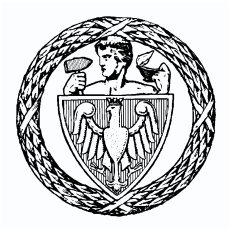
Mobile Robots

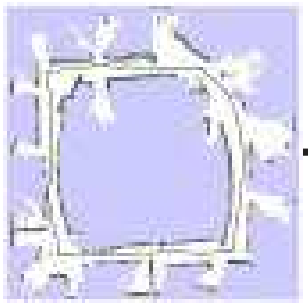
Collision Avoidance



Lecture 10

Obstacle or Collision Avoidance = Local Path Planning

Two-layered structure



map

sensor data

Planning

low frequency

subgoal

Collision avoidance

high frequency

motion command

robot

Collision Avoidance

Problem consists of computing a motion control that avoid collisions with the obstacles gathered by the sensors, whilst driving the robot towards the target location.

$$\boxed{\text{Obstacle Detection}} \Rightarrow \boxed{\text{Collision Avoidance}}$$

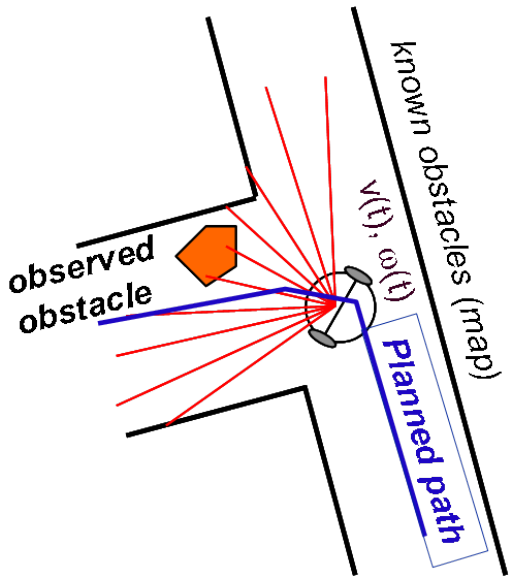
- The obstacle avoidance methods are *local* and *iterative* techniques.
- The disadvantage of locality (trap situations) is counteract with the advantage of introducing the sensory information within the control cycle (to take into account the reality of the world during execution).

- Efficient obstacle avoidance should be optimal with respect to:
  - the overall goal,
  - the actual speed and kinematics of the robot,
  - the on boards sensors,
  - the actual and future risk of collision.

Collision Avoidance

Aspects that affect the development of an obstacle avoidance method:

- the avoidance technique,
- the type of robot sensor,
- the type of an environment (static or dynamic, unknown or known, structured or not).



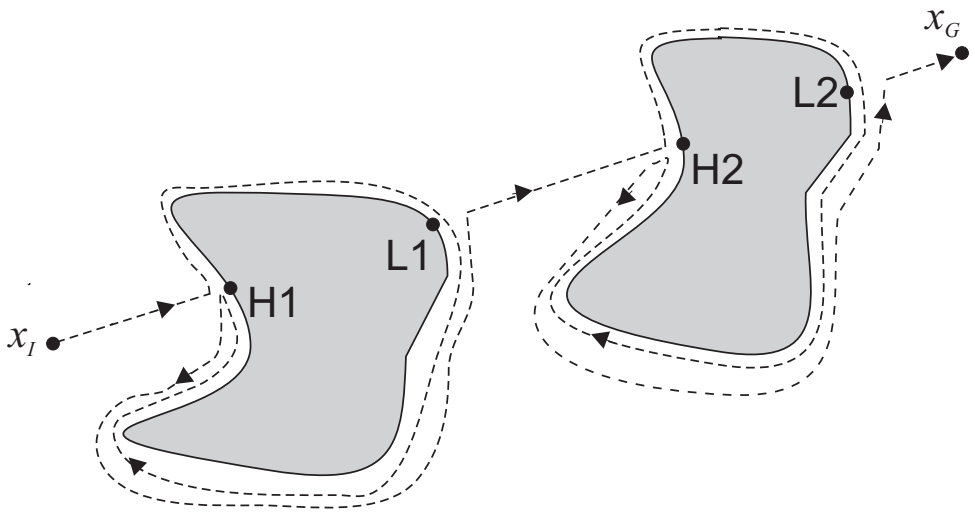
Bug Algorithms

Assumptions:

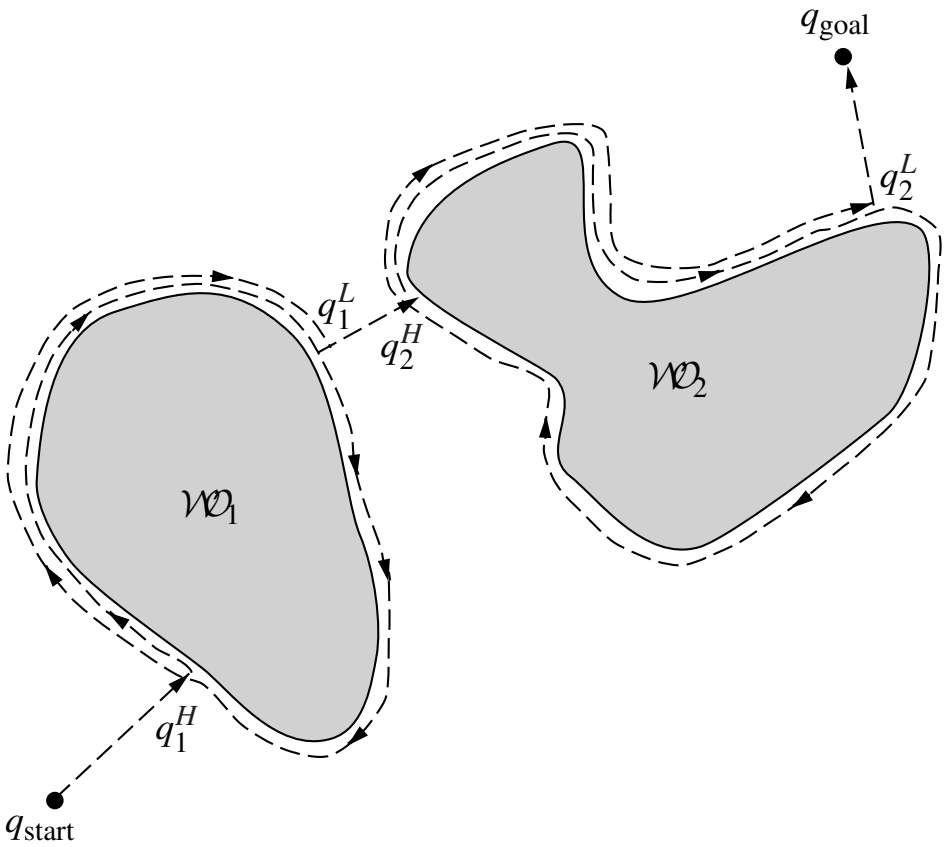
- Bug algorithms assume only *local knowledge* of the environment and a *global goal*.
- The earliest and simplest *sensor-based local* planners with the *provable guarantees*.
- It is assumed that the robot is a point operating in the plane with a contact or a range sensor to detect obstacles.
- The robot exhibits two behaviors: *motion-to-goal* (move on a straight line) and *follow-a-boundary*.
- The robot has perfect positioning (localization) system (no positioning error).
- The workspace W is bounded, i.e. $W \subset B_r(x), r < \infty$, where $B_r(x) = \{y \in \mathbb{R}^2 : d(x, y) < r\}$.
- In the workspace there is a finite number of obstacles $\mathcal{WO}_i,\ i = 1, \ldots, n$.
- Free space is given as: $W_{free} = W \setminus \bigcup_i \mathcal{WO}_i$.

Bug1 Algorithm

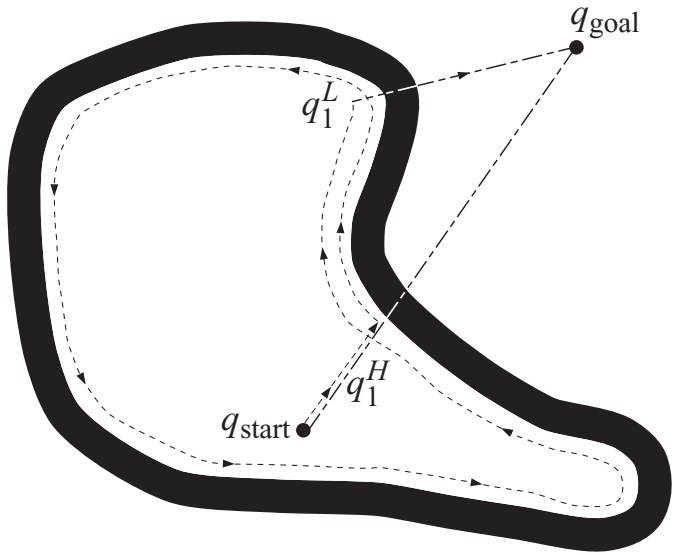
- Head toward goal.
- If an obstacle is encountered, circumnavigate it and remember how close you get to the goal.
- Return to that closest point (by wall-following) and continue.
- Each encountered obstacle is once fully circled before it is left at the point closest to the goal.
- It is an exhaustive search algorithm – it looks at all choices before committing.
- An algorithm is complete if, in finite time, it finds a path if such a path exists, or terminates with a failure if it does not.



Bug1 Algorithm



Bug1 successfully finds the goal.



Bug1 reports the goal is unreachable.

Bug1 Algorithm

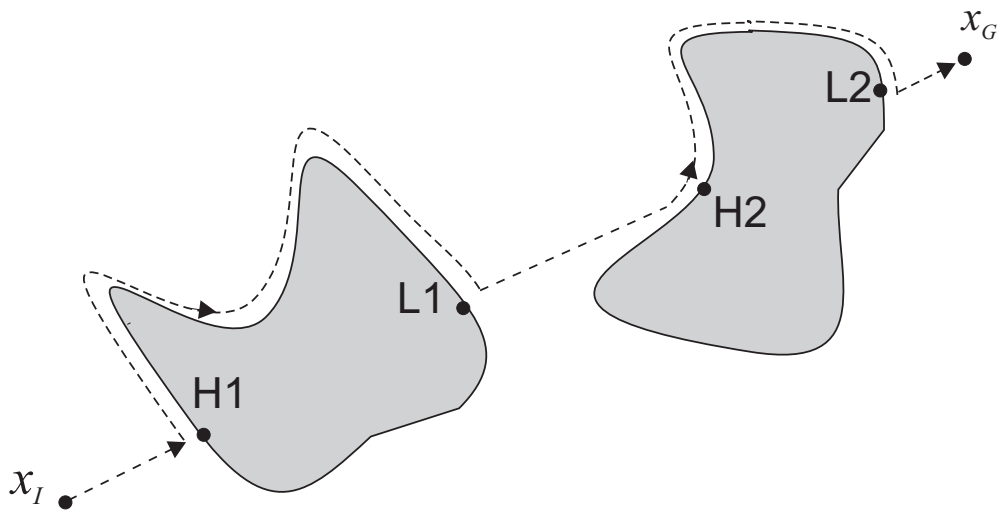
Input: A point robot with a tactile sensor
Output: A path to the q_{goal} or a conclusion no such path exists

---

while True **do**
    repeat
        From q_{i-1}^{L}, move toward q_{goal}
    until q_{goal} is reached **or** an obstacle is encountered at q_i^H
    if goal is reached **then**
        Exit
    end if
    repeat
        Follow the obstacle boundary
    until q_{goal} is reached **or** an obstacle is re-encountered at q_i^H
    Determine the point q_i^L that has the shortest distance to the goal
    Go to q_i^L
    if the robot were to move toward the goal **then**
        Conclude q_{goal} is not reachable and exit
    end if
end while

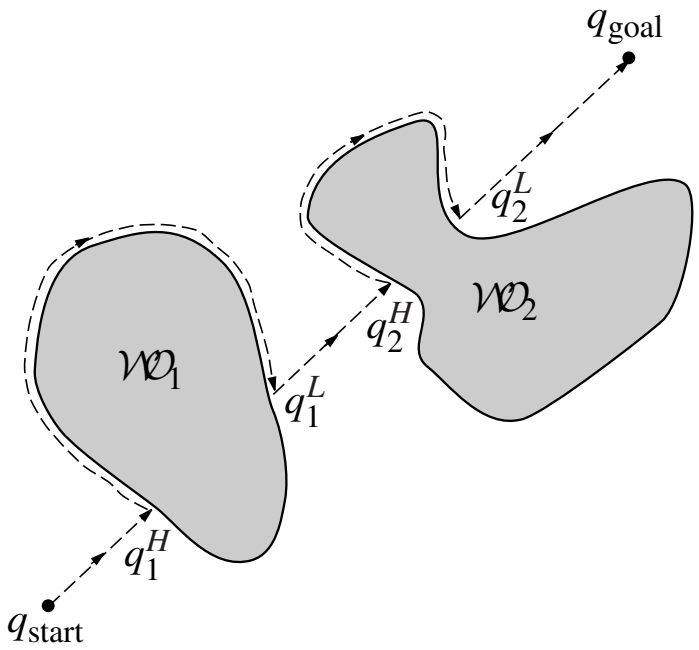
Bug2 Algorithm

- Like Bug1, 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 connecting q_{start} and q_{goal}, which remains fixed.
- The *boundary-following* behavior is invoked if the robot encounters an obstacle.
- 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.

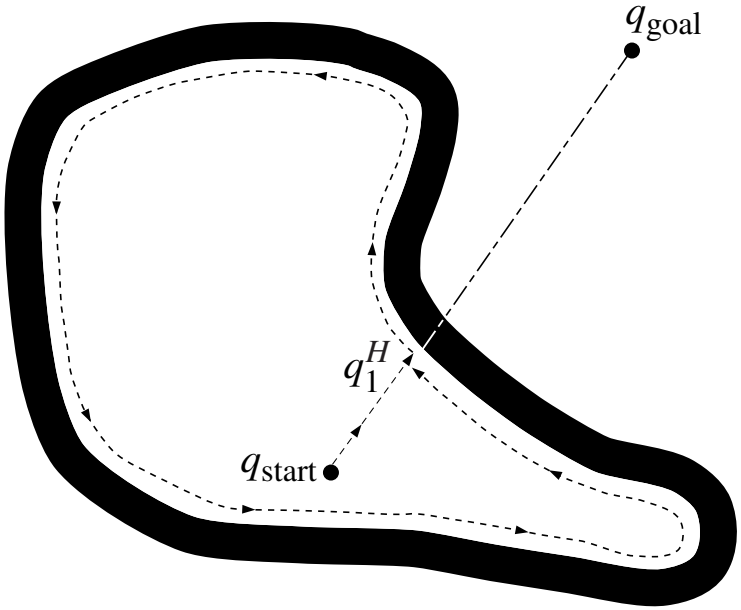


Bug2 Algorithm

The Bug2 algorithm finds a path to the goal



The Bug2 algorithm reports failure



Bug2 Algorithm

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

---

while True **do**
  repeat
    From q_{i-1}^{L}, move toward q_{goal} along m-line
  until q_{goal} is reached **or** an obstacle is encountered at hit point q_i^H
  Turn left (or right)
  repeat
    Follow the obstacle boundary
  until q_{goal} is reached **or** an obstacle is re-encountered at q_i^H **or** m-line is re-encountered
    at a point m such that $m \neq q_i^H$ (robot did not reach hit point),
    $d(m, q_{goal}) < d(m, q_i^H)$ (robot is closer), and
    if robot moves toward goal, it would not hit the obstacle
  if Goal is reached **then**
    Exit
  end if
  if q_i^H is re-encountered **then**
    Conclude goal is unreachable
  end if
  Let $q_{i=1}^{L} = m$
  $i++$
end while

Bug1 vs. Bug2

- 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.
- If the robot encounters all n obstacles the length of the path for the Bug1

$$L_{Bug1} \leqslant d(q_{start}, q_{goal}) + 1.5 \sum_{i}^{n} p_i$$

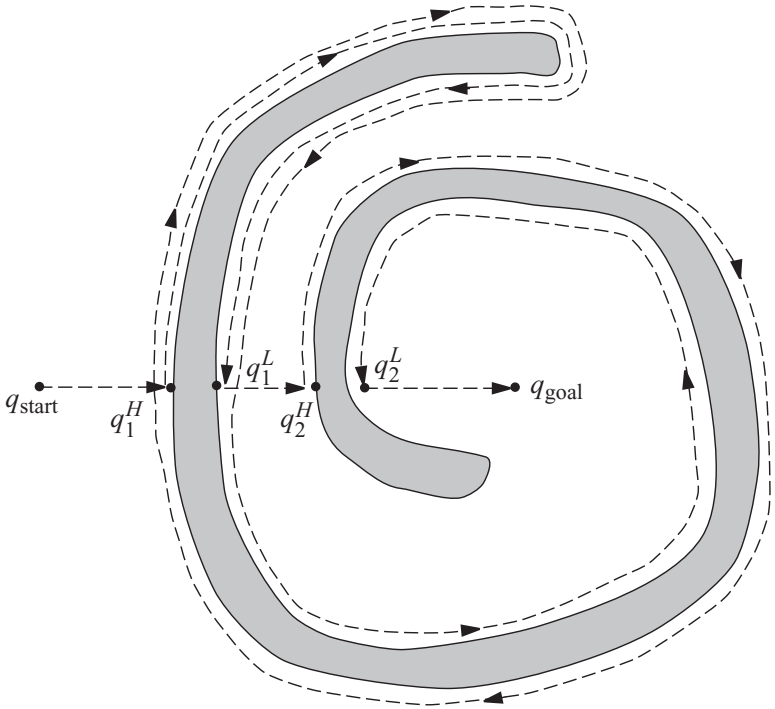
- For Bug2 if the m-line intersects the i-th obstacle n_i time, then in the worst case, the robot will traverse nearly entire perimeter of the obstacle for each leave point:

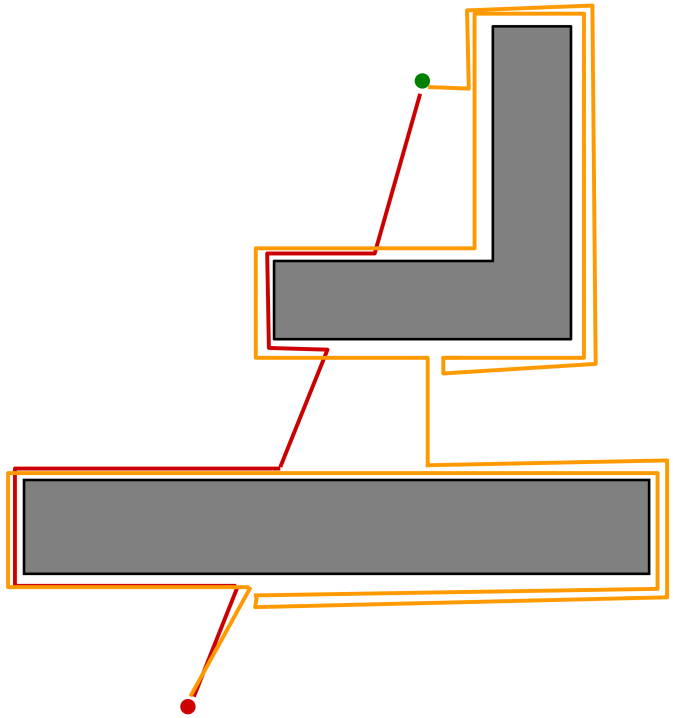
$$L_{Bug2} \leqslant d(q_{start}, q_{goal}) + 0.5 \sum_{i}^{n} n_i p_i$$

Bug1 vs. Bug2

- L_{Bug2} can be arbitrarily longer than L_{Bug2}. This can be achieved by constructing an obstacle whose boundary has many intersections with the m-line.
- 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.
- Bug2 uses an *opportunistic approach* or *greedy approach*, when it finds a leave point that is better than any it has seen before, it commits to that leave point.
- When the obstacles are simple, the Bug2 gives a quick payoff, but when the obstacles are complex, the more conservative approach of Bug1 often yields better performance.

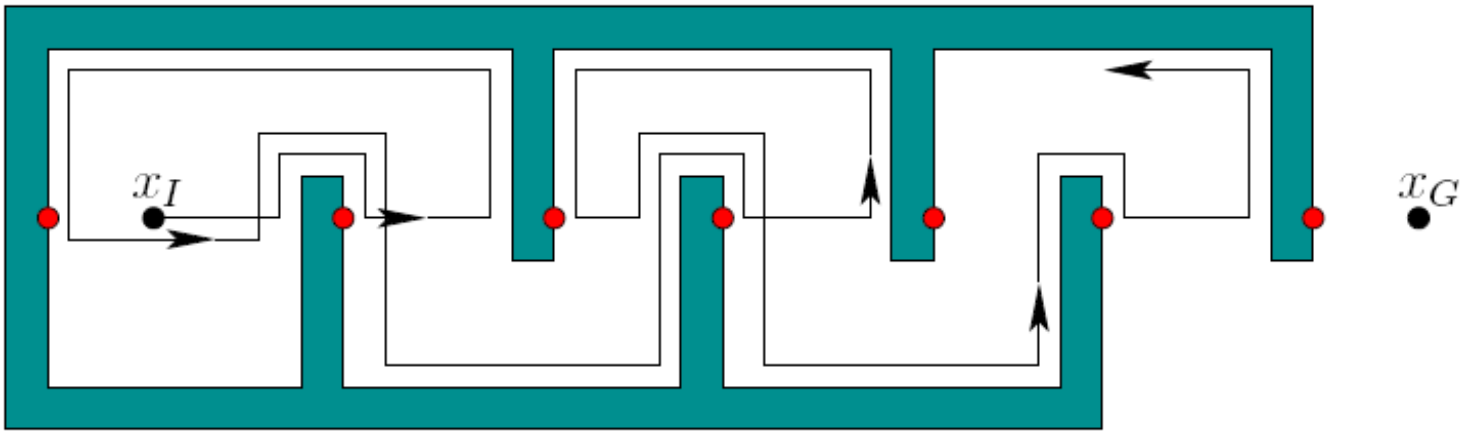
Bug1 vs. Bug2





- - - Bug2

— Bug1 — Bug2



Tangent Bug Algorithm I

- We typically use some sort of range sensing device that lets us look ahead (but it has finite resolution and is noisy).
- Tangent Bug improves the Bug2, it determines a shorter path to the goal using a range sensor with $\theta \in [0°, 360°]$ infinite orientation resolution.
- We model this sensor with *raw distance function* $\varrho : \mathbb{R}^3 \times S^1 \to \mathbb{R}$.
- A point robot $x \in \mathbb{R}^2$ with rays emanating from it. $\forall \theta \in S^1$, the value $\varrho(x, \theta)$ is the distance to the closest obstacle along the ray from x at an angle θ:

$$\varrho(x, \theta) = \min_{\lambda} d(x, x + \lambda[\cos\theta, \sin\theta]^{\mathsf{T}}),$$

$$\text{such that} \quad x + \lambda[\cos\theta, \sin\theta]^{\mathsf{T}} \in \bigcup_i \mathcal{WO}_i$$

Tangent Bug Algorithm II

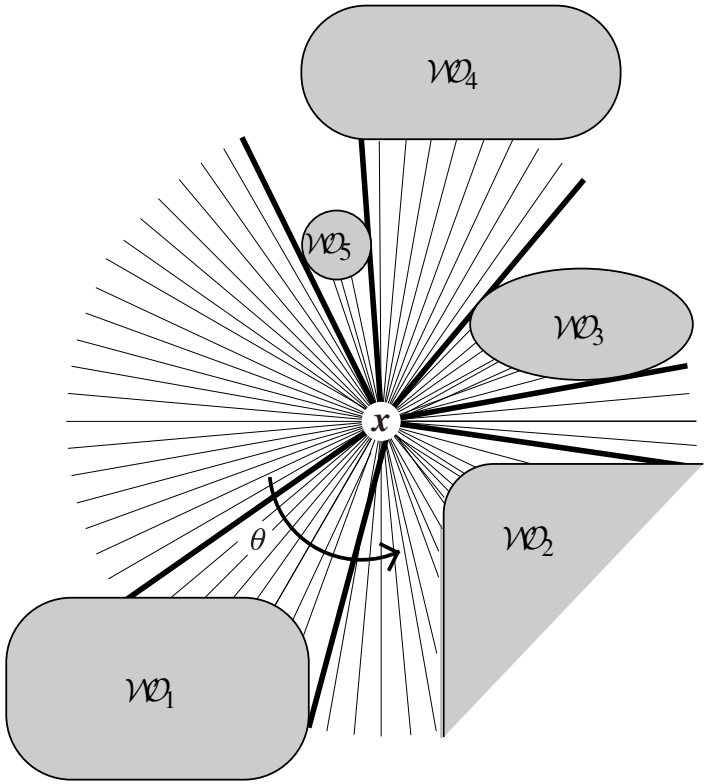
- Since the real sensors have limited range, we define the *saturated raw distance function*:

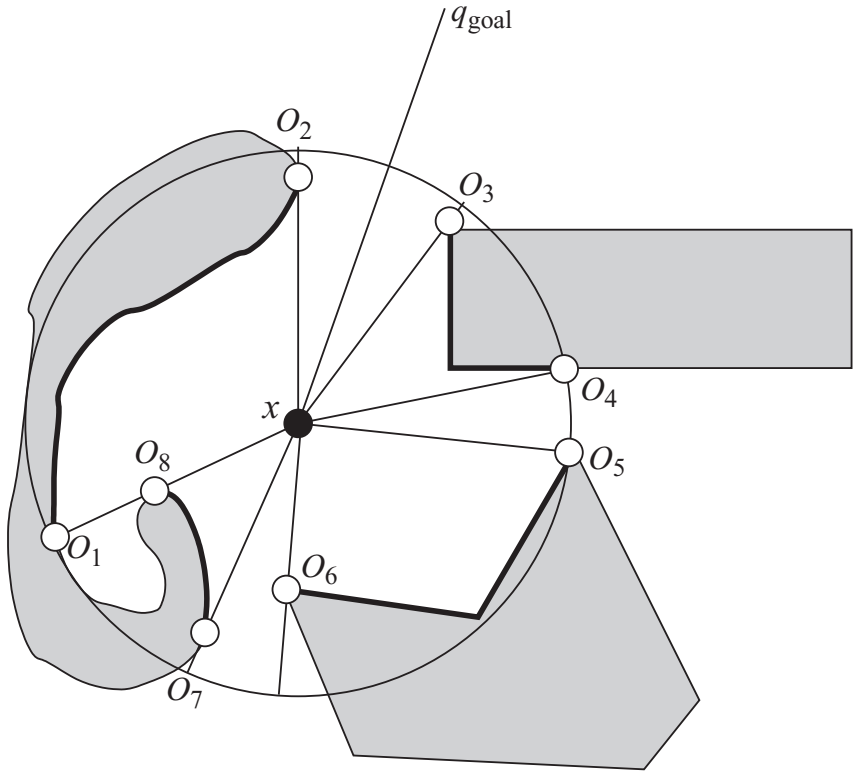
$$\varrho_R(x,\theta) = \begin{cases} \varrho(x,\theta) & \text{if } \varrho(x,\theta) < R \\ \infty, & \text{otherwise,} \end{cases}$$

  where, R, is the sensing range.

- It is assumed that the robot can detect discontinuities in $\varrho_R(x,\theta)$.
- For a fixed $x \in \mathbb{R}^2$, an *interval of continuity* is defined to be a connected set of points $x + \lambda[\cos\theta, \sin\theta]^\mathsf{T}$ on the boundary of the free space, where ϱ_R is finite and continuous with respect θ.
- The points $O_1, \ldots, O_8$ correspond to losses of continuity.

Tangent Bug Algorithm III



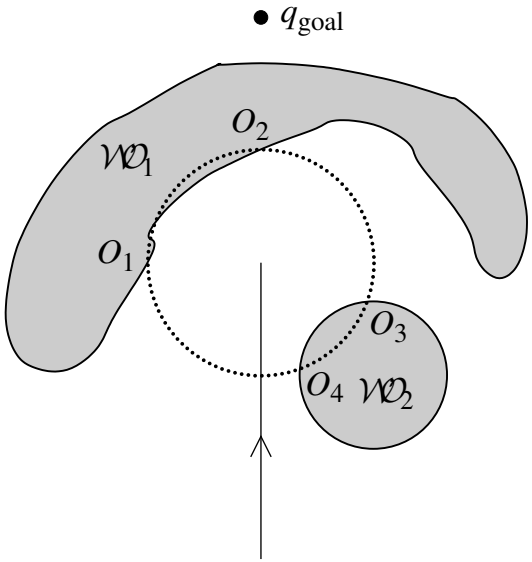


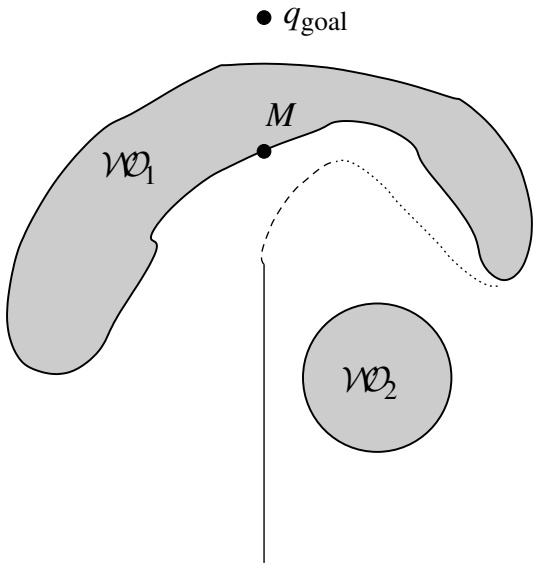
- Tangent Bug iterates between two behaviors: *motion-to-goal* and *boundary-following* (however they are different than in the Bug1 and Bug2 approaches).

Tangent Bug Algorithm IV

- The robot initially invokes the *motion-to-goal* (it moves in a straight line toward the goal until it senses an obstacle R away and directly between it and the goal.
- When the robot detects an obstacle, the circle of radius R becomes tangent to the obstacle, and this tangent point splits into two O_i's which are the endpoints of the interval.
- If the the obstacle is in the front of the robot, then this interval intersects the segment connecting the robot and the goal.
- The robot moves toward the O_i, that maximally decreases a heuristic distance $d_H = d(x, O_i) + d(O_i, q_{goal})$ to the goal.
- The robot undergoes the *motion-to-goal* behavior until it finds a point that is a local minimum of d_H.
- A boundary following behavior invoked when heuristic distance d_H increases.

Tangent Bug Algorithm V





- The *blocking obstacle* is the closest obstacle within the sensor range that intersects the segment $(1-\lambda)x + \lambda q_{goal},\ \forall \lambda \in [0,1]$.
- The *followed obstacle* is the sensed obstacle. Initially the followed obstacle and the blocking obstacle are the same.
- The robot continuously moves toward the O_i on the followed obstacle, and the planner updates two functions $d_{followed}$ oraz d_{reach}.
- A value $d_{followed}$ is the shortest distance between the sensed boundary and the goal q_{goal}.

Tangent Bug Algorithm VI

- A value d_{reach} is the shortest distance between blocking obstacle and goal (or the distance to goal if no blocking obstacle visible).
- Let $\Lambda = \{y \in \partial\mathcal{W}O : \lambda x + (1-\lambda)y \in W_{free}\ \forall\lambda \in [0,1]\}$ be a set of all points within line of sight of x with range R that are on the followed obstacle, then

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

- When $d_{reach} < d_{followed}$ the robot terminates the boundary-following behavior.
- Let T be the point where the circle centered at x of radius R intersects the segment that connects x and q_{goal}. This the point on the periphery of the sensing range that is closest to the goal.
- The algorithm starts with $x = q_{start}$ and $d_r = d(q_{start}, q_{goal})$.

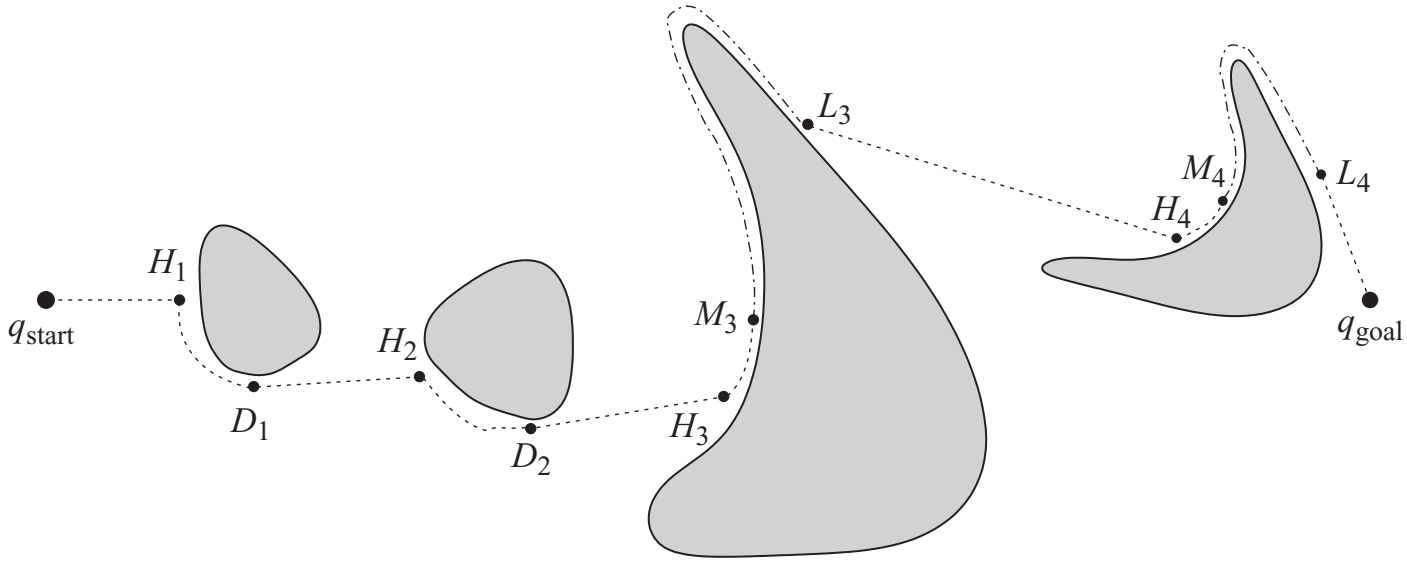
Tangent Bug Algorithm

Input: A point robot with a range sensor

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

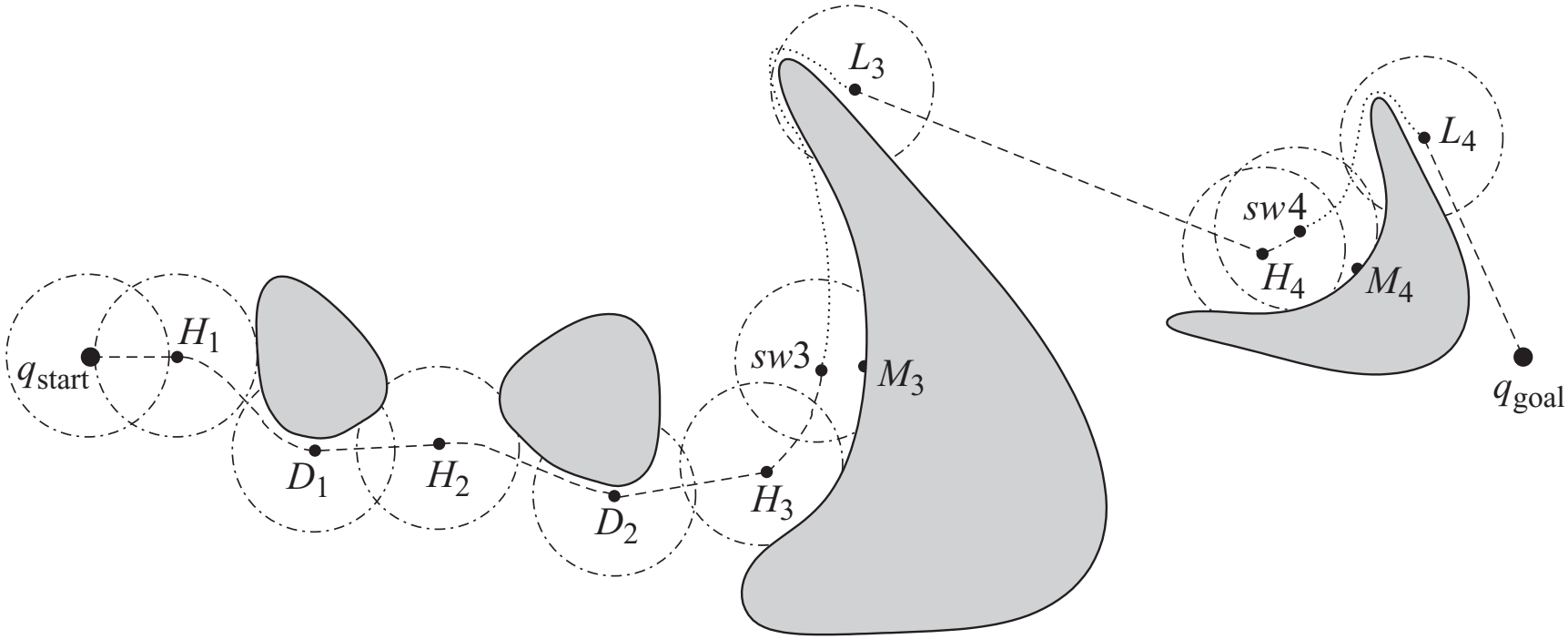
1. **while** True **do**
2. **repeat**
3. Compute continuous range segments in view
4. Move toward the point $n \in \{T, O_i\}$ that minimizes $h(x, n) = d(x, n) + d(n, q_{goal})$
5. **until**
   a) the goal is encountered, or
   b) the value of $h(x, n)$ begins to increase
6. Follow boundary continuing in same direction as before repeating
7. **repeat**
8. Update $\{O_i\}$, d_{reach} and $d_{followed}$
9. Move toward the point $n \in \{O_i\}$ that is in the chosen boundary direction
10. **until**
    a) the goal is reached, or
    b) a complete cycle is performed (goal is unreachable)
    c) $d_{reach} < d_{followed}$
11. **end while**

Tangent Bug Algorithm



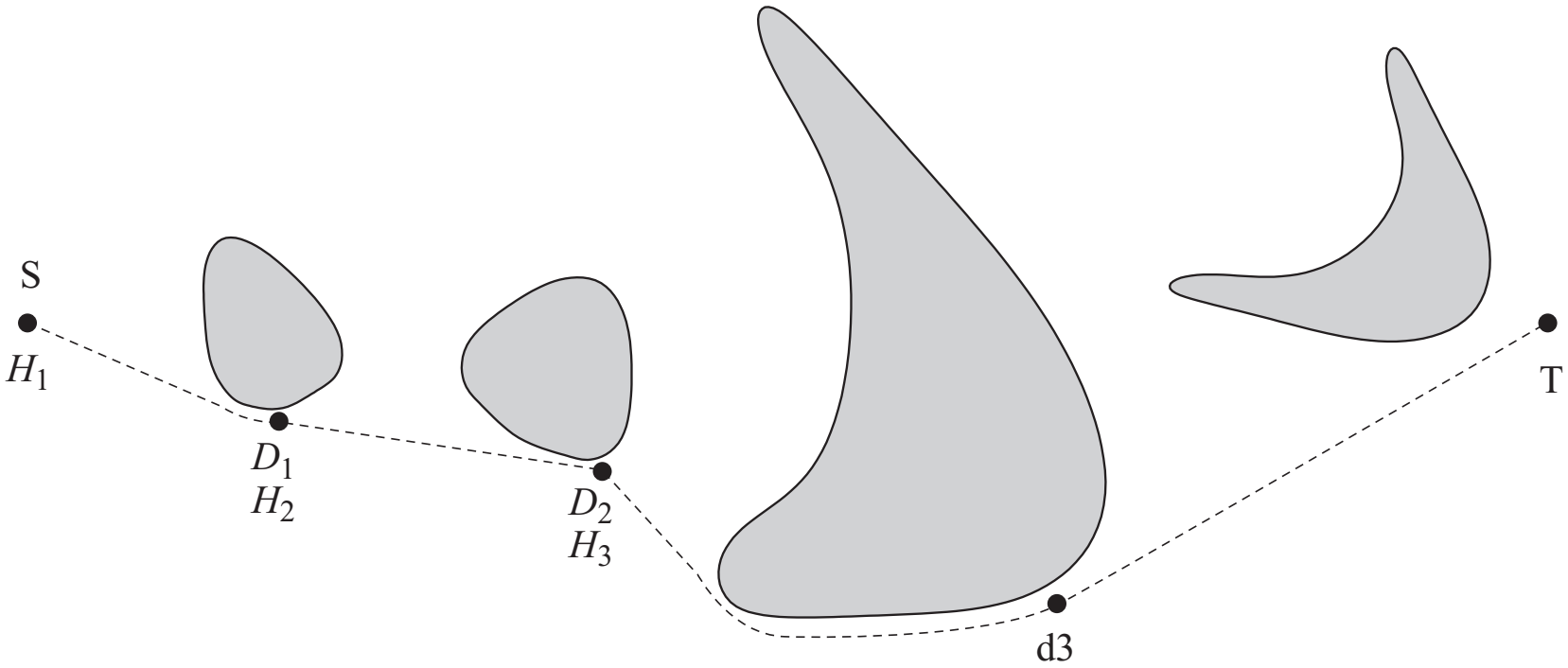
Path generated by Tangent Bug with zero sensor range

Tangent Bug Algorithm



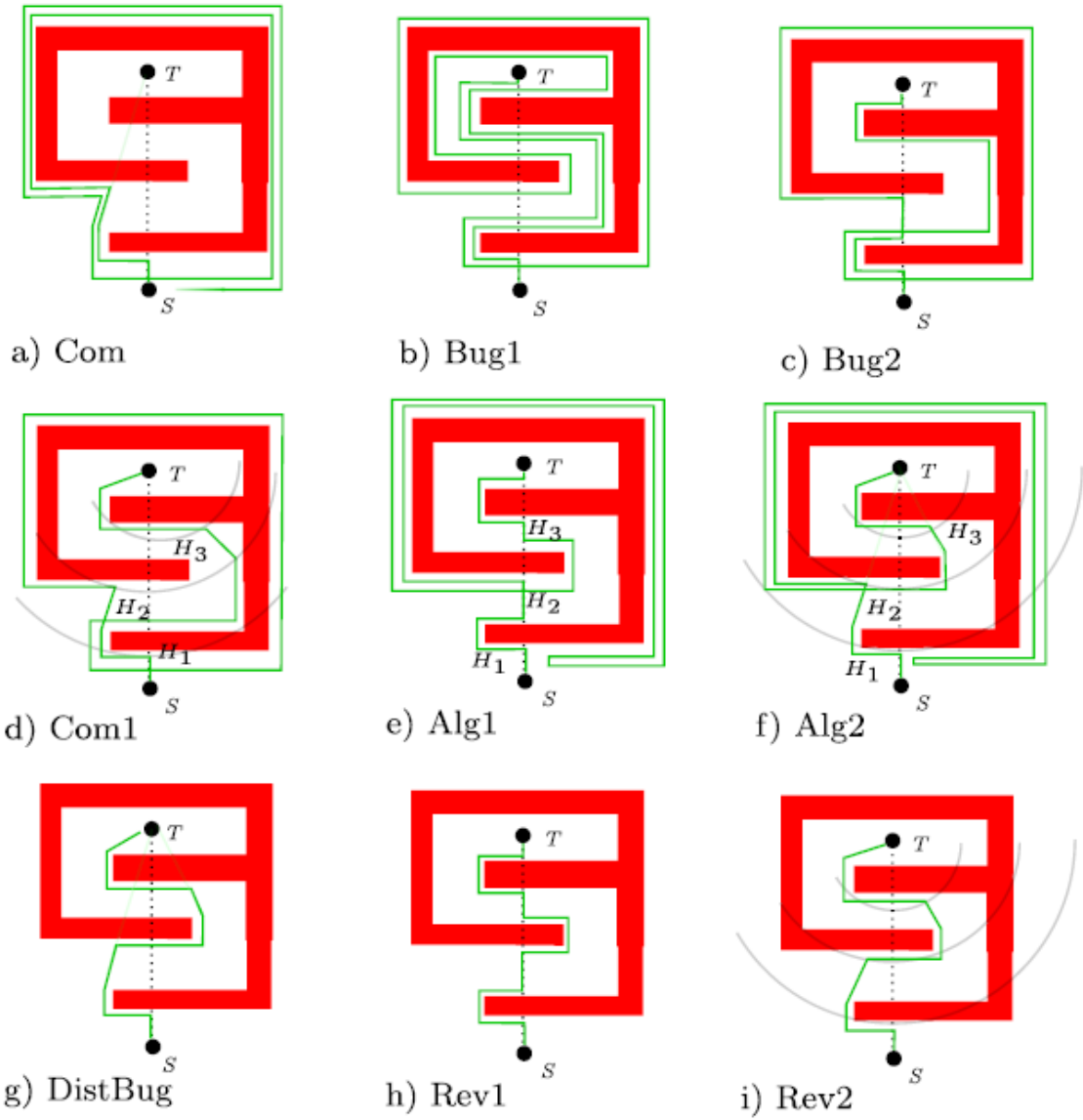
Path generated by Tangent Bug with finite sensor range

Tangent Bug Algorithm



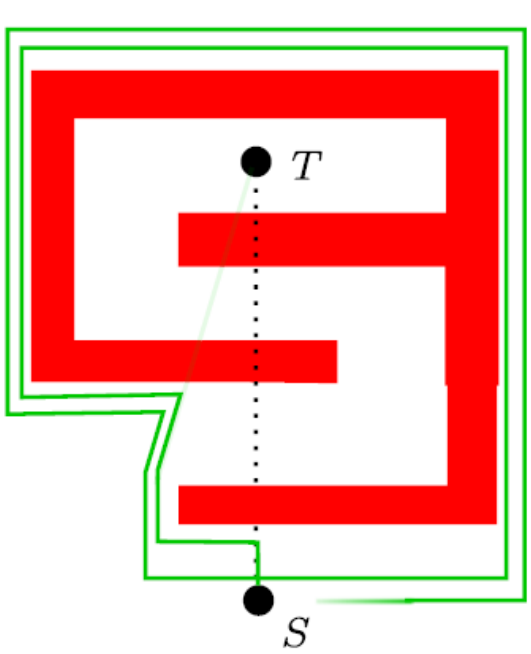
Path generated by Tangent Bug with infinite sensor range

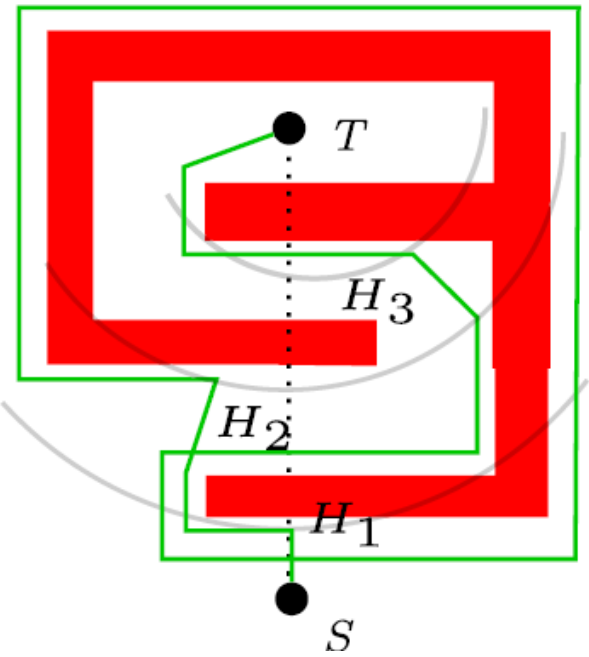
Bug Algorithms



Source: K.N. McGuire, et.al, A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

Com and Com1 Algorithms





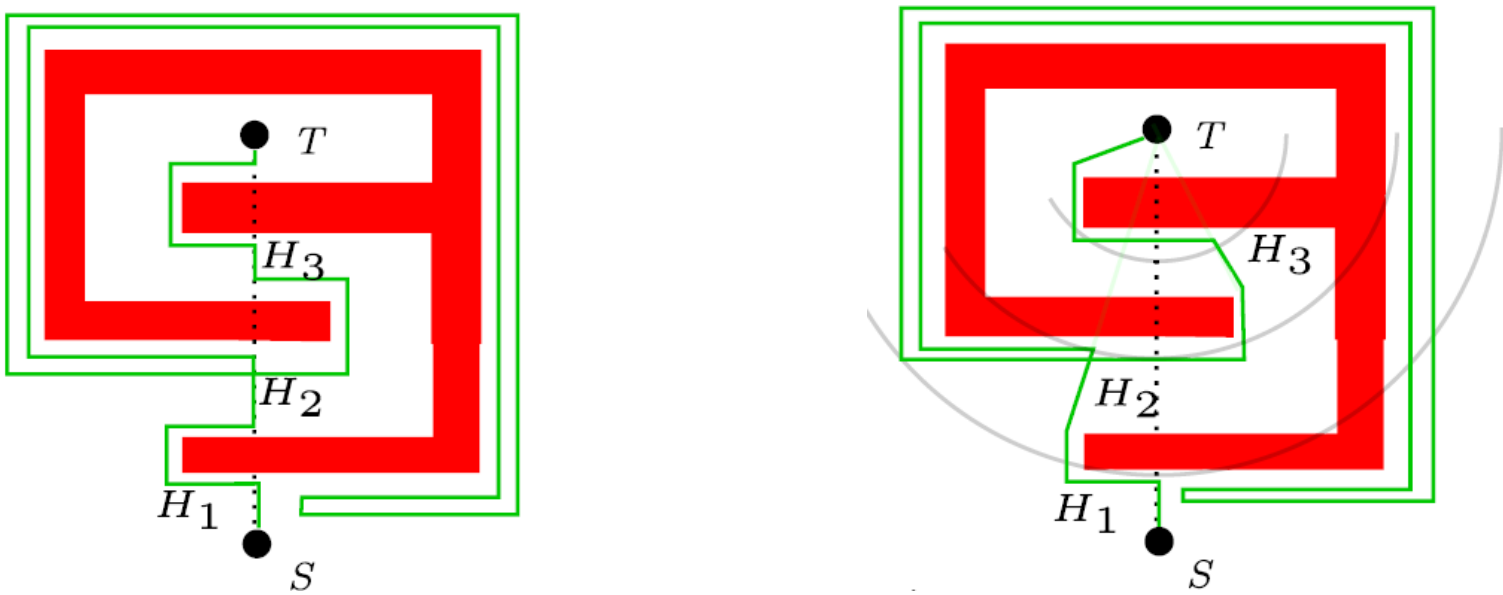
Source: K.N. McGuire, et. A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

Com

Com1

- Com the "common sense algorithm" – in more complex scenarios it cannot reach the goal.
- Com1 remembers the distance of the previous point H_i from the target T.
- Com1 utilizes this to initiate the departure from the obstacle boundary.

Alg1 and Alg2 Algorithms

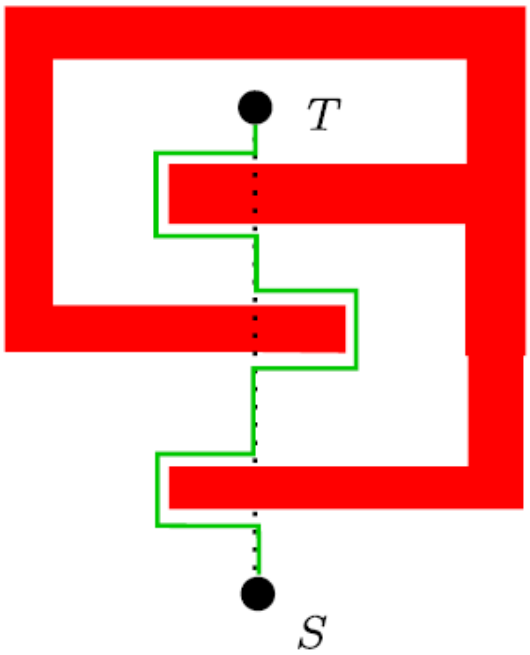


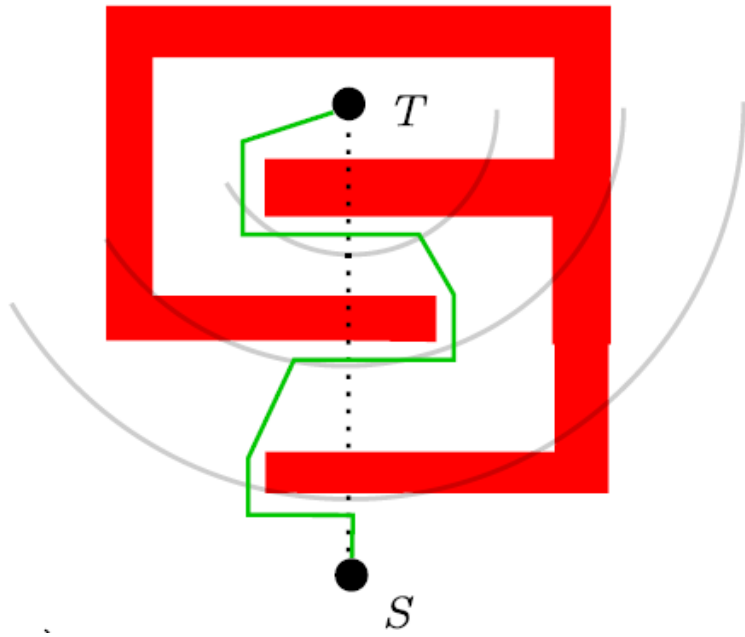
Source: K.N. McGuire, et. A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

Alg1 Alg2

- Alg1 – Bug2 modification, points H_i are remembered, it changes its wall-following direction if it comes across a previously visited hit-point H_i.
- Alg2 – Com1 modification, the distances of all points H_i from T are remembered, it reverses the wall-following direction if it encounters a previous saved hit-point.

Rev1 and Rev2 Algorithms



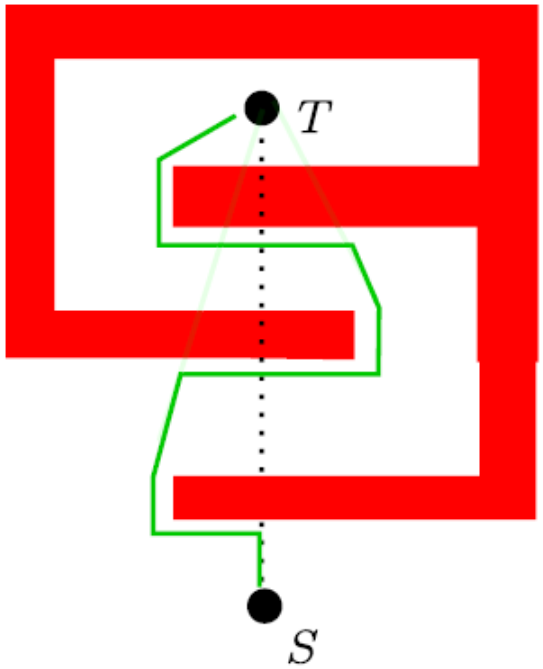


Source: K.N. McGuire, et. A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

Rev1 Rev2

- Rev1 – extension of Alg1, a change of direction occurs at each point H_i.
- Rev2 – extension of Alg2, a change of direction occurs at each point H_i.
- In some scenarios alternating the local wall-following direction is not the best policy.

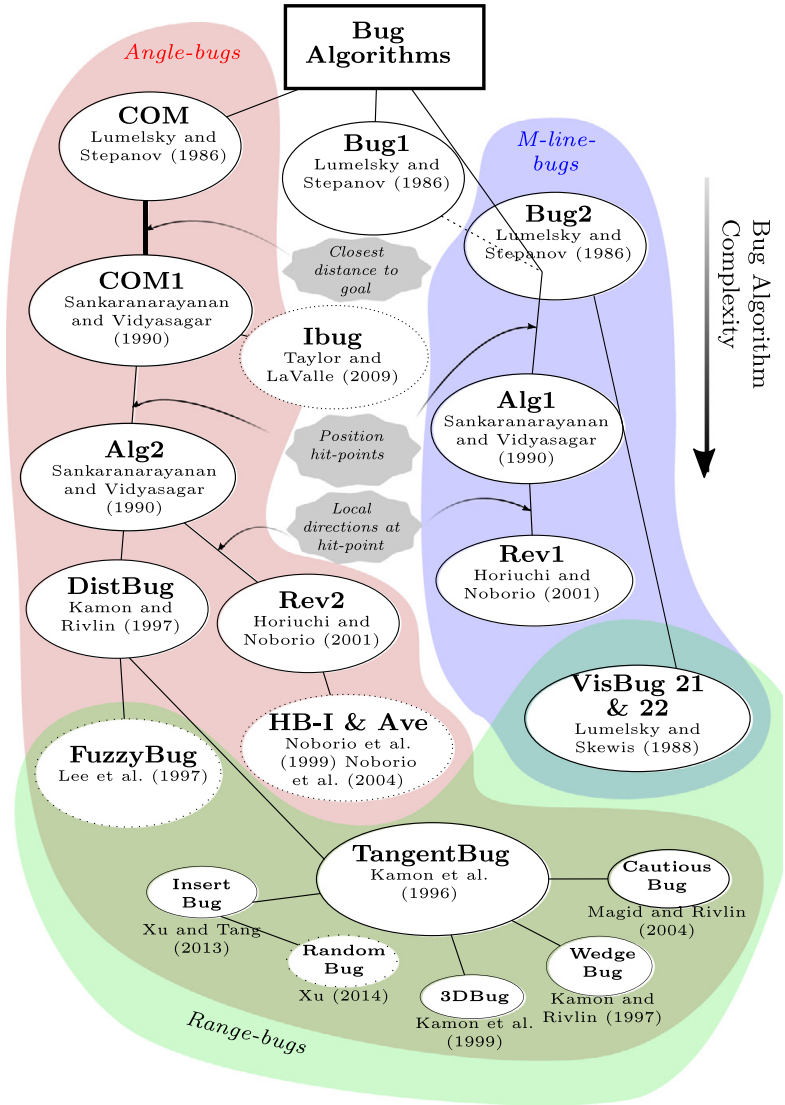
DistBug Algorithm



Source: K.N. McGuire, et. A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

- DistBug – similar to Alg2, but solely the last one H_i is remembered.
- The local wall-following direction depends on the orientation with which it touches the hit-point H_i.

Bug Algorithms



Source: K.N. McGuire, et.al, A comparative study of bug algorithms for robot navigation, *Robotics and Autonomous Systems* 121 (2019)

Obstacle Avoidance: Vector Field Histogram (VFH) I

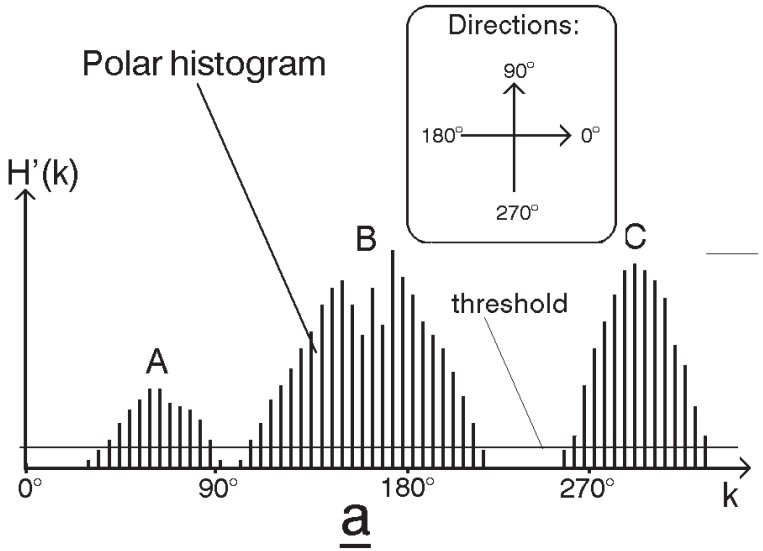
- Environment represented in an occupancy grid (2 DOF) – cell values equivalent to the probability that there is an obstacle.
- Reduction of the grid to a 1D polar histogram by tracing a dense set of rays emanating from the robot up to a maximal distance
- The steering direction is computed in two steps:
  - all openings large enough for the robot to pass are found
  - the one with lowest cost function G is selected

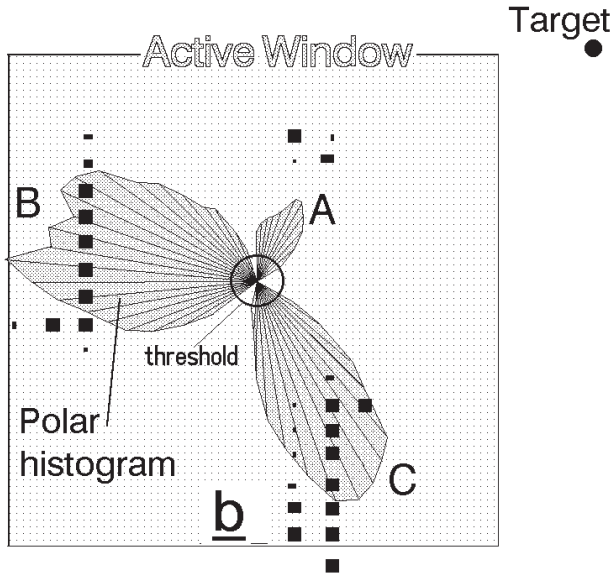
$$G = a \cdot \text{target-direction} + b \cdot \text{wheel-orientation} + c \cdot \text{previous-direction}$$

where

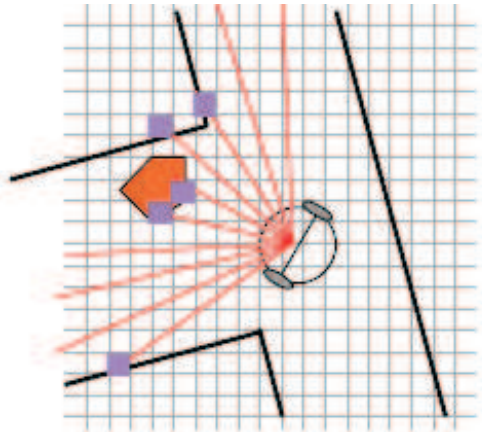
- target-direction: alignment of the robot path with the goal
- wheel-orientation: difference between the new direction and the current wheel orientation
- previous-direction: difference between the previously selected direction and the new direction

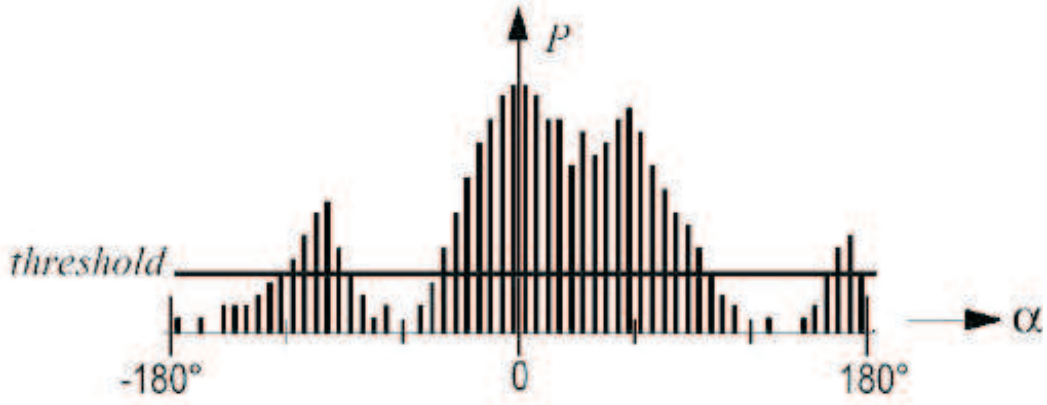
Obstacle Avoidance: Vector Field Histogram (VFH) II





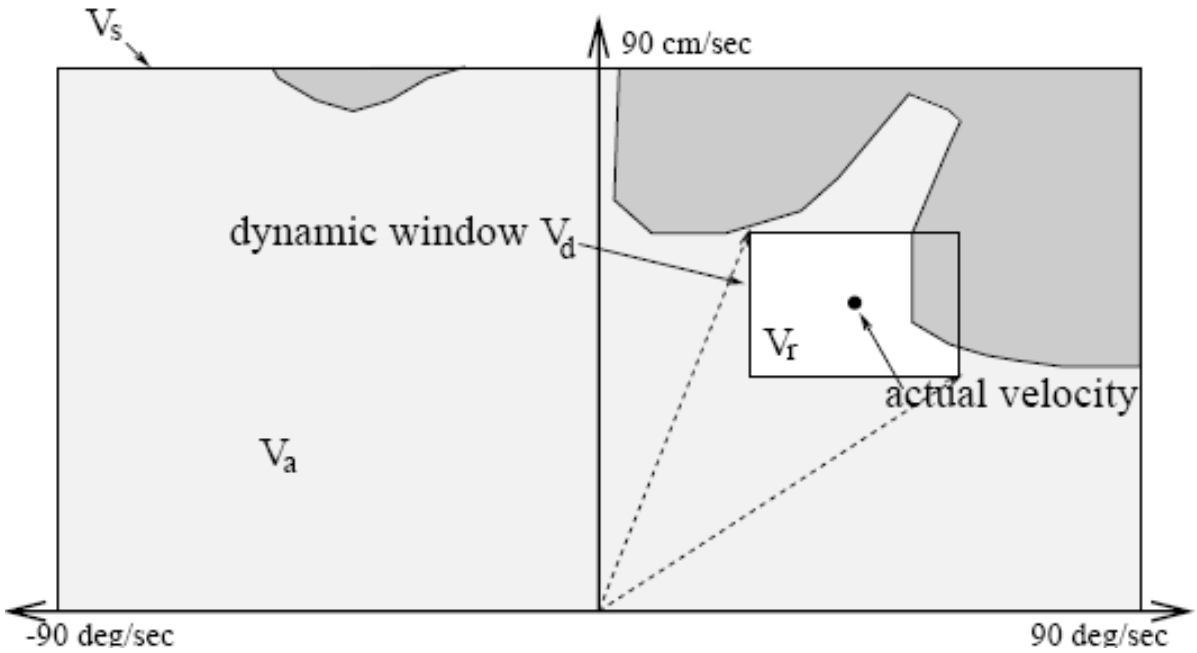
Obstacle Avoidance: Vector Field Histogram (VFH) III





- The VFH is formulated to work with probability obstacle distributions and thus is well adapted to work with uncertain sensors such as ultrasonic sonars.
- But the VFH has some limitations:
  - Might fail if narrow passageways (e.g. doors) have to be passed
  - Local minimum might not be avoided
  - Reaching of the goal can not be guaranteed
  - Dynamics of the robot not really considered
- Improved variants of the VFH algorithm have been developed (known as VFH+, VFH*).

Obstacle Avoidance: Dynamic Window Approach (DWA) I



- The kinematics of the robot is considered by searching a well chosen velocity space.
- *Circular trajectories*: DWA considers only circular trajectories uniquely determined by pairs (v, ω) of translational and rotational velocities.

Obstacle Avoidance: Dynamic Window Approach (DWA) II

- *Admissible velocities*: A pair (v, ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.

$$\mathcal{V} = \left\{(v, \omega) \in \mathbb{R}^2 : v \in [-v_{max}, v_{max}] \wedge \omega \in [-\omega_{max}, \omega_{max}]\right\}$$

- Velocities reachable by a short period of time:

$$\mathcal{V}_a = \left\{(v, \omega) : v \leqslant \sqrt{2dist(v, \omega)\ \dot{v}_b} \wedge\ \omega \leqslant \sqrt{2dist(v, \omega)\ \dot{\omega}_b}\right\},$$

  where $dist(v, \omega)$ is the distance to the obstacle, and $\dot{v}_b$ and $\dot{\omega}_b$ the maximum decelerations .

- *Dynamic window*: The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the limited accelerations of the robot

$$V_d = \{(v, \omega) : v \in [v_a - \dot{v}T,\ v_a + \dot{v}T], \wedge\ \omega \in [\omega_a - \dot{\omega}T,\ \omega_a + \dot{\omega}T]\},$$

Obstacle Avoidance: Dynamic Window Approach (DWA) III

- Resulting search space: the area V_r is defined as the intersection of the restricted areas, namely

$$V_r = V \cap V_a \cap V_d$$

- The problem is stated as the maximization of an objective function:

$$G(v, \omega) = \alpha_1 \cdot \mathrm{Goal}(v, \omega) + \alpha_2 \cdot \mathrm{Clearance}(v, \omega) + \alpha_3 \cdot Velocity(v, \omega),$$

  where $\alpha_i, i = 1, 2, 3$ are scaling factors.
- This function is a compromise among:
  - $\mathrm{Goal}(v, \omega)$, which favors velocities that offer progress to the goal.
  - $\mathrm{Clerance}(v, \omega)$, which favors velocities far from the obstacles.
  - $Velocity(v, \omega)$, that favors high speeds.