - 前后两次的 **leave points** 是否是同一个点；
 - 或者给碰到的障碍物做标识来判断检测到的是不是同一个障碍物。
- 放两张书上的图来直观感受一下



2. Bug2

2.1 Robot

同Bug1

2.2 算法伪代码

Algorithm 2 Bug2 Algorithm

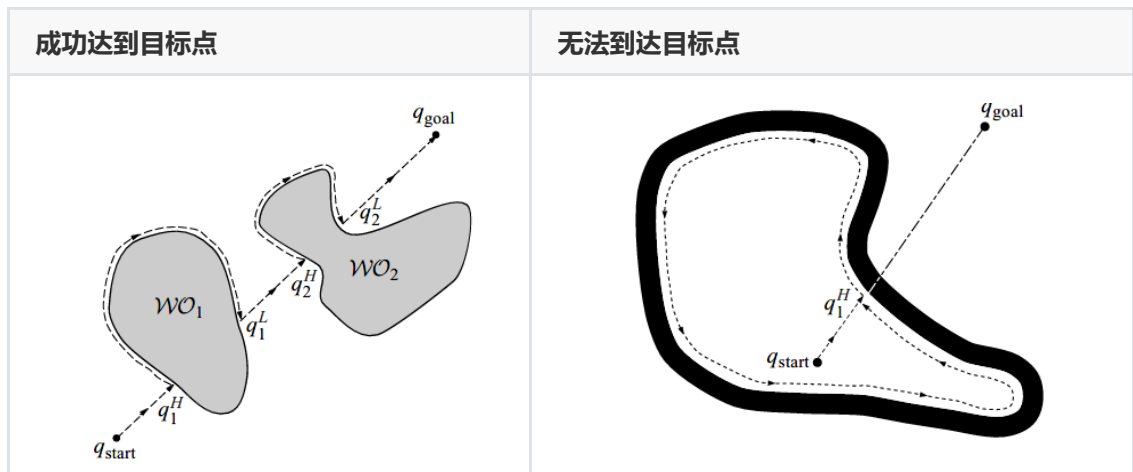
Input: A point robot with a tactile sensor

Output: A path to q_{goal} or a conclusion no such path exists

```

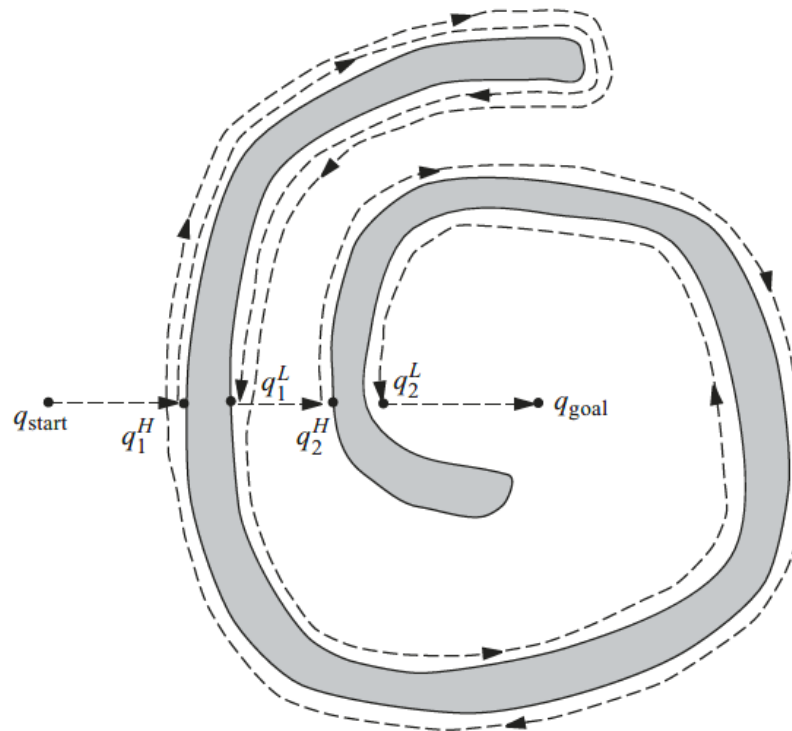
1: while True do
2:   repeat
3:     From  $q_{i-1}^L$ , move toward  $q_{goal}$  along  $m$ -line.
4:   until
      $q_{goal}$  is reached or
     an obstacle is encountered at hit point  $q_i^H$ .
5:   Turn left (or right).
6:   repeat
7:     Follow boundary
8:   until
      $q_{goal}$  is reached or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

- Bug2算法与Bug1算法不同的地方：
 - 同样会规划一条从起点 q_{start} 到目标点 q_{goal} 的直线，但是这条直线是固定不变的，即之后所有的 *hit points* 和 *leave points* 都会在这条直线上；
 - Bug2算法退出绕行障碍物的条件有三个（满足一个即可）：1) q_{goal} 点到达；2) q_i^H 点再遇（注意 i 值必须是当前值）；3) 碰到了一个新的 m 点（不等于 q_i^H ），且 $d(m, q_{goal}) < d(q_i^H, q_{goal})$ ，朝目标点前进不会碰到障碍物。
- 放两张书上的图来直观感受一下



2.5 Bug1 VS Bug2

- 两种搜索思路（基本上后续的搜索算法都是基于这两种思路）：
 - Bug1算法会详尽地搜索以找到**最优**的出发点；
 - Bug2算法会**贪婪**的将找到的第一个**比较优**的点作为出发点；
- 当障碍物很简单时，Bug2的贪婪方法会很快得到回报，但是当障碍物很复杂时，Bug2可能会因为贪婪而导致走很多**冤枉路**。
- Bug2算法的一个例子：

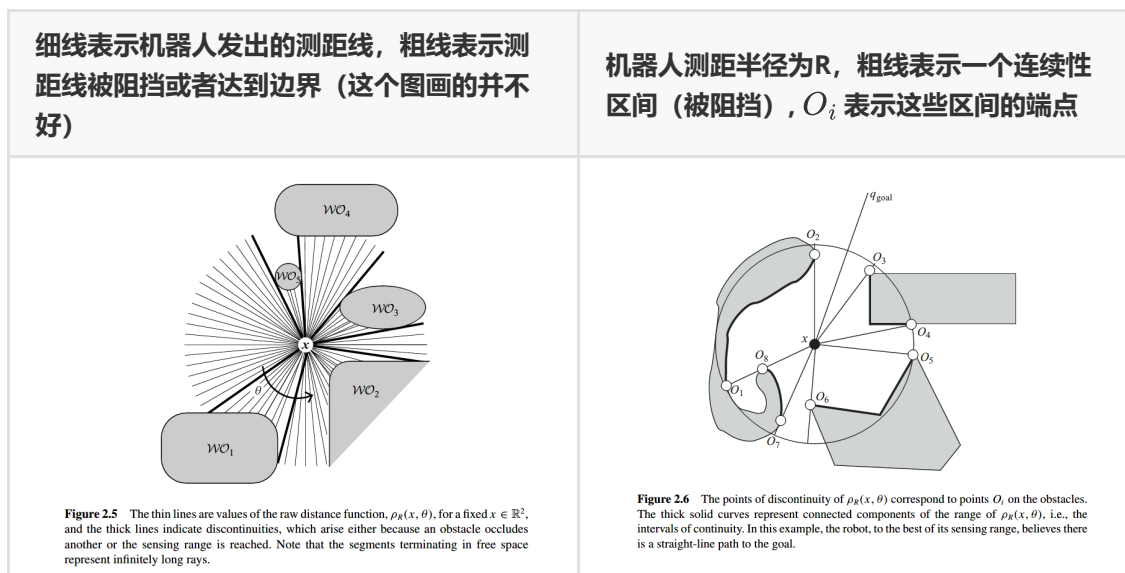


1. 机器人从 q_{start} 沿着直线 $L_{q_{start}q_{goal}}$ 前进碰到障碍物点 q_1^H 后开始绕行；
 2. 绕行过程中机器人与直线 $L_{q_{start}q_{goal}}$ 相交于点 m （判断如下：不是目标点；不是 q_1^H 点；比 q_1^H 点离目标更近，其继续前进没有障碍物），满足离开绕行条件，令 $q_1^L = m, i = 2$ ；机器人从 q_1^L 沿着直线 $L_{q_{start}q_{goal}}$ 继续前进；
 3. 前进过程中碰到障碍物点 q_2^H 后开始绕行与直线 $L_{q_{start}q_{goal}}$ 相交于点 $m = q_1^H$ ，（判断如下：不是目标点；不是 q_2^H 点；比 q_2^H 点离目标更远，其继续前进会碰到障碍物），不满足离开绕行条件，继续绕行；
 4. 绕行过程与直线 $L_{q_{start}q_{goal}}$ 相交于点 $m = q_1^L$ ，（判断如下：不是目标点；不是 q_2^H 点；比 q_2^H 点离目标更远），不满足离开绕行条件，继续绕行；
 5. 绕行过程中机器人与直线 $L_{q_{start}q_{goal}}$ 相交于点 m （判断如下：不是目标点；不是 q_2^H 点；比 q_2^H 点离目标更近，其继续前进没有障碍物），满足离开绕行条件，令 $q_2^L = m, i = 3$ ；机器人从 q_2^L 沿着直线 $L_{q_{start}q_{goal}}$ 继续前进；
 6. 达到目标点 q_{goal} 。
- 考虑最坏情况：（ p_i 表示障碍物 i 的周长， n 表示障碍物的数目）
 - Bug1算法: $L_{Bug1} \leq d(q_{start}, q_{goal}) + 1.5 \sum_{i=1}^n p_i$ （1.5是考虑最糟糕的情况：检测1圈，然后需要绕0.5圈到leave point）
 - Bug2算法: $L_{Bug2} \leq d(q_{start}, q_{goal}) + 0.5 \sum_{i=1}^n n_i p_i$ （ n_i 表示碰到障碍物 i 的次数）
 - 从式子对比可以发现，Bug2算法并不是一定比Bug1算法好，反而在某一个障碍物与 $m - lines$ 多次相交的情况下效率变得很差，如上面的例子所示。

3. Tangent Bug

3.1 Robot

- 机器人的探测能力加强，不仅仅是前面提到的接触传感器（测量半径很小，几乎为0）。
- 假设机器人具有360°无限分辨率的测距能力，同时可以检测不连续性。



- Tangent Bug算法利用这些端点来完成避障，使路径变得更短和平滑。

3.2 算法伪代码

Algorithm 3 Tangent Bug Algorithm

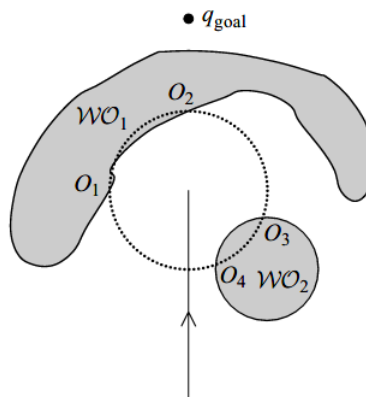
Input: A point robot with a range sensor

Output: A path to the q_{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_{reach} , d_{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**
-

Implementing Tangent Bug

- Basic problem: compute tangent to curve forming boundary of obstacle at any point, and drive the robot in that direction
- Let $D(x) = \min_c d(x, c) \quad c \in \cup_i WO_i$
- Let $G(x) = D(x) - W^* \leftarrow$ some safe following distance
- Note that $\nabla G(x)$ points radially away from the object
- Define $T(x) = (\nabla G(x))$ the tangent direction
 - in a real sensor (we'll talk about these) this is just the tangent to the array element with lowest reading
- We could just move in the direction $T(x)$
 - open-loop control
- Better is $\delta x = \mu (T(x) - \lambda (\nabla G(x)) G(x))$
 - closed-loop control (predictor-corrector)
- Tangent Bug 算法与Bug1/Bug2算法一样都包含两种运动模式：motion-to-goal 和 boundary-following。
 - 首先，机器人朝目标点 q_{goal} 直线前进，下图中的虚线圈表示机器人的检测范围，一开始检测到障碍物时是与障碍物相切于一点，随着运动的进行这个切点会变为一条线段的两个端点，如图中的 O_1 、 O_2 和 O_3 、 O_4 ，如果这两个端点之间的障碍物边界区间与机器人运动直线不相交则不会影响其运动，否则就将影响机器人的运动。



- 机器人无法继续向目标点直线前进，会考虑选择其中一个端点为子目标前进，选取的标准是最大程度地减少到达目标的启发式距离，启发式距离函数的选取可以根据可以获得的环境信息来确定，可简单可复杂，例如 $d(x, O_i) + d(O_i, q_{goal})$ 。

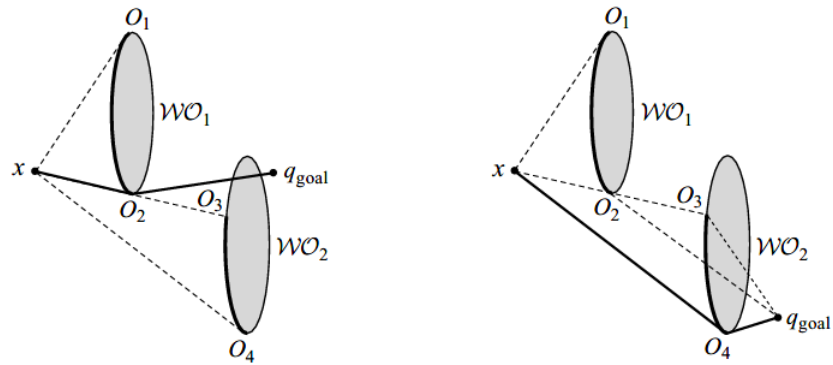


Figure 2.8 (Left) The planner selects O_2 as a subgoal for the robot. (Right) The planner selects O_4 as a subgoal for the robot. Note the line segment between O_4 and q_{goal} cuts through the obstacle.

- 上图中左右两边的图对比说明这种启发式距离函数是**短视**的，它只能得到当前状态下最优的选择，却并不一定是全局最优的选择。

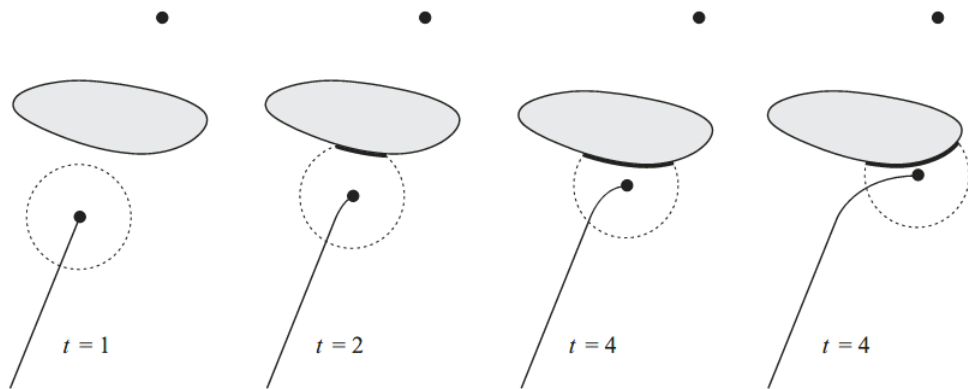
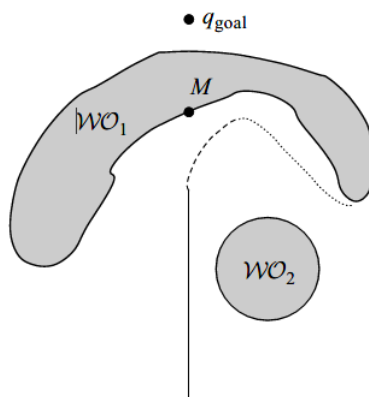
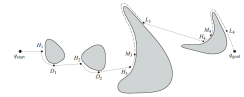


Figure 2.9 Demonstration of motion-to-goal behavior for a robot with a finite sensor range moving toward a goal which is “above” the light gray obstacle.

- 上图表面随着时间 t 的变化，机器人会朝着每一个时刻的子目标前进，直到可以直线前进到目标点或者启发式距离无法继续变小。
- 当启发式距离无法继续变小时，切换到绕行模式。如下图所示，点M是机器人找到的障碍物上离目标点最近的局部最小值点，机器人开始绕行，如果在绕行过程中发现了距离目标点更近的点，机器人将回到motion-to-goal模式。



- 机器人探测范围对算法的影响

说明	对比图
$q_{start} \rightarrow H_1 \rightarrow D_1 \rightarrow H_2 \rightarrow D_2 \rightarrow H_3 \rightarrow M_3$ 都是 motion-to-goal 模式; $M_3 \rightarrow L_3$ 是 boundary-following 模式; $L_3 \rightarrow H_4 \rightarrow M_4$ 是 motion-to-goal 模式; $M_4 \rightarrow L_4$ 是 boundary-following 模式; $L_4 \rightarrow q_{goal}$ 是 motion-to-goal 模式。	 <p>Figure 2.11 Path generated by Tangent Bug with zero sensor range. The dashed lines correspond to the motion-to-goal behavior and the solid lines correspond to boundary following.</p>
对比上图，机器人的探测范围不再局限于脚底下，路径整体趋势是一样的，只不过变得更加平滑，一些点的位置更加靠前了。	 <p>Figure 2.12 Path generated by Tangent Bug with finite sensor range. The dashed lines correspond to the motion-to-goal behavior and the solid lines correspond to boundary following. The dashed-dotted circles correspond to the sensor range of the robot.</p>
当探测为无限时，没有 boundary-following 模式的运动过程， 那为什么机器人不会直接直线运动到第三个障碍物呢？ 因为 Tangent 算法判断这个障碍物是否会对运动产生影响是通过机器人与目标点的连线与障碍物有没有相交。	 <p>Figure 2.13 Path generated by Tangent Bug with infinite sensor range. The dashed lines correspond to the motion-to-goal behavior and the solid lines correspond to boundary following.</p>

上面的这三幅图表明了机器人的探测能力对轨迹规划的影响，探测能力越强意味着获得的环境信息越多，继而越容易找到最优路径。

- 直观感受 Tangent Bug 可以参考资料 2。

4. 其他

- 一些基本的证明
 - 证明一个机器人从障碍物 leave point 离开一个障碍物之后不会再次碰到该障碍物？
 - 证明 Bug1, Bug2 等算法是完备的。
- Bug-like 算法主要就是两个动作：朝着一个目标点前进和绕行障碍物
 - 朝着一个目标点前进可以理解为 $d(\cdot, n)$ 的梯度下降形式，其中 $d(\cdot, n)$ 表示机器人到目标点 n 的距离；
 - 障碍物边界跟随（boundary-following）就复杂很多，因为我们并没有障碍物边界的先验知识；
 - Bug-like 算法之间主要的区别在于离开障碍物的位置选择；
- 为了到达目的地，我们只能借助传感器信息，但是一个传感器的能力毕竟是有限的，我们只能先确定机器人需要什么信息，然后在确定机器人应该移动到哪个位置来获取更多信息，**这是所有基于传感器的规划都需要克服的问题。**
- 对于一个移动机器人，我们希望输入传感器信息，输出机器人的移动速度和转向速度指令。
- 需要解决的三个问题：
 - 机器人需要什么信息来绕过障碍物？
 - 机器人如何从传感器数据中获取这些信息？
 - 机器人如何使用这些信息来确定（局部）路径？

4.1 需要的信息：切线

- 如果障碍物是平坦的，例如走廊的墙壁，机器人只需要确定障碍物表面的法线，然后确保与障碍物表面平行的方向即可。
- 机器人可以沿着一条与障碍物表面法向量始终垂直的路径，这条路径 $c(t)$ 满足条件 $\dot{c}(t) = v$ ，方向为 $(n(c(t)))^\perp$ ， v 的方向基于前面的 \dot{c} 的方向。

- 持续判断障碍物的表面法向量实现起来也并不简单，一般假设障碍是**局部平坦**的，那么我们就可以在这一段范围里垂直法线移动，然后重复这个过程，这样就构成了一系列有序的短直线段，这些短直线段可以近似认为是**切线**。

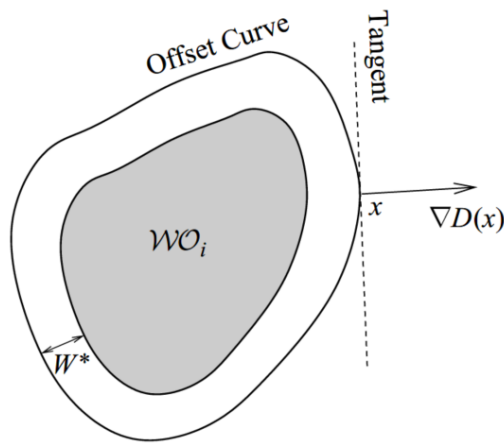


Figure 2.14 The solid curve is the offset curve. The dashed line represents the tangent to the offset curve at x .

曲线上一点 x 的切线可以近似认为是曲线在该点的一阶近似；

4.2 如何从传感器数据中获取信息：距离和梯度

- 如果我们不讲机器人视为一个质点，而是一个有着 360° 触碰传感器的圆（如下图所示），当这个机器人触碰到障碍物时，接触面指向圆心的方向可以**近似**认为是障碍物表面的法向量，利用这个信息，机器人可以构造出一系列切线来沿着障碍物。

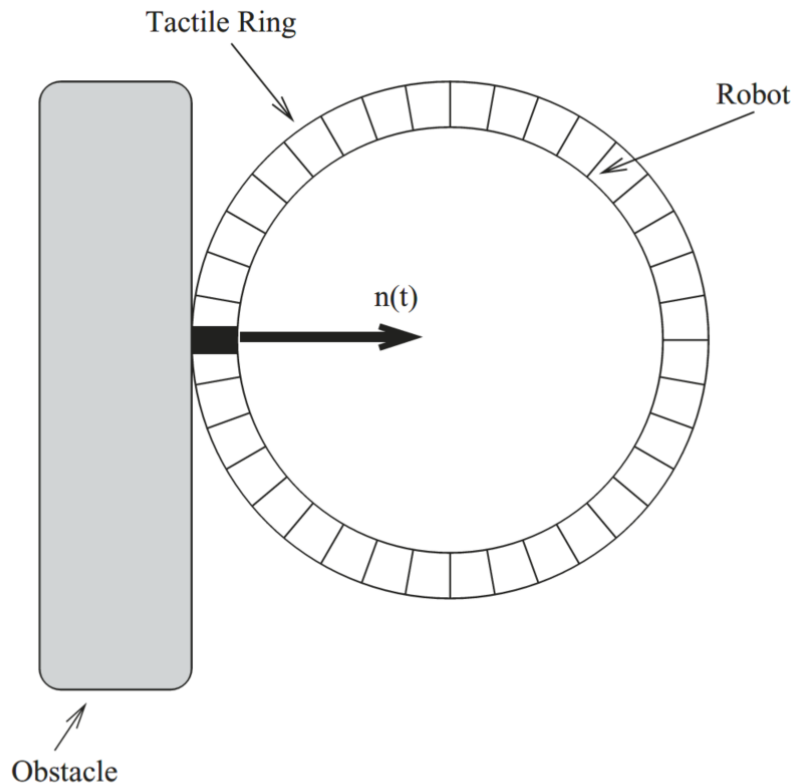


Figure 2.15 A fine-resolution tactile sensor.

- 然而，接触式传感器在一定程度上会危害到障碍物和机器人，所以，我们一般要求机器人沿着距离最近障碍物一个安全距离 W^* 的偏移曲线（offset curve），可以理解为将障碍物进行**扩大**一定尺寸；

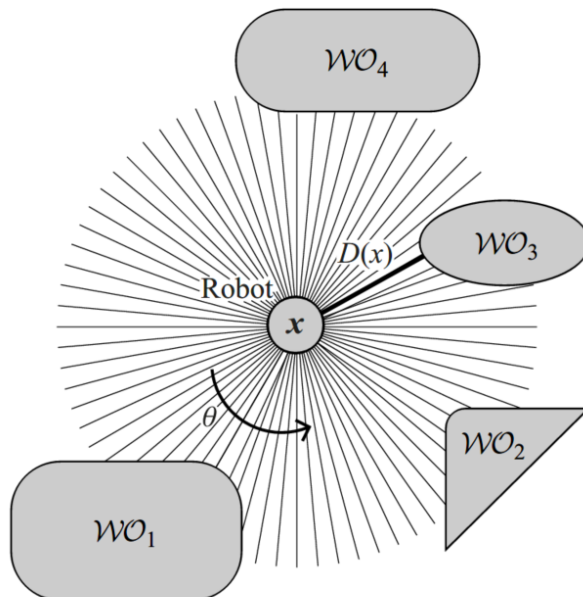


Figure 2.16 The global minimum of the rays determines the distance to the closest obstacle; the gradient points in a direction away from the obstacle along the ray.

$D(x) = \min_{c \in \cup_i \mathcal{WO}_i} d(x, c)$ 表示距离机器人最近的障碍物距离

- 一个移动机器人携带板载测距传感器（可以全角度测距），我们寻找最小的值点

$D(x) = \min_s \rho(x, s)$ ，注意区别于介绍 Tangent Bug 算法时，我们是在寻找不连通区域。

- 除了距离信息之外，我们还需要距离的**梯度信息**，如 $\nabla D(x) = [\frac{\partial D(x)}{\partial x_1} \quad \frac{\partial D(x)}{\partial x_2}]$ （平面情况下），其指向距离增加最大的方向，在上图中， $D(x)$ 的梯度就是一个沿着最小障碍物距离的单位向量。
- 需要了解常用的测量传感器有什么异同，比如超声波测距、激光测距、声纳等，这样才能根据实际的工作环境选择合适的传感器。

4.3 如何处理传感器信息

- 切线 $(\nabla D(x))^\perp$ 垂直于 $\nabla D(x)$ （距离增大最大的方向）和 $-\nabla D(x)$ （距离减小最大的方向）。
- 偏移曲线上的点满足映射 $G(x) = D(x) - \mathcal{W}^* \rightarrow 0$ ，集合中映射到 0 的这些非零点构成的集合被称为这个映射的**零空间**，偏移曲线上一点 x 的**切线空间**是 $\mathcal{D}G(x)$ 的零空间， $\mathcal{D}G(x)$ 表示 G 的雅可比矩阵（**Jacobian**），雅可比矩阵可以看成是不同切线空间之间的映射关系。（ \mathcal{D} 表示微分）
- 由于 G 是一个实值函数，因此其对应的雅可比矩阵是一个**行向量** $\mathcal{D}D(x)$ ，在**欧式空间**中， $(\mathcal{D}D(x))^T = \nabla D(x)$ ，又因为切线空间是 $\mathcal{D}D(x)$ 的零空间，所以平面障碍物边界跟随的切线垂直于 $\nabla D(x)$ 。
- 根据距离信息，机器人可以确定偏移曲线的切线方向：
 - 如果障碍物的表面是平坦的，那么其对应的偏移曲线也是平坦的，直接沿着切线方向前进就可以保证机器人是跟随障碍物边界的；
 - 如果障碍物的表面不是平坦的（即有曲率），那么我们就假设其表面是局部平坦，机器人可以沿着切线走一小段距离，但是由于障碍物不平坦，机器人实际上并没有沿着偏移曲线前进（可能远离，也可能是靠近），所以必须将其拉回到安全距离 \mathcal{W}^* ，即沿着 $\nabla D(x)$ 或 $-\nabla D(x)$ 方向移动。

- 我们可以将整个过程看成是机器人在不断的**预测**和**校正**，机器人根据切线来局部预测偏移曲线的形状，并且当这个切线的近似条件不再满足时进行校正。整个过程中机器人并不是完全跟着偏移曲线，而是在离散化的跟随一些曲线上的点。

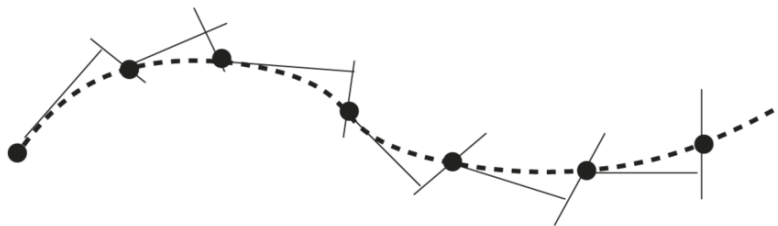


Figure 2.20 The dashed line is the actual path, but the robot follows the thin black lines, predicting and correcting along the path. The black circles are samples along the path.

- 数学上，我们可以将整个过程看成是跟随表达式 $G(x) = 0$ 的根。根据**隐函数定理**（具体可以参考资料1，也可以参考书中的附录D），我们可以用 $G(y, \lambda) = D(y, \lambda) - \mathcal{W}^*$ 隐性的定义了偏移曲线，其中 λ 坐标对应切线方向， y 坐标对应与切线正交的直线或超平面。如果 $D_Y G(y, \lambda)$ 在 $x = (\lambda, y)^T$ 是**满射**，那么根据隐函数定理， $G(y, \lambda)$ 的根局部定义了一条曲线，例如 $y(\lambda)$ ，沿着障碍物的边界，并且到障碍物的距离为 \mathcal{W}^* 。
- 假设机器人位于位置 x 点，（这个点到障碍物边界的距离固定为 \mathcal{W}^* ），这个机器人根据预测沿着 λ 方向走了 $\Delta\lambda$ ，导致其**脱离**了偏移路径。校正过程就是找到**偏移路径与校正平面的相交位置**，主要是用到了牛顿迭代法。**校正平面**是指垂直于 λ 方向，且到 x 位置距离为 $\Delta\lambda$ 处的一个平面。

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

参考资料

- <https://zh.wikipedia.org/wiki/%E9%9A%90%E5%87%BD%E6%95%B0%E5%AE%9A%E7%90%86> 隐函数定理
- <https://user.ceng.metu.edu.tr/~akifakkus/courses/ceng786/tangentMouse.html>
- <https://cs.gmu.edu/~plaku/teaching/LecRoboBugAlgorithms.pdf>