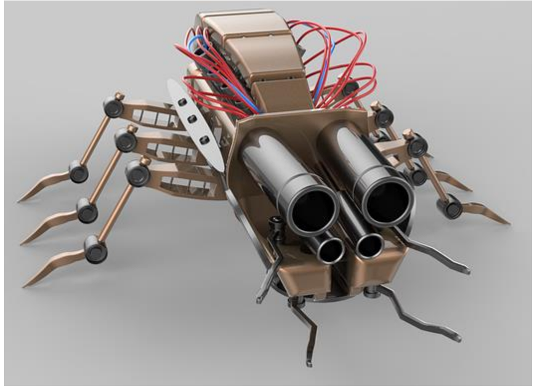
COMS W4733: Computational Aspects of Robotics

Lecture 12: Bug Algorithms 2



Slide materials from H. Choset, G. D. Hager, and Z. Dodds

Instructor: Tony Dear

Review: Bug Algorithms

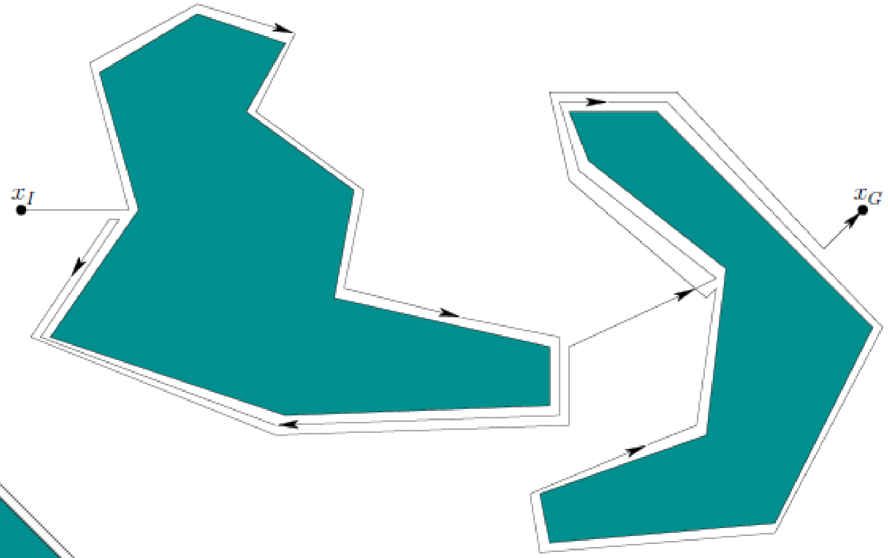
- Path planning algorithms for point particle robots in environments with obstacles
- Incremental, reactive planning with simple behaviors
- Sensing limited to tactile and range sensors

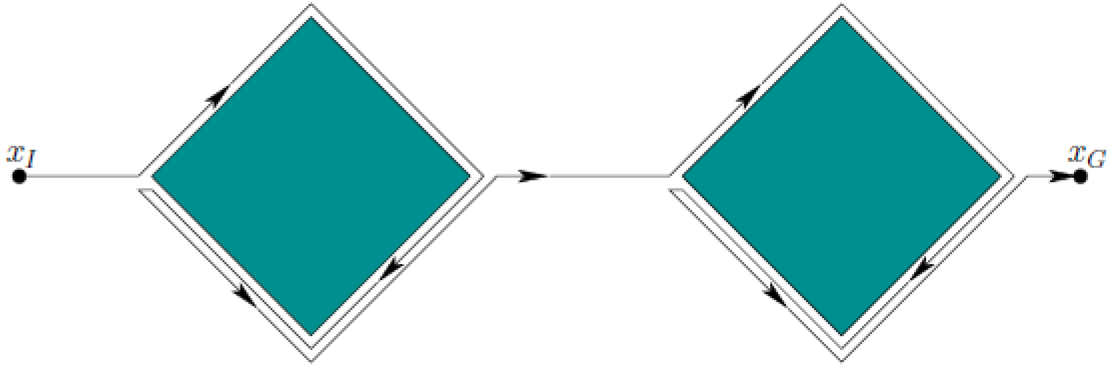
- Bug 1: Exhaustive search, traverse all obstacle boundaries
- Bug 2: Greedy search, head along m-line whenever possible

- Bug 2 often outperforms Bug 1 but can be less predictable
- Both are *complete*

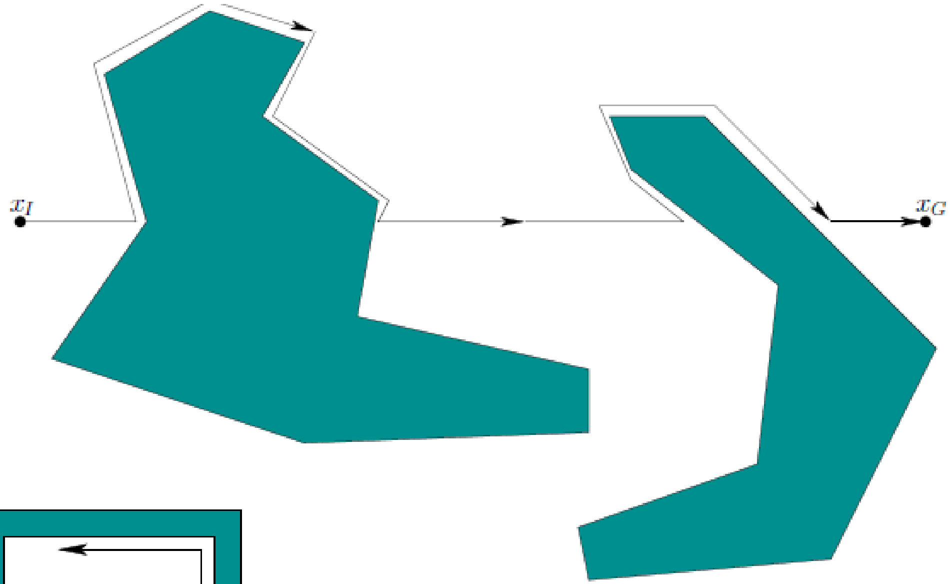
Bug 1

Distance $\leq D + 1.5 \sum_i P_i$

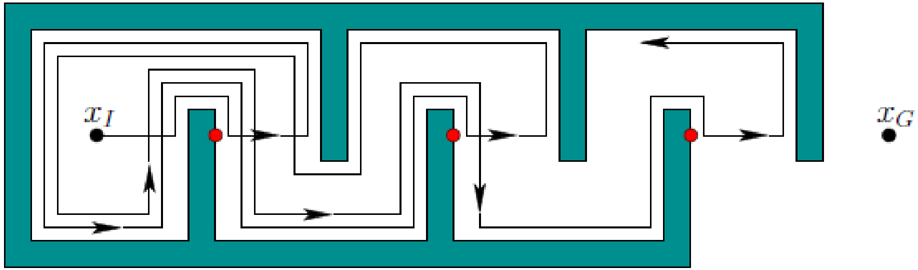




Bug 2



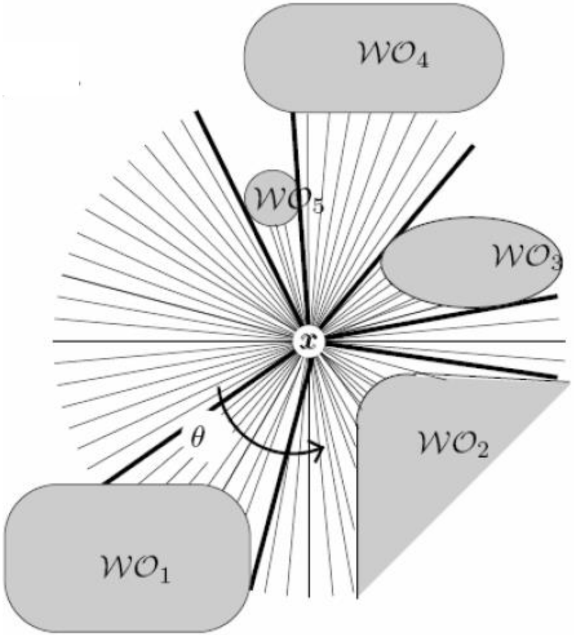
Distance $\leq D + 0.5 \sum_i n_i P_i$



Range Sensing

- Detect obstacles some finite distance away
- Infrared, sonar, radar, lidar…

- $\boldsymbol{x} = (x, y)^T$ is the robot's current position
- $\boldsymbol{p} = \boldsymbol{x} + \lambda(\cos\theta, \sin\theta)^T$ is any other point in the plane a distance λ away at an angle θ



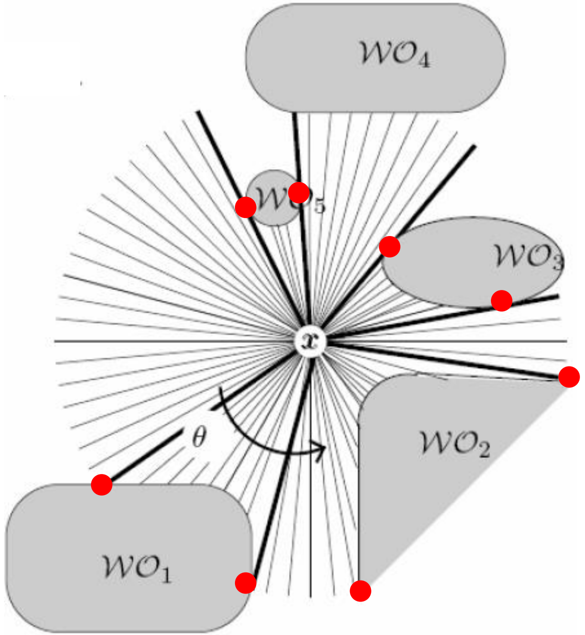
- Raw distance function

$$\rho(\boldsymbol{x}, \theta) = \min_{\lambda} d(\boldsymbol{x}, \boldsymbol{p}) \text{ s.t. } \boldsymbol{p} \in \cup_i \text{ Boundary}(WO_i)$$

- Saturated raw distance function $$\rho_R(\boldsymbol{x}, \theta) = \begin{cases} \rho(\boldsymbol{x}, \theta), & \rho(x, \theta) < R \\ \infty, & otherwise \end{cases}$$

Range Sensing

- Distance function $\rho_R(\boldsymbol{x}, \theta)$ is discontinuous!
- Values of θ at which ρ_R jumps values indicate limits of a sensed obstacle
- Obstacle boundary interpolated along continuous segment of finite ρ_R



$$\rho(\boldsymbol{x}, \theta) = \min_{\lambda} d(\boldsymbol{x}, \boldsymbol{p}) \text{ s.t. } \boldsymbol{p} \in \cup_i \text{ Boundary}(WO_i)$$

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

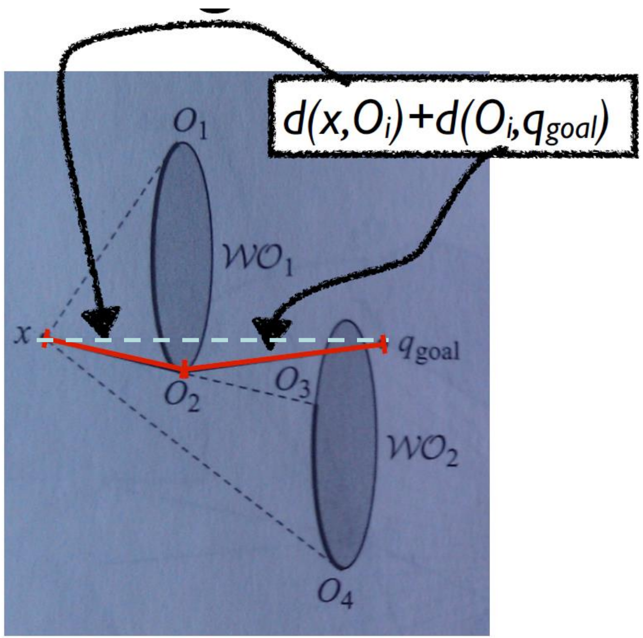
Range Sensing for Bug

- Suppose robot sees obstacle on path to goal
- No point in heading along this path as we'll eventually have to turn

- O_i: visible obstacle extents
- $d(\boldsymbol{x}, O_i)$: distance to any obstacle extent
- $d(O_i, \boldsymbol{q}_{goal})$: best distance from an obstacle extent to goal—this is an estimate!

- Idea: Minimize the heuristic (à la A*)

$$d(\boldsymbol{x}, O_i) + d(O_i, \boldsymbol{q}_{goal})$$



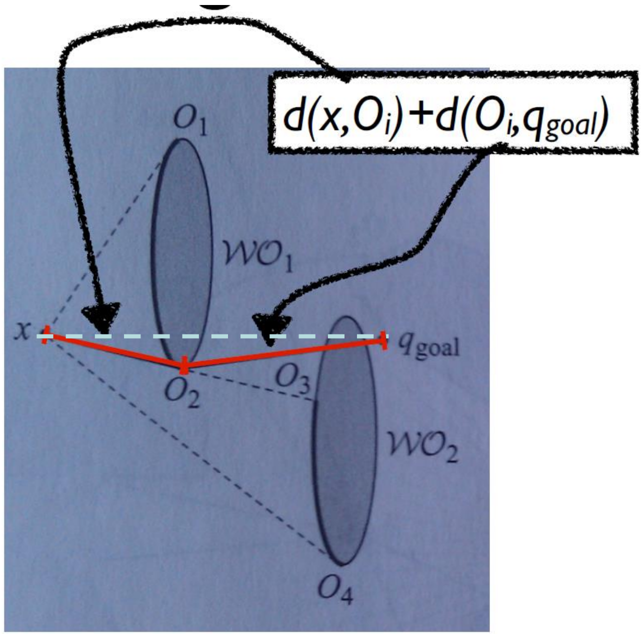
Quick A* Review

- Recall that A* is a *search* algorithm that finds a path from a start to a goal state
- All state transitions have associated costs

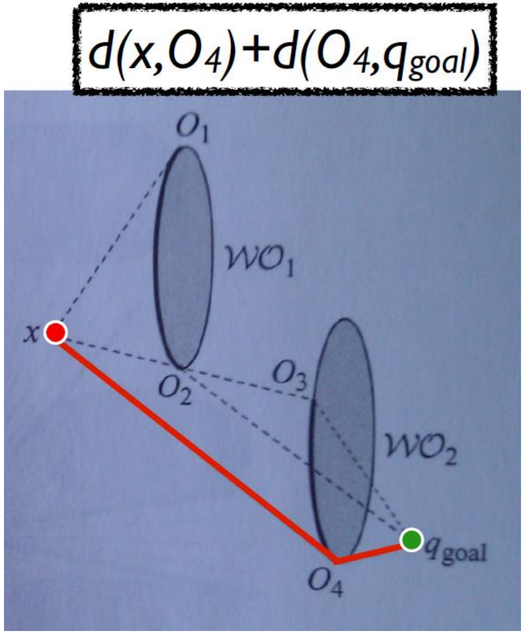
- A* expands nodes according to a sum of *backward* and *forward costs (heuristics)*
- Unlike A* though, Bug is moving while planning

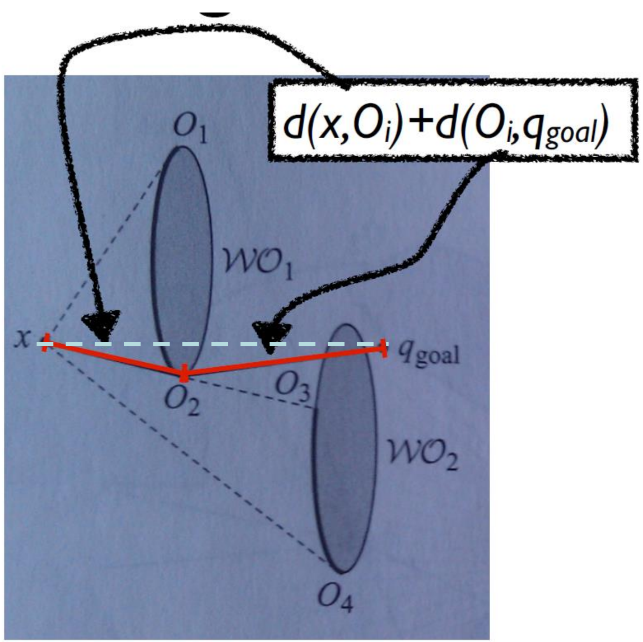
- Is $d(O_i, \boldsymbol{q}_{goal})$ admissible and consistent?

$$d(\boldsymbol{x}, O_i) + d(O_i, \boldsymbol{q}_{goal})$$



Range Sensing for Bug





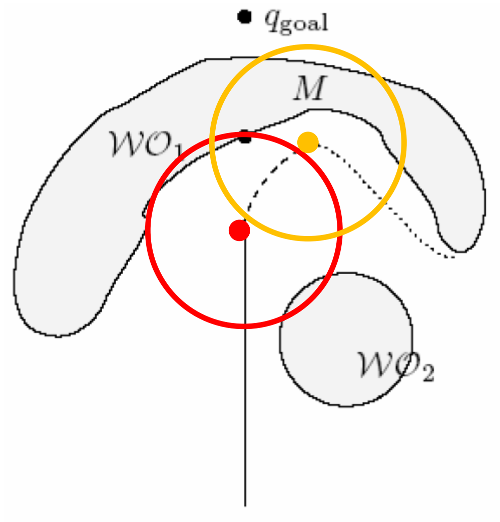
Tangent Bug

- As robot moves toward obstacle, O_i continually updates and moves along boundary
- Robot should keep moving toward O_i as it updates

- What if O_i starts moving away from the goal?
- Our "heuristic" will start increasing!
- Robot may be tempted to move back toward other extent O_j

- Instead, robot behavior should switch to "follow boundary"
- Same behavior as with Bug 1 and Bug 2

- Then when do we switch back to "head to goal"?



Switching Behaviors

- $d_{followed}$: shortest distance between current obstacle boundary and goal
- Distance from the point where our distance heuristic was minimal

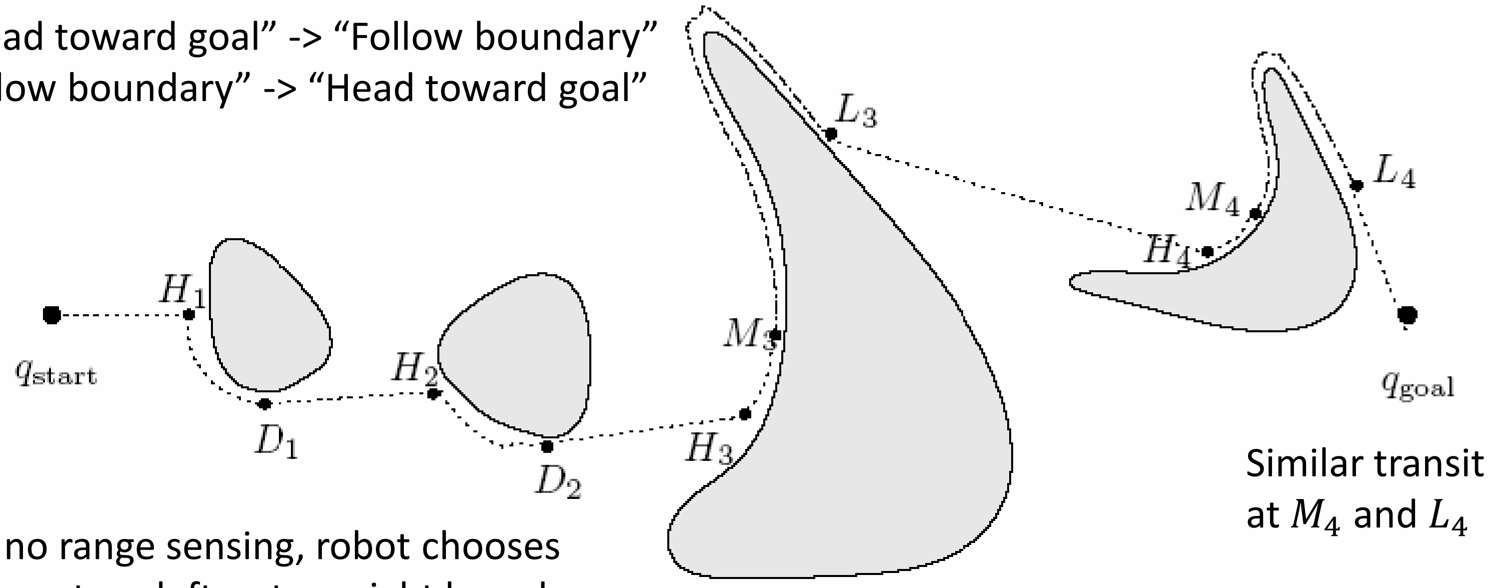
- d_{reach}: shortest distance between goal and closest free workspace point in range
- Typically will increase relative to $d_{followed}$ as heuristic increases, until robot starts heading to goal again or finds unobstructed path to goal

$$d_{reach} = \min_{c \in \Lambda} d(g_{goal}, c) \qquad \Lambda = \{y \in boundary(WO): \lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}$$

- Stop boundary following when $d_{reach} < d_{followed}$!

Example: Zero Sensor Range

M_3: “Head toward goal” -> “Follow boundary”
L_3: “Follow boundary” -> “Head toward goal”



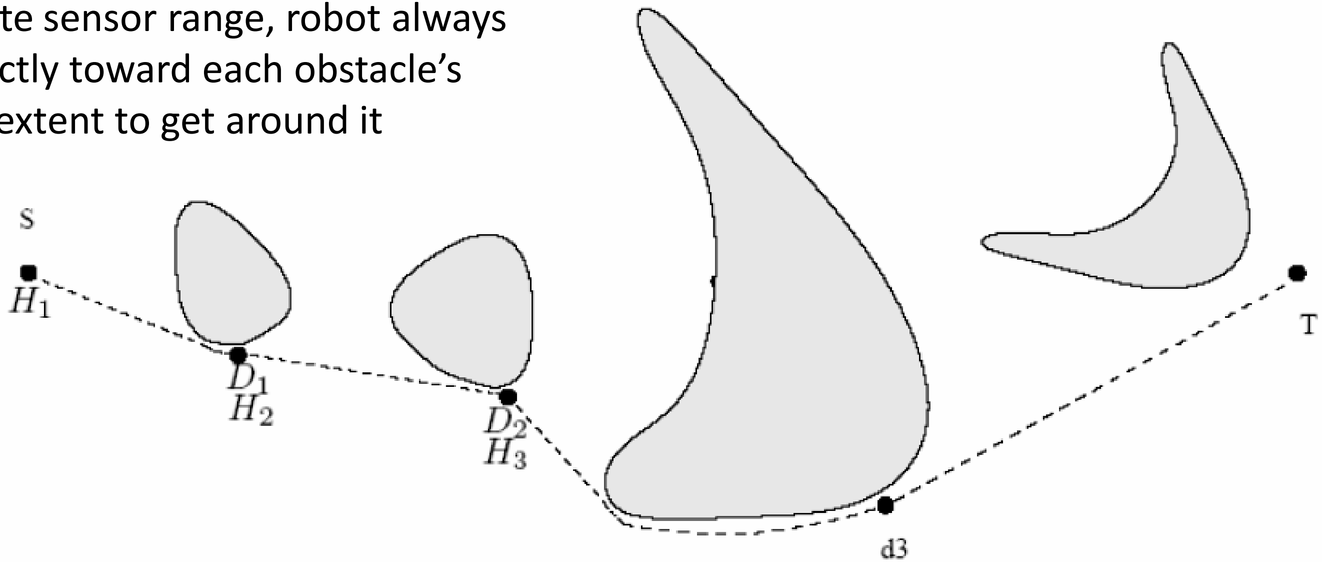
Similar transitions at M_4 and L_4

With no range sensing, robot chooses between turn left or turn right based on which direction gets us closer to goal

On left side of WO_3, M_3 is closest point to q_{goal}, i.e. $d_{followed} = d(M_3, q_{goal})$

Example: Infinite Sensor Range

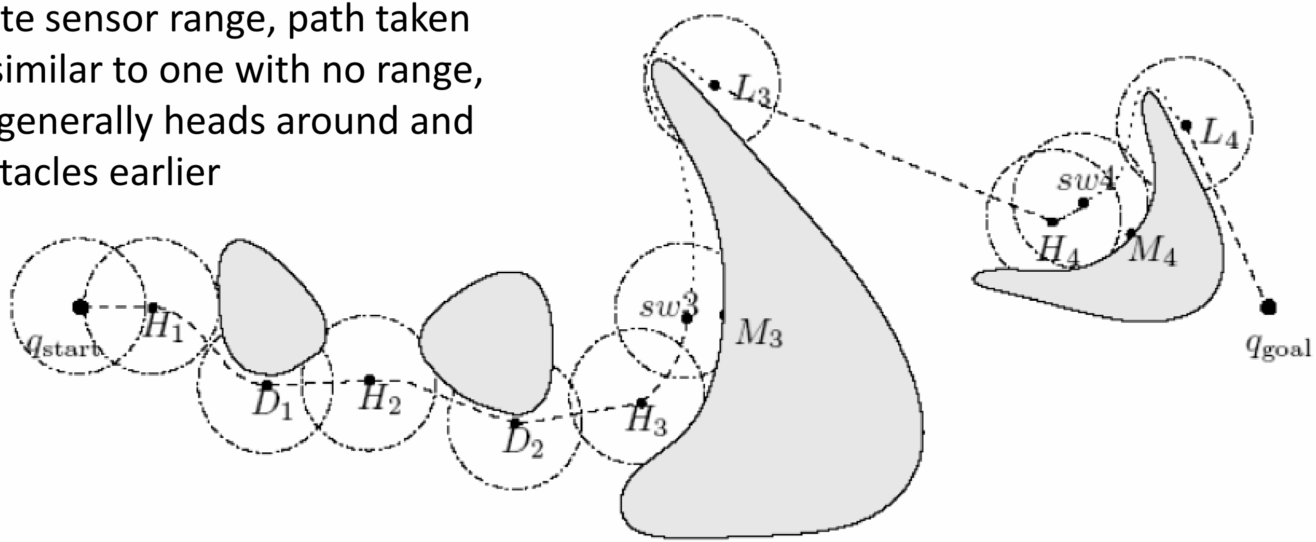
With infinite sensor range, robot always heads directly toward each obstacle's boundary extent to get around it



Still no guarantee of optimality! Infinite range but not when obstacles lie in line of sight

Example: Finite Sensor Range

With a finite sensor range, path taken may look similar to one with no range, but robot generally heads around and leaves obstacles earlier



Robot tends to hug obstacles less closely but still tracks around them in a "tangent" manner

Tangent Bug Algorithm

- Repeat:
  - Repeat "head toward goal":
    - Take sensor range readings, compute range segments
    - Move toward point $\boldsymbol{n} \in \{\boldsymbol{q}_{goal}, O_1, O_2, \dots\}$ that minimizes $d(\boldsymbol{x}, \boldsymbol{n}) + d(\boldsymbol{n}, \boldsymbol{q}_{goal})$
  - Until:
    - Goal is reached: Return success
    - Value of $d(\boldsymbol{x}, \boldsymbol{n}) + d(\boldsymbol{n}, \boldsymbol{q}_{goal})$ increases
  - Repeat "follow boundary":
    - Update discontinuity points $\{O_1, O_2, \dots\}$, distances $d_{followed}, d_{reach}$
  - Until:
    - Goal is reached: Return success
    - Obstacle is circumnavigated: Return failure
    - $d_{reach} < d_{followed}$

Implementing Boundary Following

- How to get robot to smoothly follow an obstacle's boundary?

- $D(\boldsymbol{x}) = \min_{\boldsymbol{c}} d(\boldsymbol{x}, \boldsymbol{c}), \boldsymbol{c} \in \cup\, Boundary(WO_i)$
- $G(\boldsymbol{x}) = D(\boldsymbol{x}) - W^*$, where W^* is some safe following distance
- Contour lines of $G(\boldsymbol{x})$ encircle the boundary and take on obstacle's shape

- $\nabla G(\boldsymbol{x})$ points radially away from the object
- We can just move orthogonal to this direction
- If $G(\boldsymbol{x})$ becomes negative, take a step in the direction of $\nabla G(\boldsymbol{x})$

Bug Variations

- Lots of variations and extensions: Alg1, Alg2, DistBug, Class1, Rev1, Rev2, VisBug, …

- May perform more computations for more optimal paths
- Different algorithms excel in different environments (no clear winner)

- Example strategies:
  - Find ways to leave current obstacle earlier
  - Constrain search space to perform less computation
  - Alternate between turning left and turning right

Practical Issues

- How does the robot know where on the map it is?
- How does Bug 2 find the m-line after deviating from it?

- **Odometry**: Using sensors to estimate position over time (*dead reckoning*)
- No way to “recalibrate” measurements against a known ground truth!
- Very susceptible to buildup of errors over time

- Robot may start out with a map of the world with known **landmarks**
- Landmarks can be a source of calibration

Landmark Navigation

- Some landmarks require some degree of understanding or object identification
- E.g.: Visual landmarks that look different at different angles or lighting

- Others do not require as much comprehension from robot's perspective
- E.g. heat, light, sound intensities

- In general, robot should have some internal model of any landmark
- Landmarks should also be stationary or move in predictable ways

- Combination with sensor readings help determine robot estimates

Summary

- Range sensing allows robot to react sooner and avoid unnecessary obstacle tracking
- Optimize between what we can see and what we can predict
- Still maintain only simple behaviors: “head to goal”, “follow boundary”
- Lots of different strategies to better optimize to particular environments
- Also many other issues in practice: how to incorporate different sensor inputs, how to maintain and recognize map and objects therein, etc