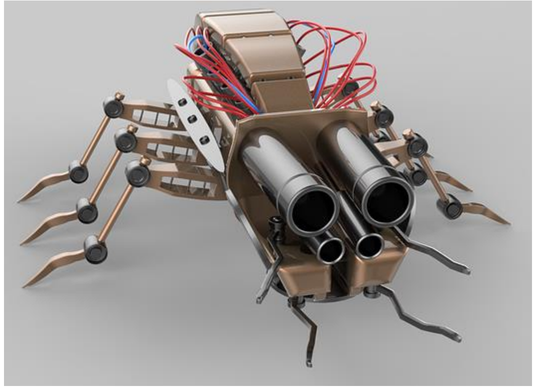
COMS W4733: Computational Aspects of Robotics

Lecture 11: Bug Algorithms 1



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

Instructor: Tony Dear

Planning Algorithms

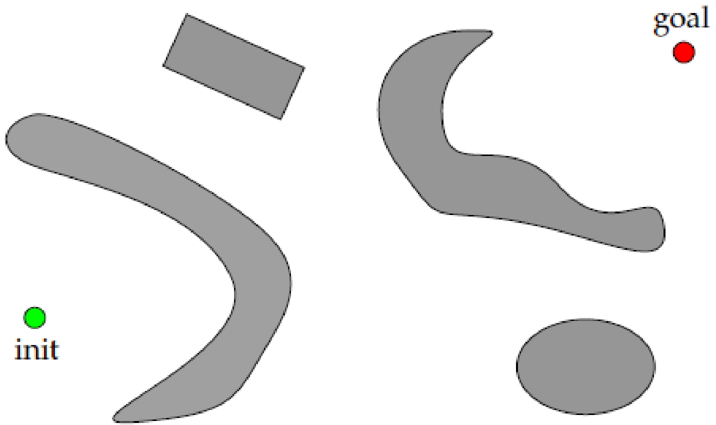
- Up to now: Continuous trajectories represented as mathematical functions
- Easy to formulate, guaranteed solution given initial and final conditions
- But *extremely* limited to obstacle-free, static, fully observable environments!

- Robots operate in the real world
- Full of obstacles (possibly dynamic), oftentimes not fully observable
- Plans and trajectories need to update in response to environment feedback

- Choice of algorithms will depend on what we know and what we don't

Bug Algorithms

- How do bugs traverse the environment?
- Assume only local knowledge of surroundings
  - Tactile sensing
  - Finite distance sensing
- Behaviors are simple and can be enumerated
  - Follow a wall
  - Move toward a goal
- Overall planning is *incremental* and *reactive*

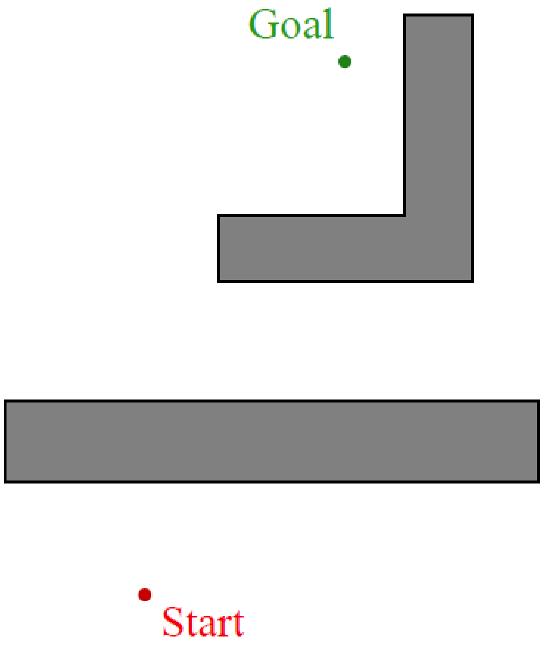


Environment Setup

- Assume point robot—not worrying about kinematics or constraints here!
- Environment is bounded, finite number of obstacles

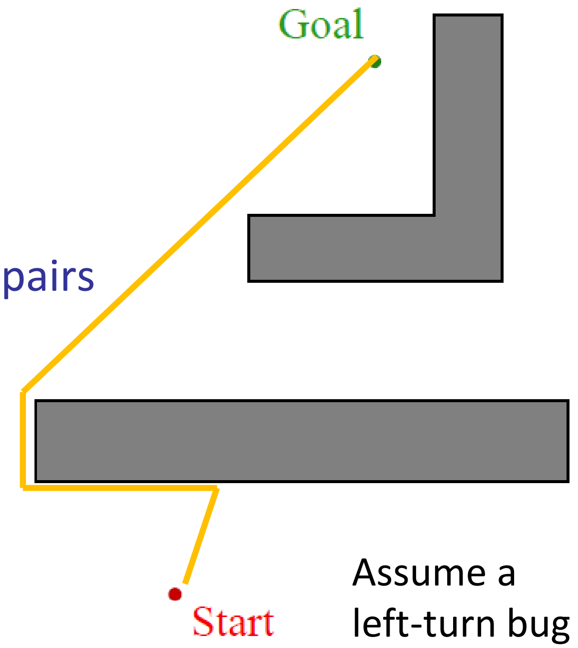
- Robot knows its global position
- Robot is able to measure distance between points
  - Does not know layout of obstacles

- Workspace: (x, y) or $(x, y, \theta) \in W$
- Set of obstacles: WO_i
- *Free workspace*: $W_{free} = W - \cup_i WO_i$



Bug 0

- If no obstacles, best path to the goal is a simple straight line
- May encounter obstacles between q_{start} and q_{goal}
  - *Hit point* on obstacle i: q_i^H
  - *Leave point* on obstacle i: q_i^L
- *Path* can be represented as a sequence of hit-leave point pairs
- While not at goal:
  - If at obstacle according to sensors:
    - Follow obstacle until we can head toward goal
  - Else: Head toward goal



Assume a left-turn bug

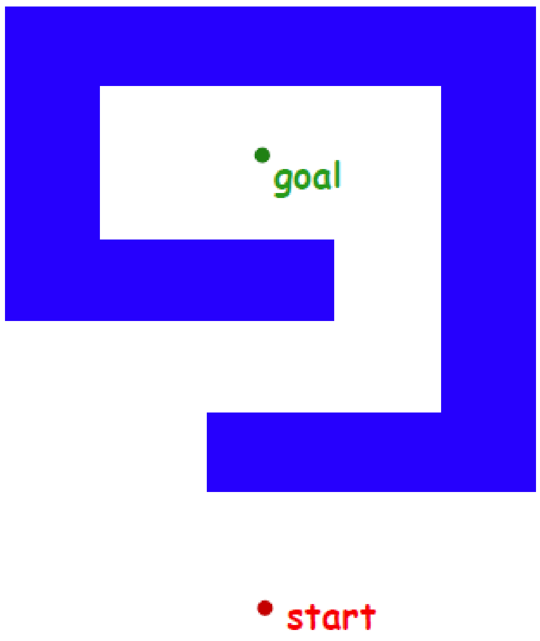
Bug 0

- When does bug 0 fail?

- Drawback: Bug 0 has no memory
- Going around in loops can be a problem

- Can we improve Bug 0 without adding any features?

- What if we always turn right rather than left?
- Unfortunately we wouldn't know when to switch

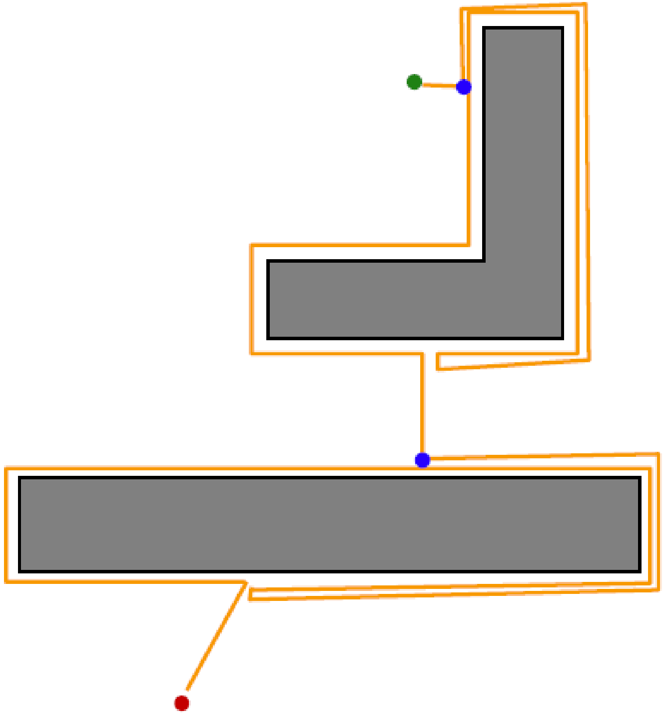


Bug 1

- How to improve Bug 0 if we had memory?
- We should realize if we have circumnavigated an obstacle—remember q_i^H

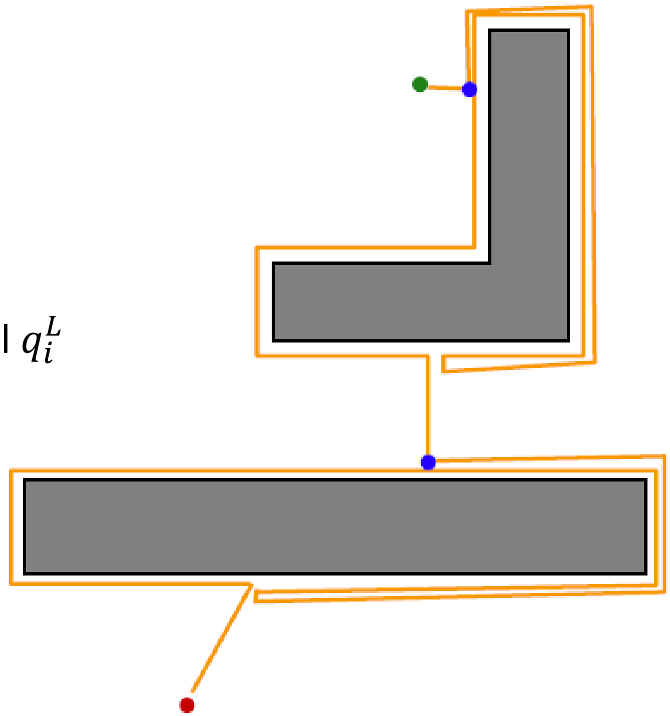
- What else should we keep track of?
- If we have seen the obstacle's entire perimeter, we would also know the best point to leave it

- Once we reach q_i^H again, return to that closest point and move forward

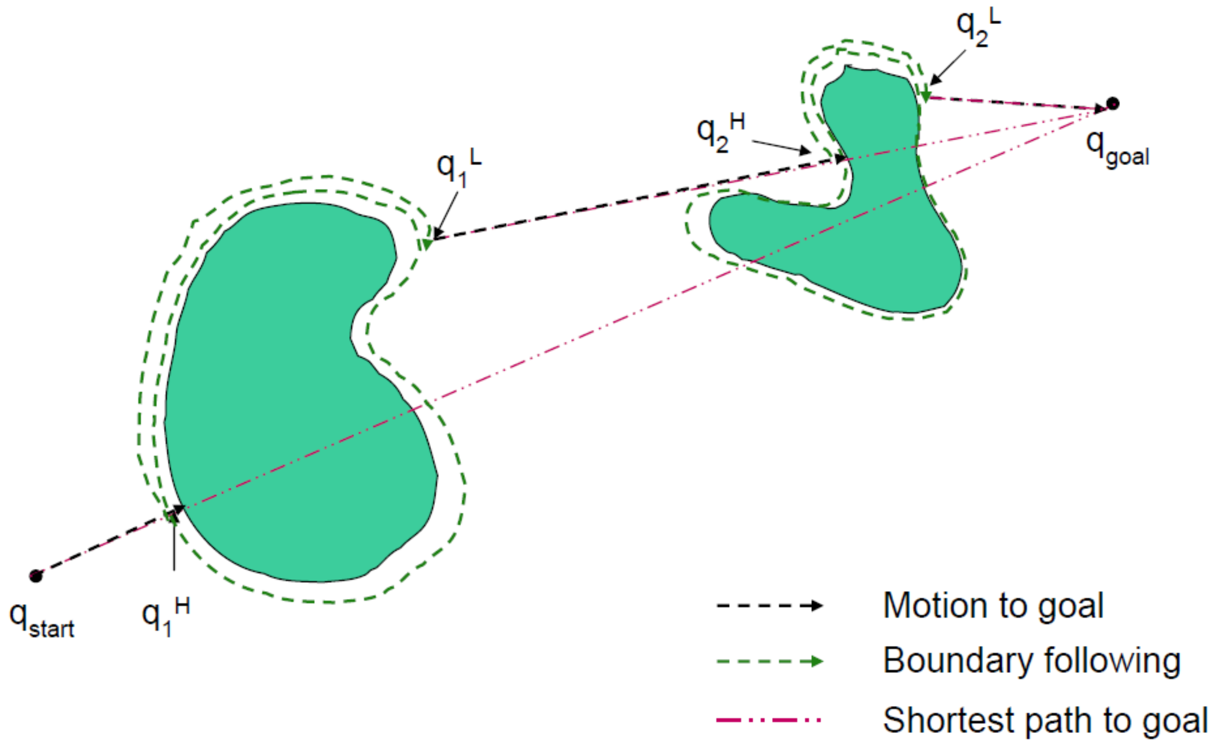


Bug 1

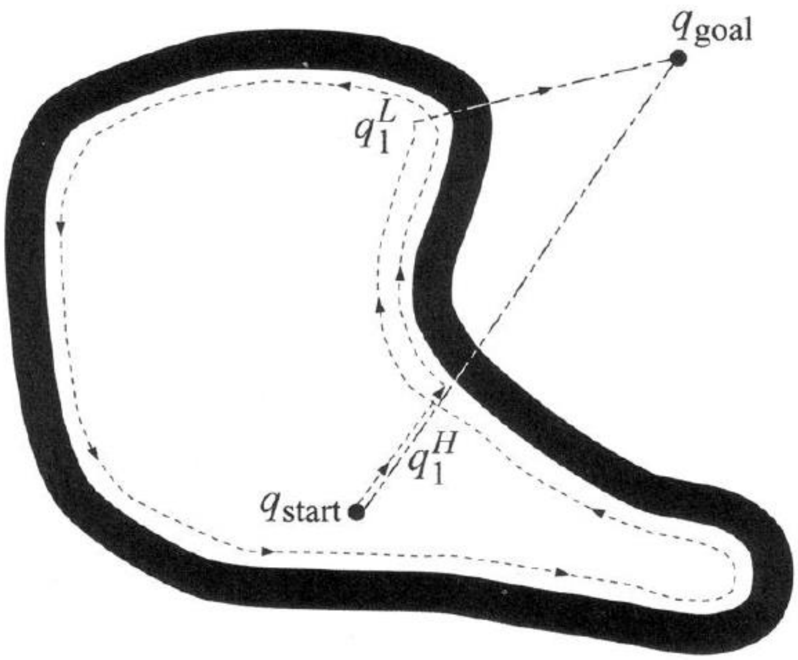
- $q_0^L \leftarrow q_{start}, i = 1$
- Loop:
  - Repeat: From q_{i-1}^L move toward q_{goal}
  - If goal reached: Exit and return success
  - If obstacle WO_i encountered at q_i^H:
    - Circumnavigate WO_i and record closest point to goal q_i^L
    - If goal reached: Return success
    - If q_i^H re-encountered: return to q_i^L
  - If move toward q_{goal} encounters obstacle:
    - Return failure
  - Else: $i \leftarrow i + 1$



Bug 1 Examples



Bug 1 Examples



Bug 1 Completeness

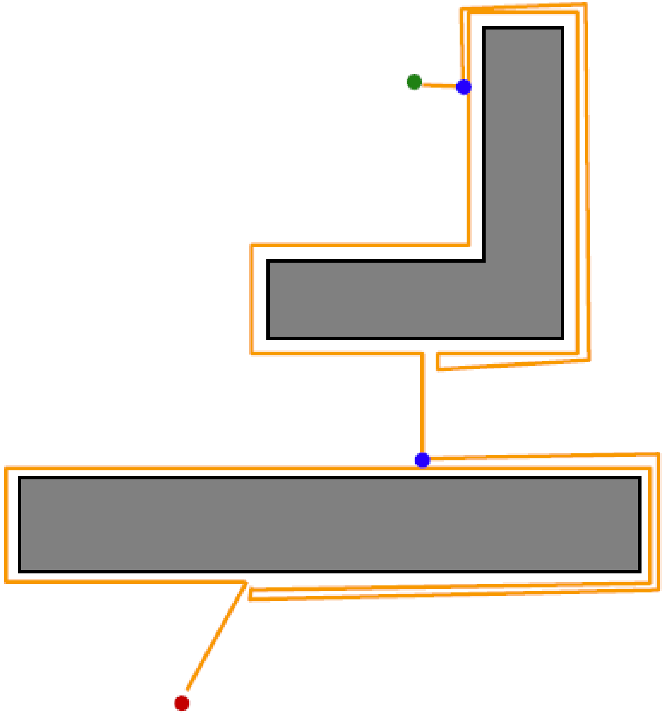
- Suppose Bug 1 never terminates…
  - But each leave point is closer to the goal than the hit point, since we pick q_i^L to be the closest point on the obstacle: $d\left(q_i^L, q_{goal}\right) < d(q_i^H, q_{goal})$
  - Each hit point is also closer to the goal than the last hit point, since the robot moved toward the obstacle from q_{i-1}^L to arrive at q_i^H
  - There a finite number of hit/leave point pairs, which the robot will eventually exhaust
- Suppose Bug 1 incorrectly returns failure…
  - Then it will attempt to leave from the closest leave point q_i^L that will run into an obstacle
  - But there must be at least one other possible leave point $q_i^{L^*}$ on the obstacle along line to goal
  - Since a path exists, robot would have encountered $q_i^{L^*}$ while circumnavigating obstacle and would not leave from q_i^L

Bug 1 Performance

- What are the best- and worst-case scenarios?

- Best case: Robot goes straight to goal
- Distance = D (distance from start to goal)

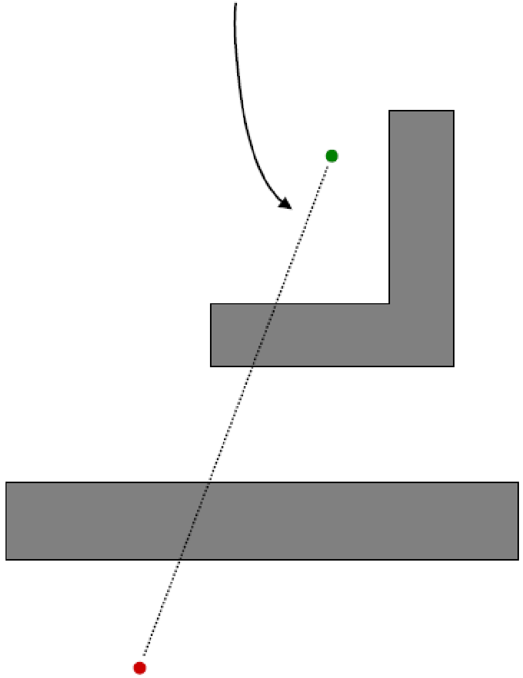
- Worst case: Robot travels D distance, plus completely circumnavigating all obstacles
- Distance = $D + 1.5 \sum_i P_i$
- $P_i =$ perimeter of WO_i



Bug 2

- Do we really need to explore every obstacle's perimeter all the time?
- We already had some idea of the "shortest path" to the goal from the start!
- New idea: Just follow the **m-line** whenever we are able to instead of computing new shortest paths
- Only need to remember m-line and hit points

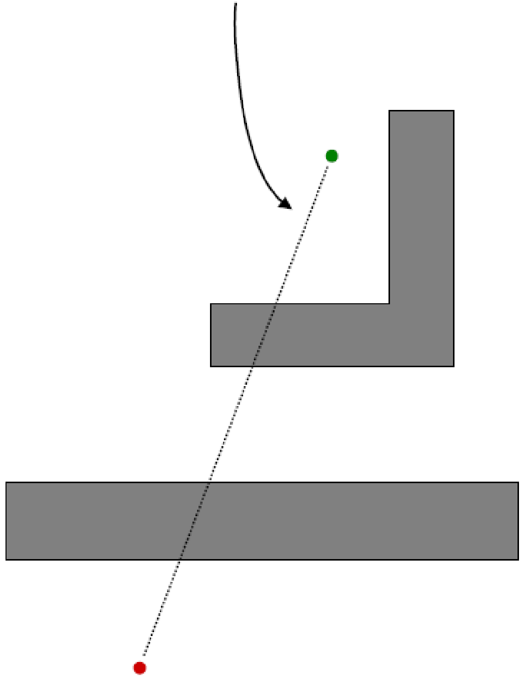
Call the line from the starting point to the goal the ***m-line***



Bug 2

- $q_0^L \leftarrow q_{start}, i = 1$
- Loop:
  - Repeat: From q_{i-1}^L move toward q_{goal} along m-line
  - If goal reached: Exit and return success
  - If obstacle WO_i encountered at q_i^H:
    - Repeat: Turn left or right and follow boundary of WO_i
      - If goal reached: Return success
      - If q_i^H re-encountered: Return failure
      - If m-line encountered at m s.t. $d(m, q_{goal}) < d(q_i^H, q_{goal})$ and line(m, q_{goal}) does not encounter obstacle:
        - Stop following WO_i
  - $q_i^L \leftarrow m; i \leftarrow i + 1$

Call the line from the starting point to the goal the ***m-line***



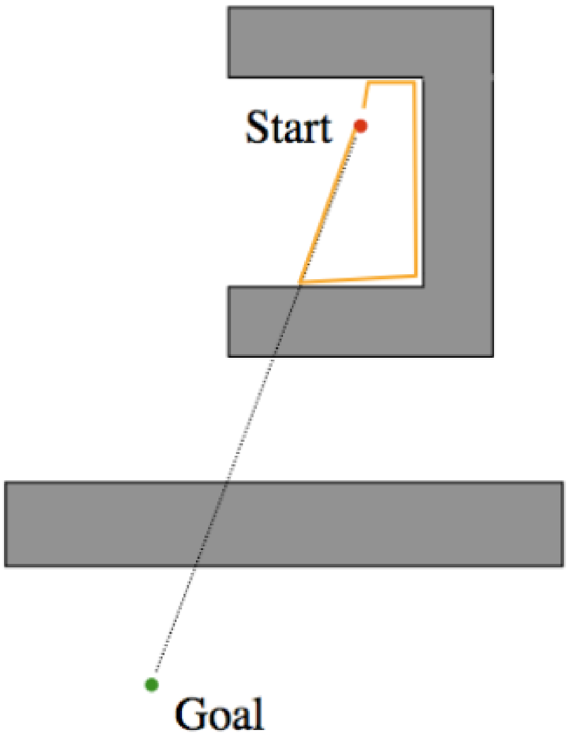
Bug 2

- Why do we need the distance condition when circling WO_i?
- If m-line encountered at m s.t. $\boldsymbol{d}(\boldsymbol{m}, \boldsymbol{q}_{goal}) < \boldsymbol{d}(\boldsymbol{q}_i^H, \boldsymbol{q}_{goal})$ and line(m, q_{goal}) does not encounter obstacle

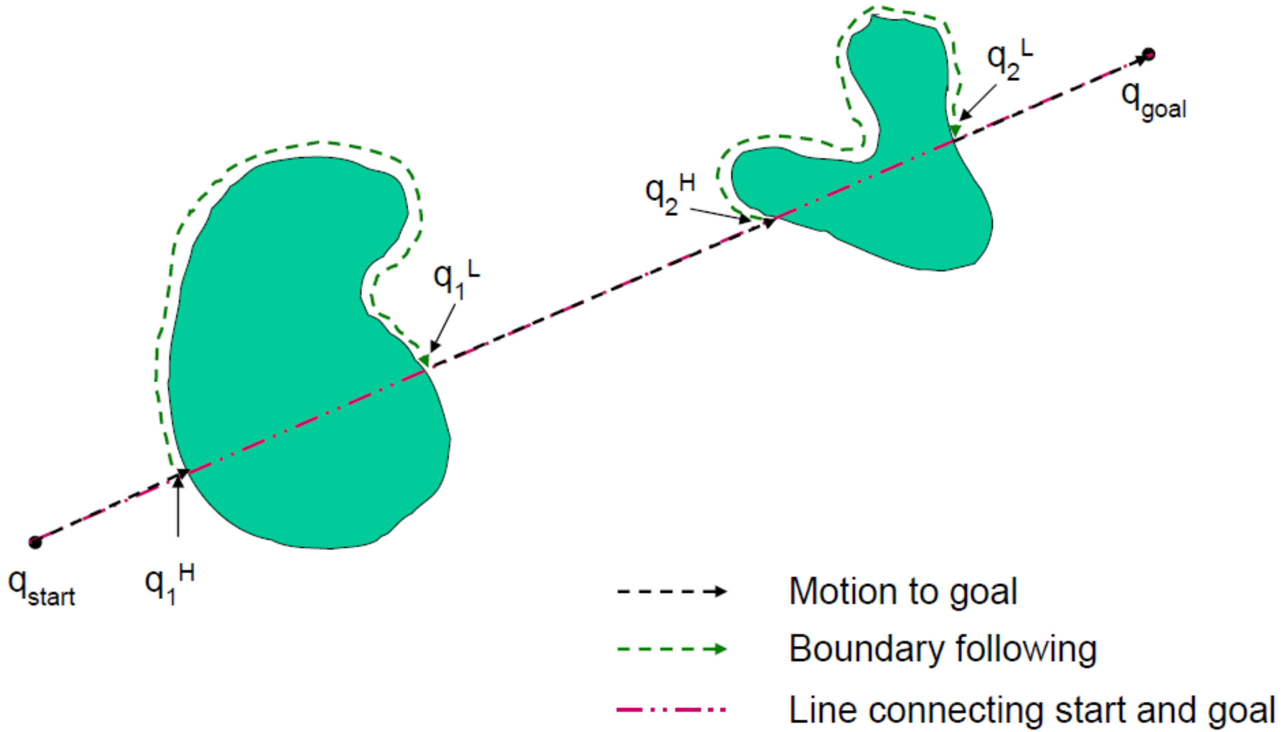
- Is it possible to re-encounter m-line such that we're actually farther away from the goal than when we started?

- In other words, can we somehow follow an obstacle to move behind the start or hit point?

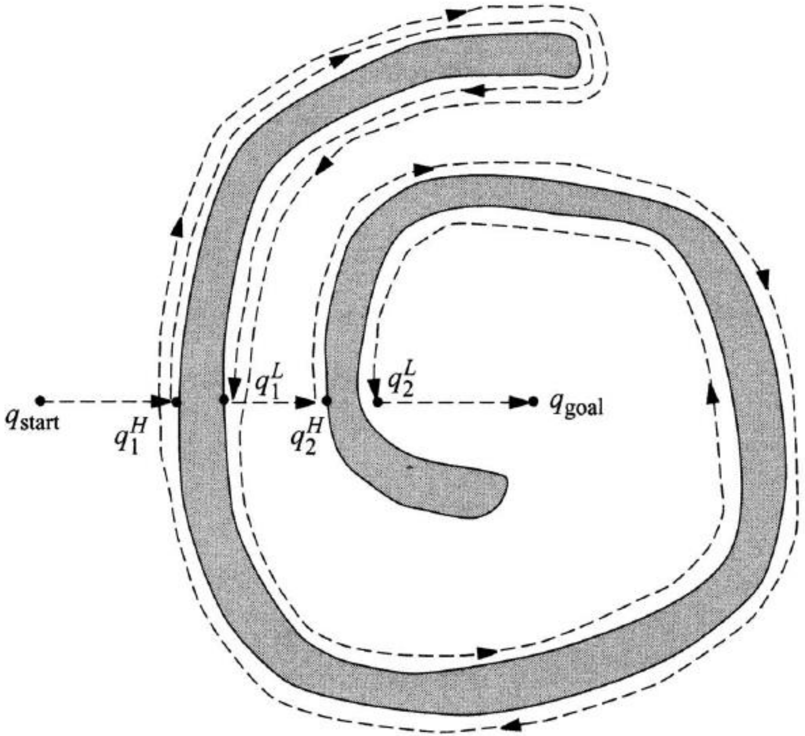
- Distance condition ensures we don't get stuck in infinite loop!



Bug 2 Examples

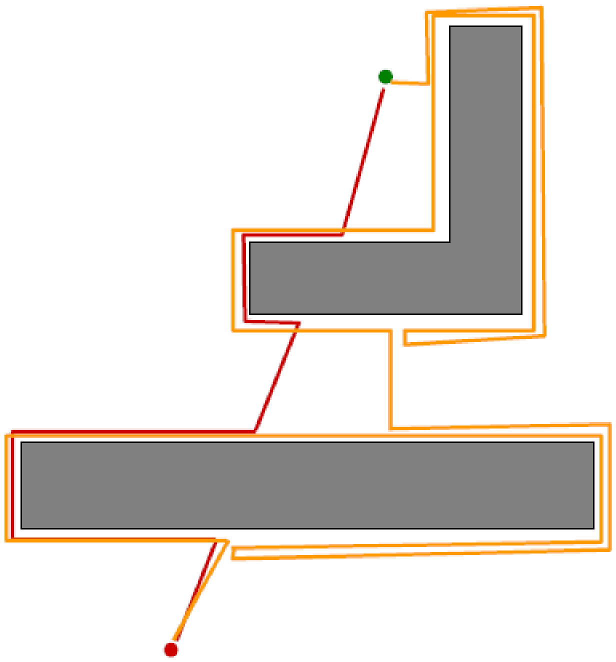


Bug 2 Examples

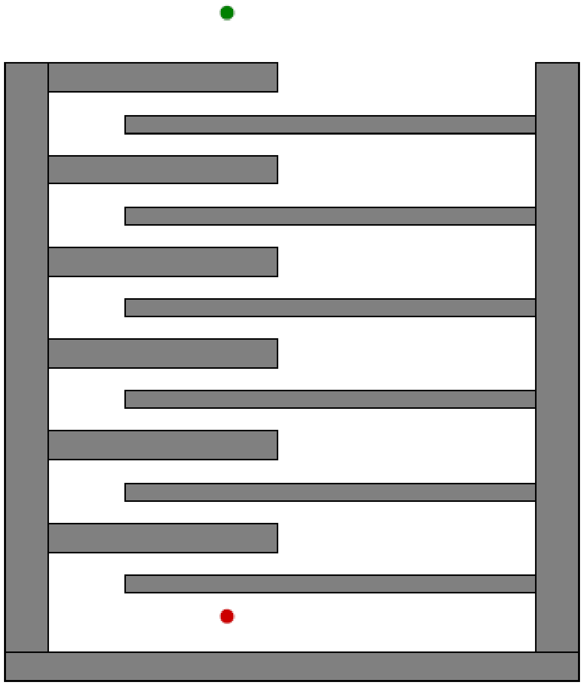


Bug 1 vs Bug 2

Bug 2 beats Bug 1?



Bug 1 beats Bug 2?

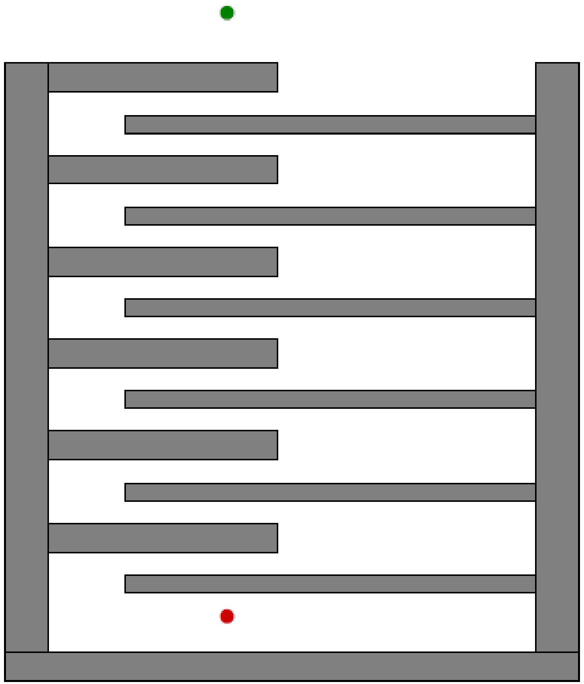


Bug 2 Performance

- Best case: Robot goes straight to goal, distance = D
- Same as Bug 1

- What is the worst case?
- Each time we follow the m-line and find a hit point q_i^H, we could potentially traverse the perimeter!
- Suppose n_i = number of intersections with WO_i

- Worst case: distance $= D + \frac{1}{2}\sum_i n_i P_i$



Bug Algorithm Properties

- Bug 1 is an exhaustive search algorithm
- Circumnavigate every obstacle, check all options before choosing the best one

- Bug 2 is a greedy search algorithm
- Takes the first thing that looks good—leave points along stored m-line

- Bug 2 can outperform Bug 1 in many situations, but not always
- Bug 1 has more predictable performance and results

- Both algorithms are *complete*—either return a path or failure if none exists