Lecture 10
Planning - II - Bugs

Course Logistics

- Project 4 was posted on 02/19 and will be due on 03/05.
  - Start early!
- Quiz 5 will be posted tomorrow at noon and will be due at noon on Wed.

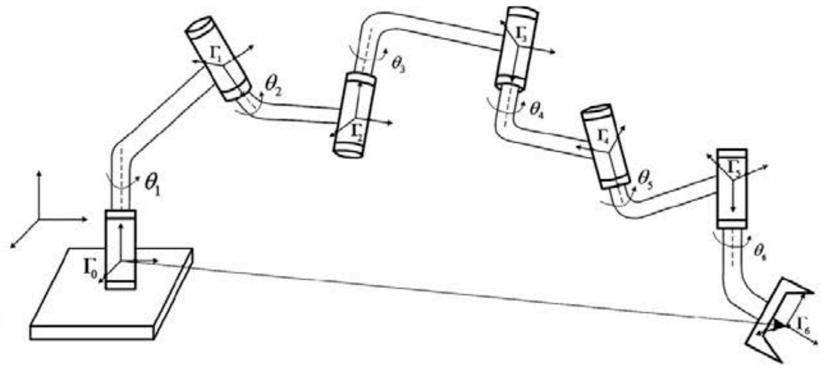
Previously in Manipulation Lectures

Robot Kinematics

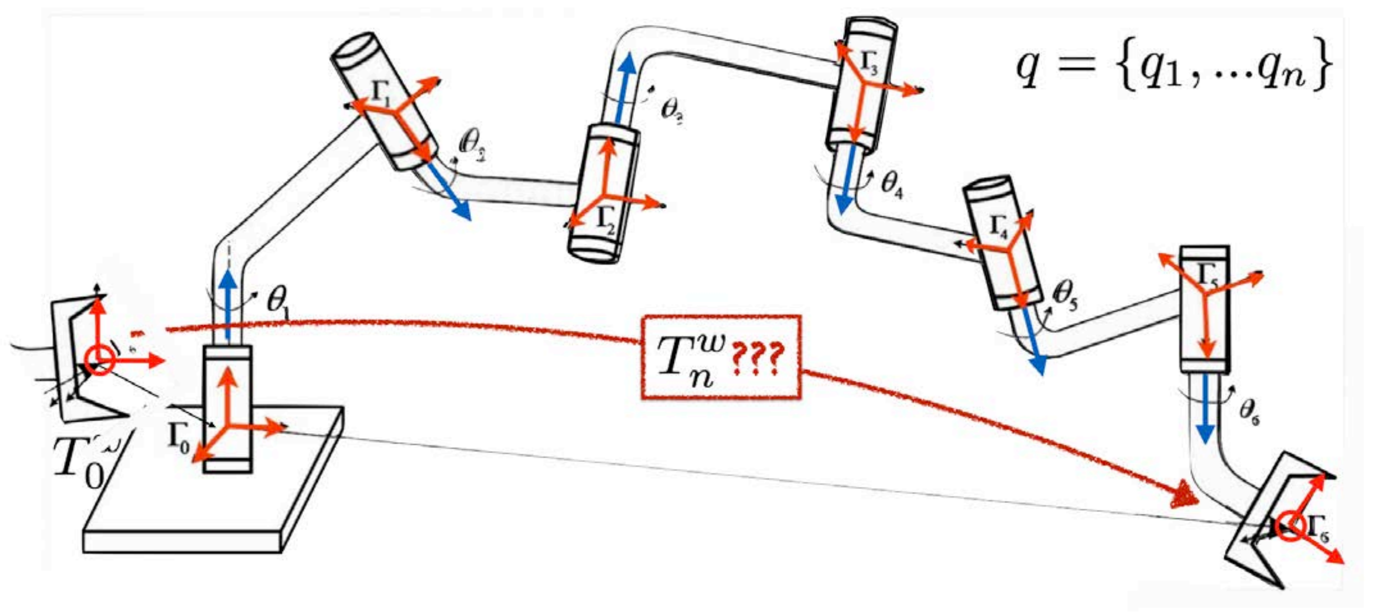
Goal: Given the structure of a robot arm, compute

– **Forward kinematics**: infer the pose of the end-effector, given the state of each joint.

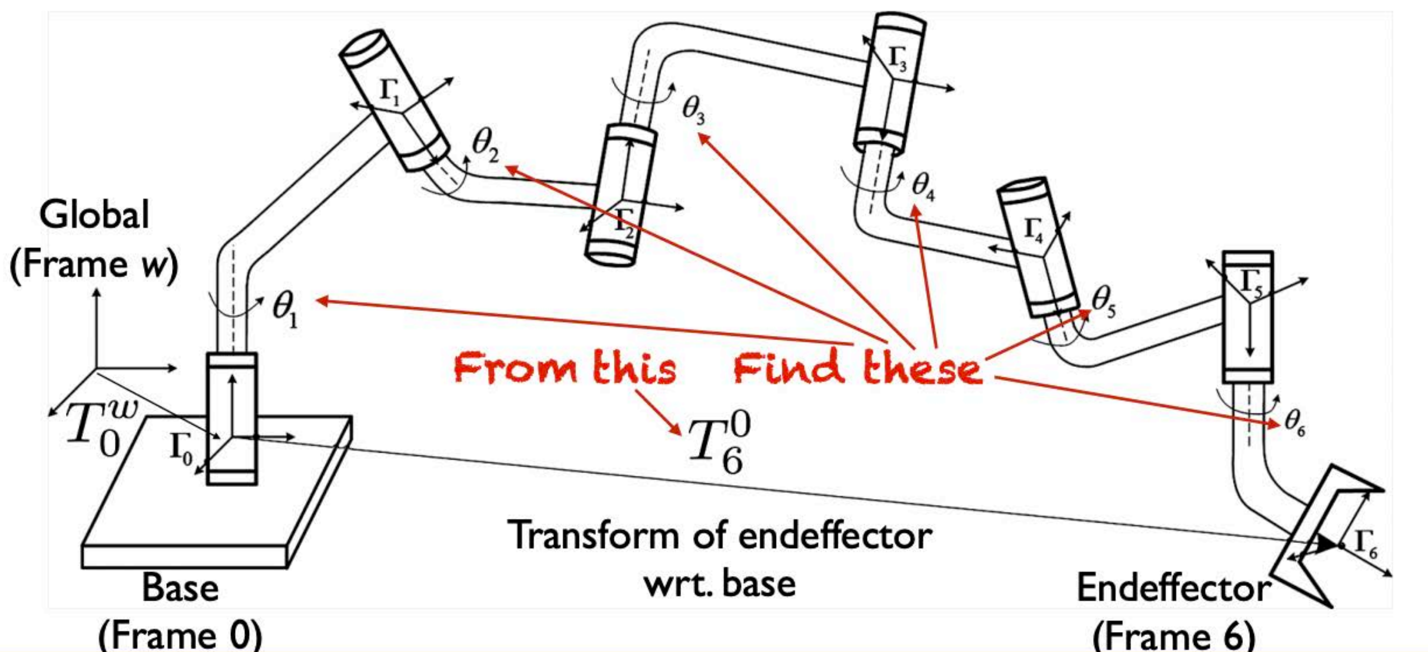
– **Inverse kinematics**: infer the joint states to reach a desired end-effector pose.



Forward kinematics restated: Given **q**, *find* T^w_n;
T^w_n *transforms endeffector into workspace*



Inverse kinematics: how to solve for q = {θ1, …,θN} from T^0_N?



Inverse Kinematics: 2 possibilites

- **Closed-form solution**: geometrically infer satisfying configuration
  - *Speed*: solution often computed in constant time
  - *Predictability*: solution is selected in a consistent manner
- **Solve by optimization**: minimize error of endeffector to desired pose
  - often some form of Gradient Descent (a la Jacobian Transpose)
  - *Generality*: same solver can be used for many different robots

Robot arm and its Jacobian

Lets focus on i-1th frame

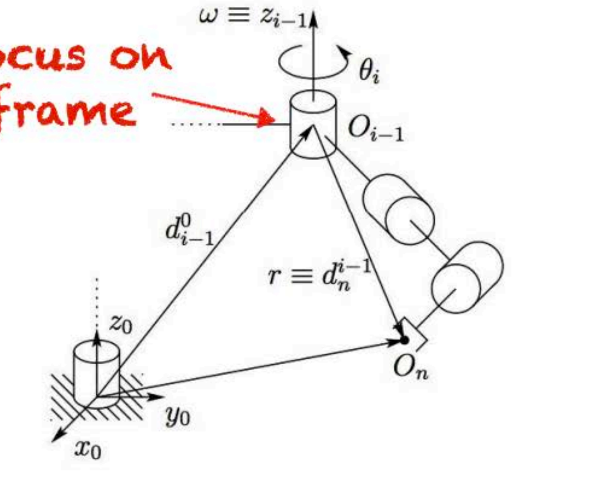


Figure 5.1: Motion of the end-effector due to link *i*.

i-1th frame maps to ith column in

The Jacobian

A 6xN matrix

$$J = [J_1 J_2 \cdots J_n]$$

consisting of two 3xN matrices

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

J_i for a prismatic joint

$$J_i = \begin{bmatrix} z_{i-1} \\ 0 \end{bmatrix}$$

J_i for a rotational joint

$$J_i = \begin{bmatrix} z_{i-1} \times (o_n - o_{i-1}) \\ z_{i-1} \end{bmatrix}$$

How to use this Jacobian for IK as optimization?

IK Procedure restated:

compute endpoint error: $\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n$

compute step direction: $\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n$

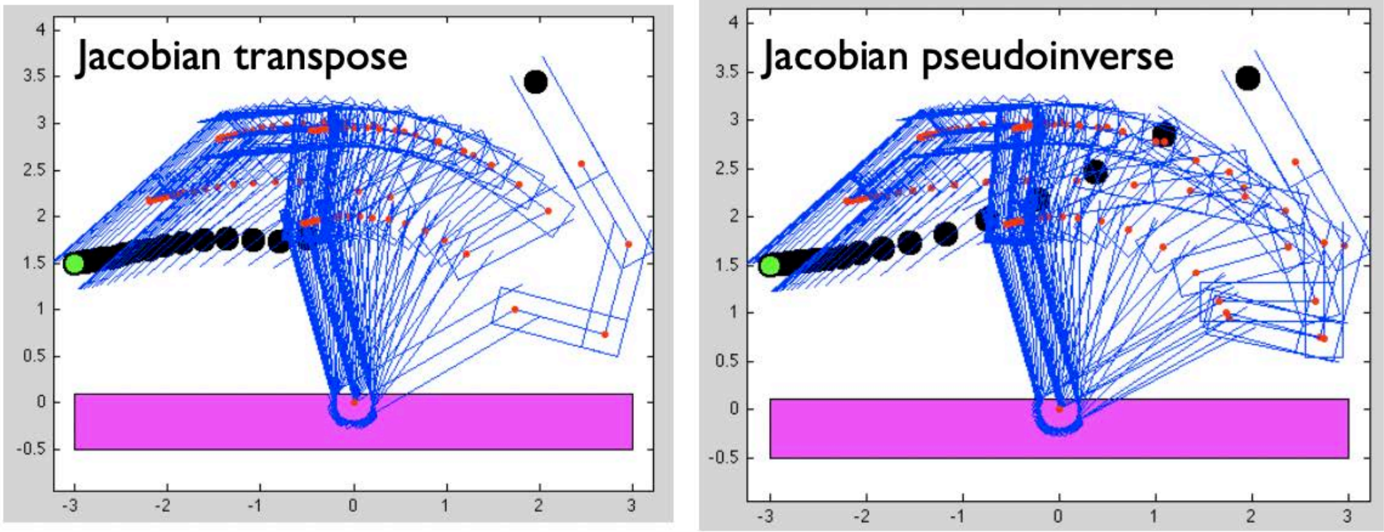
perform step direction: $\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n$

repeat

Check point:

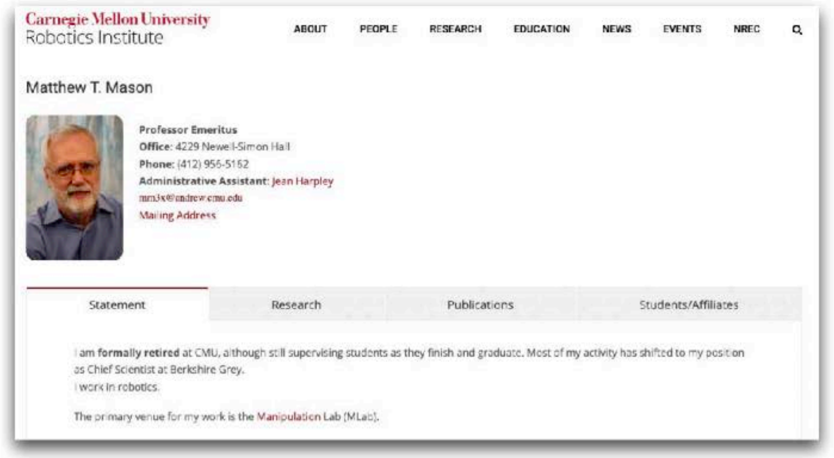
How will you get x_{n+1} given q_{n+1}?

Matlab 5-link arm example: Jacobian transpose



Definition of **Manipulation**

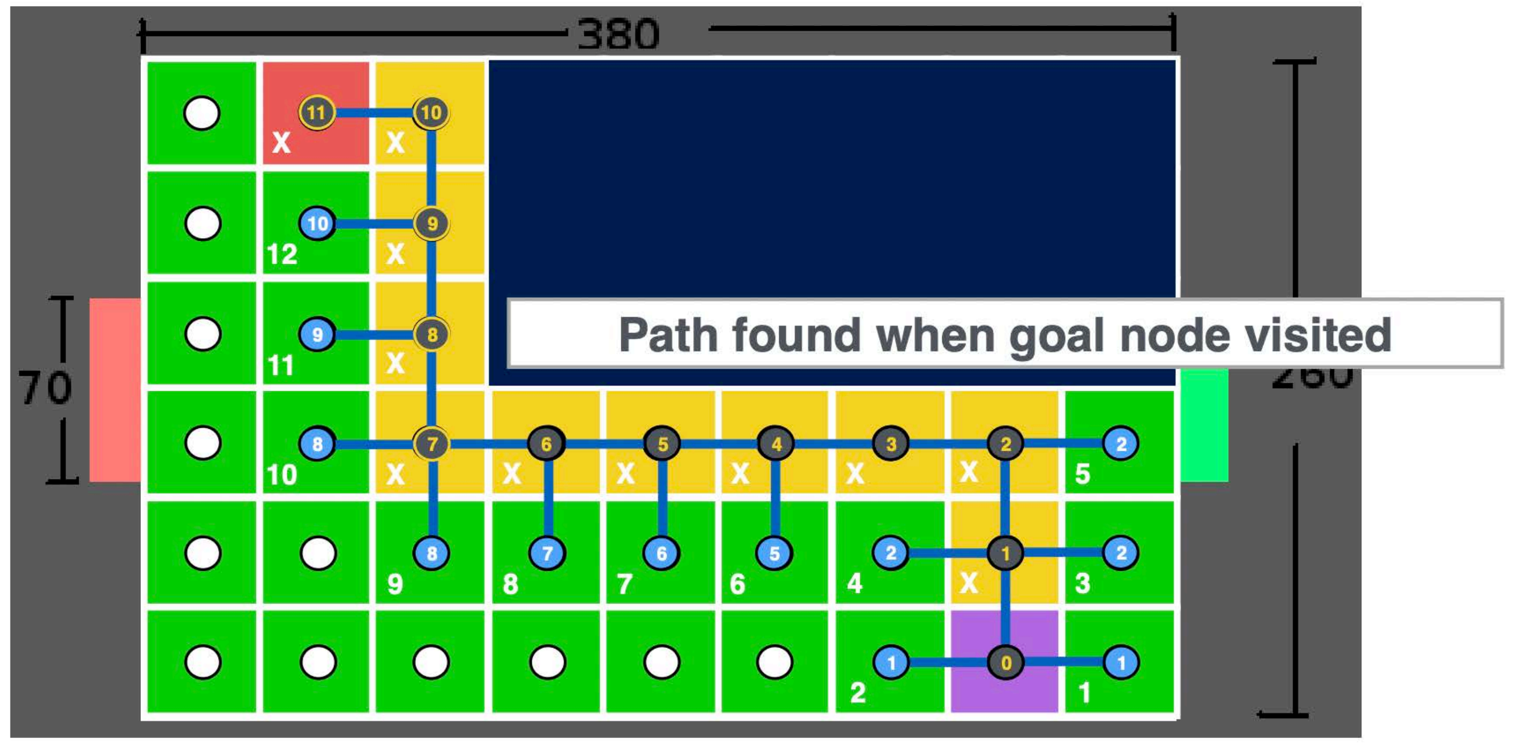
Mason, Matthew T. "Toward robotic manipulation."
Annual Review of Control, Robotics, and Autonomous Systems 1 (2018): 1-28.



This lecture uses the structure and material from this review paper!

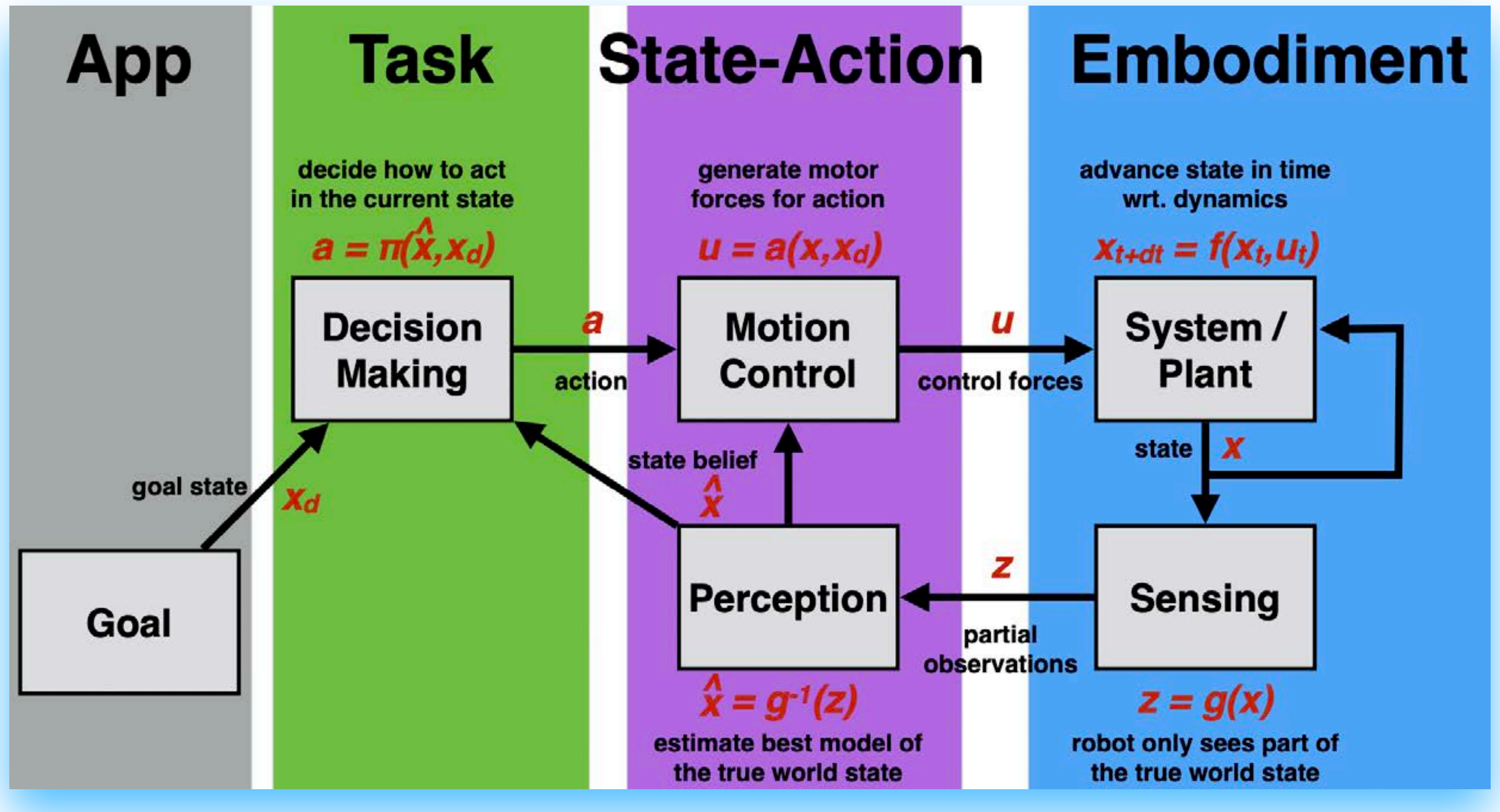
Previously in
Planning, Decision Making, Control

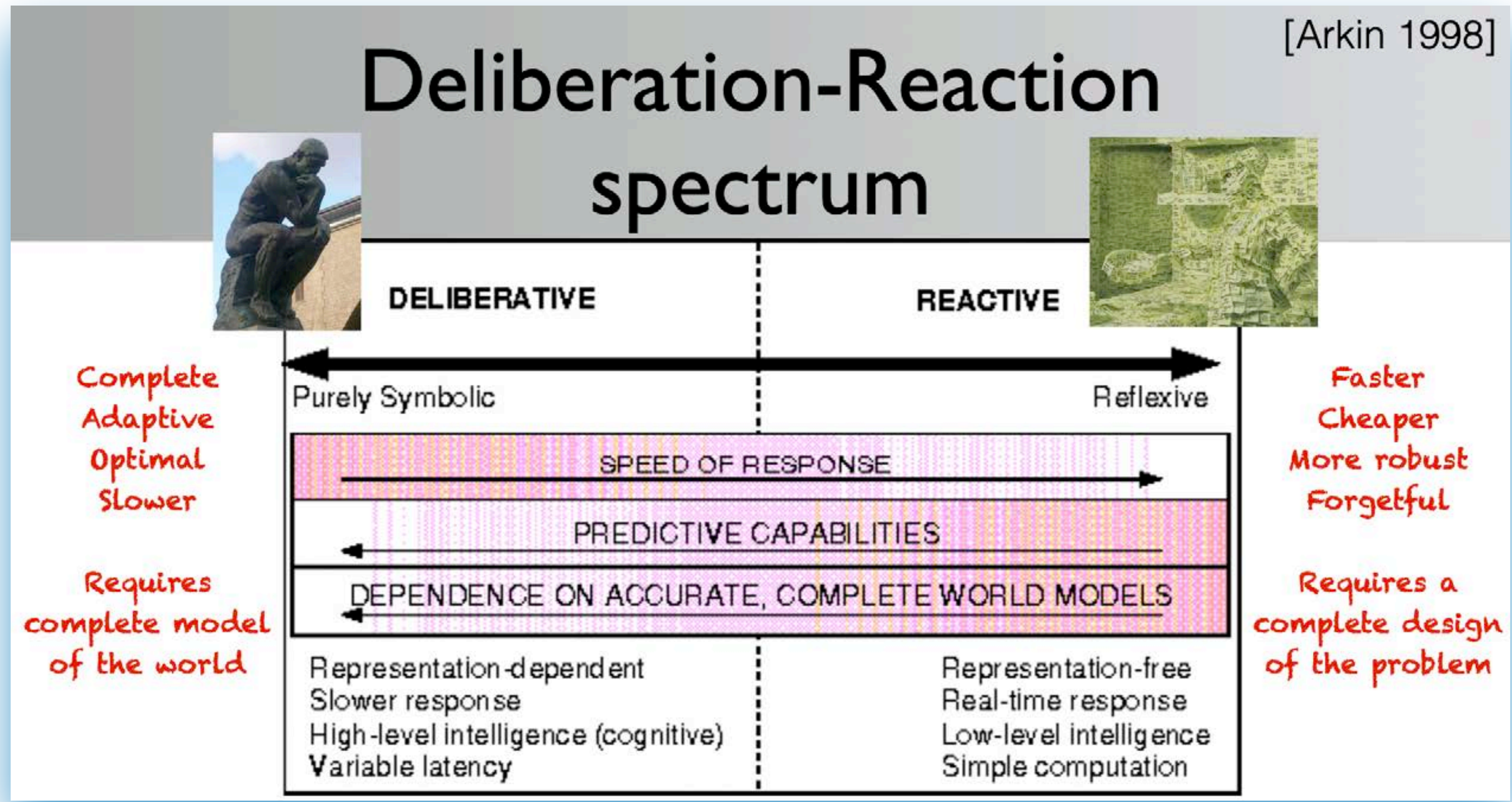
Depth-first search



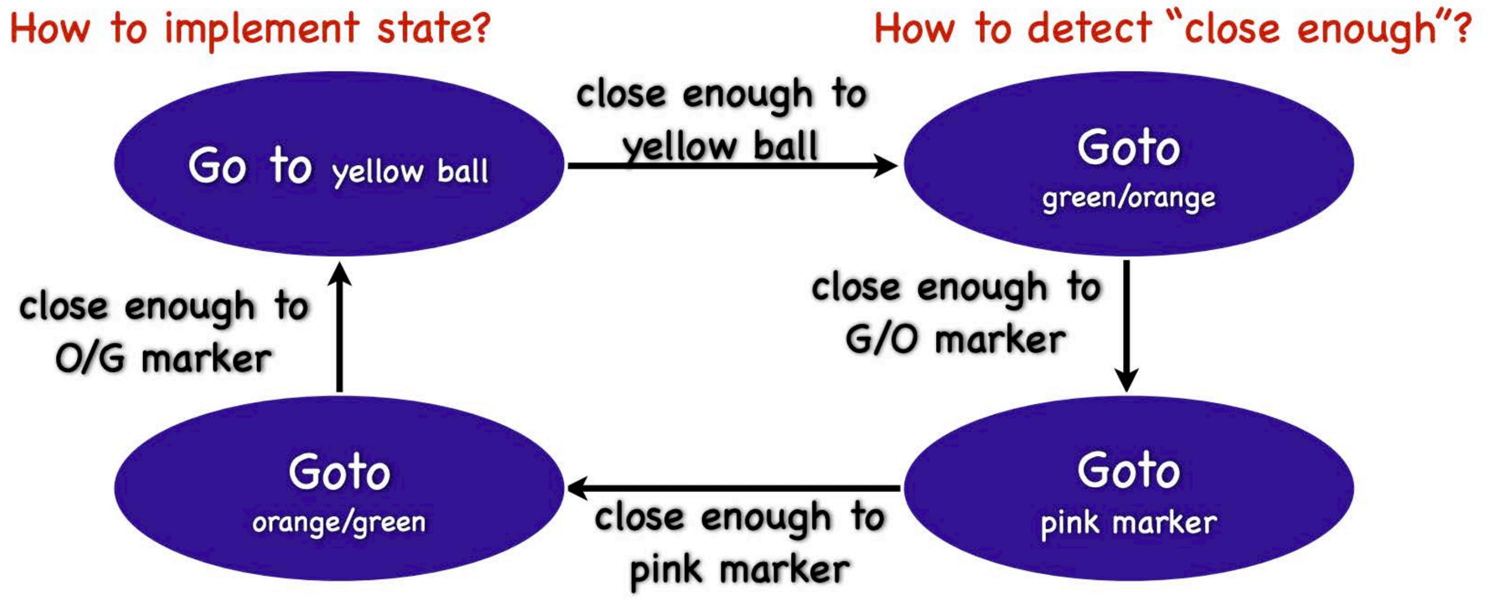
Search algorithm template

```
all nodes ← {dist_start ← infinity, parent_start ← none, visited_start ← false}
start_node ← {dist_start ← 0, parent_start ← none, visited_start ← true}
visit_list ← start_node
    while visit_list != empty && current_node != goal
        cur_node ← highestPriority(visit_list)
        visited_cur_node ← true
        for each nbr in not_visited(adjacent(cur_node))
            add(nbr to visit_list)
            if dist_nbr > dist_cur_node + distStraightLine(nbr,cur_node)
                parent_nbr ← current_node
                dist_nbr ← dist_cur_node + distStraightLine(nbr,cur_node)
            end if
        end for loop
    end while loop
output ← parent, distance
```





Object seeking FSM



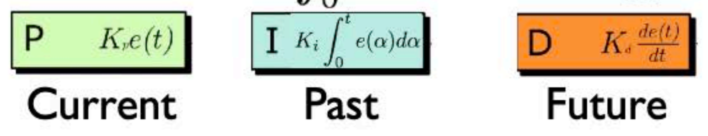
PID Control

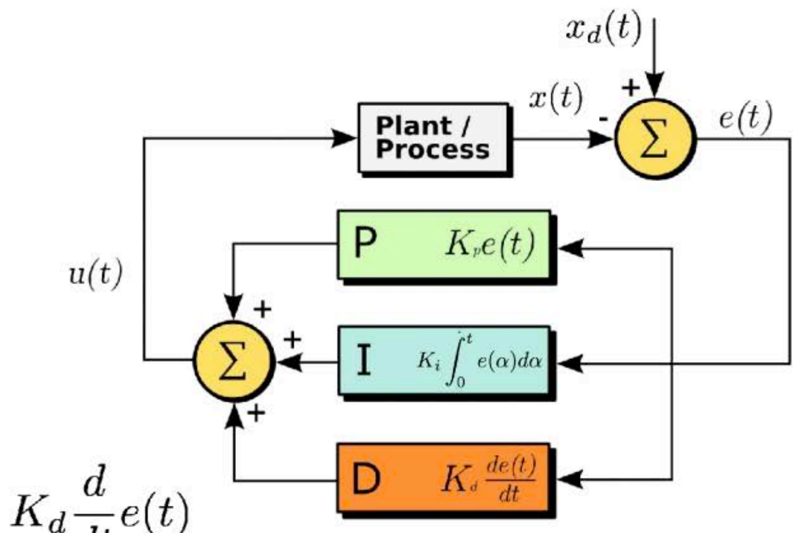
Error signal:

$$e(t) = x_{desired}(t) - x(t)$$

Control signal:

$$u(t) = K_p e(t) + K_i \int_0^t e(\alpha) d\alpha + K_d \frac{d}{dt} e(t)$$



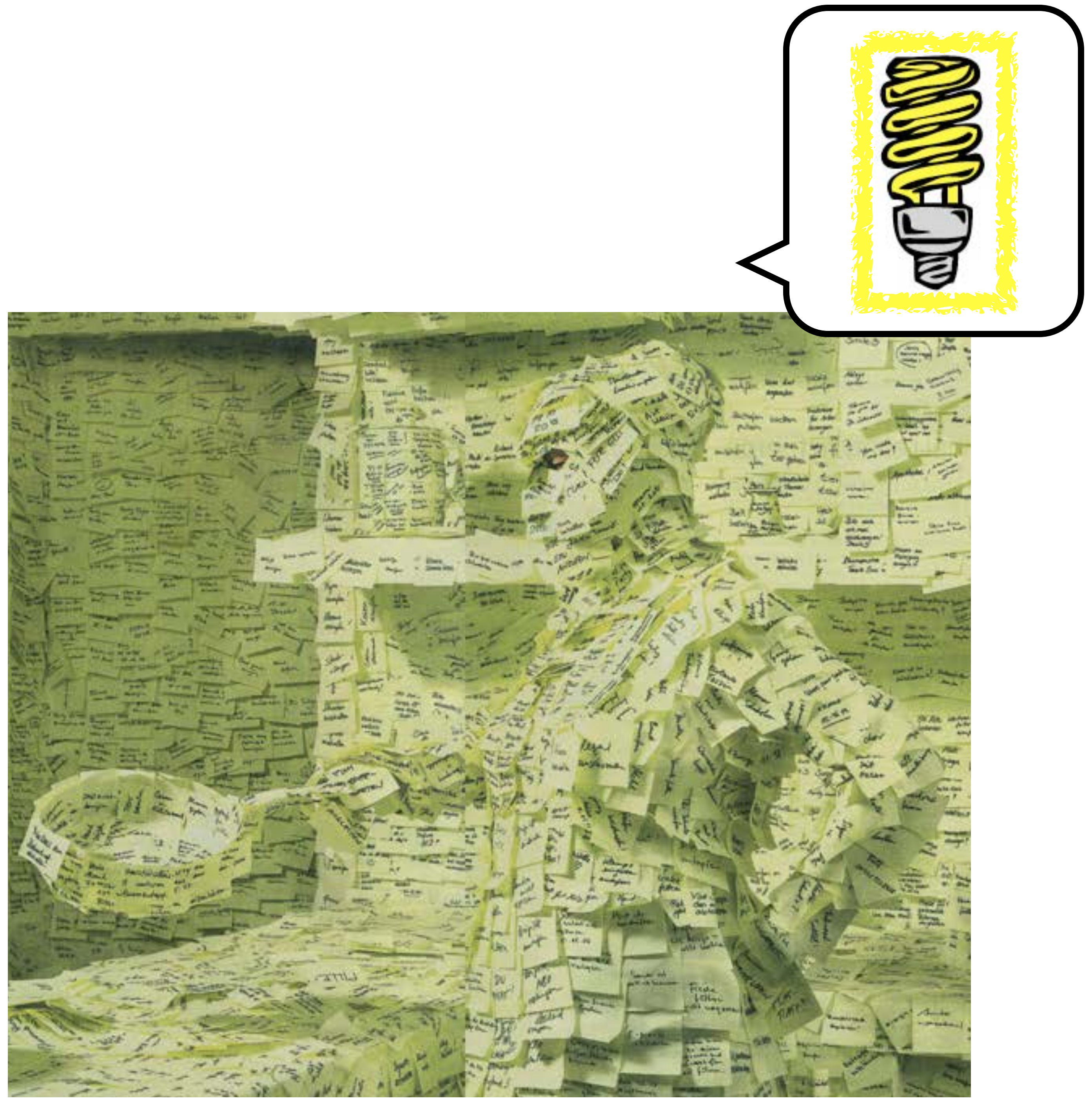


Approaches to motion planning

- **Bug algorithms: Bug[0-2], Tangent Bug**
- Graph Search (fixed graph)
  - Depth-first, Breadth-first, Dijkstra, A-star
- Sampling-based Search (build graph):
  - Probabilistic Road Maps, Rapidly-exploring Random Trees
- Optimization (local search):
  - Gradient descent, potential fields, Wavefront

Should your robot's decision making

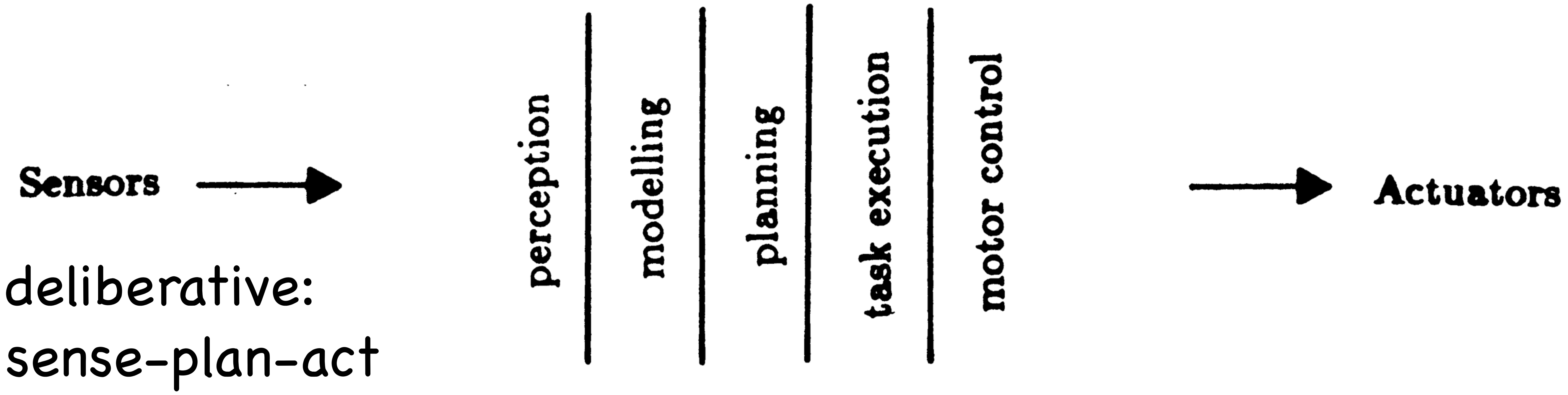
OR



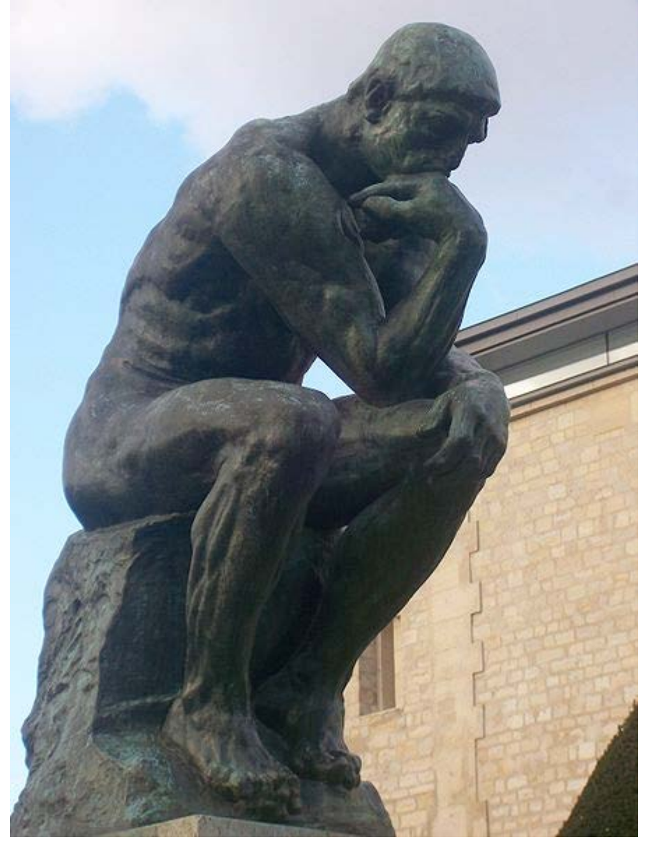
fully think through solving a problem?

react quickly to changes in its world?

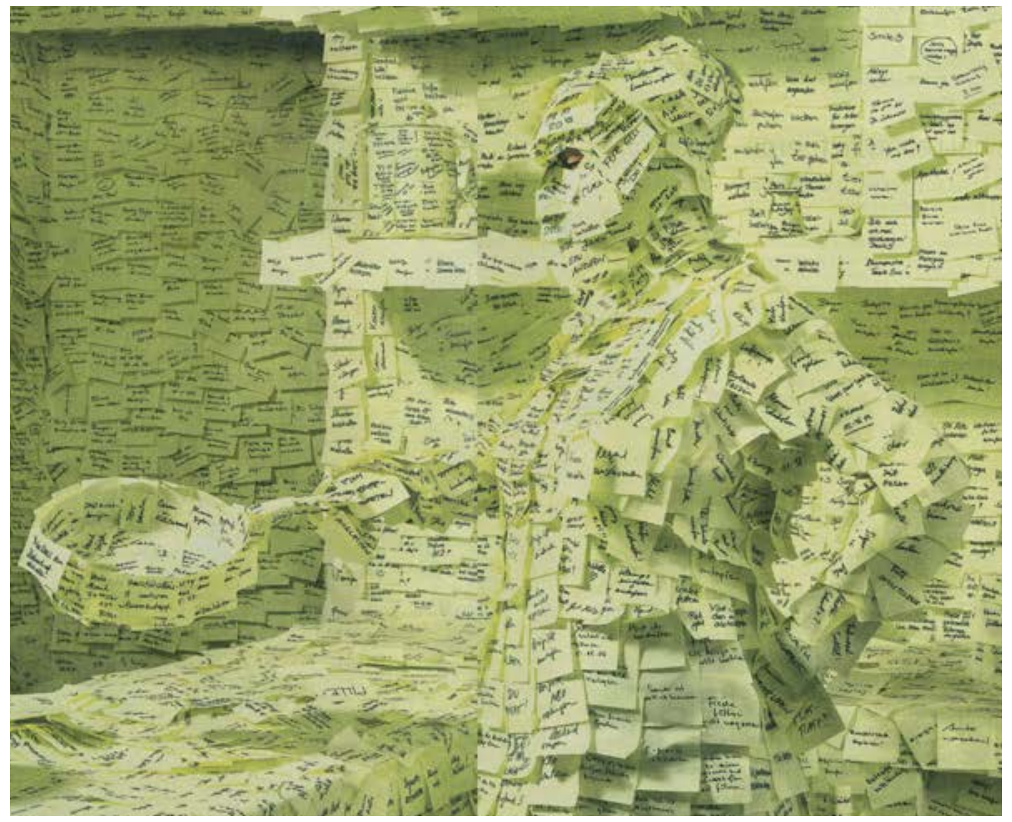
Deliberation v. Reaction

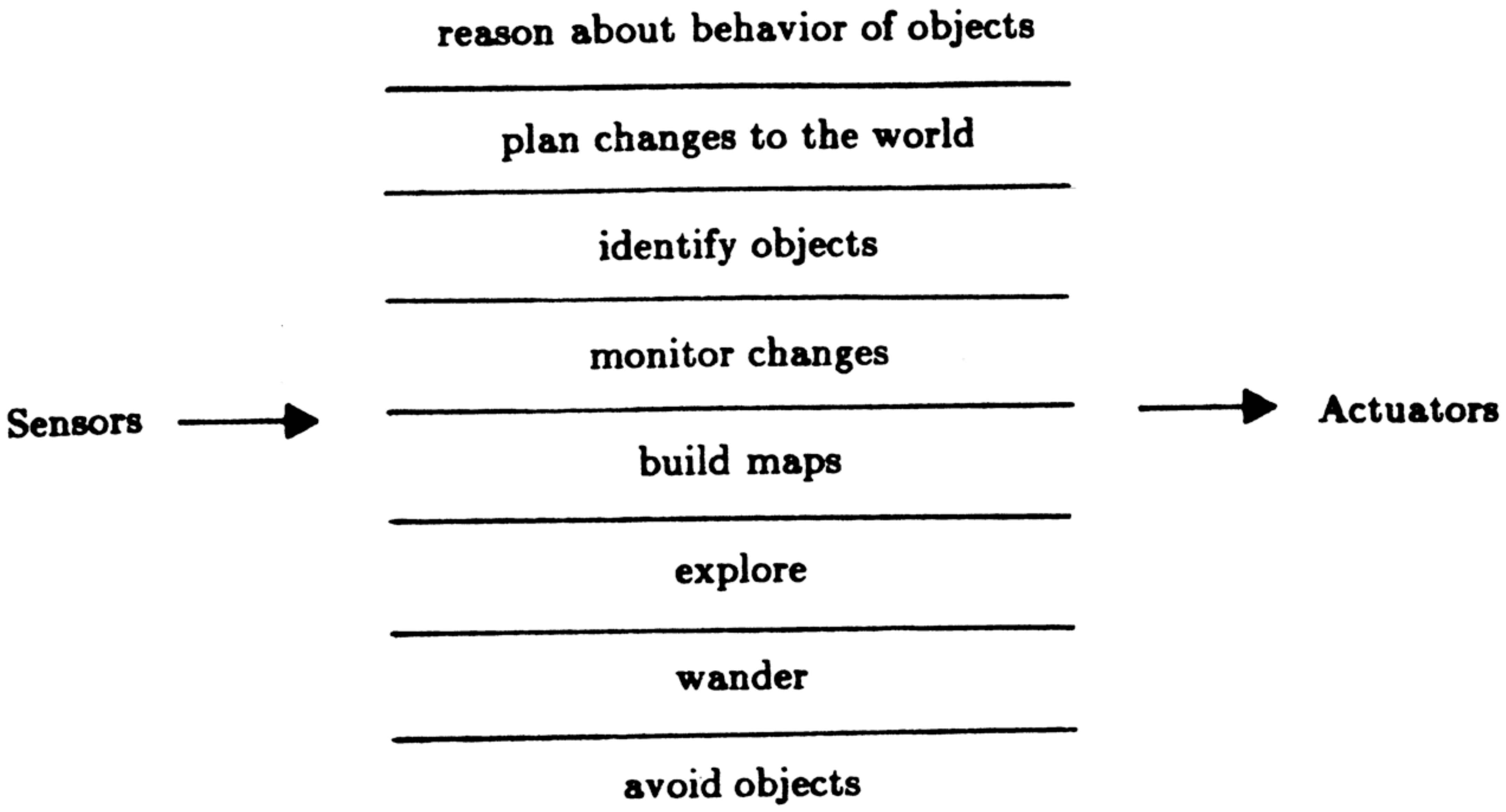


deliberative:
sense-plan-act

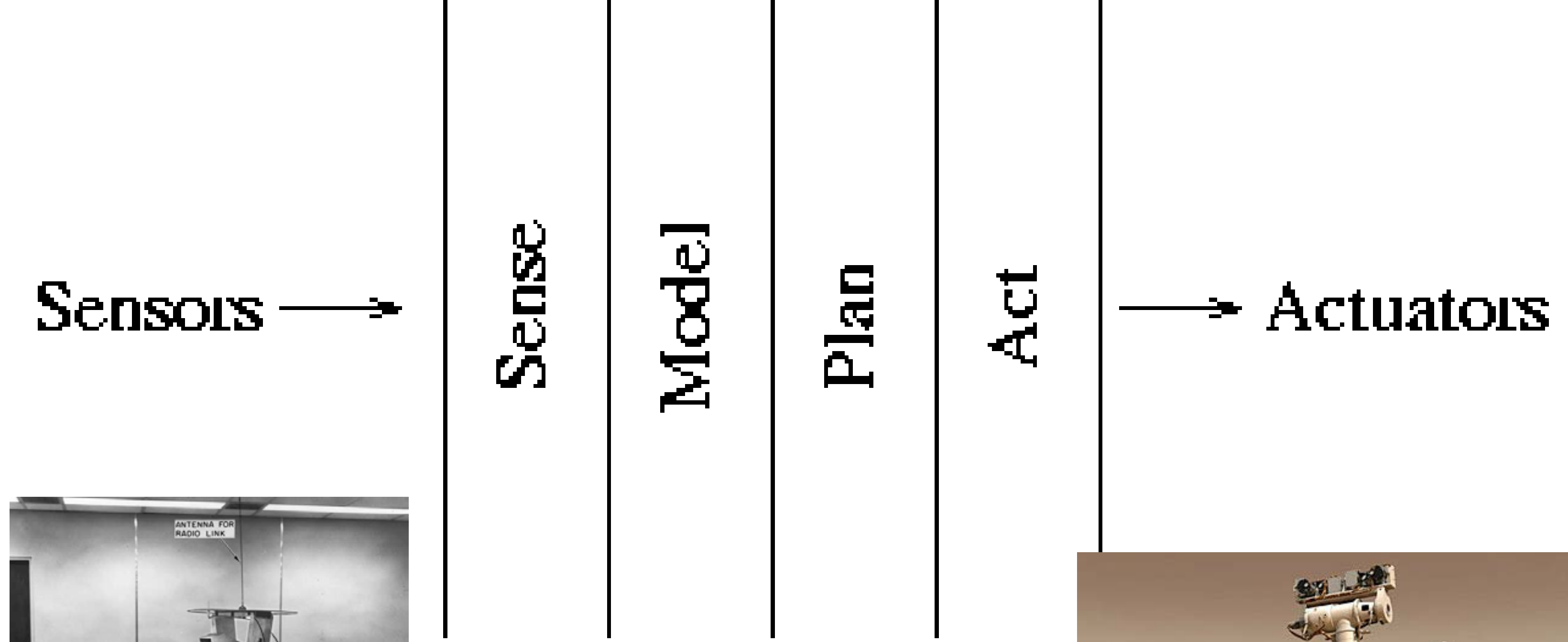


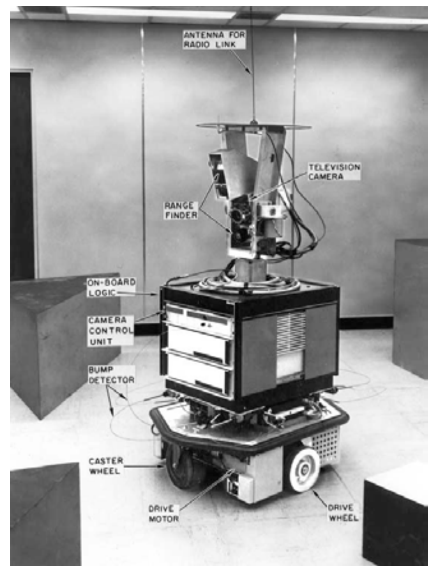
reaction: subsumption,
Finite State Machine
controllers act in parallel



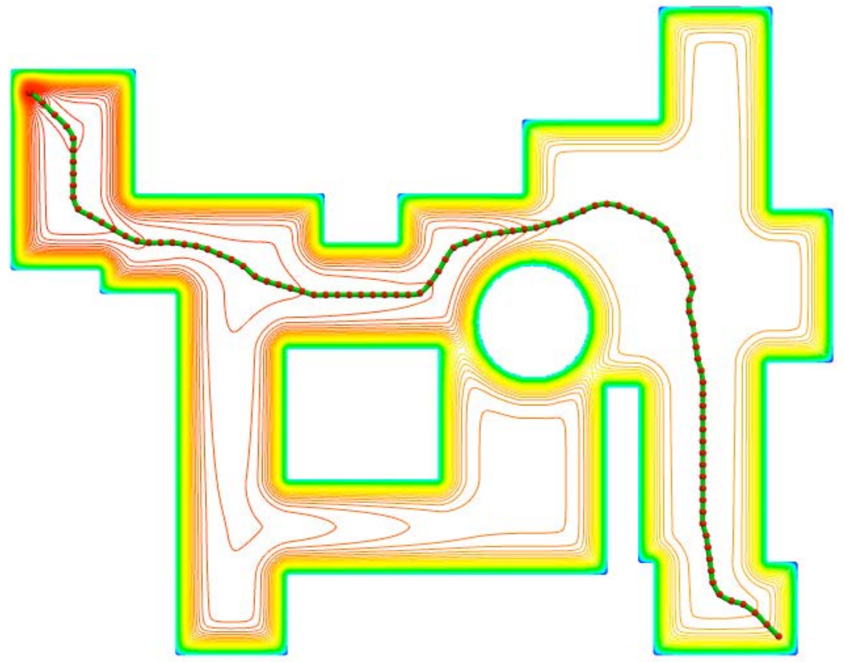


Deliberation

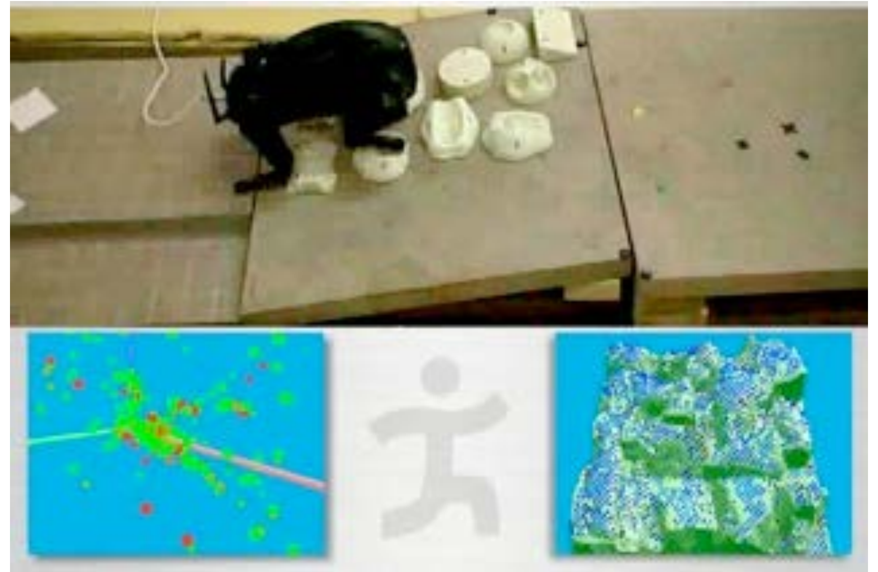




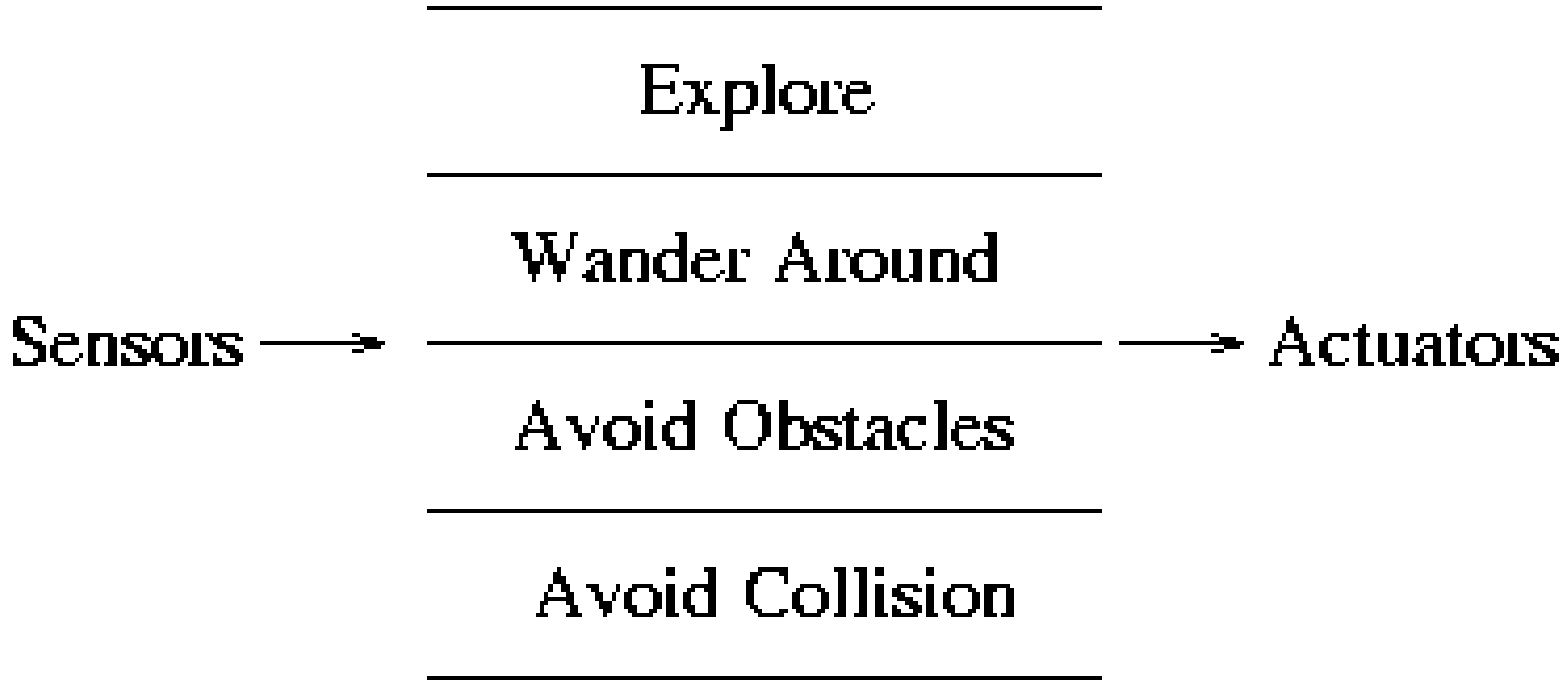
“Sense-Plan-Act” paradigm



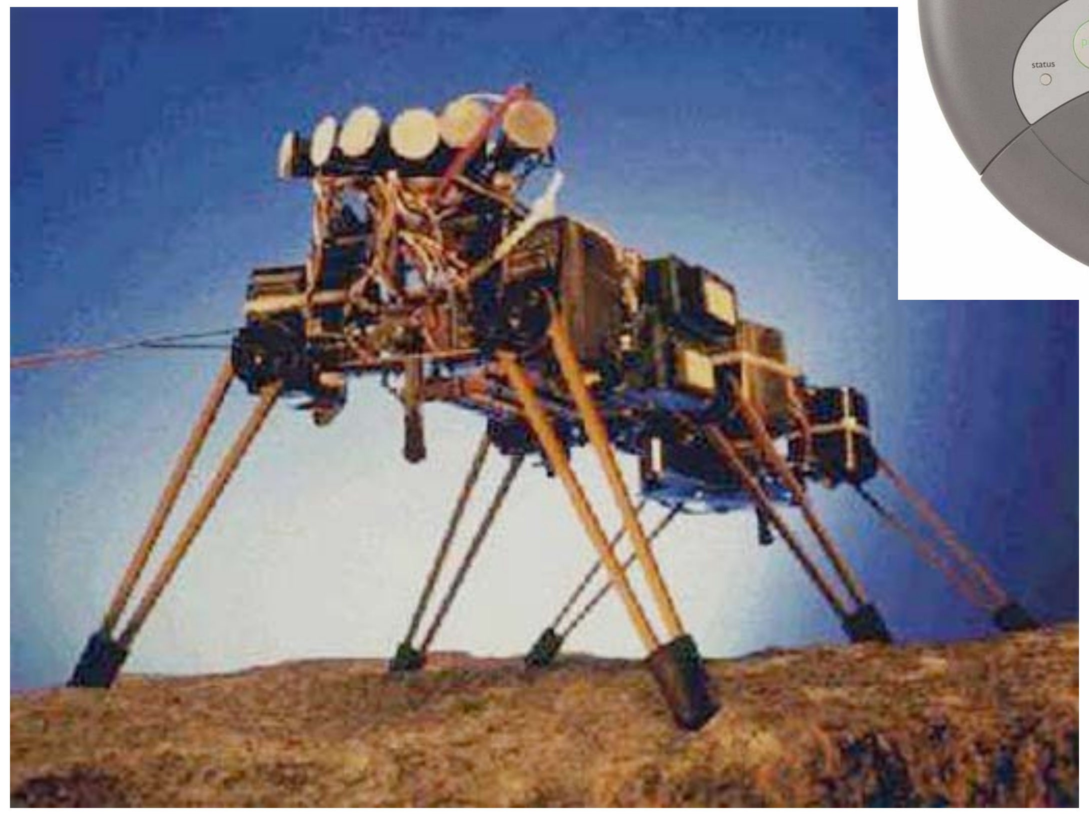
- sense: build most complete model of world
  - GPS, SLAM, 3D reconstruction, affordances
- plan: search over all possible outcomes
  - BFS, DFS, Dijkstra, A*, RRT
- act: execute plan through motor forces



Reaction



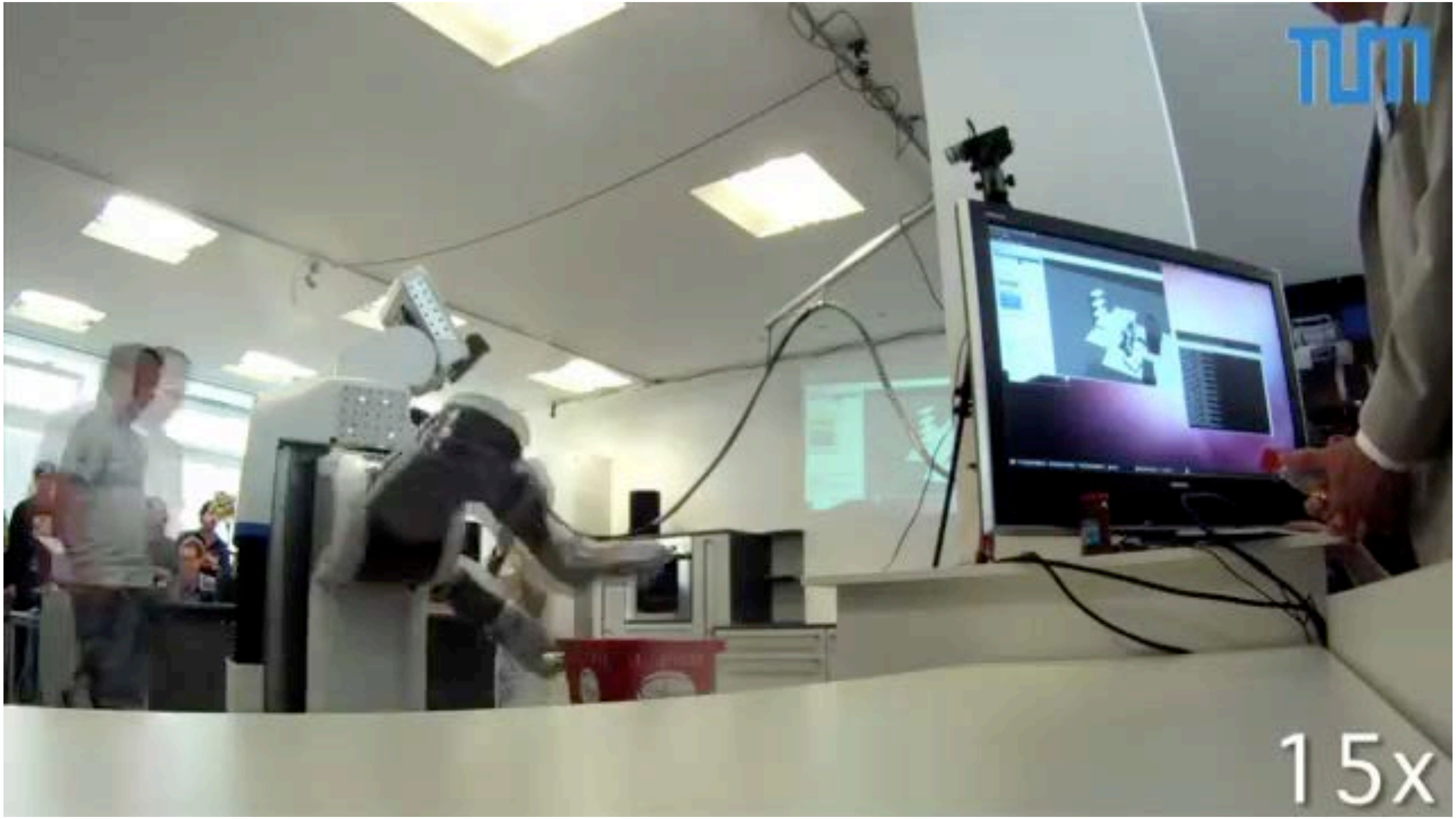
- No representation of state
- Typically, fast hardcoded rules
- Embodied intelligence
  - behavior := control + embodiment
  - ant analogy, stigmergy
- Subsumption architecture
  - prioritized reactive policies
- Ghengis hexpod video



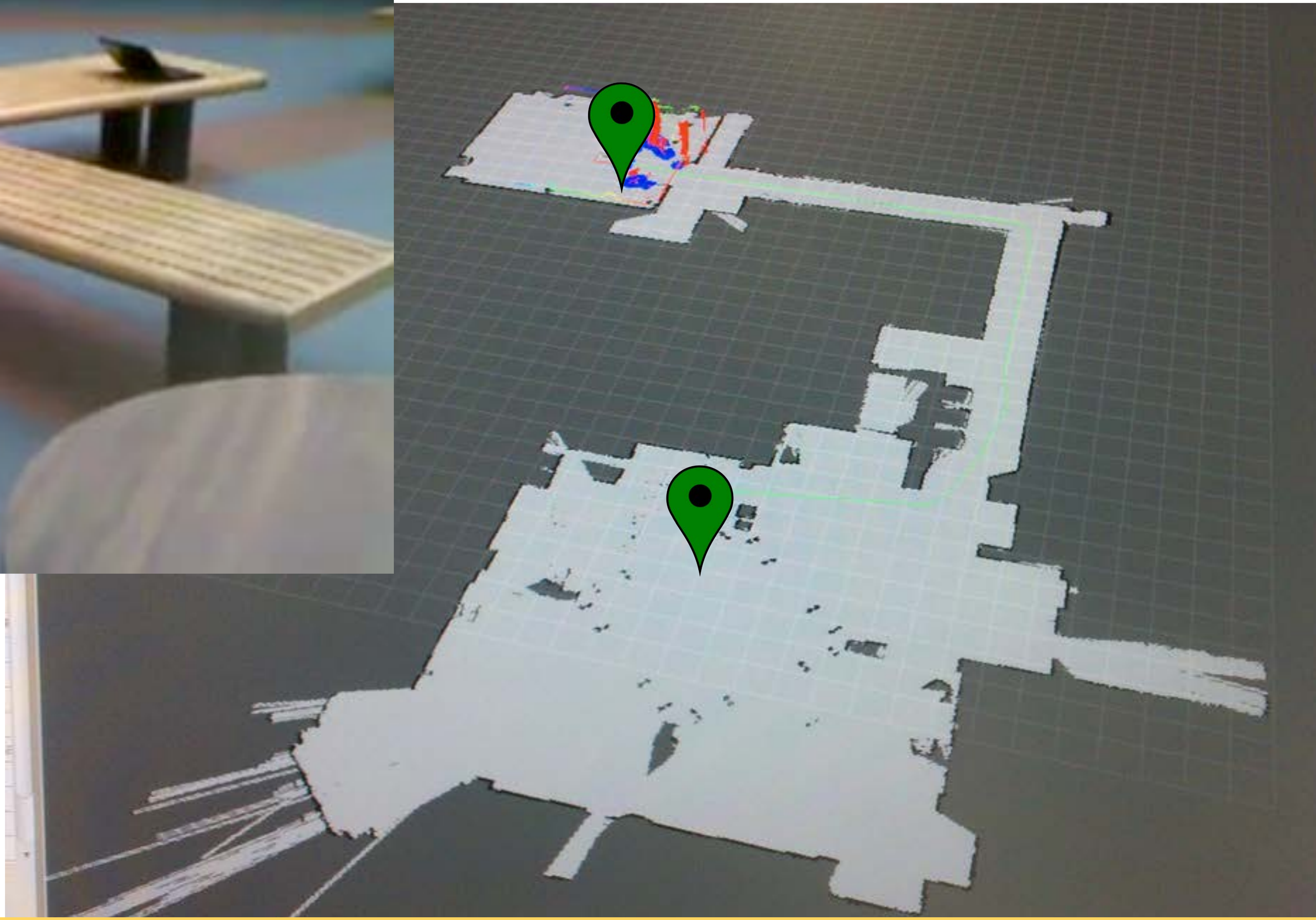
Slide borrowed from Michigan Robotics autorob.org

MIT Genghis

https://www.youtube.com/watch?v=1j6CliOwRng



Robots have to make lots of decisions



Remember your path planner?

How to get from Location A to Location B?

Base Navigation

- How get from point A to point B
- **What is the simplest policy to perform navigation?**
    - Remember: simplest reactive policy?

Random Walk: Goal Seeking

- Move in a random direction until you hit something
- Then go in a new direction
- Stop when you get to the goal, assuming it can be recognized

goal: exit here

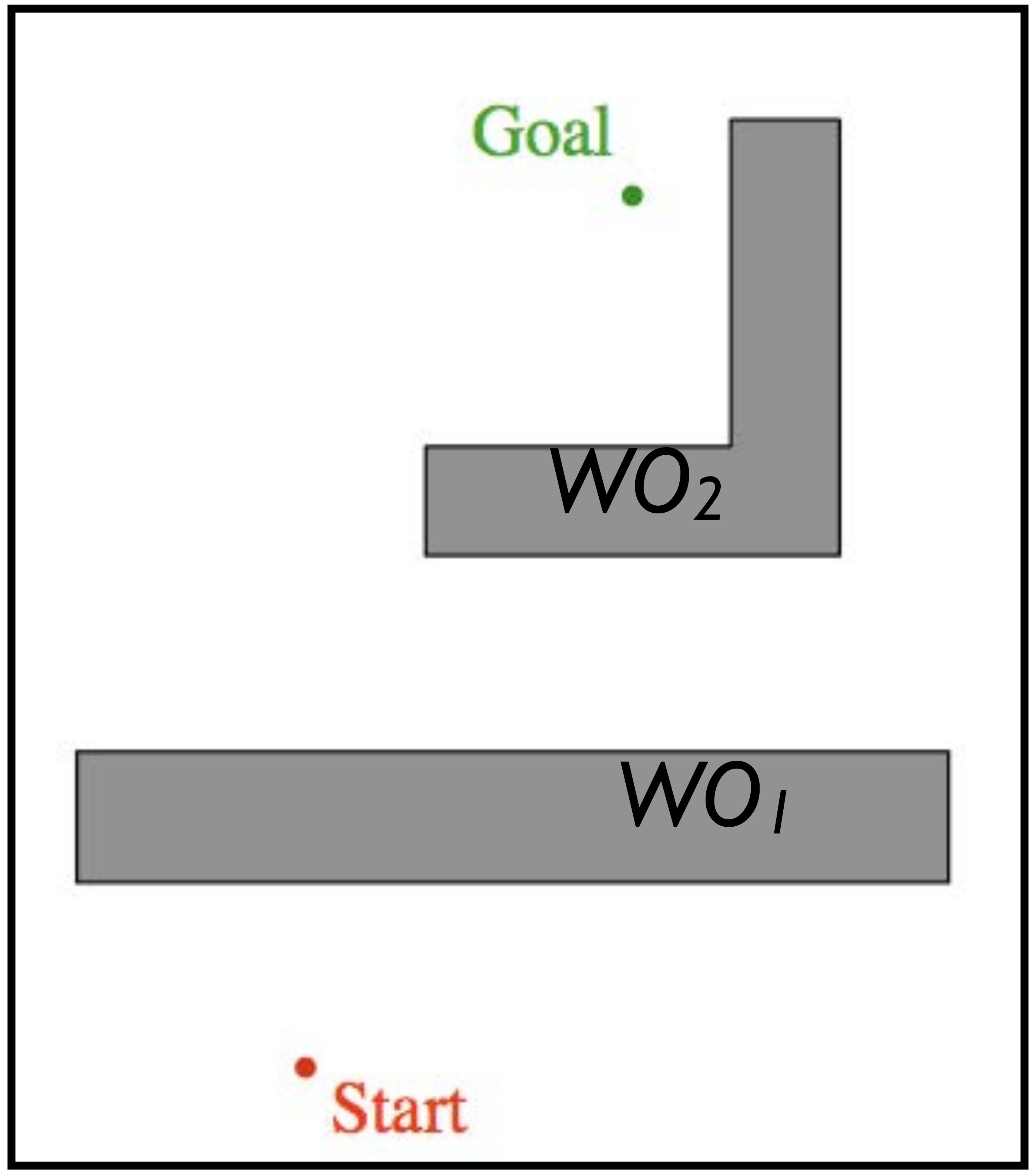
Lisa Miller, http://www.youtube.com/watch?v=VBzXDrz8rMI

Base Navigation

- How get from point A to point B
- What is the simplest policy to perform navigation?
  - random walk
  - reactive: embodied intelligence
- **What is a “simple” deliberative policy?**

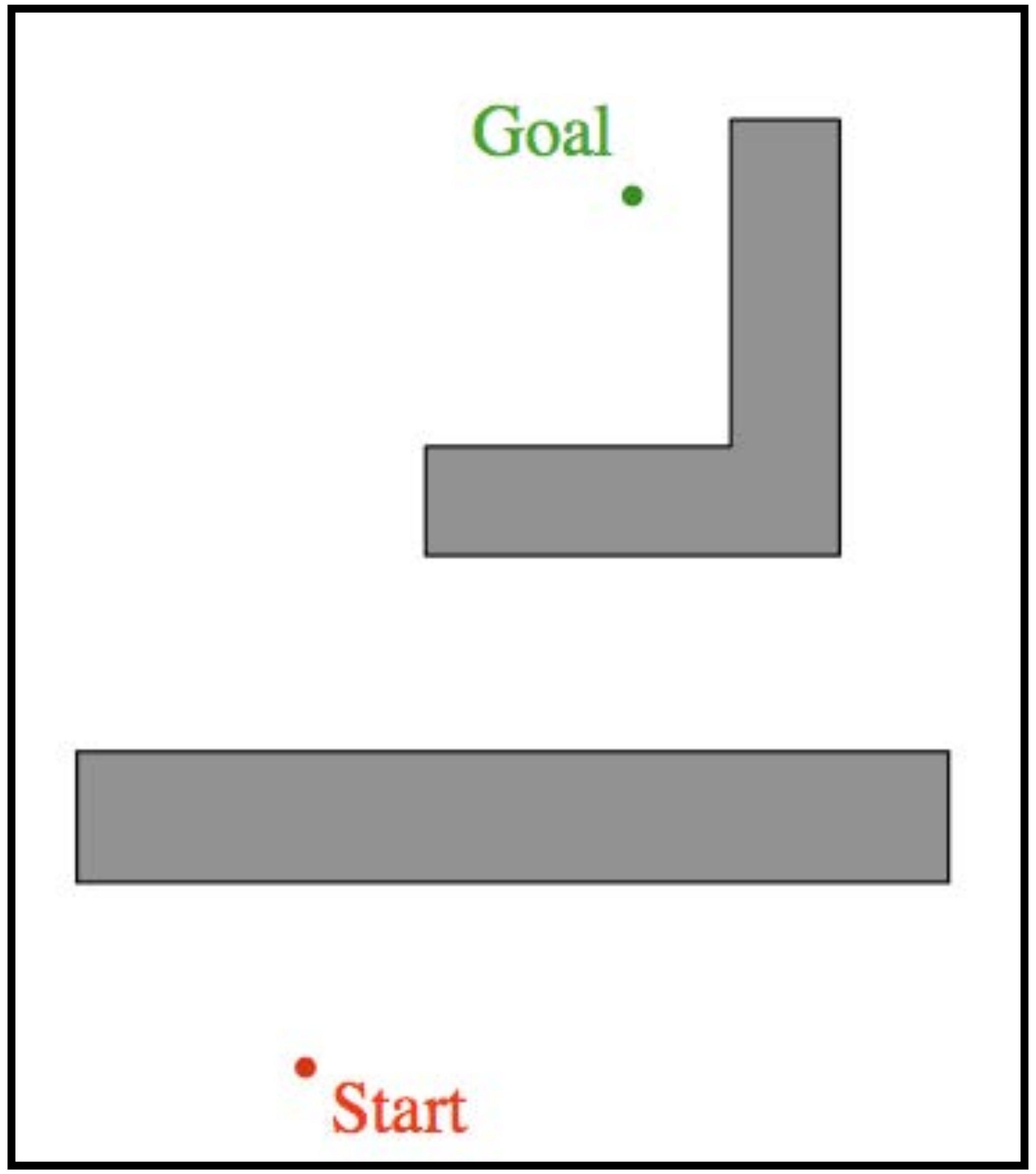
Bug Algorithms

- Assume bounded world *W*
- Known: global goal
  - measurable distance *d(x,y)*
- Unknown: obstacles WO_i
- Local sensing
  - tactile
  - distance traveled



Bug Algorithms

- Assume bounded world *W*
- Known: global goal
  - measurable distance *d(x,y)*
- Unknown: obstacles WO_i
- Local sensing
  - **bump sensor**
  - distance traveled

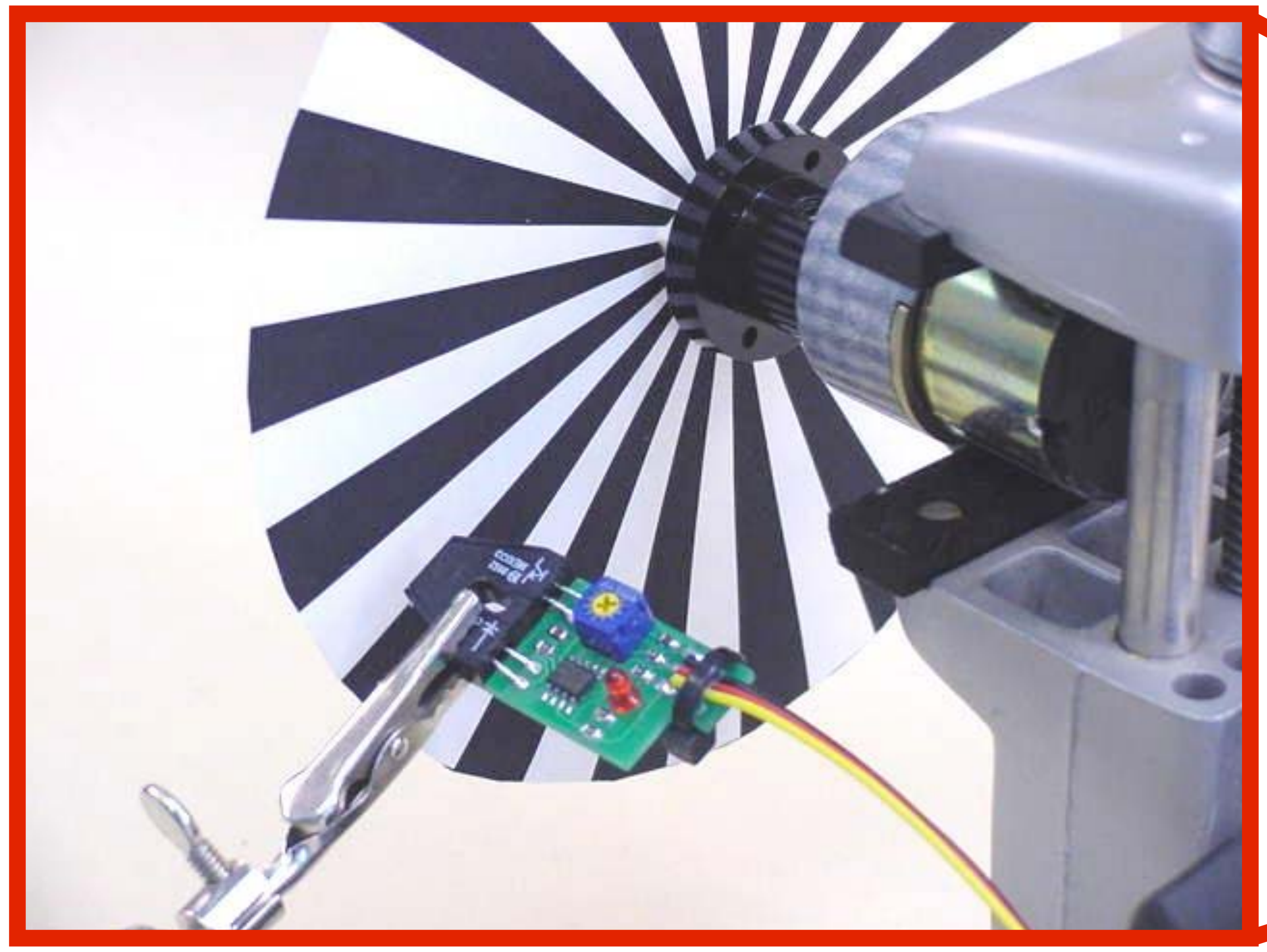


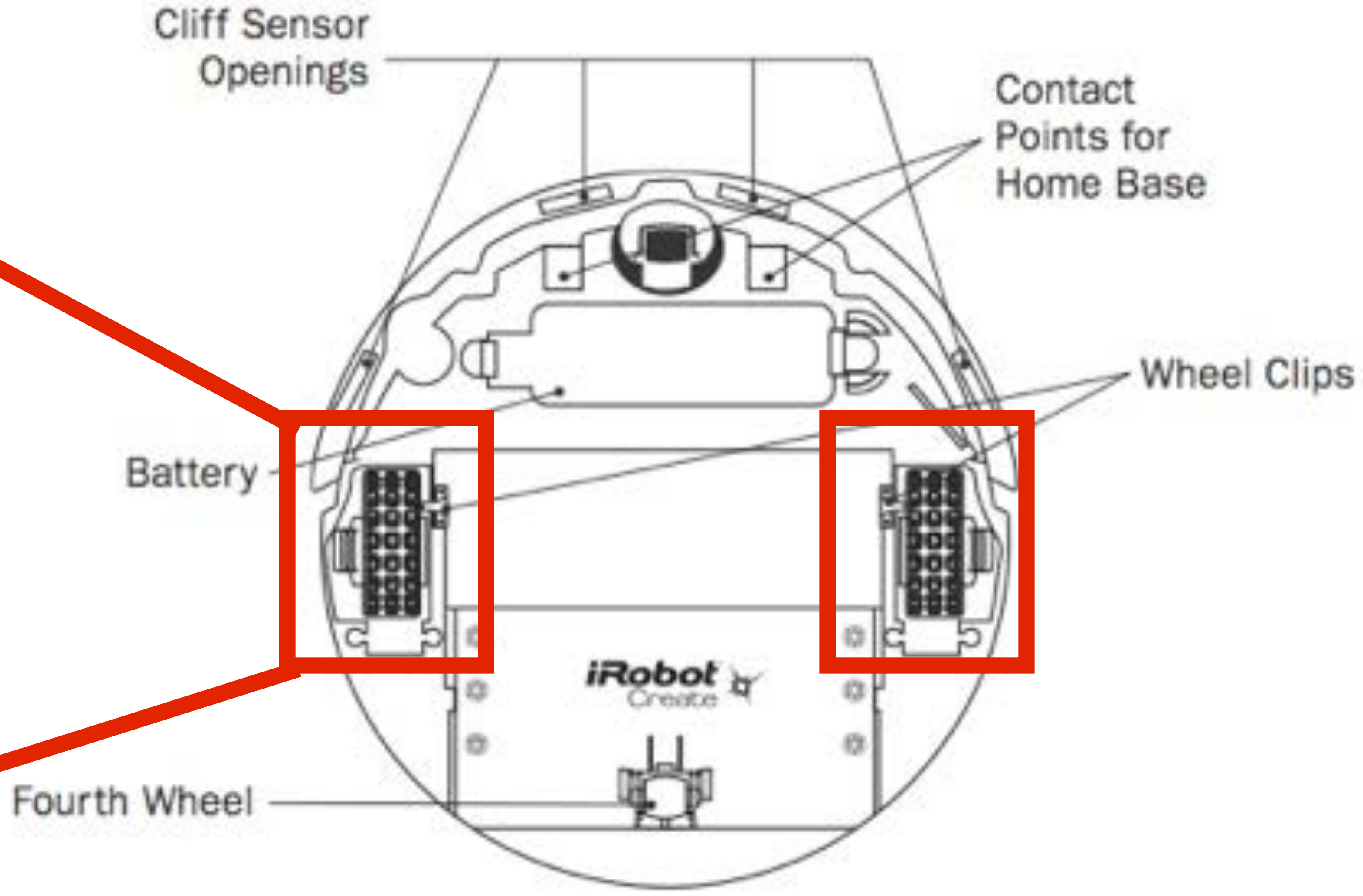
≈

bumper is essentially an on/off button

Bug A

Optical encoders



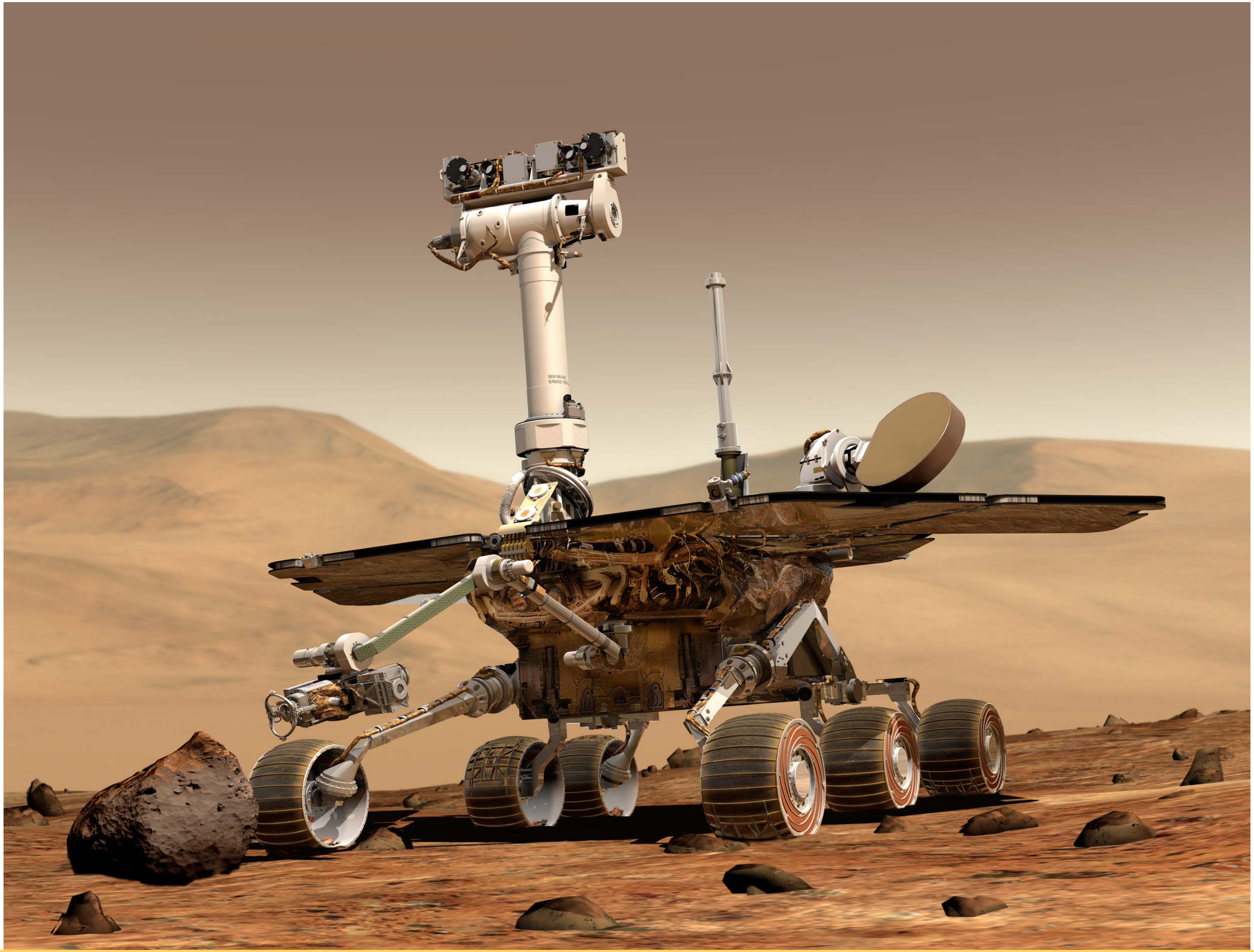


- Assume
- Known:
  - meas
- Unknow
- Local sensing
  - bump
  - odometry

≈

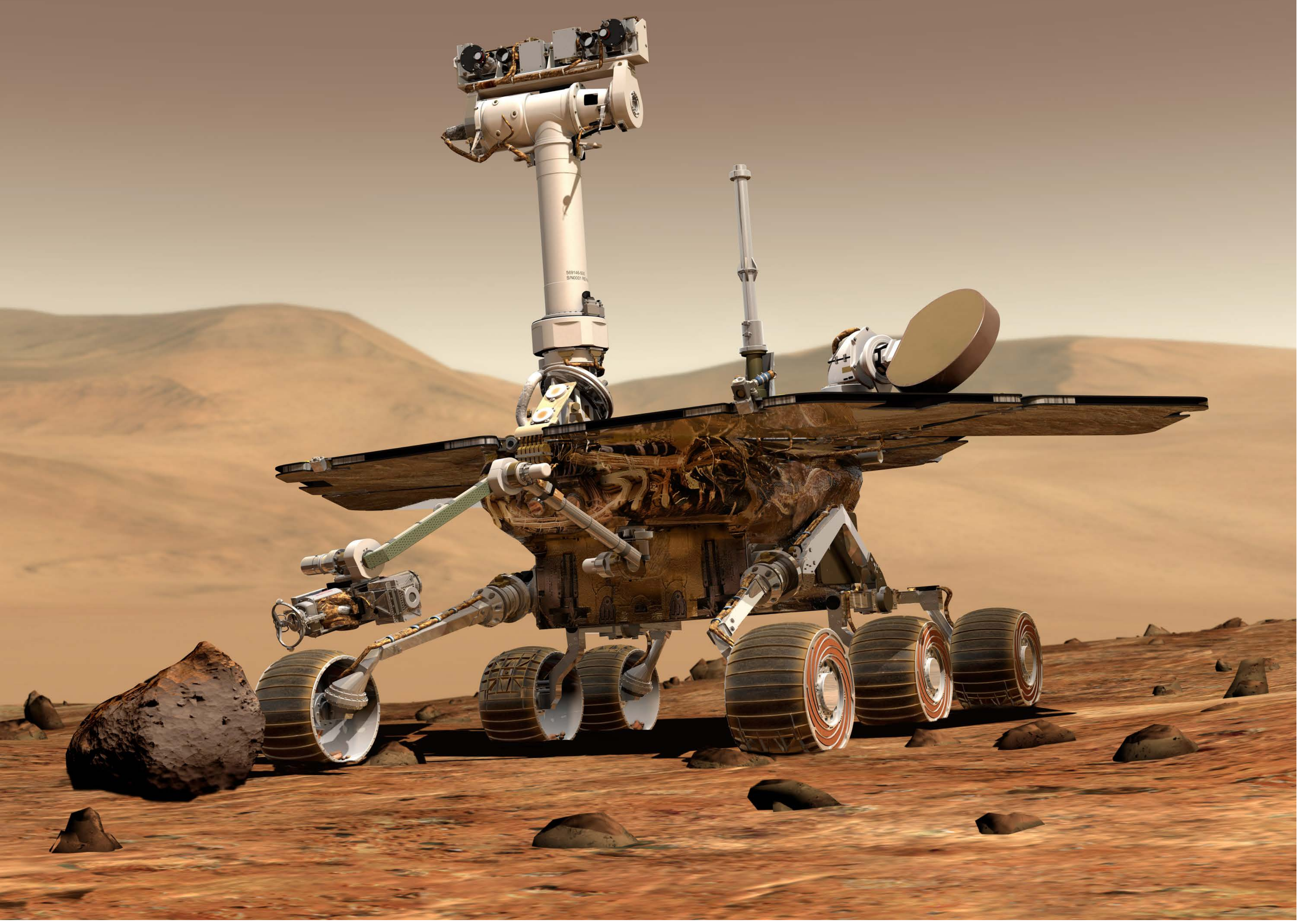
Slide borrowed from Michigan Robotics autorob.org

Interesting application of Bug algorithms ?

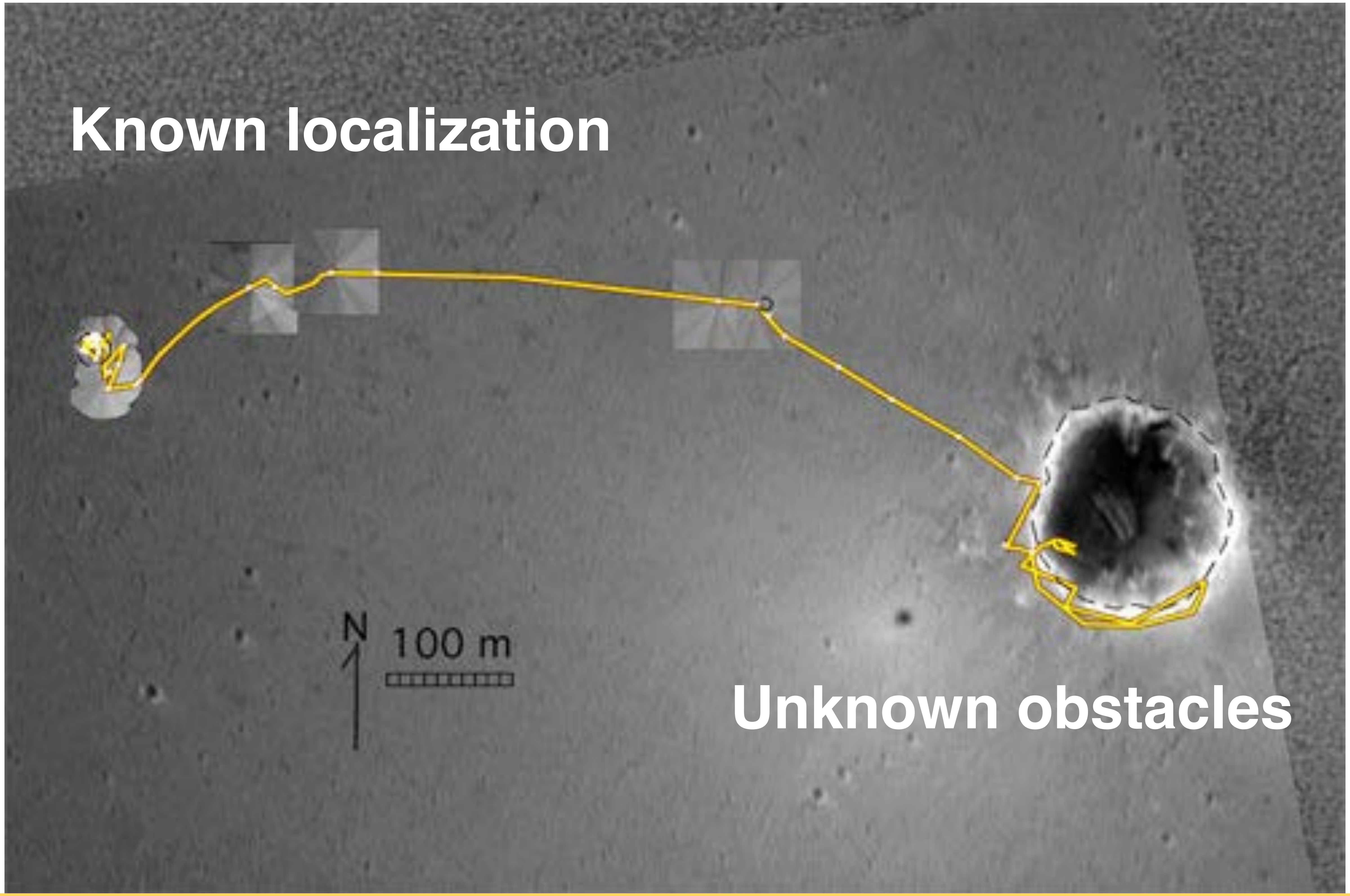


Slide borrowed from Michigan Robotics autorob.org

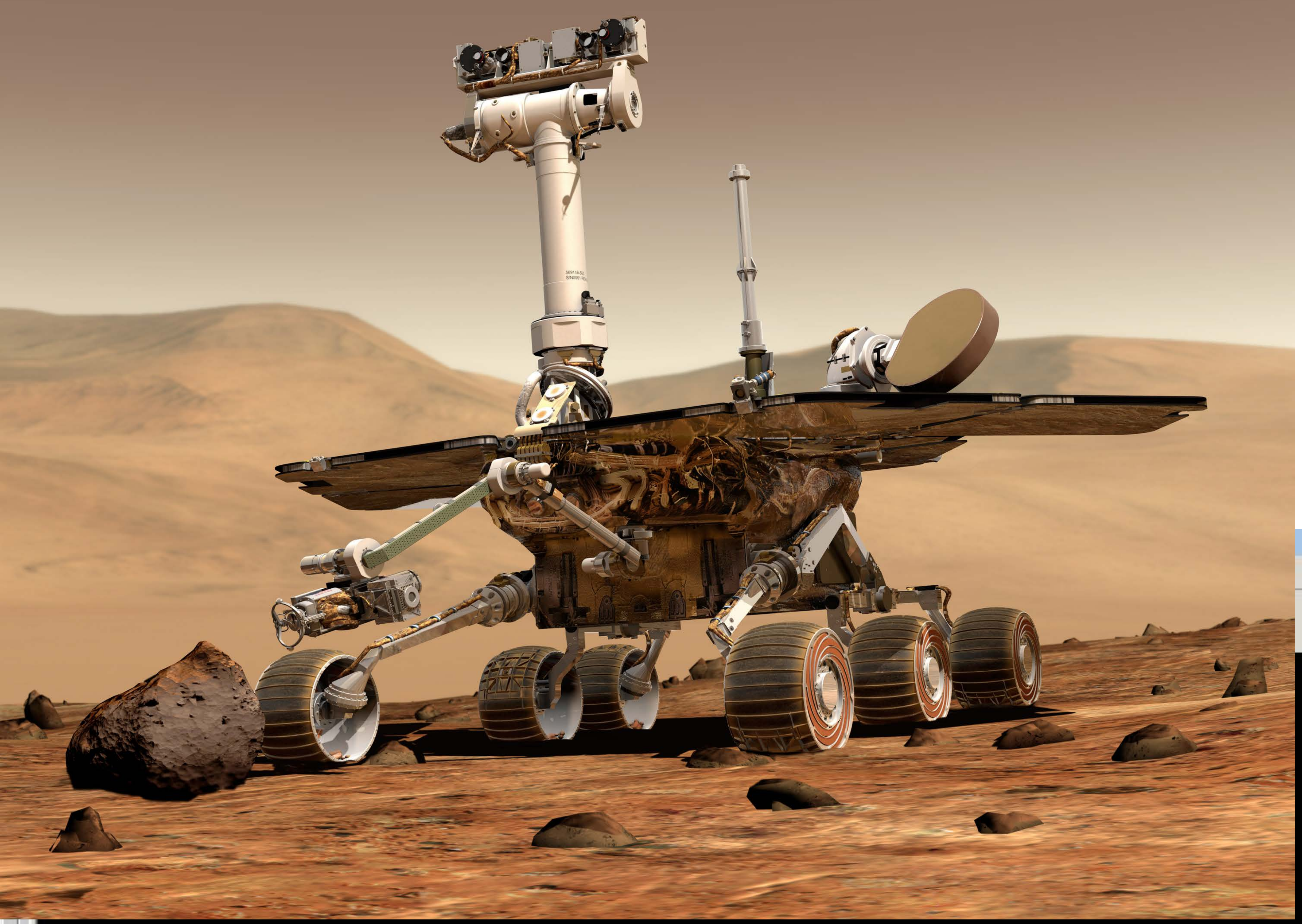
Mars Exploration Rover



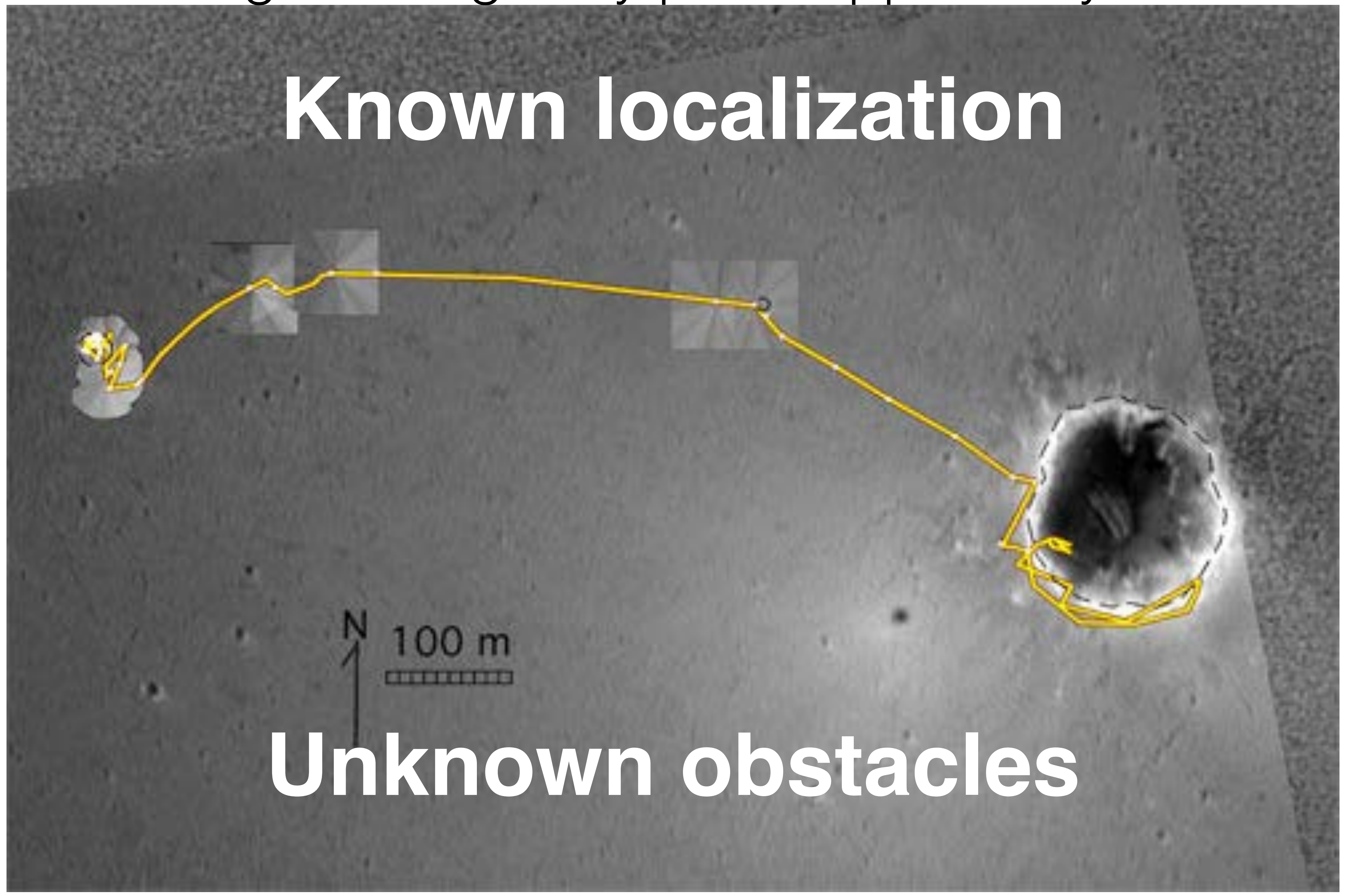
http://mars.nasa.gov/mer/gallery/press/opportunity/20040921a.html

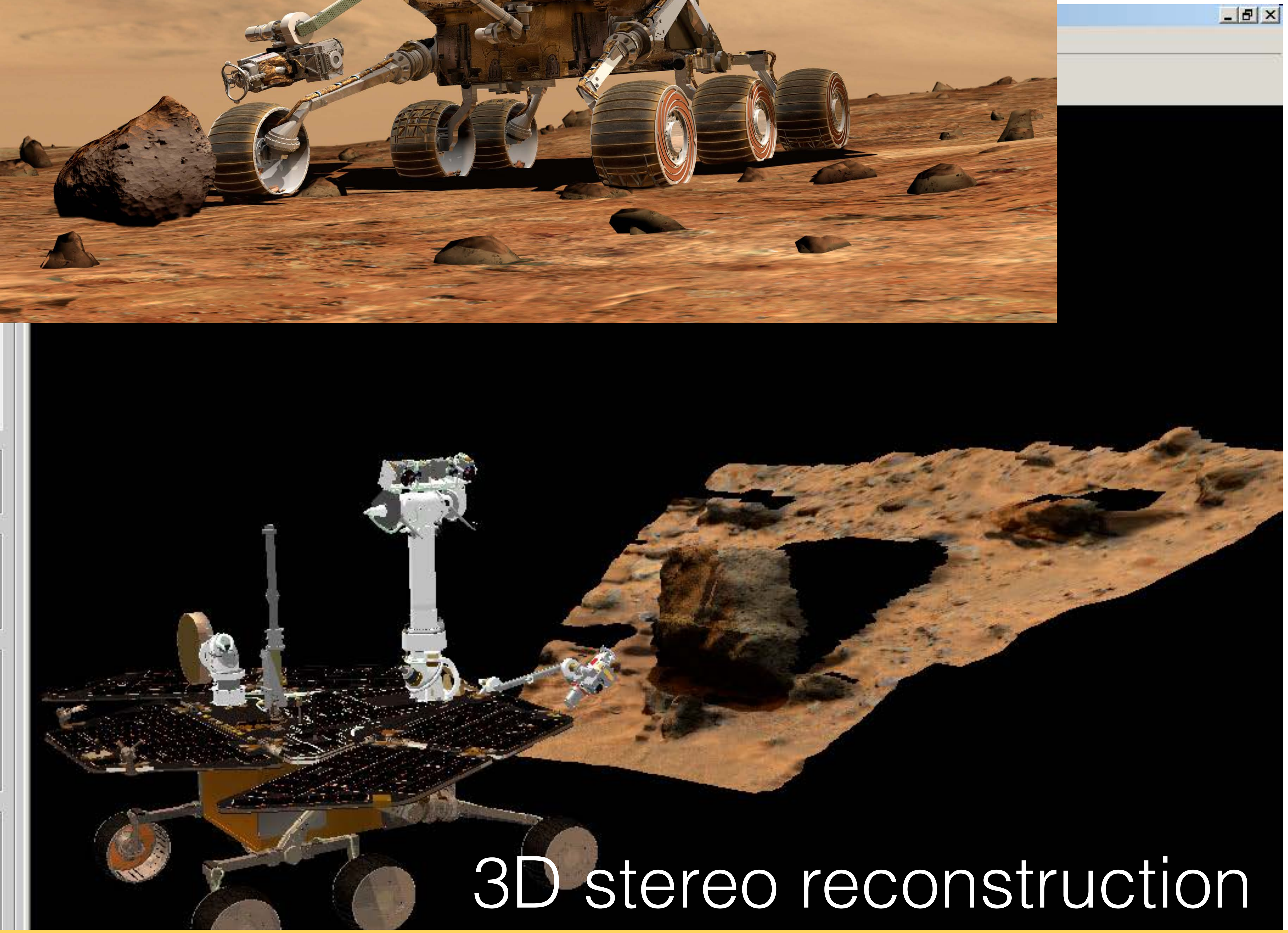


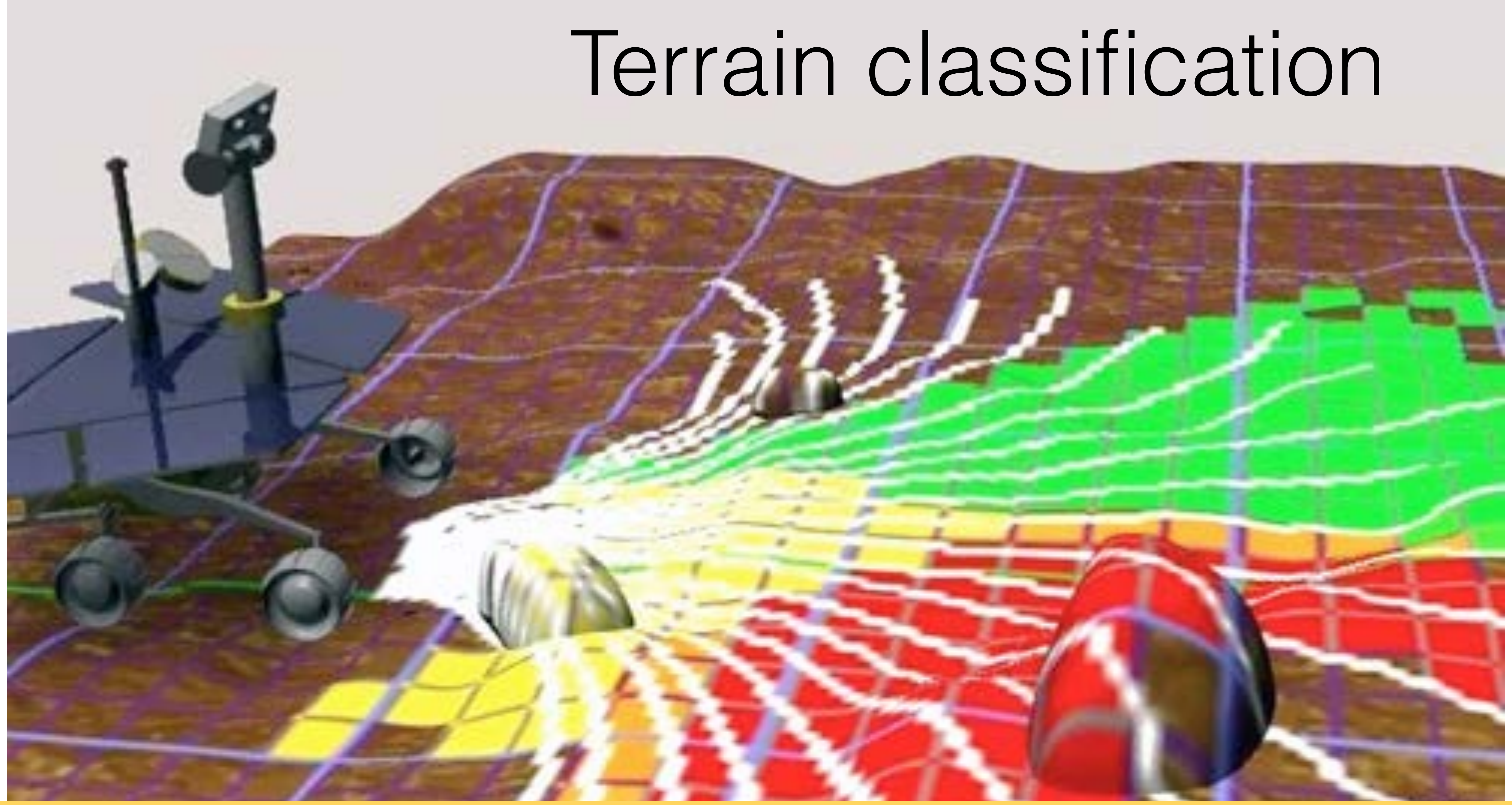
Mars Exploration Rover



http://mars.nasa.gov/mer/gallery/press/opportunity/20040921a.html

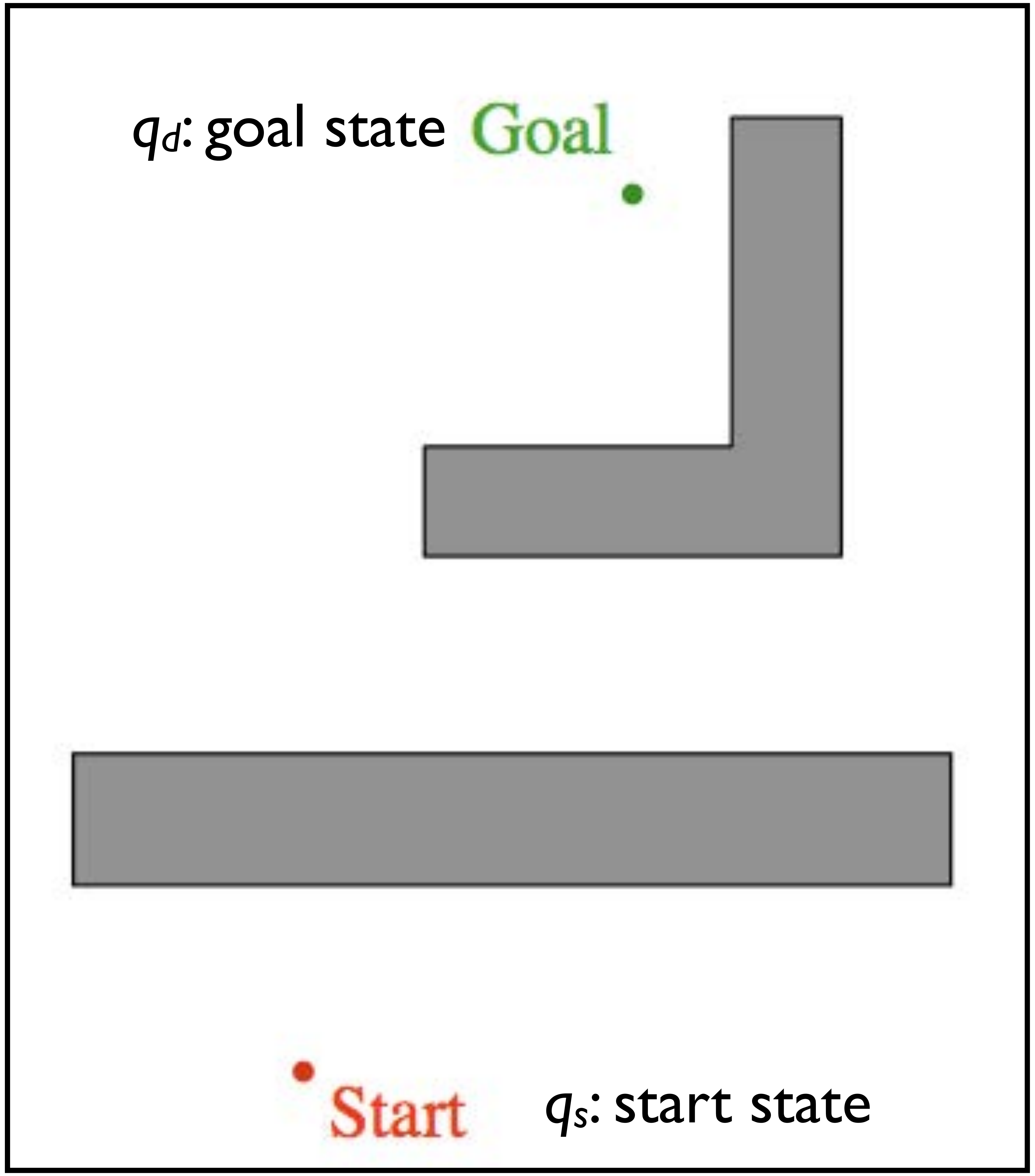






Slide borrowed from Michigan Robotics autorob.org

Bug Navigation

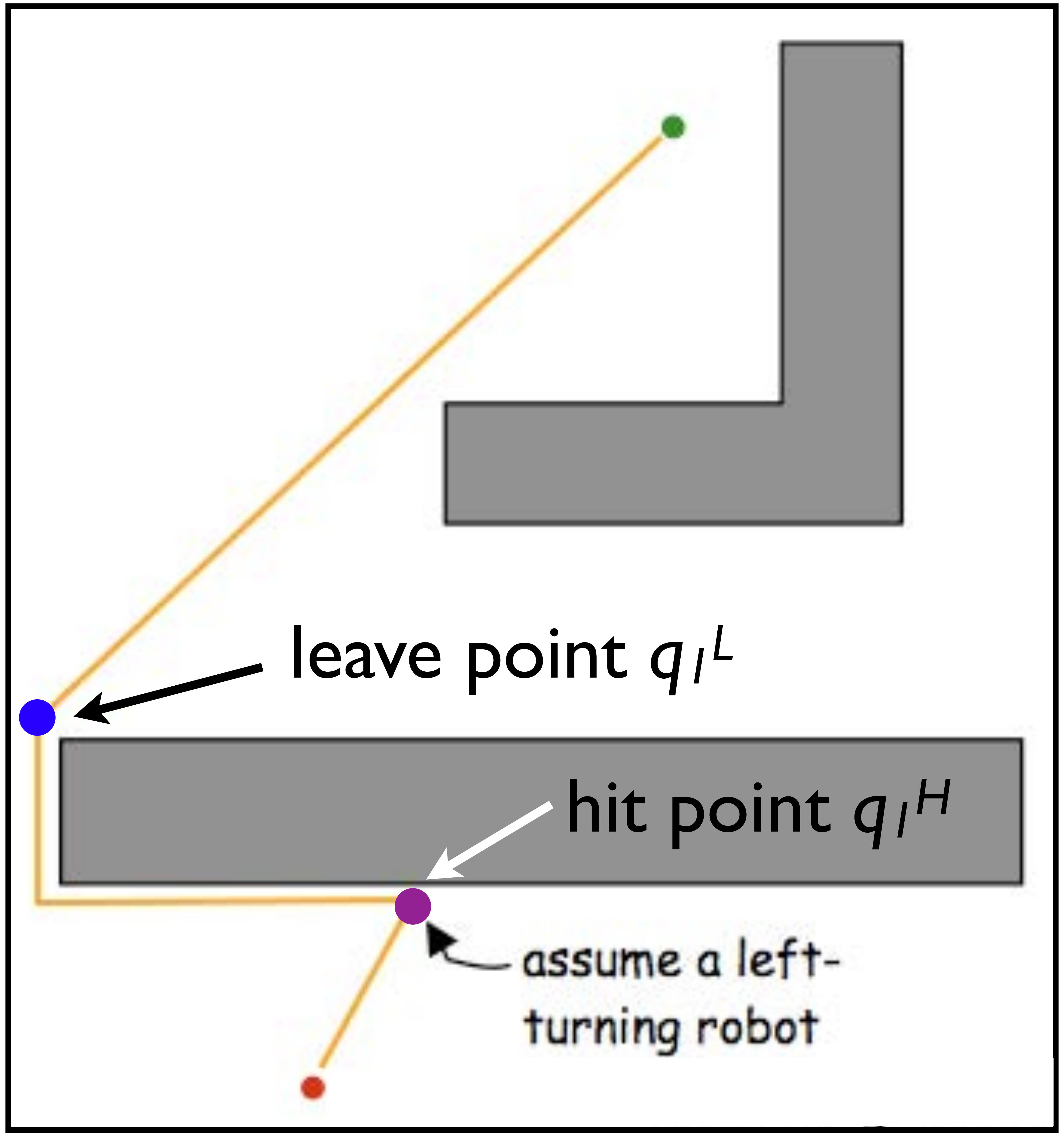


Plan navigation path from start q_s to goal q_d

as a sequence of hit/leave point pairs on obstacles

Hit point: q_i^H
Leave point: q_i^L

Bug 0



1) Head towards goal

2) When hit point set, follow wall, until you can move towards goal again (leave point)

3) continue from (1)

Slide borrowed from Michigan Robotics autorob.org

Wall following

follow wall

Trevor Jay

One approach:

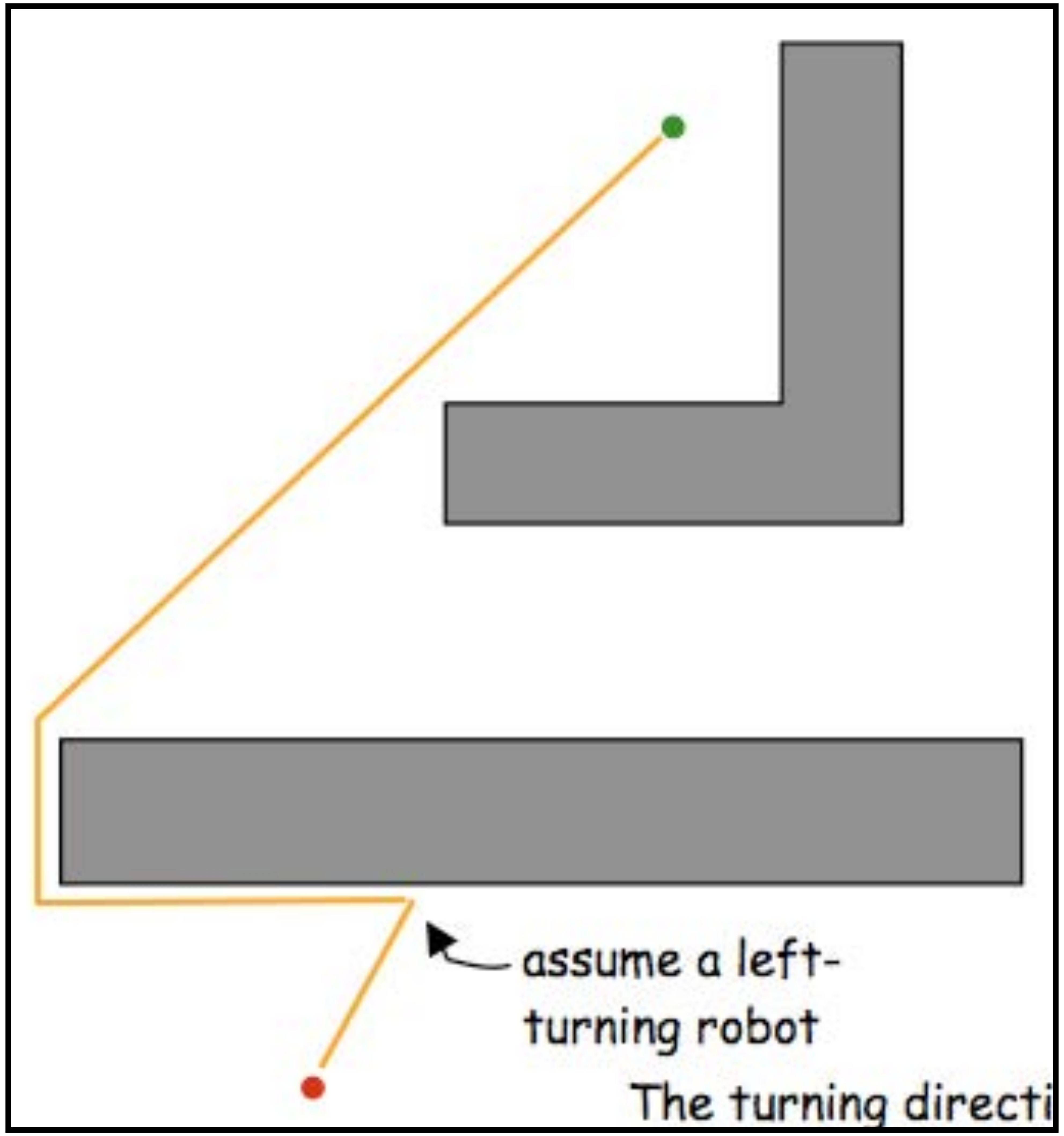
a) move forward with slight turn

b) when bumped, turn opposite direction

c) goto (*a*)

What map would foil Bug 0?

Bug 0

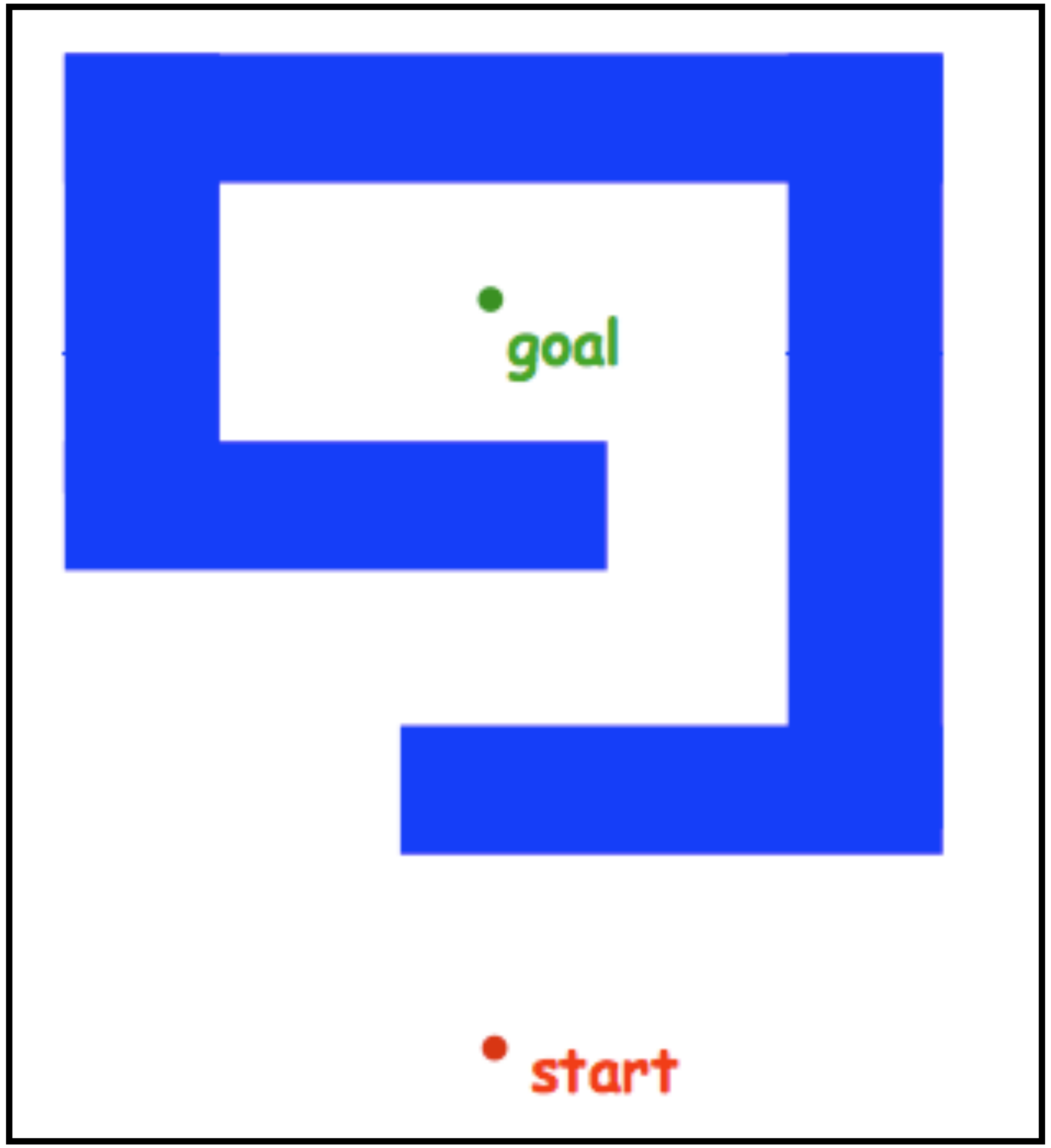


1) Head towards goal

2) When hit point set, follow wall, until you can move towards goal again (leave point)

3) continue from (1)

Bug 0



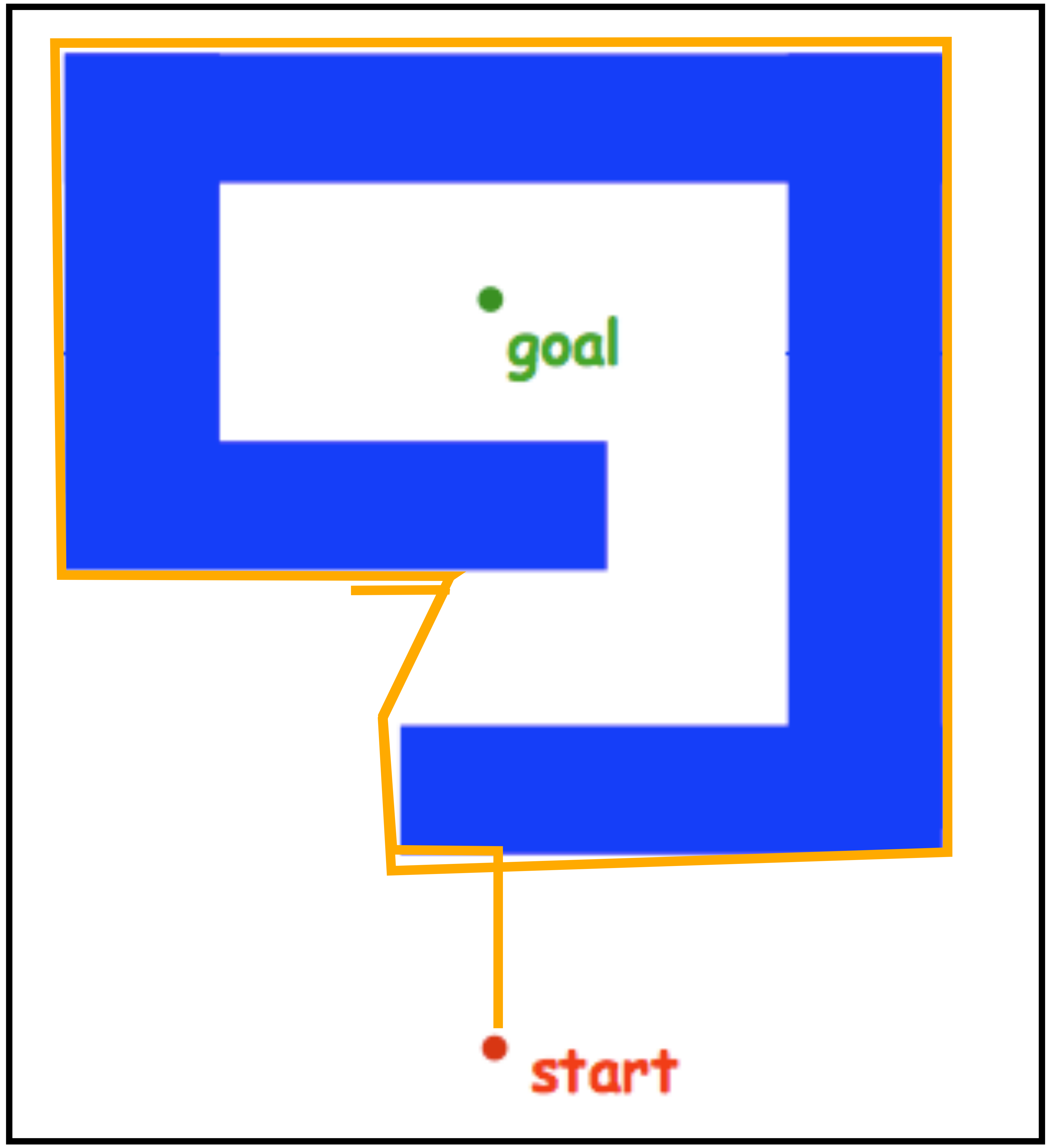
1) Head towards goal

2) When hit point set, follow wall, until you can move towards goal again (leave point)

3) continue from (1)

Can you trace the Bug 0 path?

Slide borrowed from Michigan Robotics autorob.org



Bug 0

1) Head towards goal

2) When hit point set, follow wall, until you can move towards goal again (leave point)

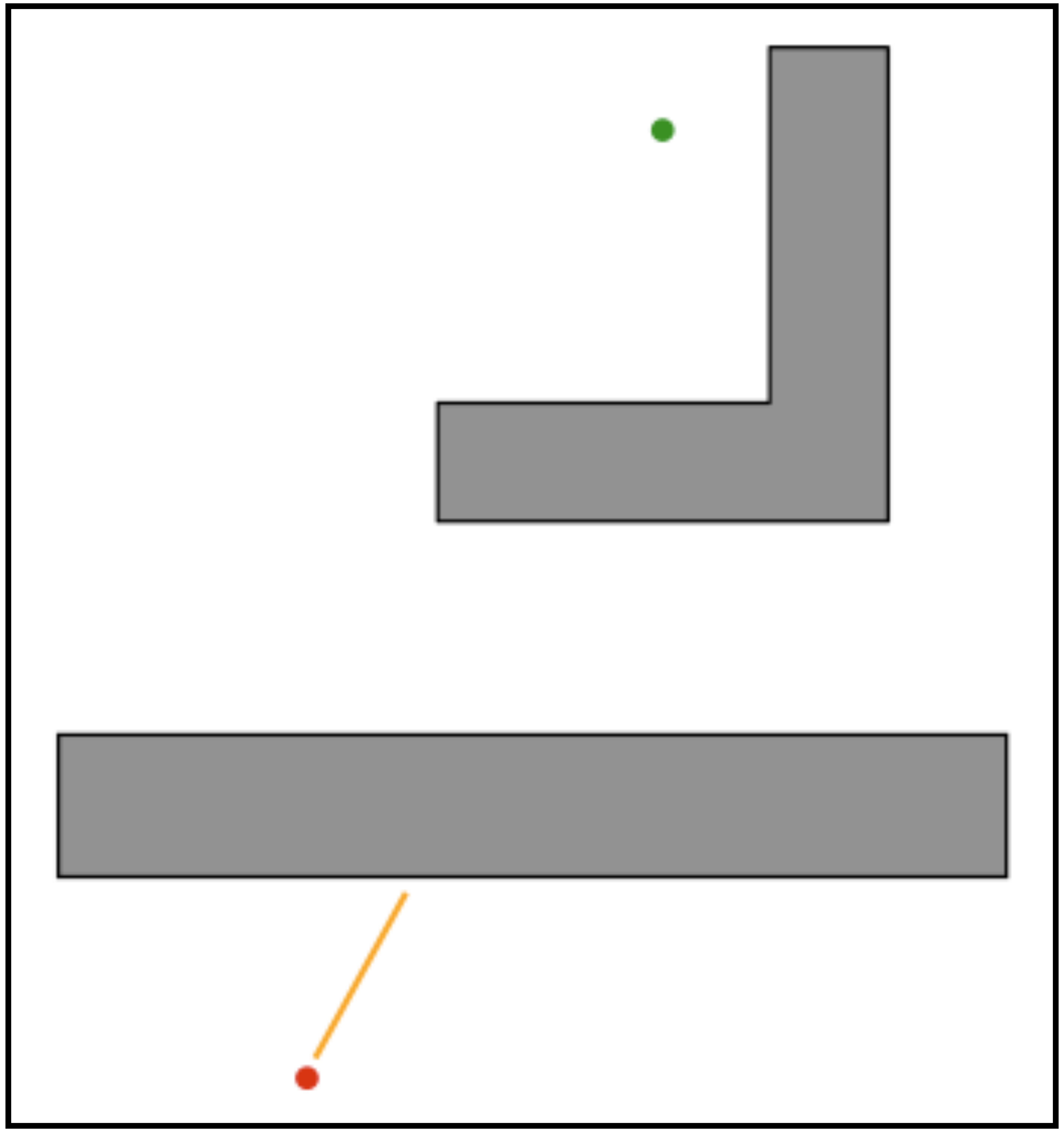
3) continue from (1)

Can you trace the Bug 0 path?

Can we make a better bug? How?

Slide borrowed from Michigan Robotics autorob.org

Bug 1



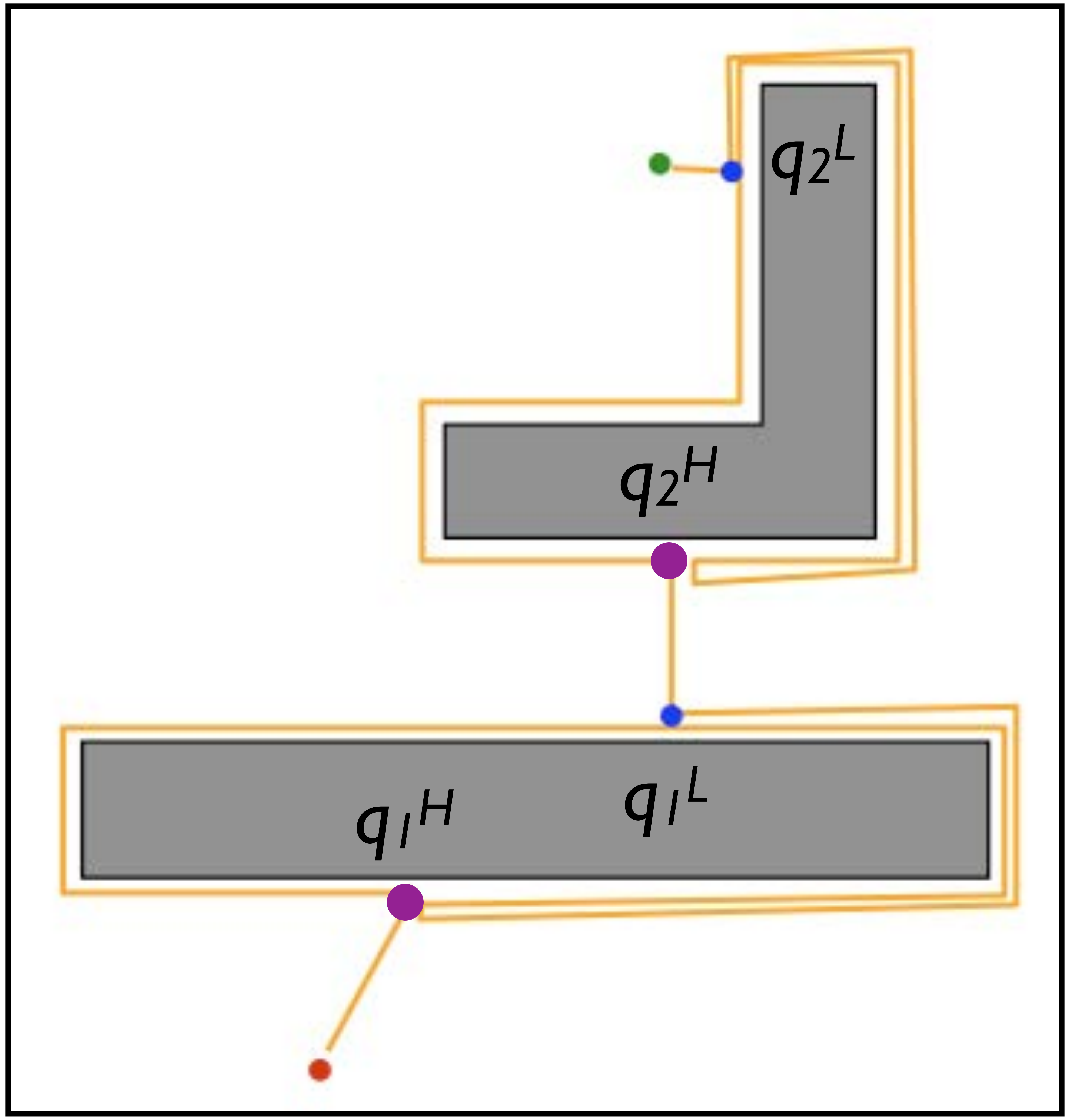
1) Head towards goal

2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) continue from (1)

Bug 1



1) Head towards goal

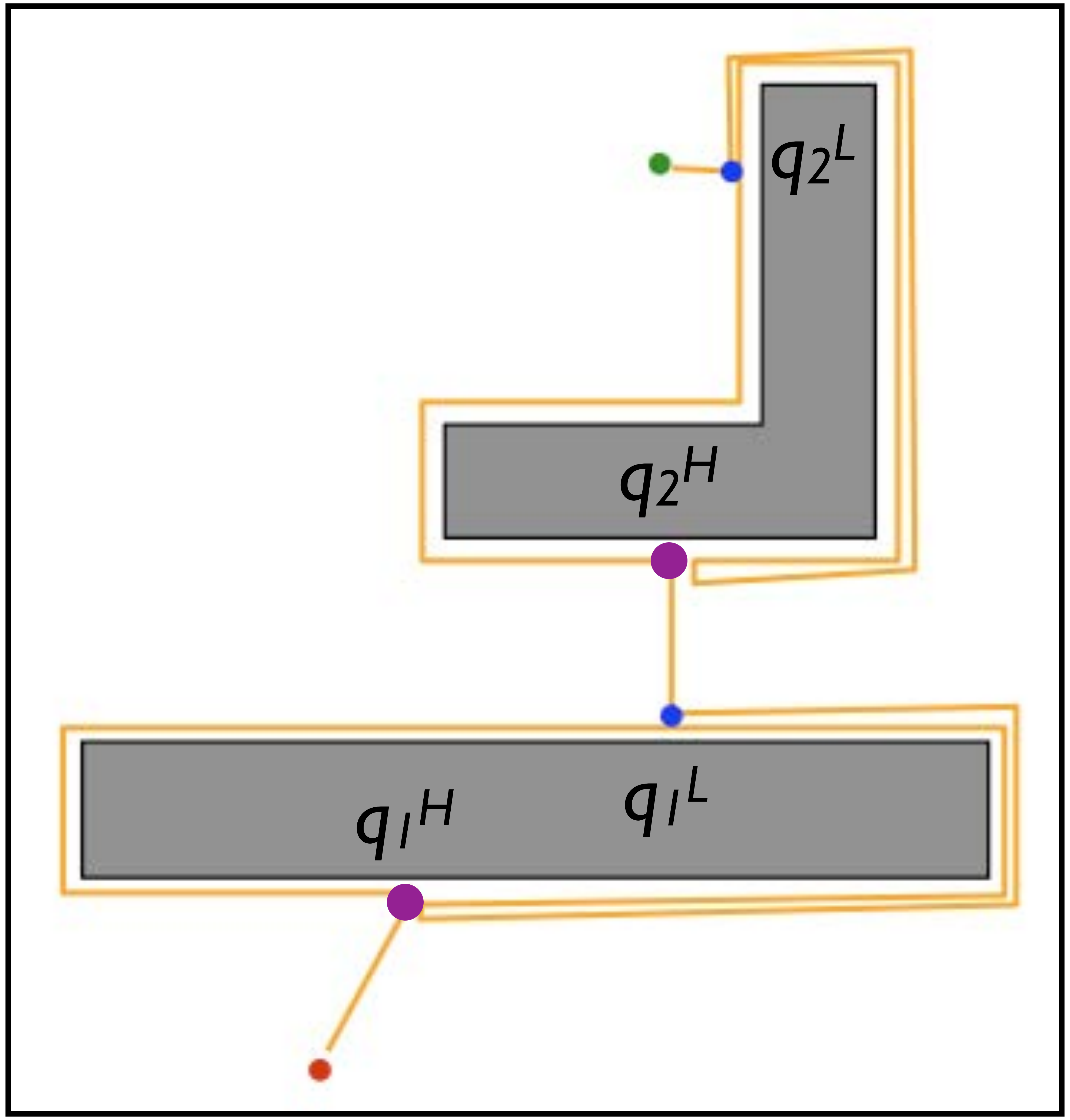
2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) continue from (1)

What map would foil Bug 1?

Bug 1



1) Head towards goal

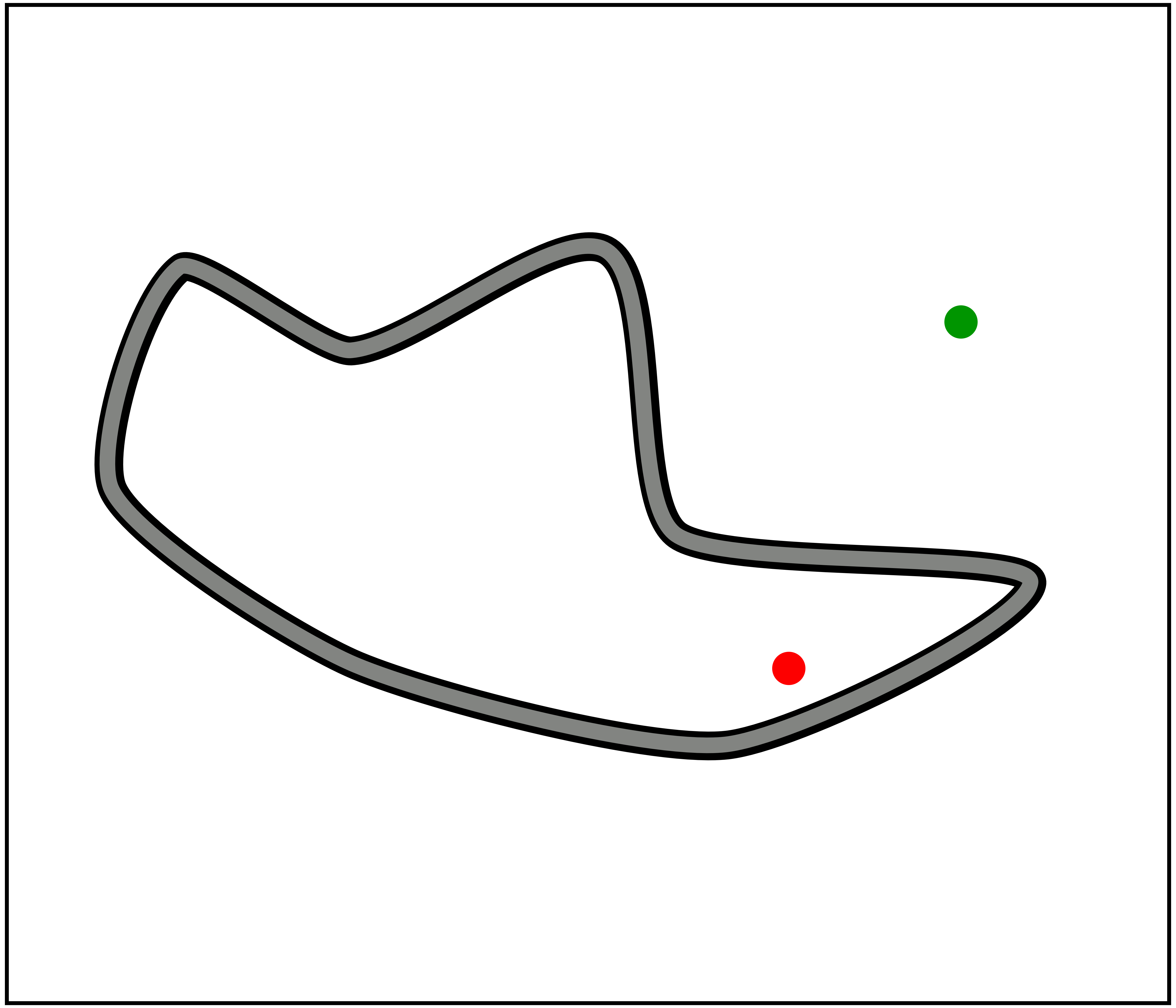
2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) continue from (1)

What map would foil Bug 1?

Bug 1



1) Head towards goal

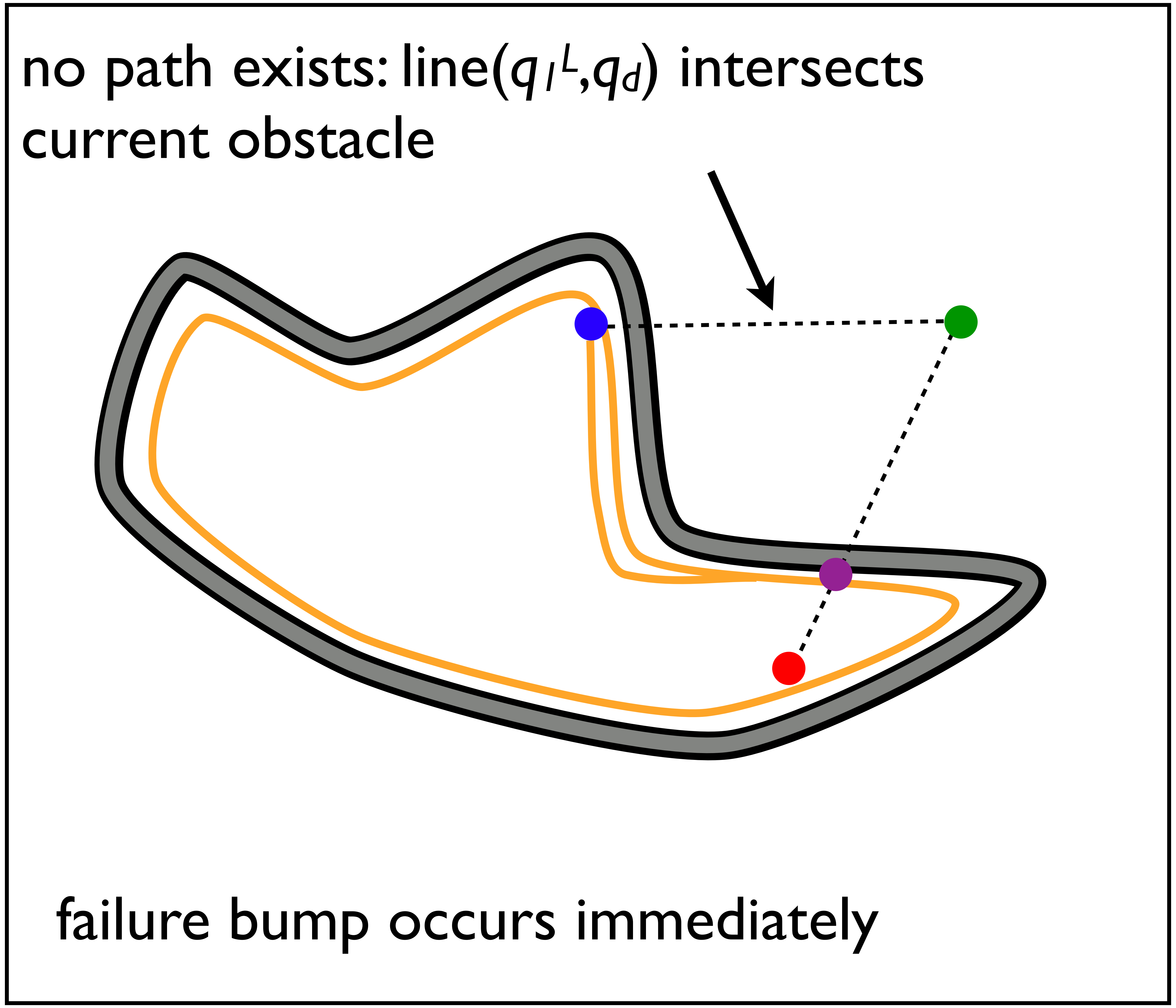
2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) if bump current obstacle,
return fail;
else, continue from (1)

What map would foil Bug 1?

Bug 1: Detecting Failure



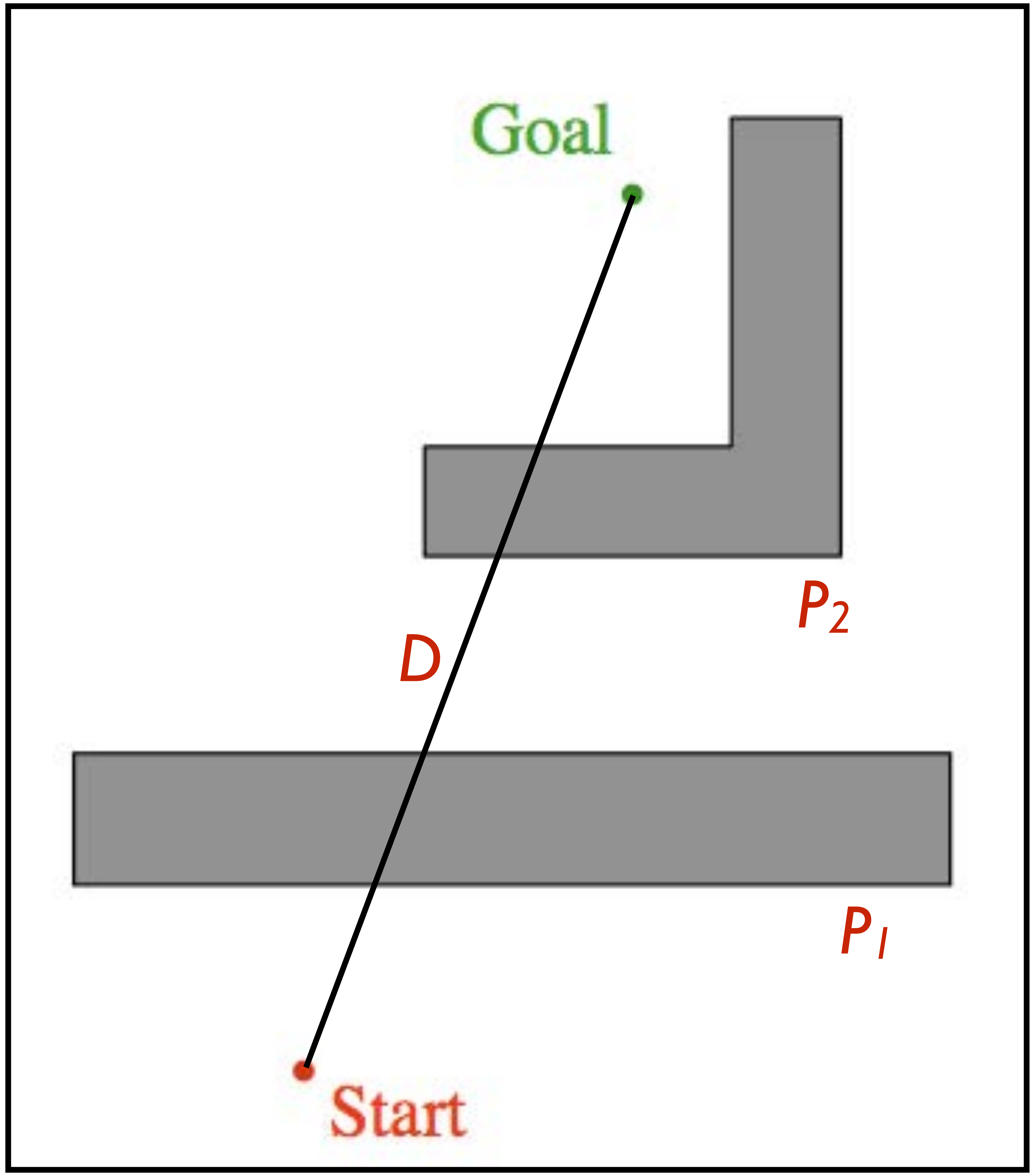
1) Head towards goal

2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) if bump current obstacle,
   return **fail**;
   else, continue from (1)

Bug 1: Search Bounds

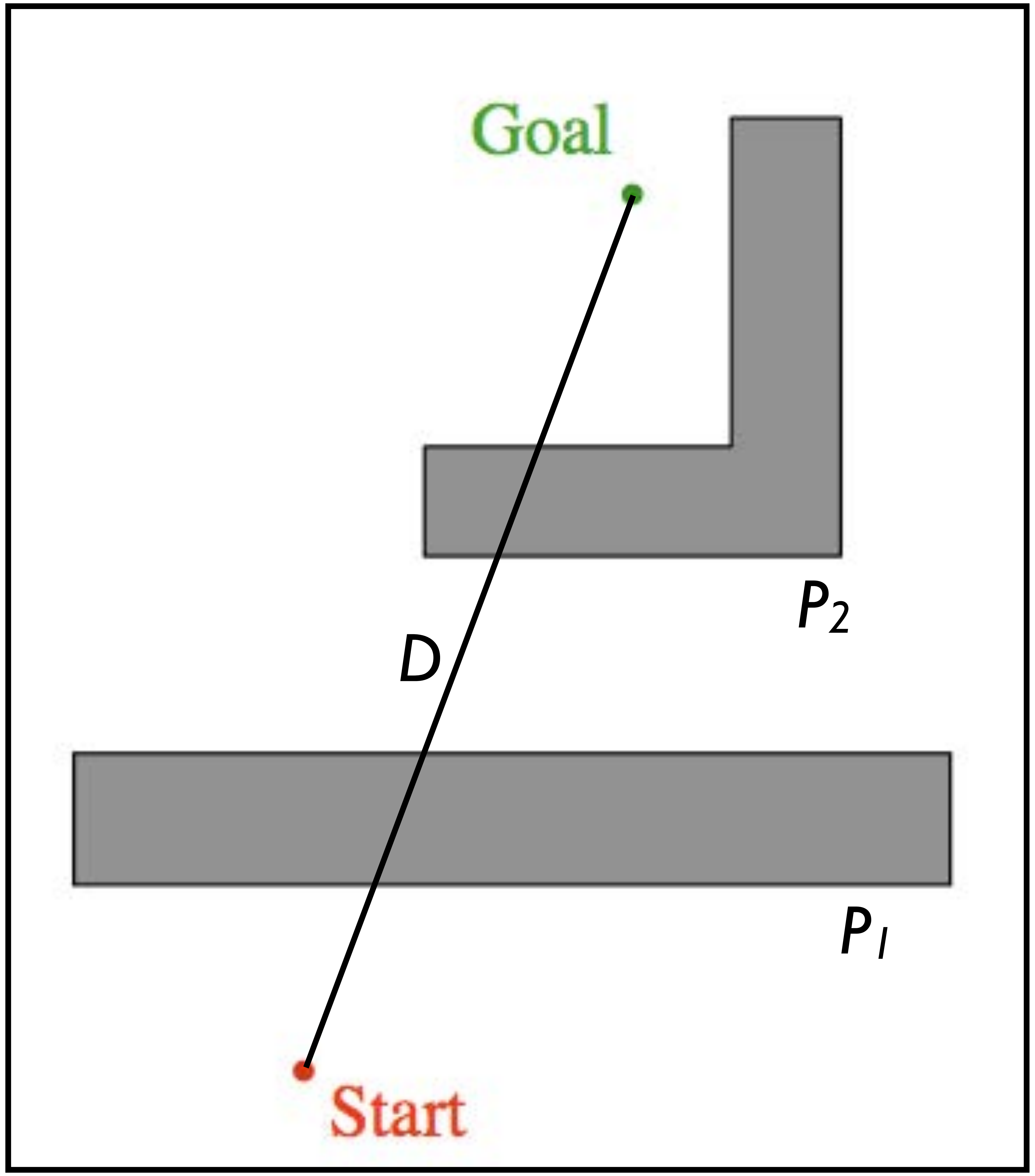


Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: [redacted]

Worst case: [redacted]

Bug 1: Search Bounds

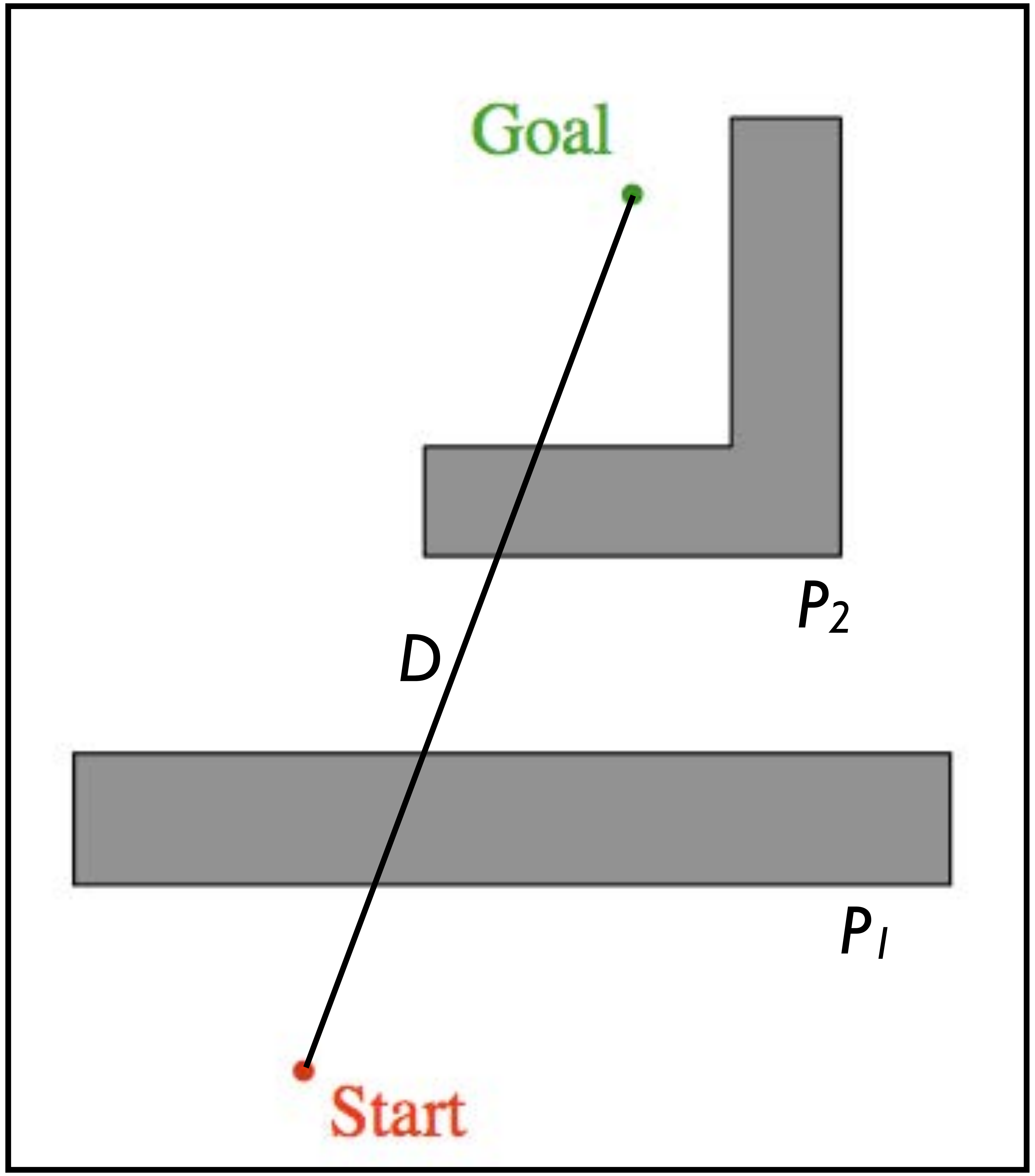


Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: D

Worst case: [redacted]

Bug 1: Search Bounds



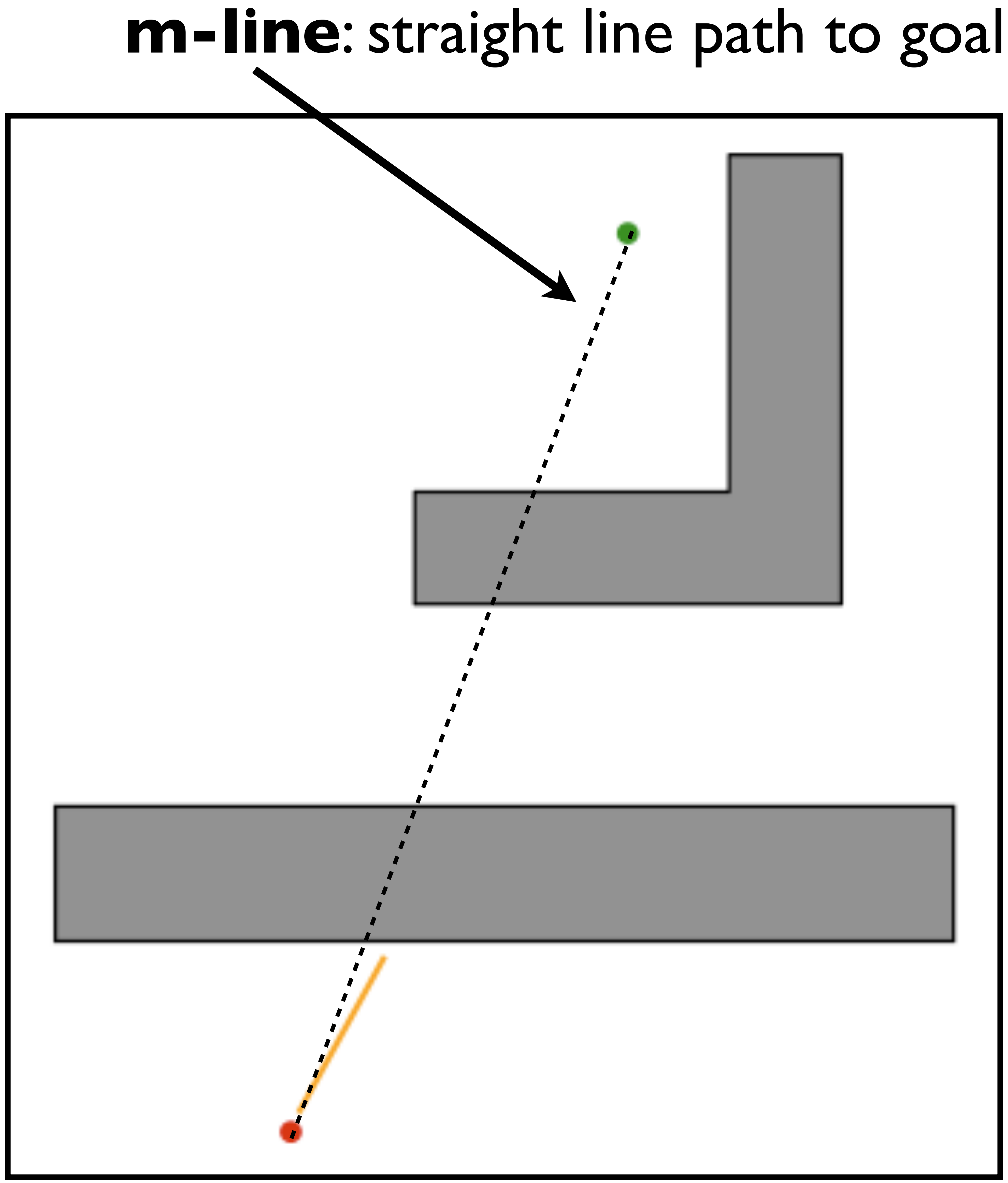
Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: D

Worst case: $D + 1.5\sum_i P_i$

Is there a faster bug?

Bug 2



1) Head towards goal on m-line

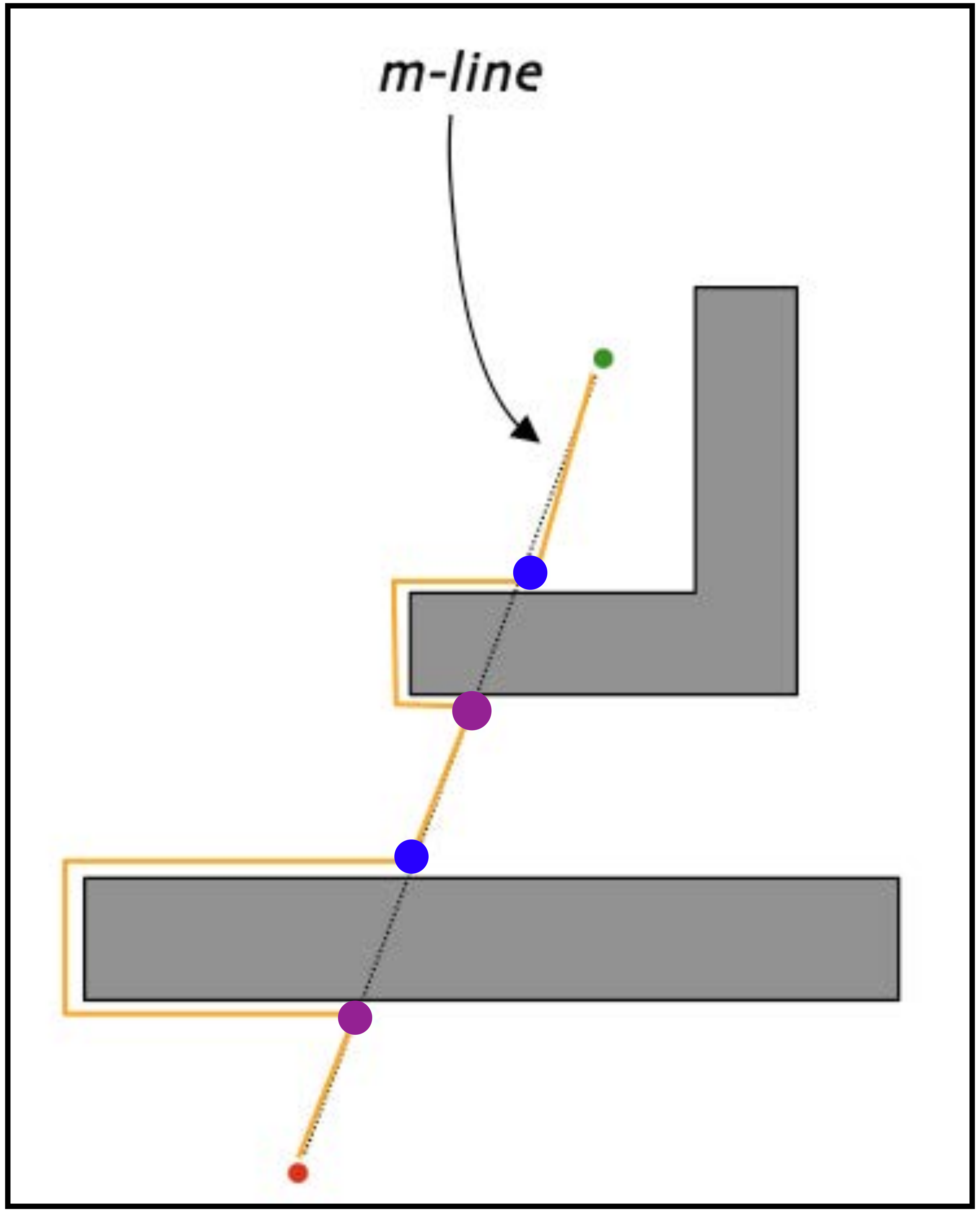
2) When hit point set, traverse obstacle until m-line is encountered

3) set leave point and exit obstacle

4) continue from (1)

Slide borrowed from Michigan Robotics autorob.org

Bug 2



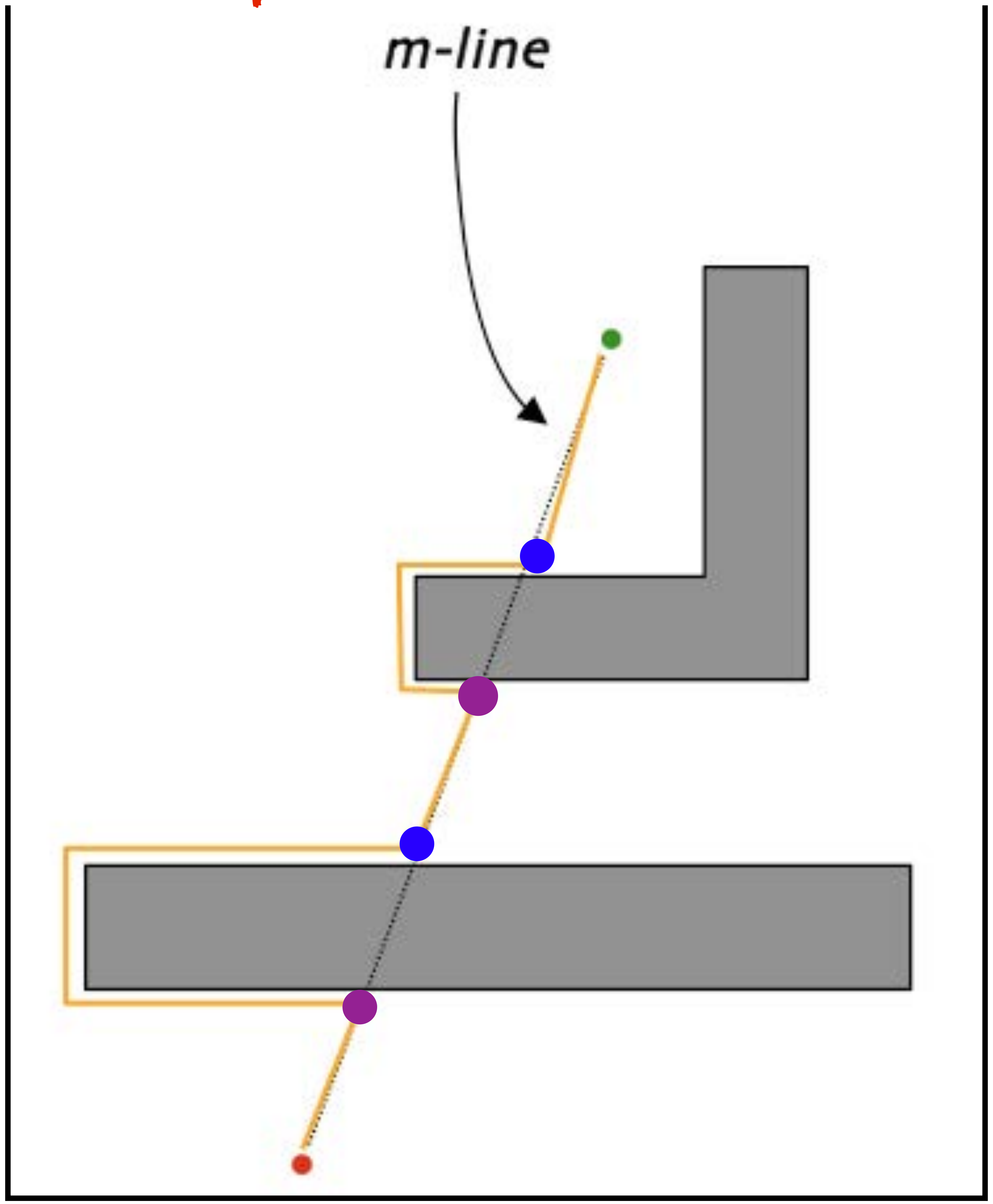
1) Head towards goal on m-line

2) When hit point set, traverse obstacle until m-line is encountered

3) set leave point and exit obstacle

4) continue from (1)

What map would foil Bug 2?



Bug 2

1) Head towards goal on m-line

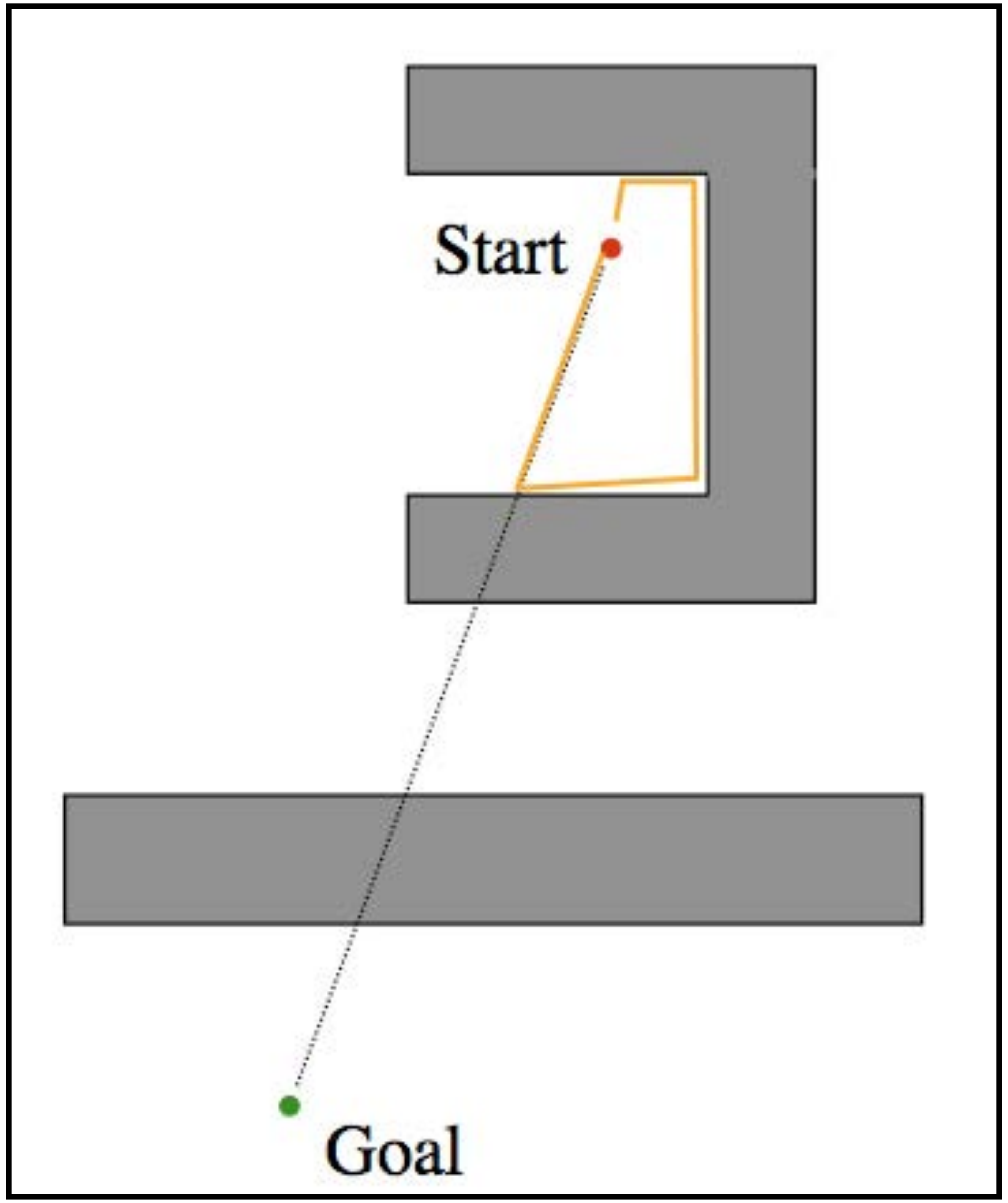
2) When hit point set, traverse obstacle until m-line is encountered

3) set leave point and exit obstacle

4) continue from (1)

Slide borrowed from Michigan Robotics autorob.org

Bug 2



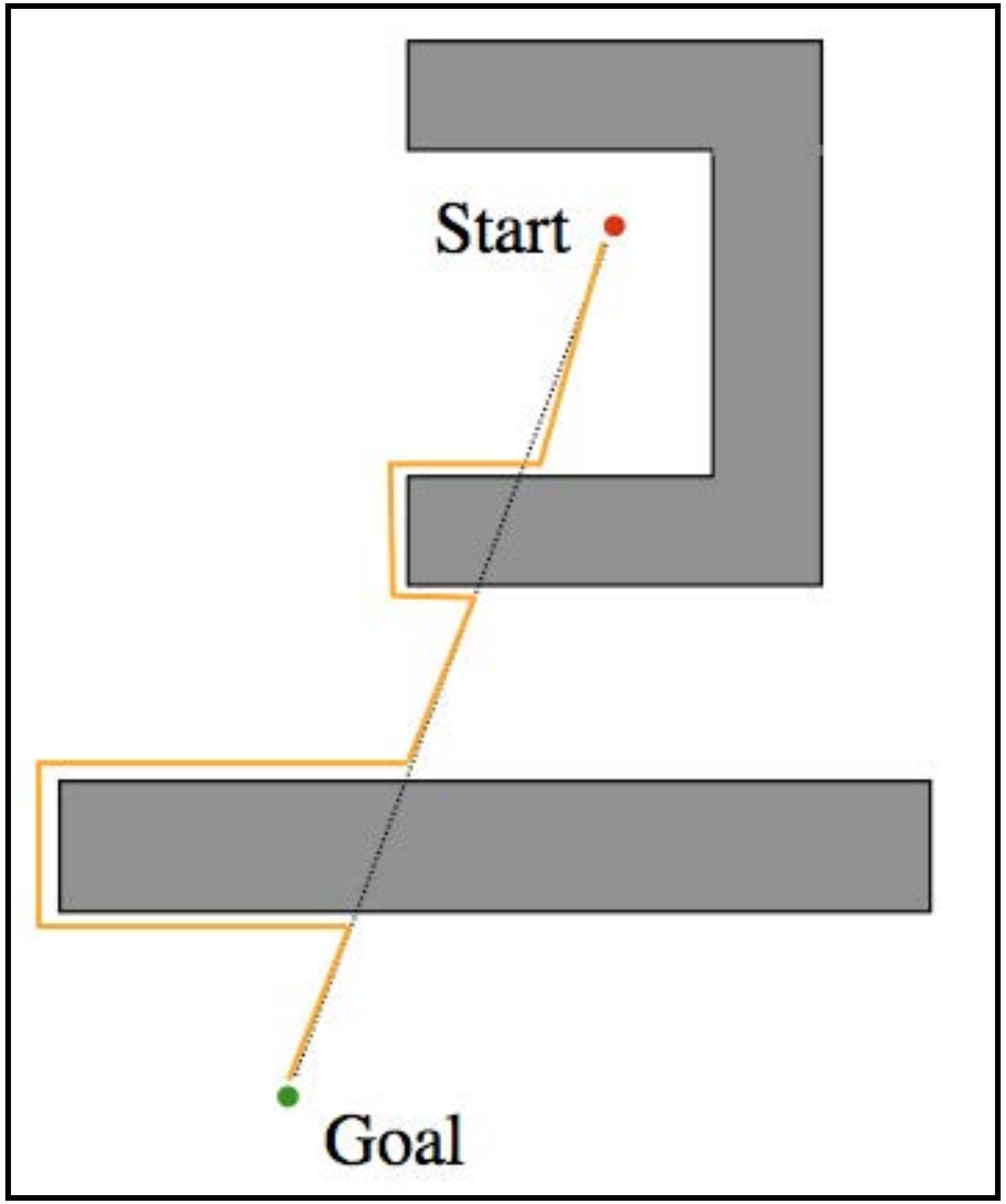
1) Head towards goal on m-line

2) When hit point set, traverse obstacle until m-line is encountered

3) set leave point and exit obstacle

4) continue from (1)

Bug 2



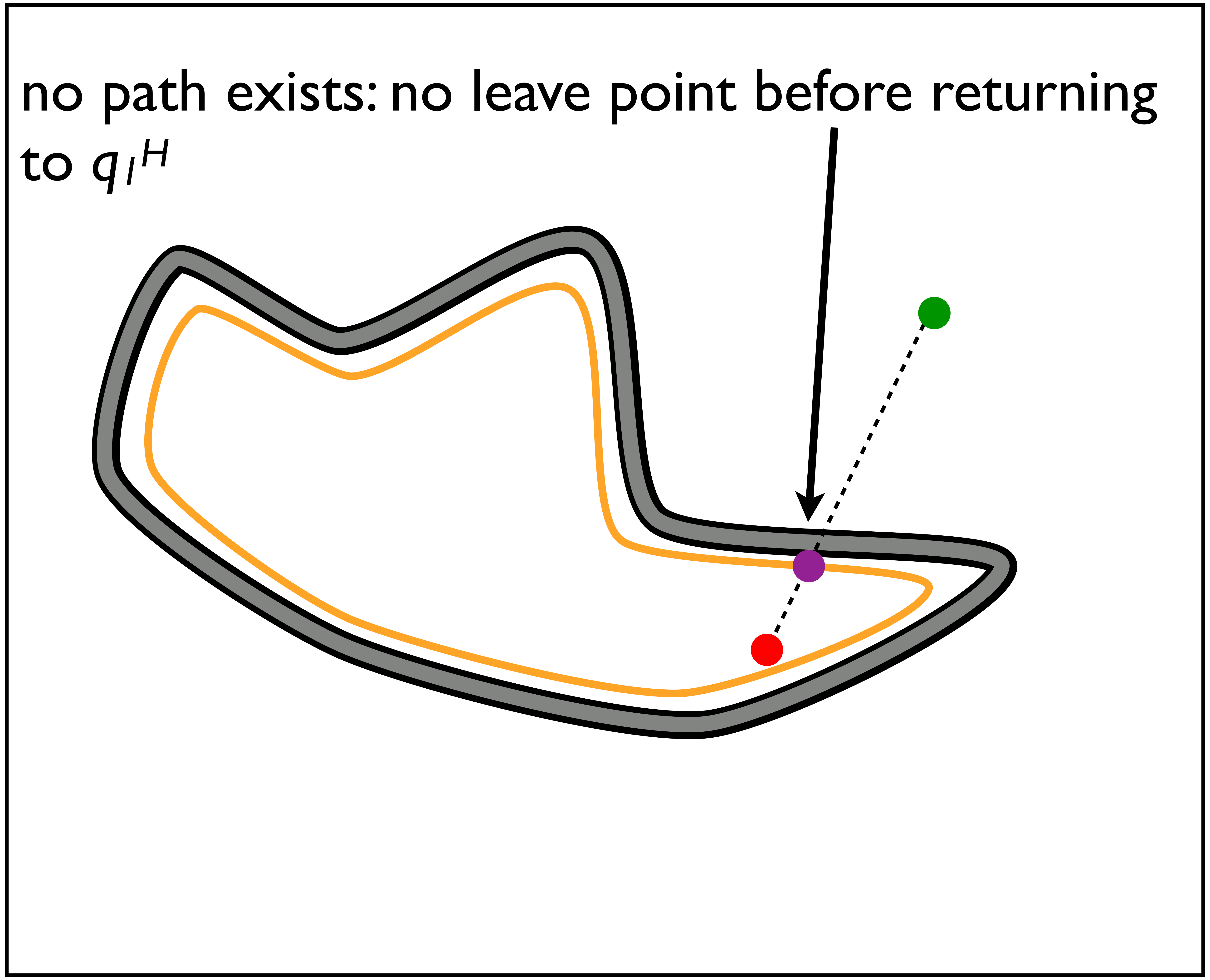
1) Head towards goal on m-line

2) When hit point set, traverse obstacle until m-line is encountered **& closer to the goal**

3) set leave point and exit obstacle

4) continue from (1)

Bug 2: Detecting Failure



1) Head towards goal on m-line

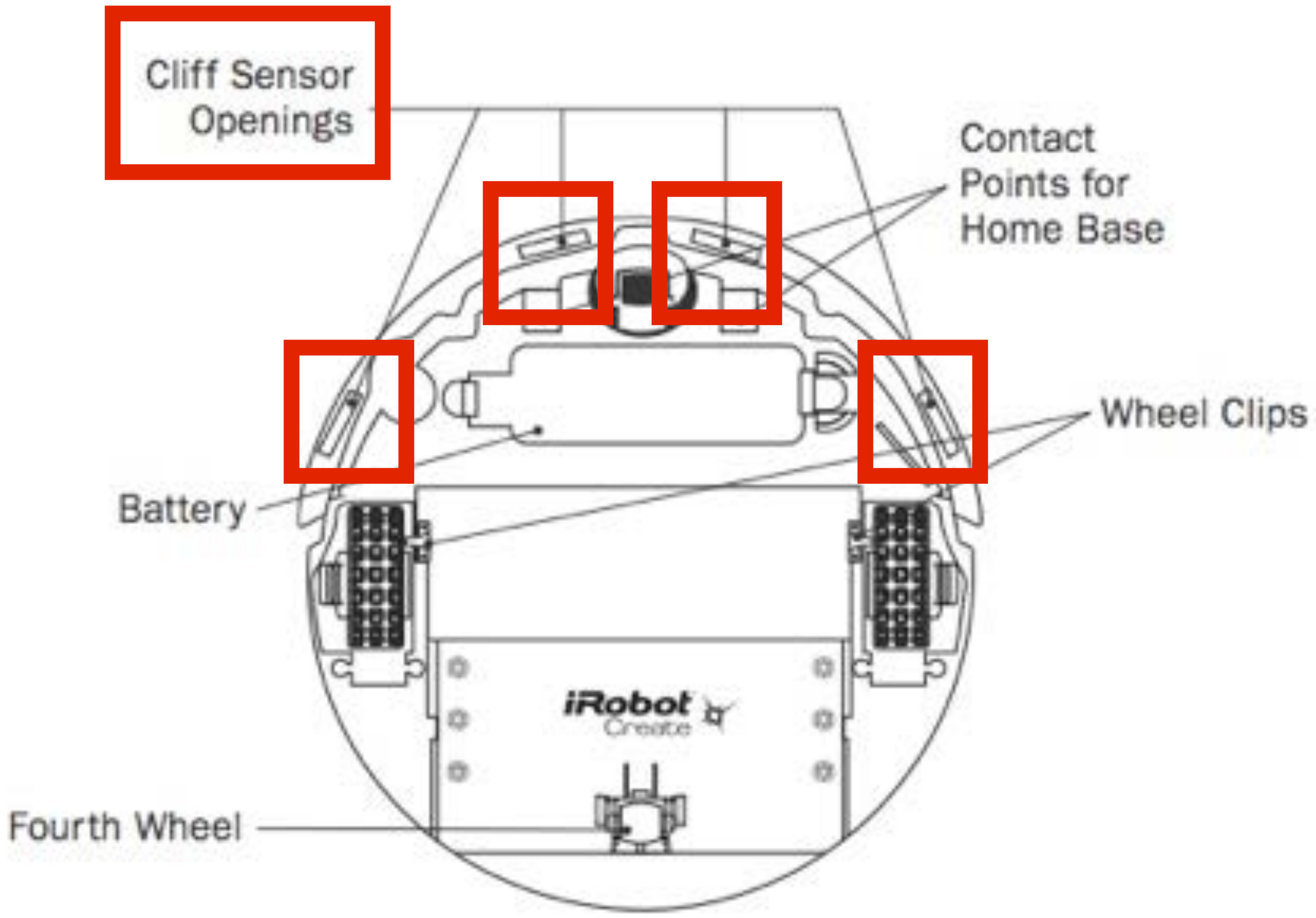
2) When hit point set, traverse obstacle until m-line is encountered & closer to the goal
or hit point reached

3) **if not i^{th} hit point,** set leave pt. and exit

4) continue from (1)

Bug 2 in action

m-line drawn on floor with tape recognizable by Create cliff sensor



Kayle Gishen

Is Bug2 better than Bug1?

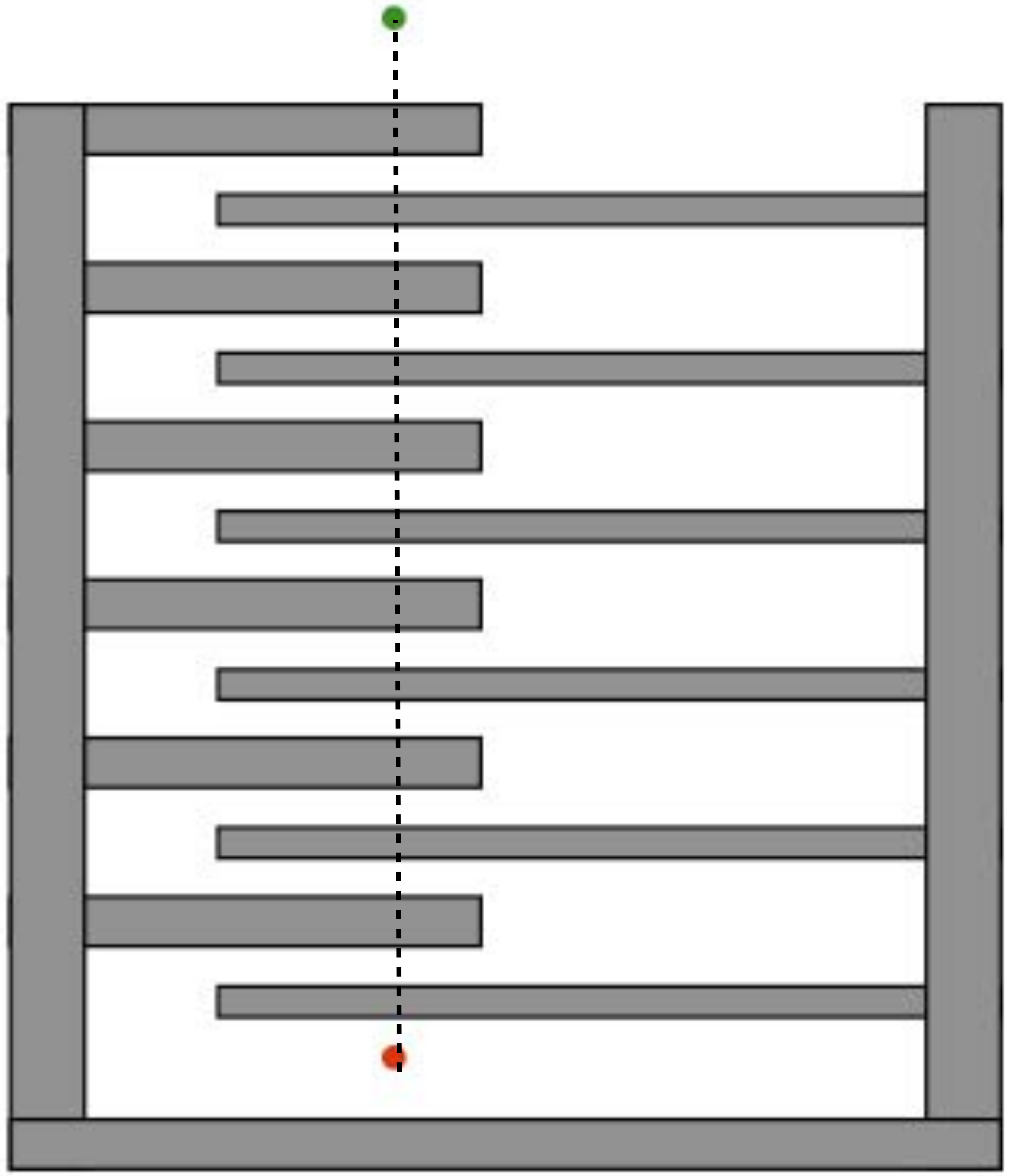
Bug 1 v. Bug 2:

Draw worlds where Bug 2 performs better than Bug 1 (and vice versa)

Bug 2 beats Bug 1	Bug 1 beats Bug 2

Home work!

Bug 2: Search Bounds

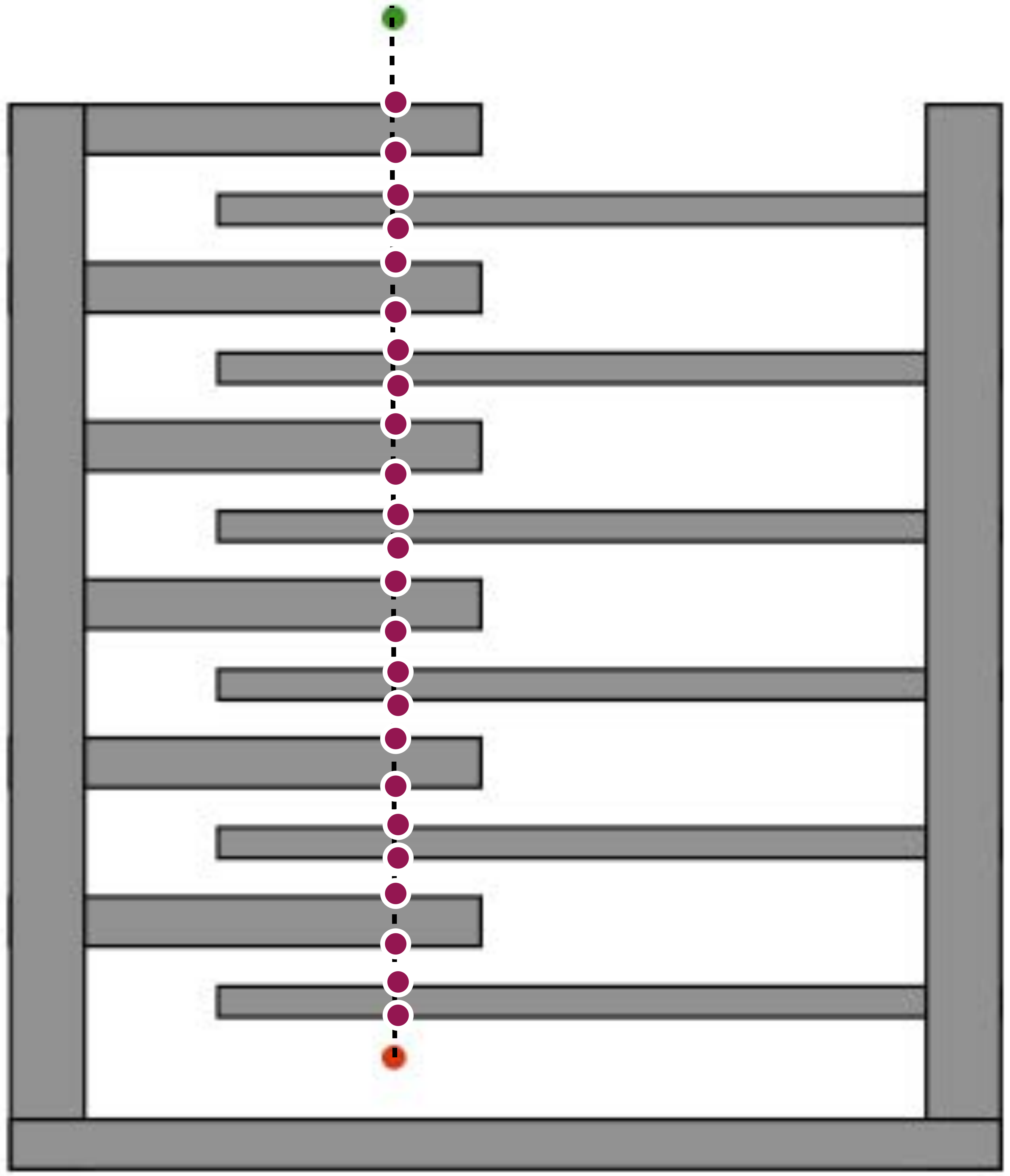


Bounds on path distance, assuming

- *D*: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: [redacted]

Worst case: [redacted]



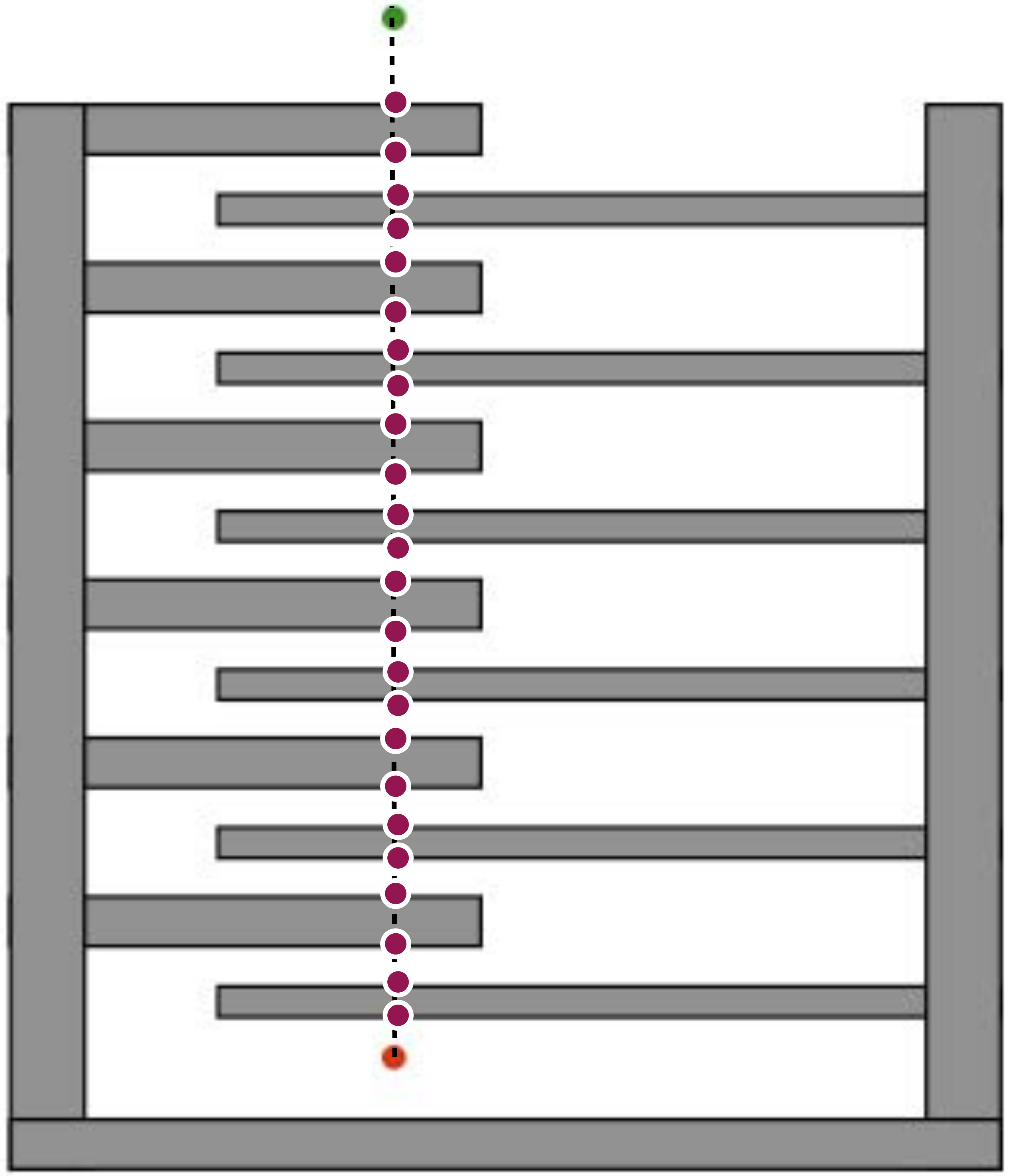
Bug 2: Search Bounds

Bounds on path distance, assuming

- D: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: D

Worst case: [redacted]



Bug 2: Search Bounds

Bounds on path distance, assuming

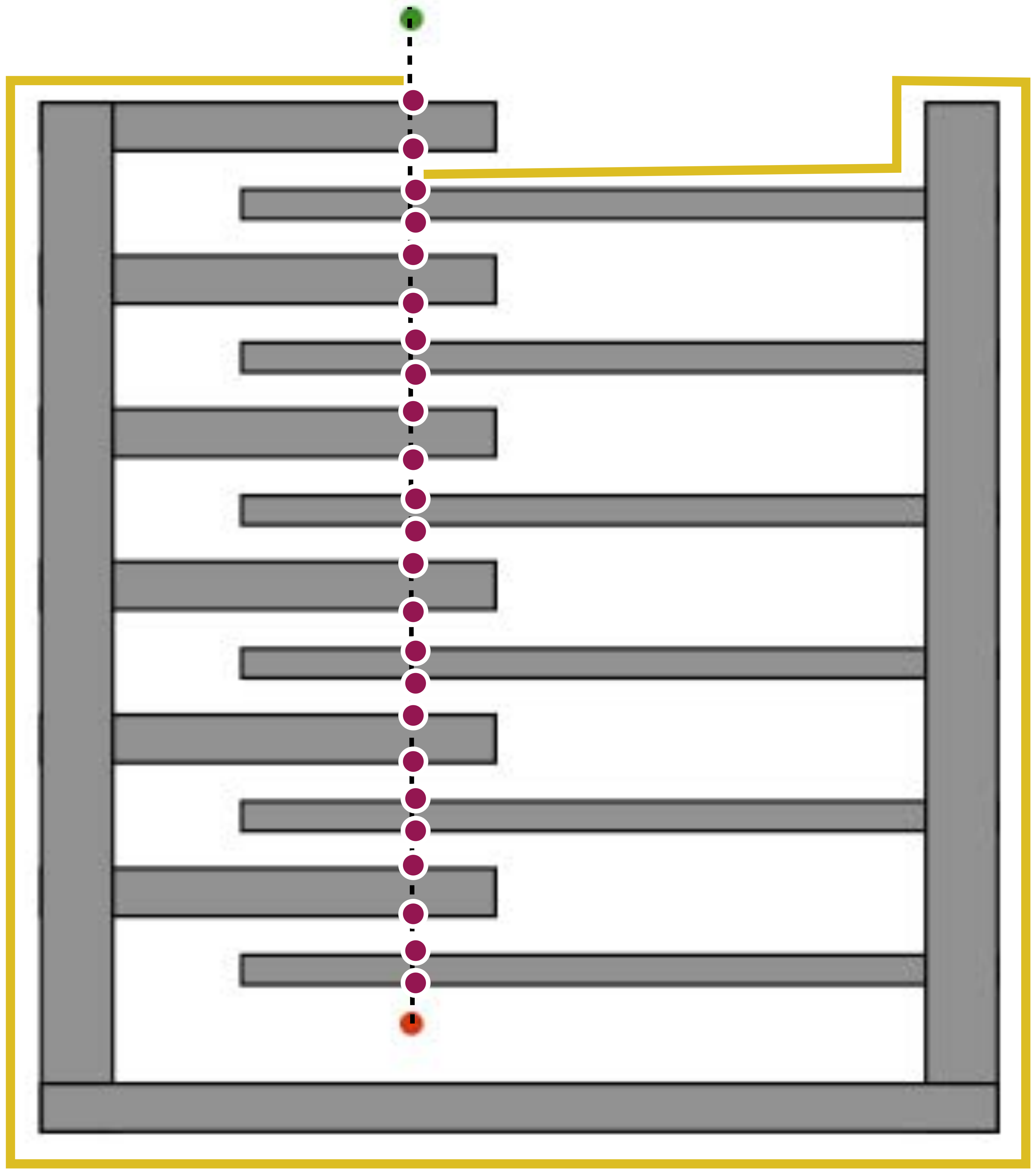
- D: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: D

Worst case: $D + \sum_i (n_i/2)P_i$

Why?

Why? Consider all leave points on m-line; only half are valid



Each leave pt might require traversing entire obstacle perimeter, including the outside

Bug 2: Search Bounds

Bounds on path distance, assuming

- D: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: D

Worst case: $D + \sum_i (n_i/2)P_i$

Search Bounds:

Bug 1

Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: *D*

Worst case: $D + 1.5\sum_i P_i$

Bug 2

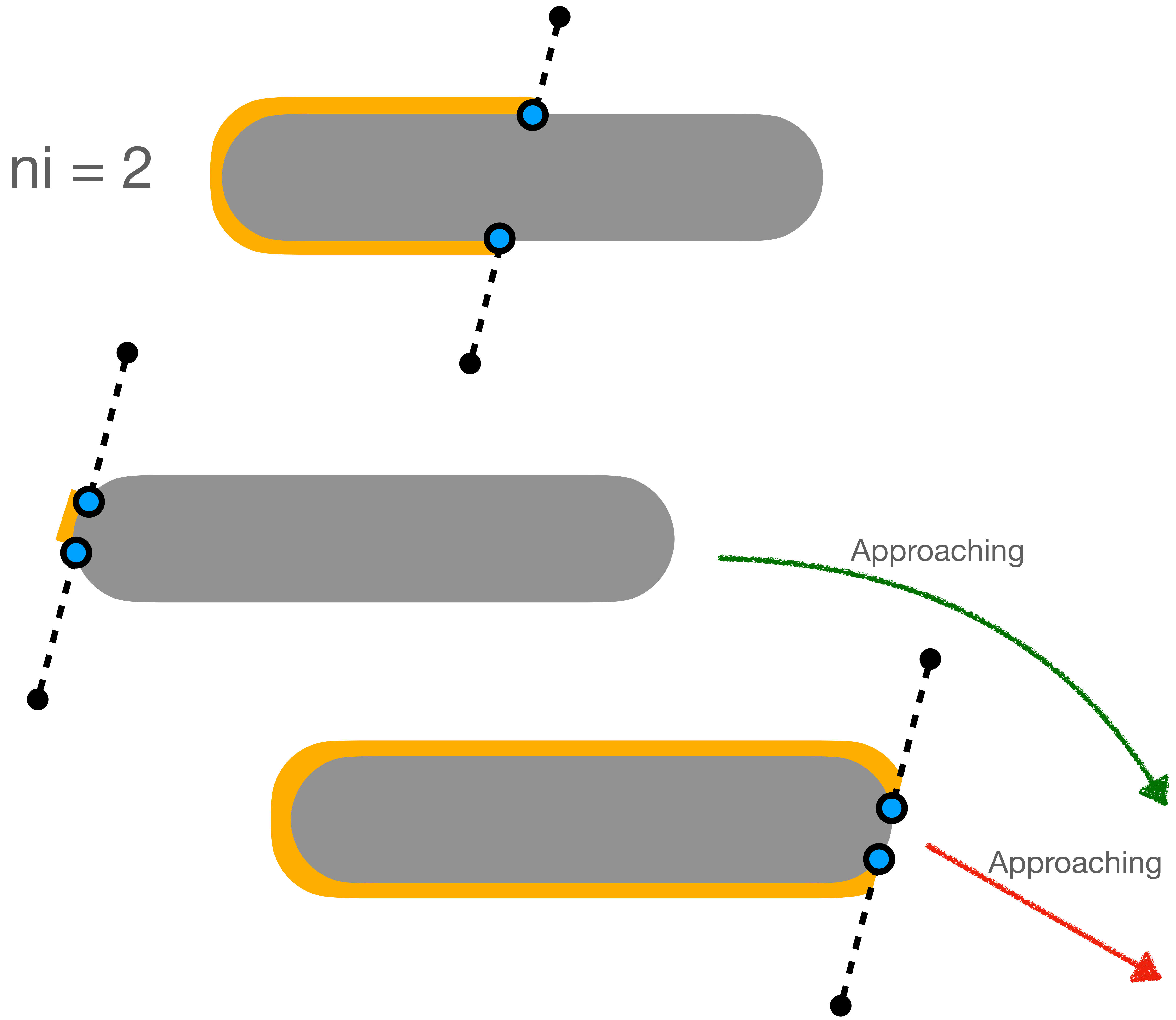
Bounds on path distance, assuming

- *D*: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: *D*

Worst case: $D + \sum_i (n_i/2)P_i$

Search Bounds:



Bug 2

Bounds on path distance, assuming

- *D*: distance start-to-goal
- P_i: obstacle perimeter
- n_i: number of m-line intersections for WO_i

Best case: *D*

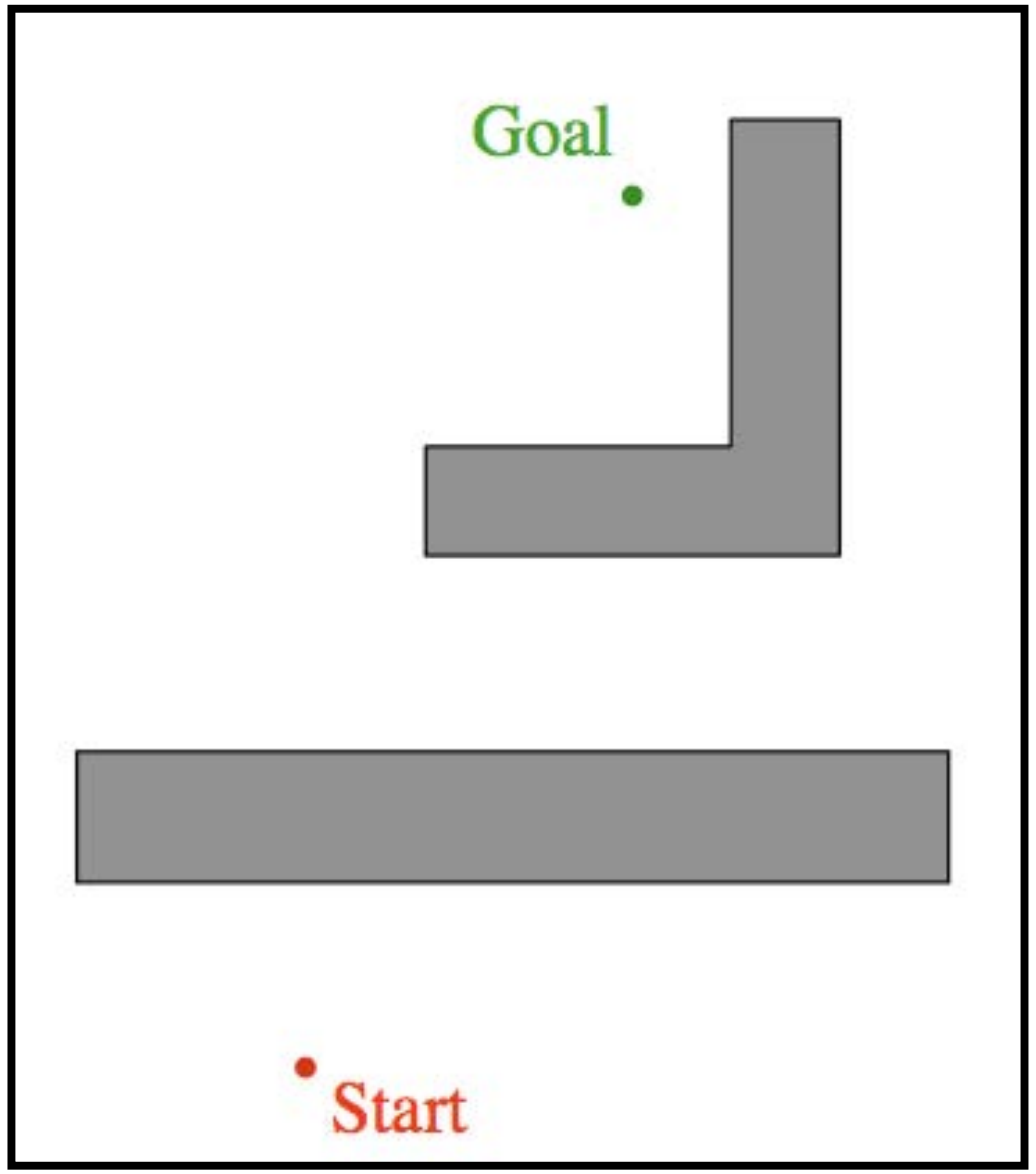
Worst case: $D + \sum_i (n_i/2) P_i$

Suppose robot has a range sensor.

Is there a better Bug algorithm?

Tangent Bug

- Assume bounded world
- Known: global goal
  - measurable distance *d(x,y)*
- Local sensing
  - **range finding**
  - odometry



>

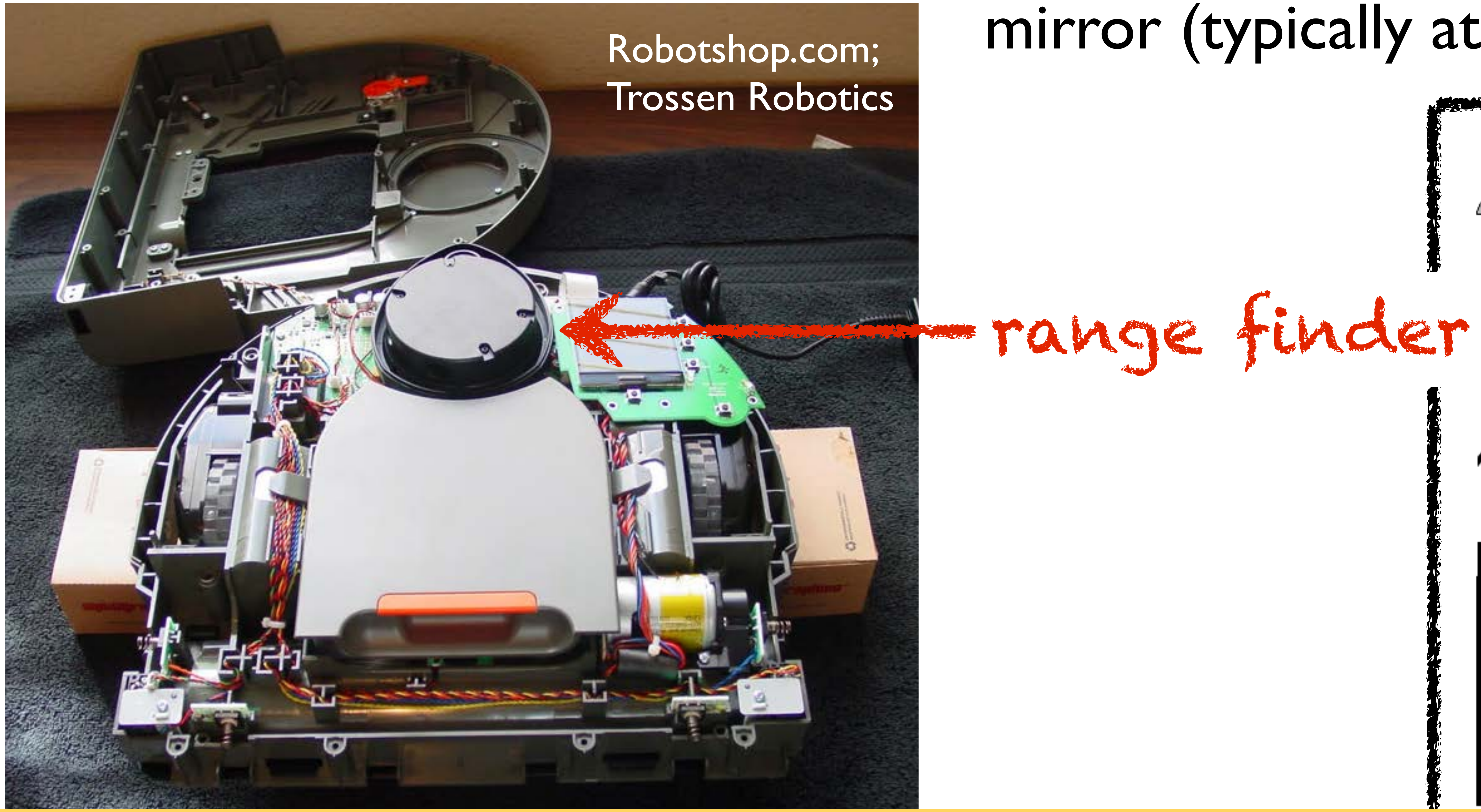
Laser Rangefinding

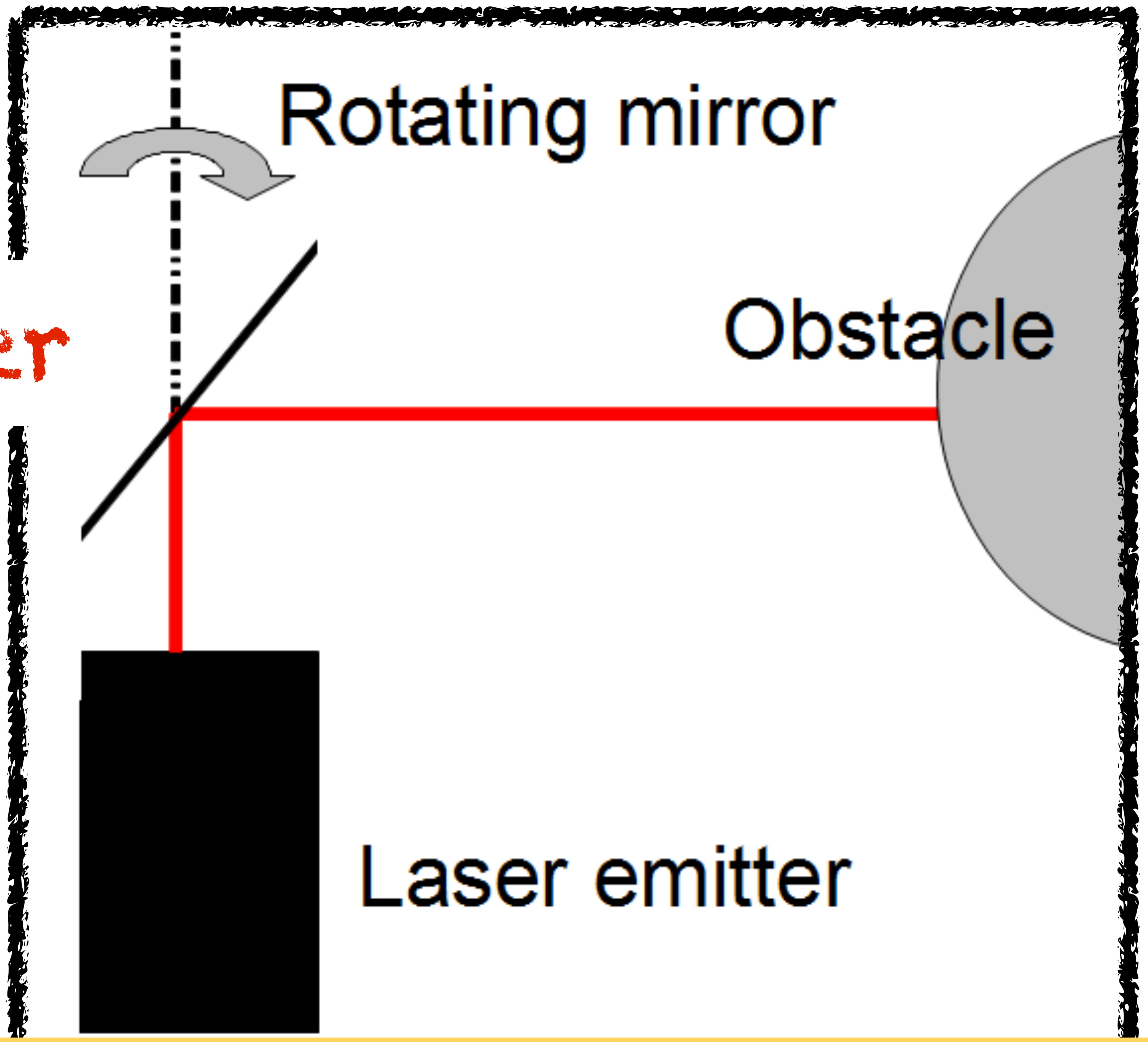
(briefly)

Emit laser beam in a direction

Distance to nearest object related to time from emission to sensing of beam
(assumes speed of light is known)

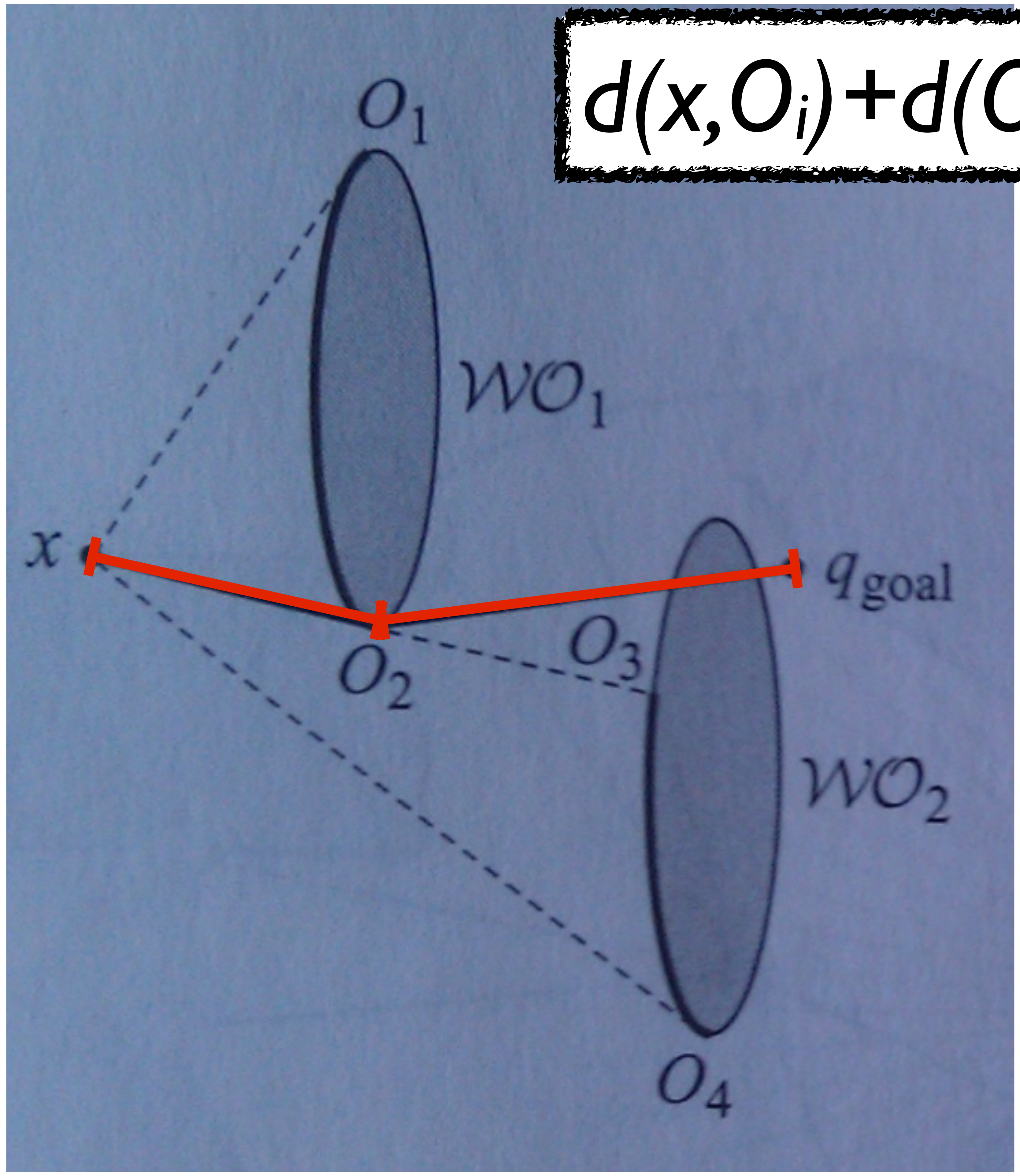
Planar range finding : reflect laser on spinning mirror (typically at 10Hz)





Slide borrowed from Michigan Robotics autorob.org

Tangent Bug: Heuristic Distance-to-Goal



$d(x,O_i)+d(O_i,q_{goal})$

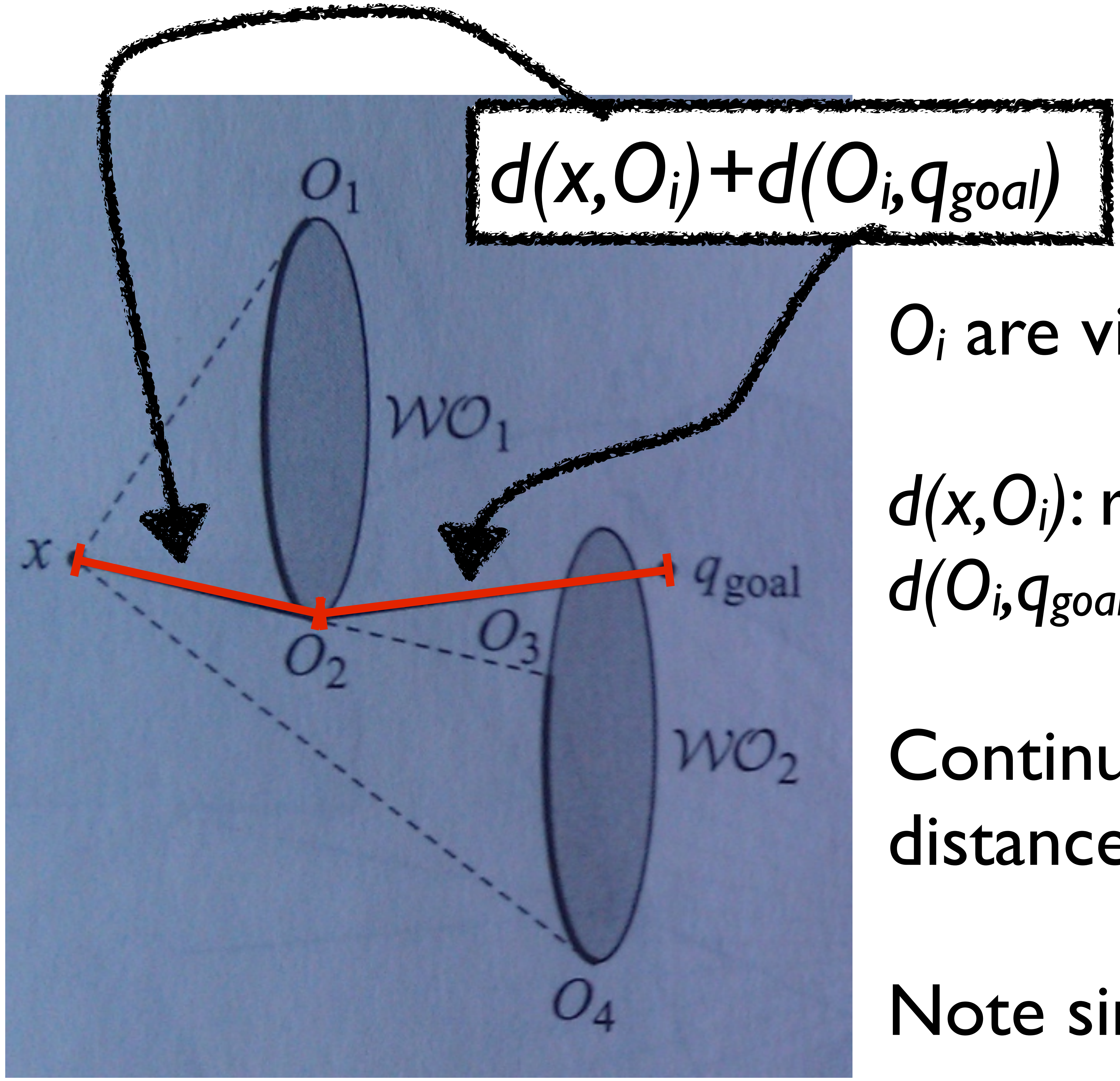
O_i are visible obstacle extents

$d(x,O_i)$: robot can see
$d(O_i,q_{goal})$: best path robot cannot see

Continually move robot such that distance to goal is decreased

Note similarity to A* search heuristic

Tangent Bug: Heuristic Distance-to-Goal



O_i are visible obstacle extents

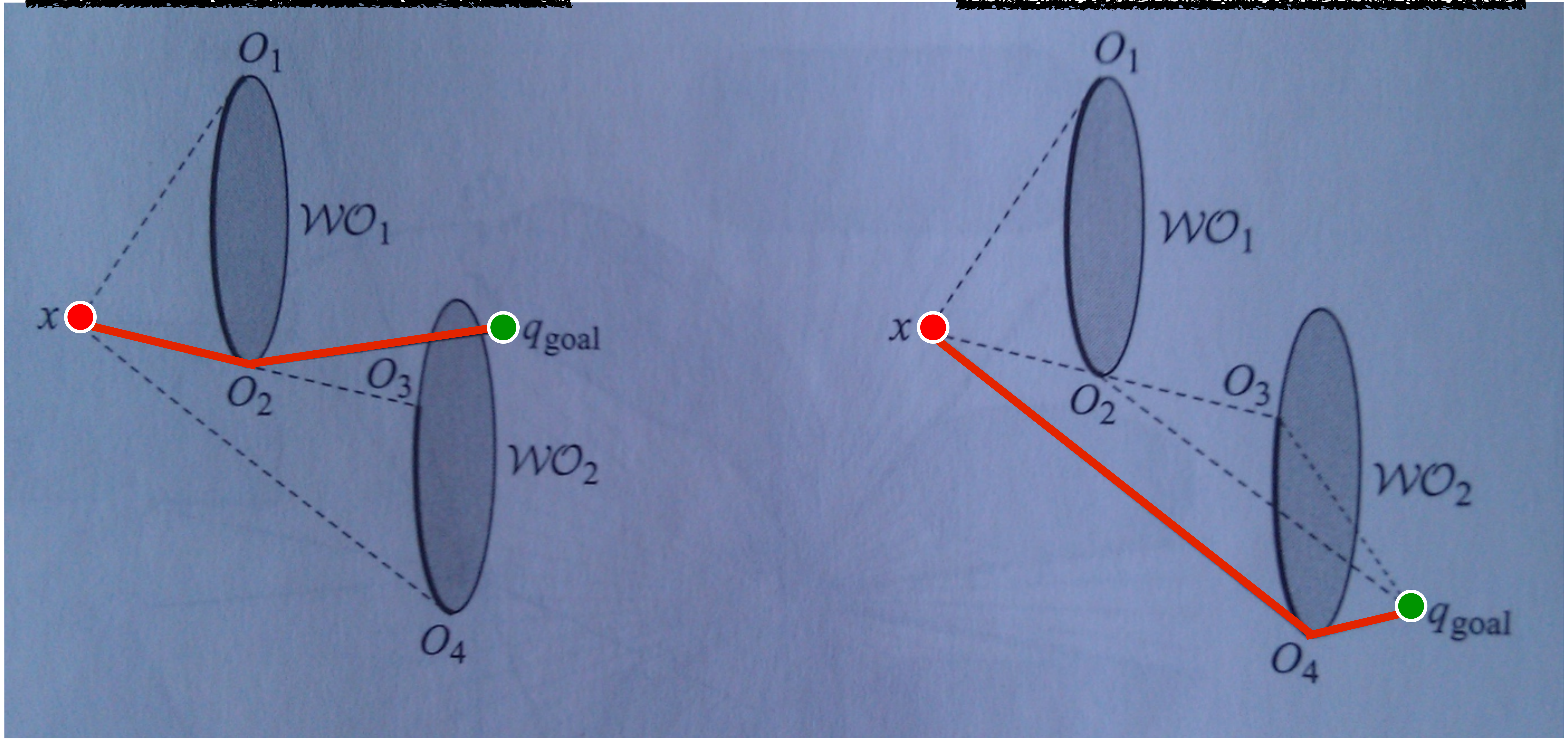
$d(x,O_i)$: robot can see
$d(O_i,q_{goal})$: best path robot cannot see

Continually move robot such that distance to goal is decreased

Note similarity to A* search heuristic

$d(x, O_2) + d(O_2, q_{goal})$

$d(x, O_4) + d(O_4, q_{goal})$

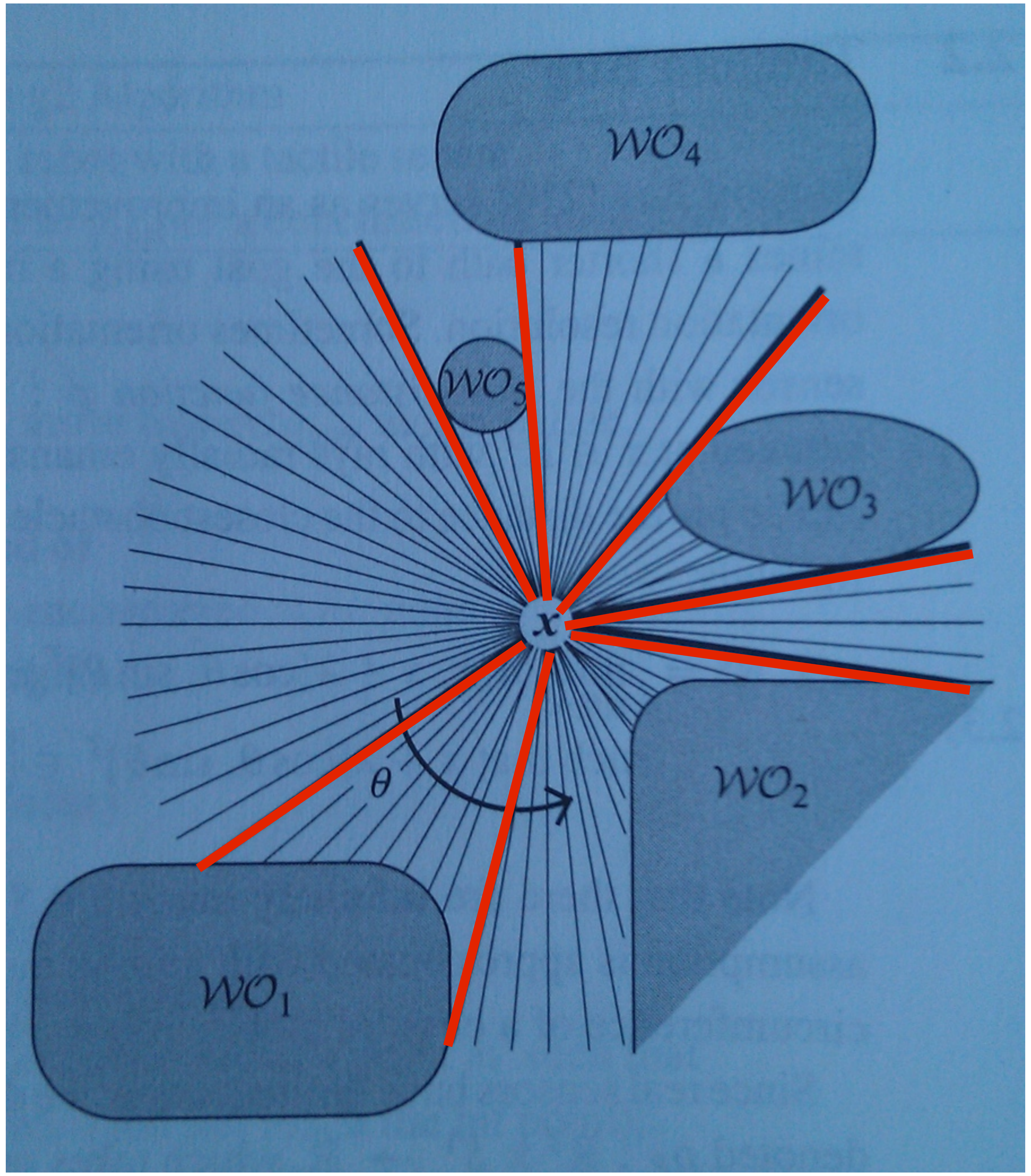


Range Segmentation

range scan ρ(x,Θ): sensed distance along ray at angle Θ within limit R

discontinuities {O_i} in scan result from obstacles

{O_i} segments scan into intervals continuity, with obstacles and free space

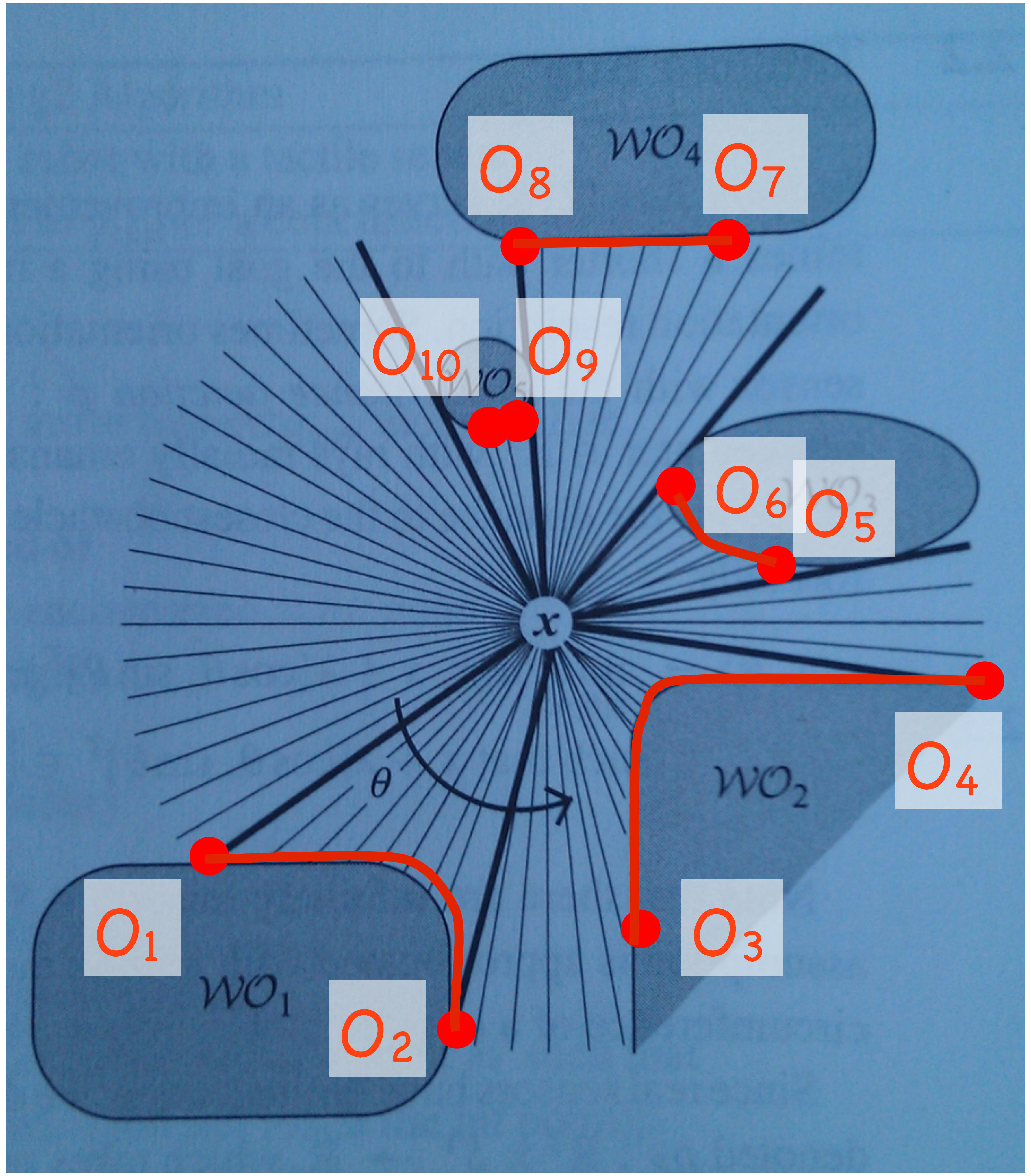


Range Segmentation

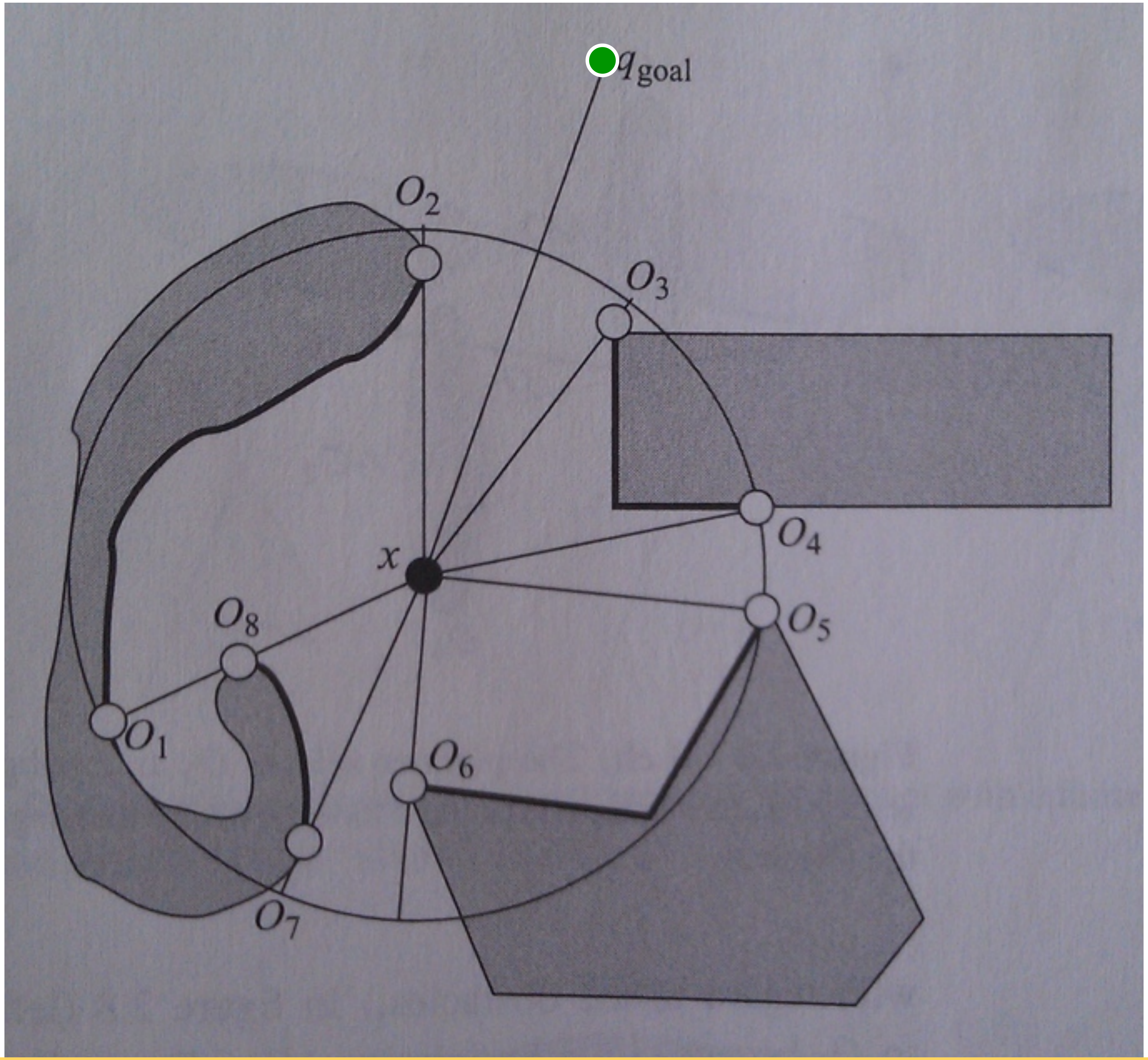
range scan ρ(x,Θ): sensed distance along ray at angle Θ within limit R

discontinuities {O_i} in scan result from obstacles

{O_i} segments scan into intervals continuity, with obstacles and free space



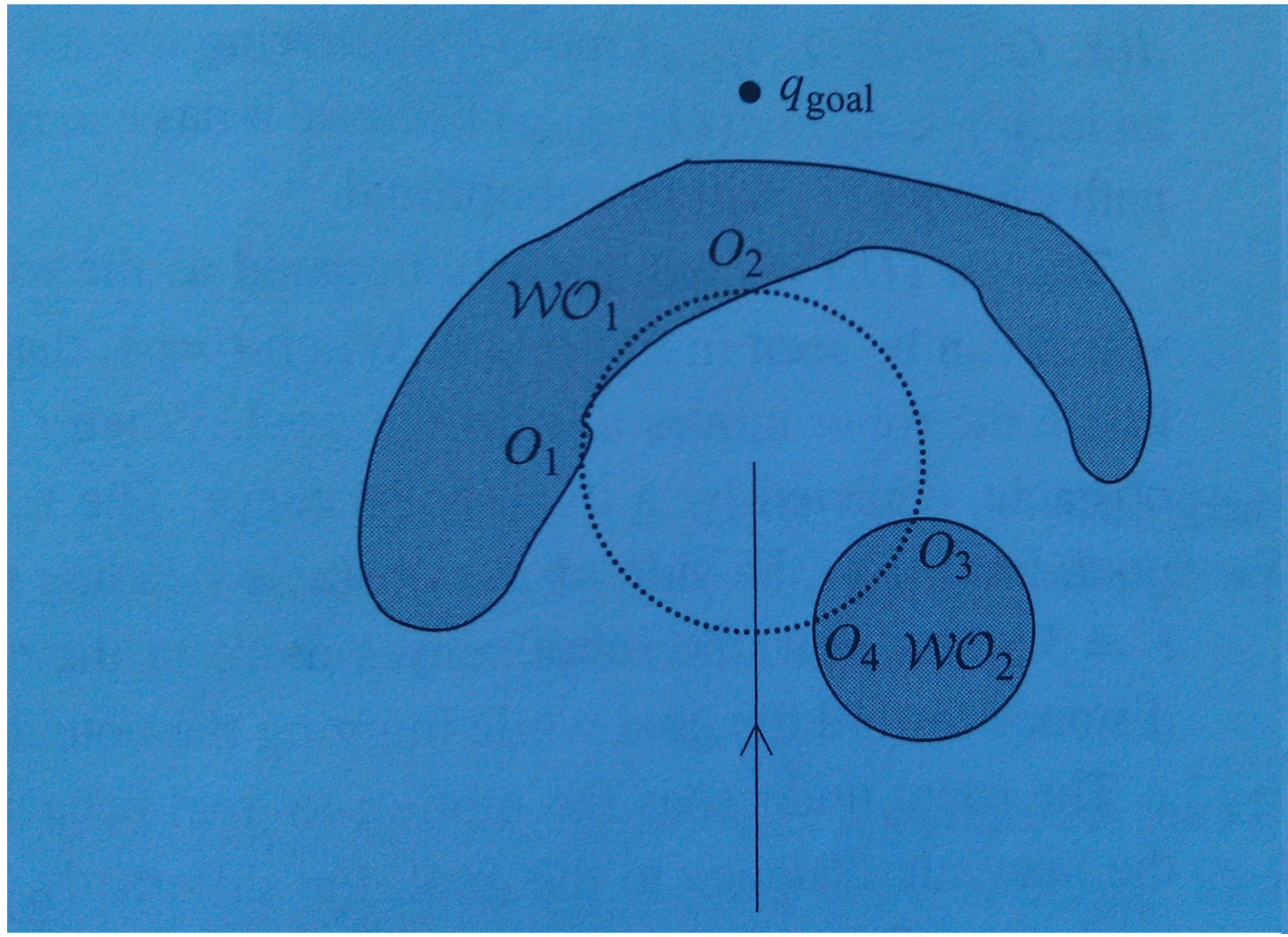
Slide borrowed from Michigan Robotics autorob.org



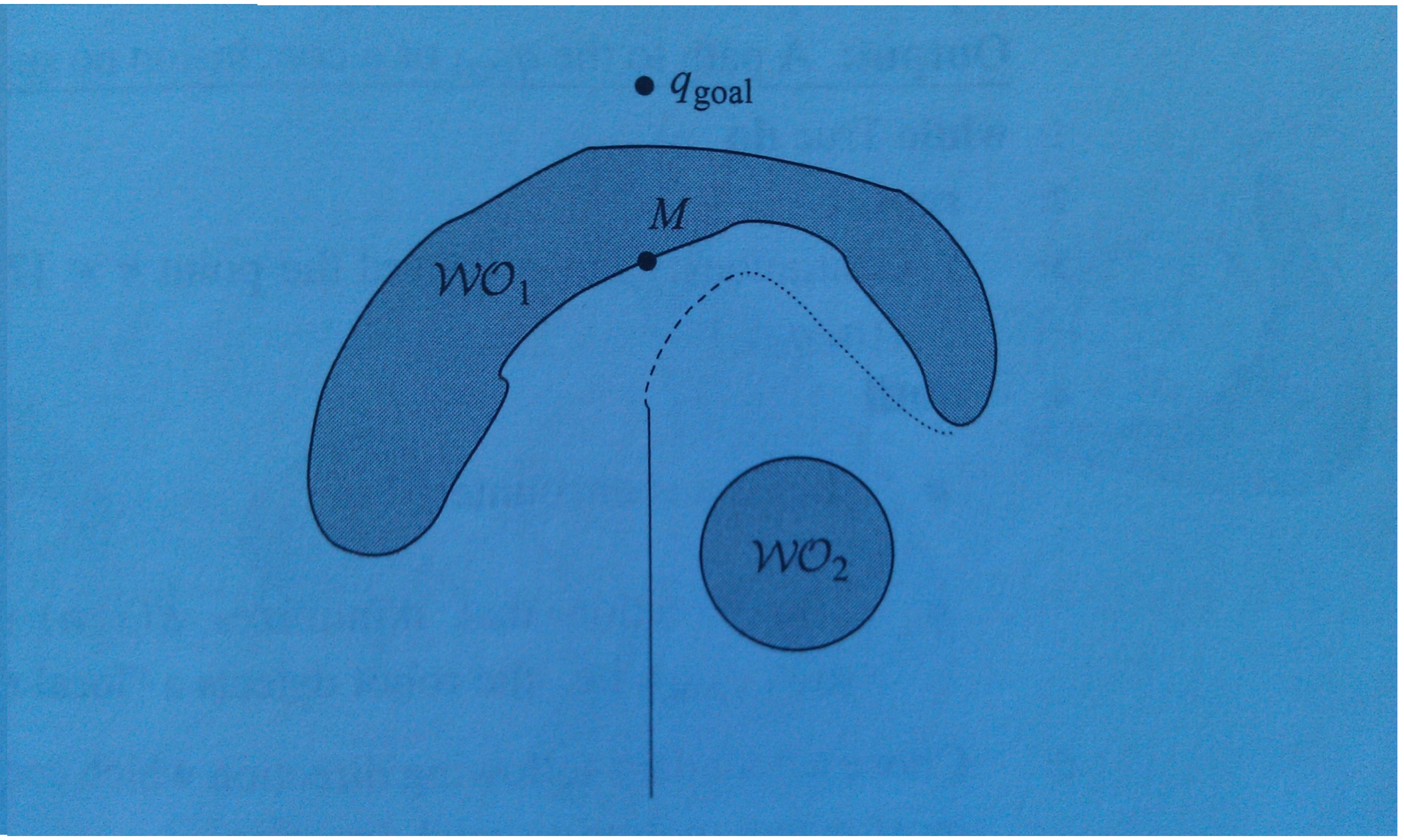
q_{goal}
O_2
O_3
O_4
x
O_5
O_8
O_1
O_6
O_7

Tangent Bug Behaviors

Similar to other bug algorithms, Tangent Bug uses two behaviors:



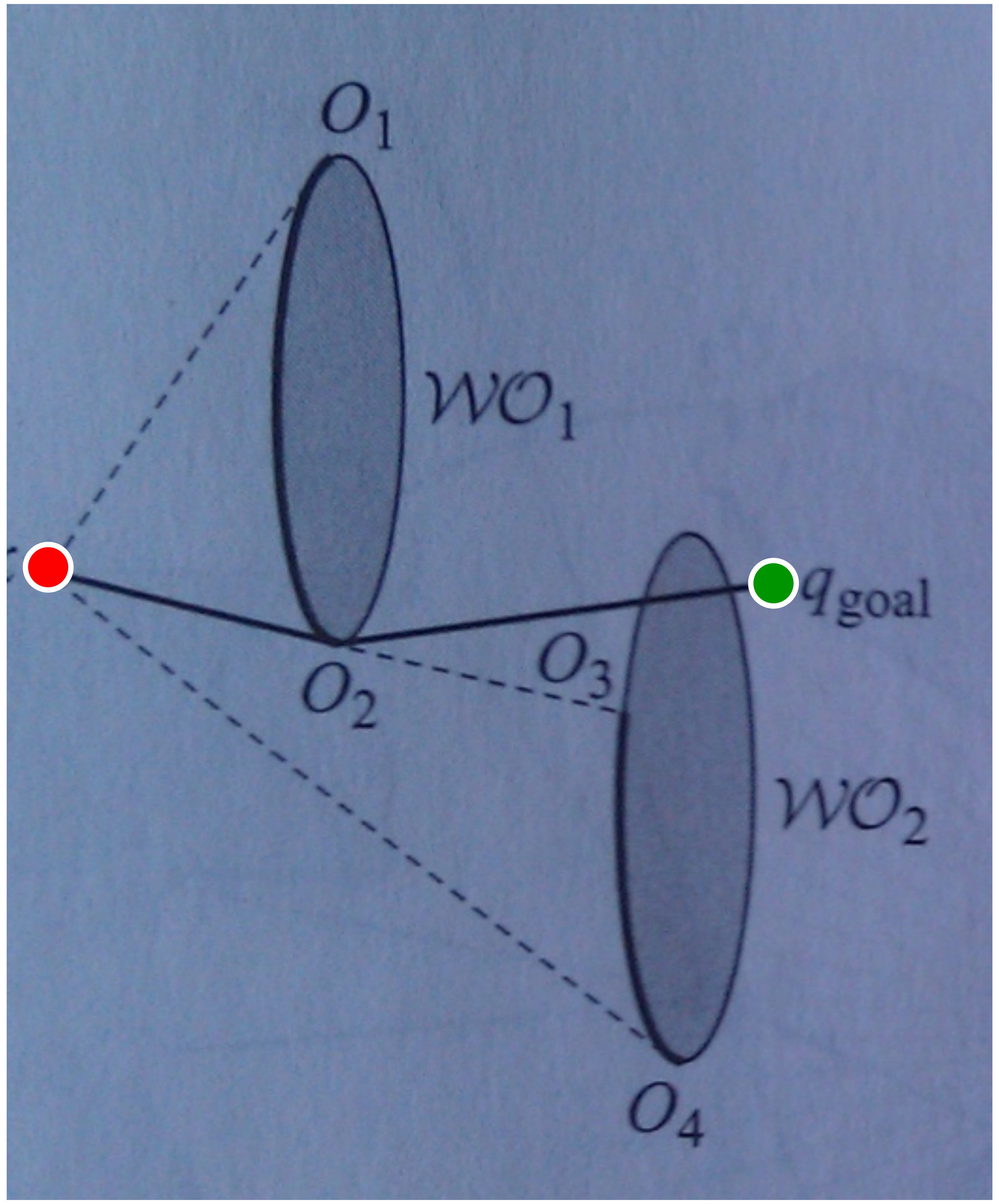
motion-to-goal



boundary-follow

Tangent Bug

$G(x) = d(x,O_i)+d(O_i,q_{goal})$



1) motion-to-goal: Move to current O_i to minimize $G(x)$, until goal (success) or $G(x)$ increases (local minima)

2) boundary-follow: move in while loop:

   a) repeat updates

      $d_{reach} = \min d(q_{goal}, \{\text{visible } O_i\})$

      $d_{follow} = \min d(q_{goal}, \text{sensed}(WO_j))$

      $O_i = \text{argmin}_i\ d(x,O_i)+d(O_i,q_{goal})$

   b) until

      goal reached, (**success**)

      robot cycles around obstacle, (**fail**)

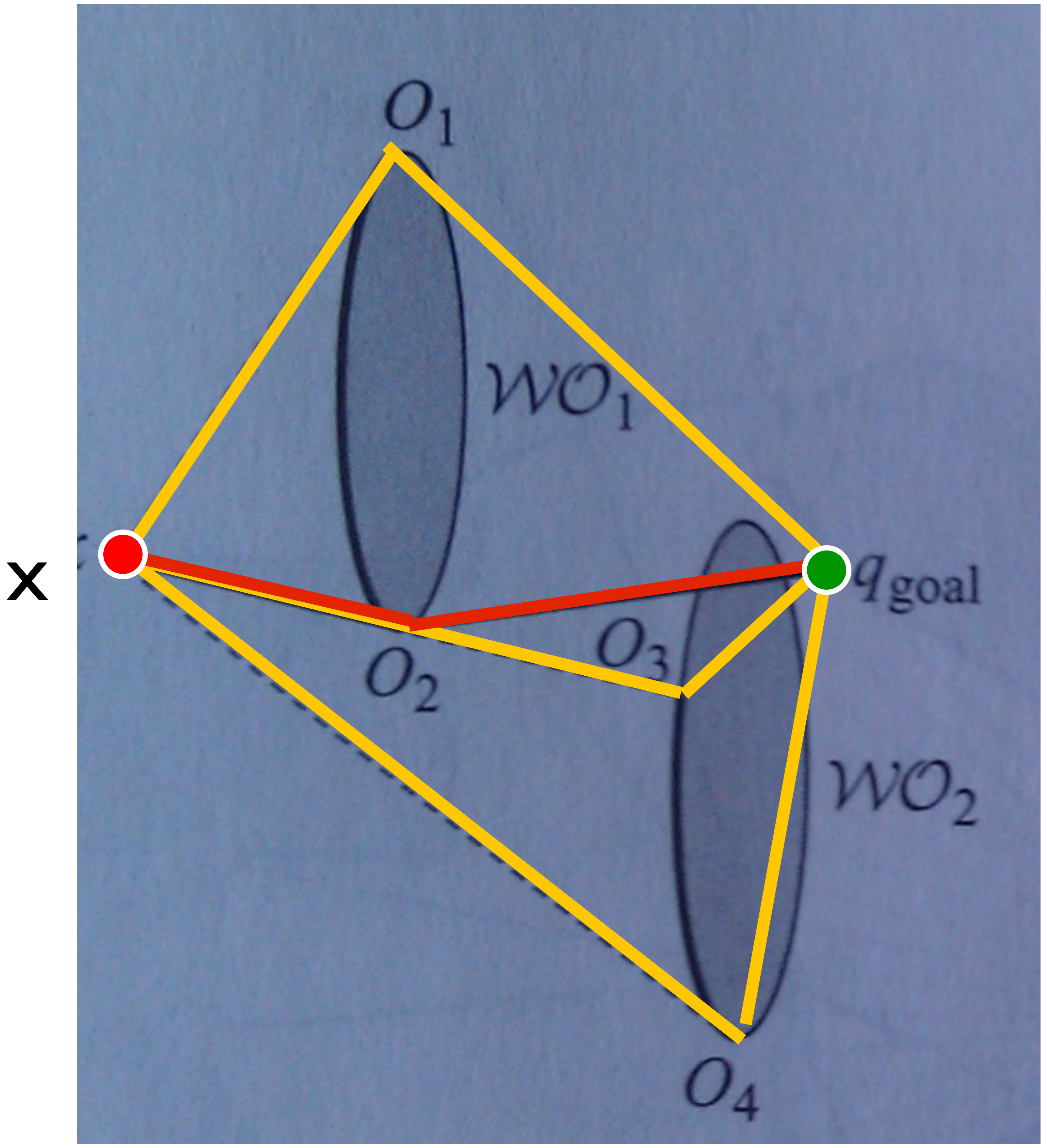
      $d_{reach} < d_{follow}$,

         (**cleared obstacle or local minima**)

3) continue from (1)

Tangent Bug

$G(x) = d(x,O_2)+d(O_2,q_{goal})$



min $G(x)$ in red, others in yellow

1) motion-to-goal: Move to current O_i to minimize $G(x)$, until goal (success) or $G(x)$ increases (local minima)

2) boundary-follow: move in while loop:

a) repeat updates

d_{reach} = min $d(q_{goal}$, {visible O_i})

d_{follow} = min $d(q_{goal}$, sensed(WO_j))

O_i = argmin$_i$ $d(x,O_i)+d(O_i,q_{goal})$

b) until

goal reached, (**success**)

robot cycles around obstacle, (**fail**)

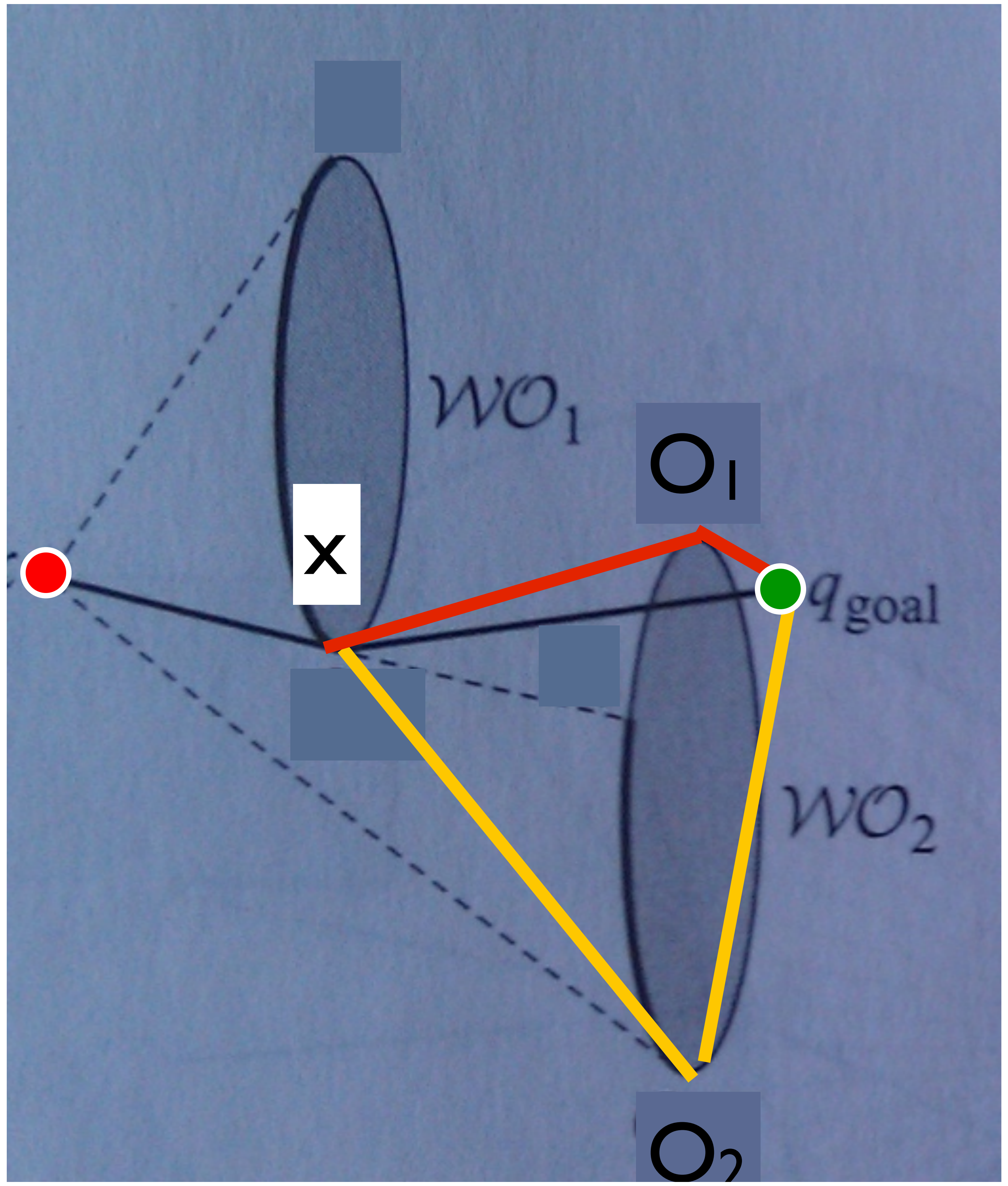
$d_{reach} < d_{follow}$,

(**cleared obstacle or local minima**)

3) continue from (1)

Tangent Bug

$G(x) = d(x,O_1)+d(O_1,q_{goal})$



min $G(x)$ in red, others in yellow

1) motion-to-goal: Move to current O_i to minimize $G(x)$, until goal (success) or $G(x)$ increases (local minima)

2) boundary-follow: move in while loop:

a) repeat updates

$d_{reach} = \min d(q_{goal}, \{\text{visible } O_i\})$

$d_{follow} = \min d(q_{goal}, \text{sensed}(WO_j))$

$O_i = \text{argmin}_i\ d(x,O_i)+d(O_i,q_{goal})$

b) until

goal reached, (**success**)

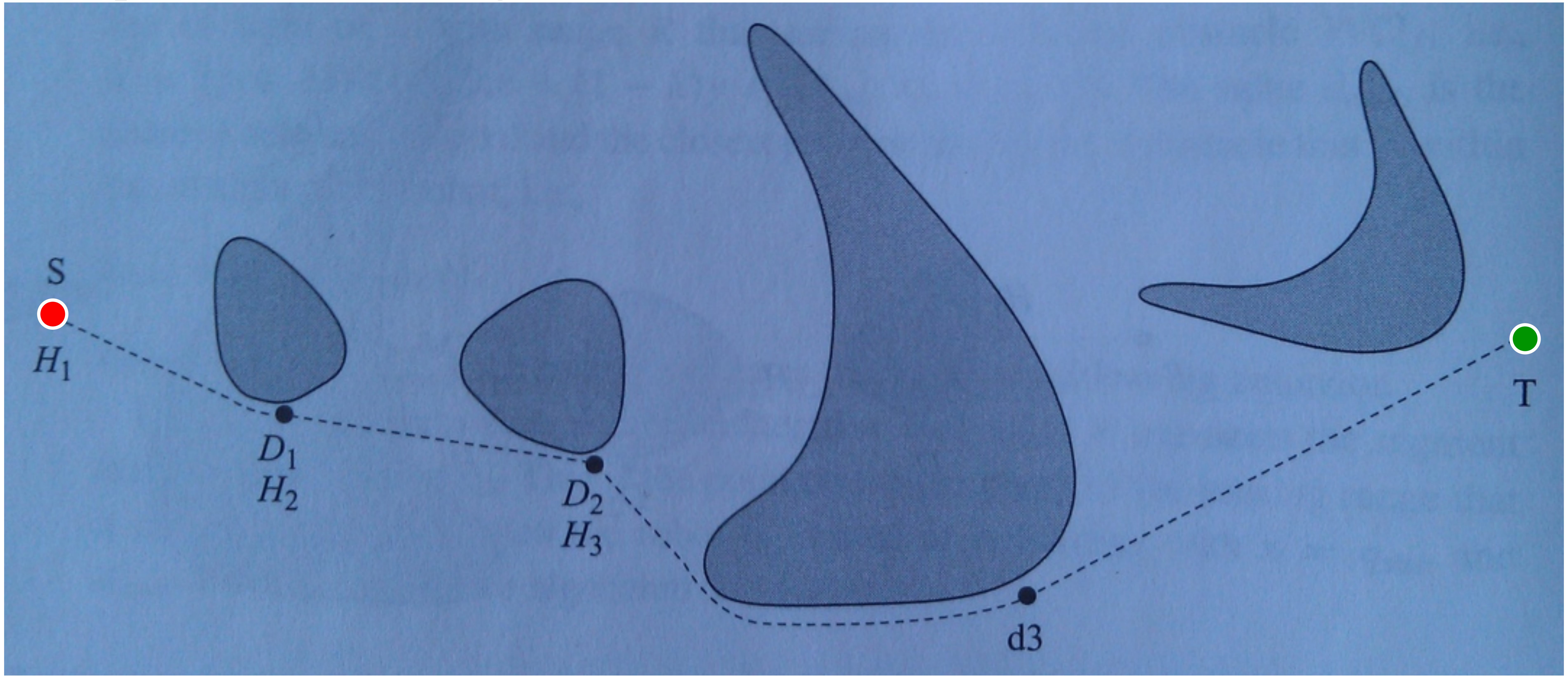
robot cycles around obstacle, (**fail**)

$d_{reach} < d_{follow}$,

(**cleared obstacle or local minima**)

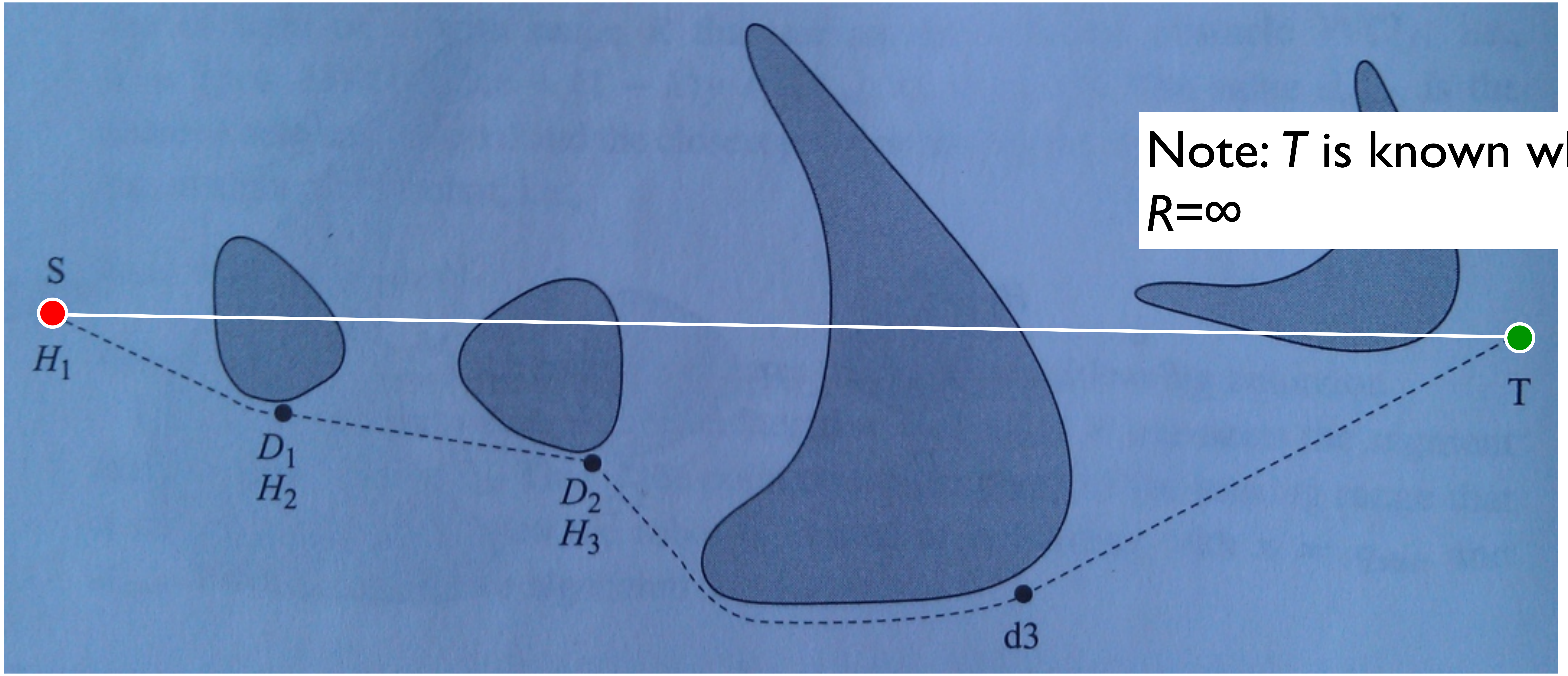
3) continue from (1)

Example: range $R=\infty$



H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

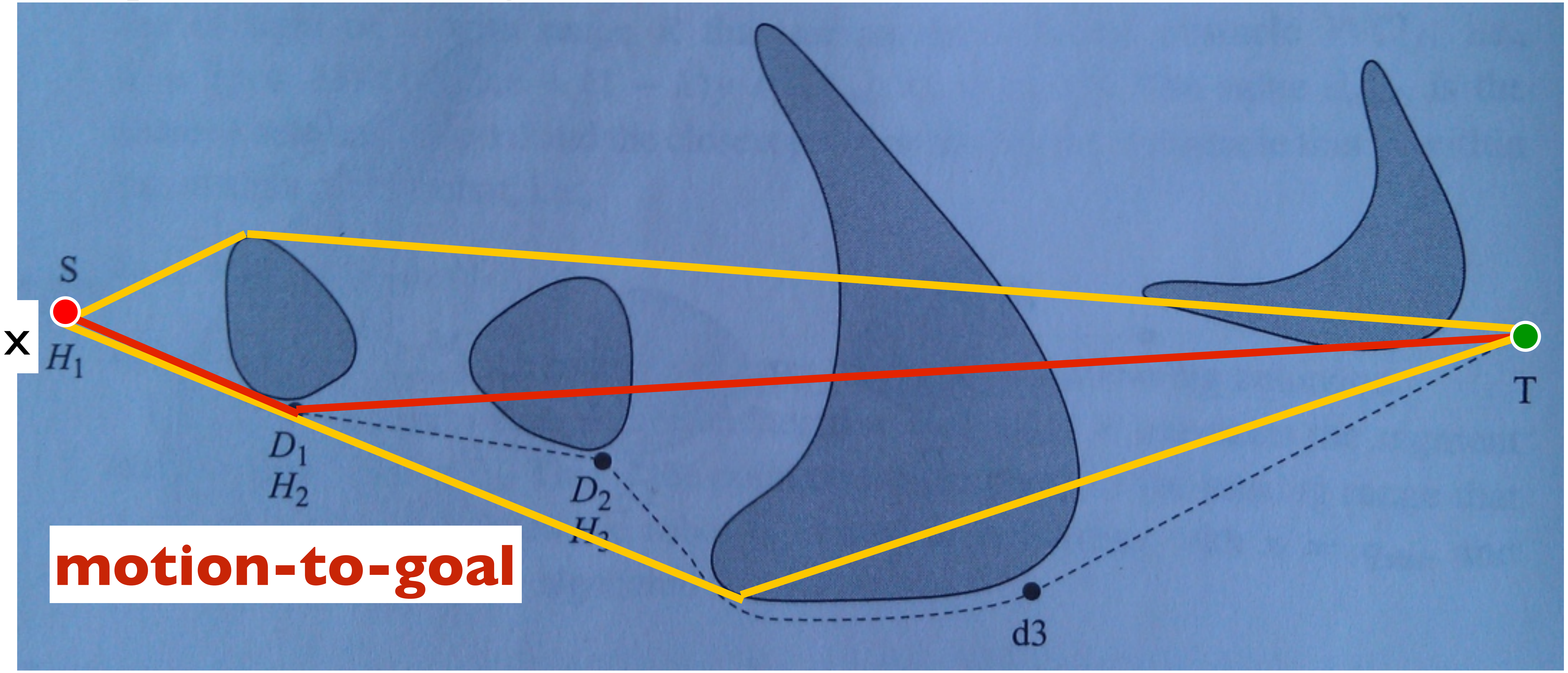
Example: range $R=\infty$



Note: T is known when $R=\infty$

H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

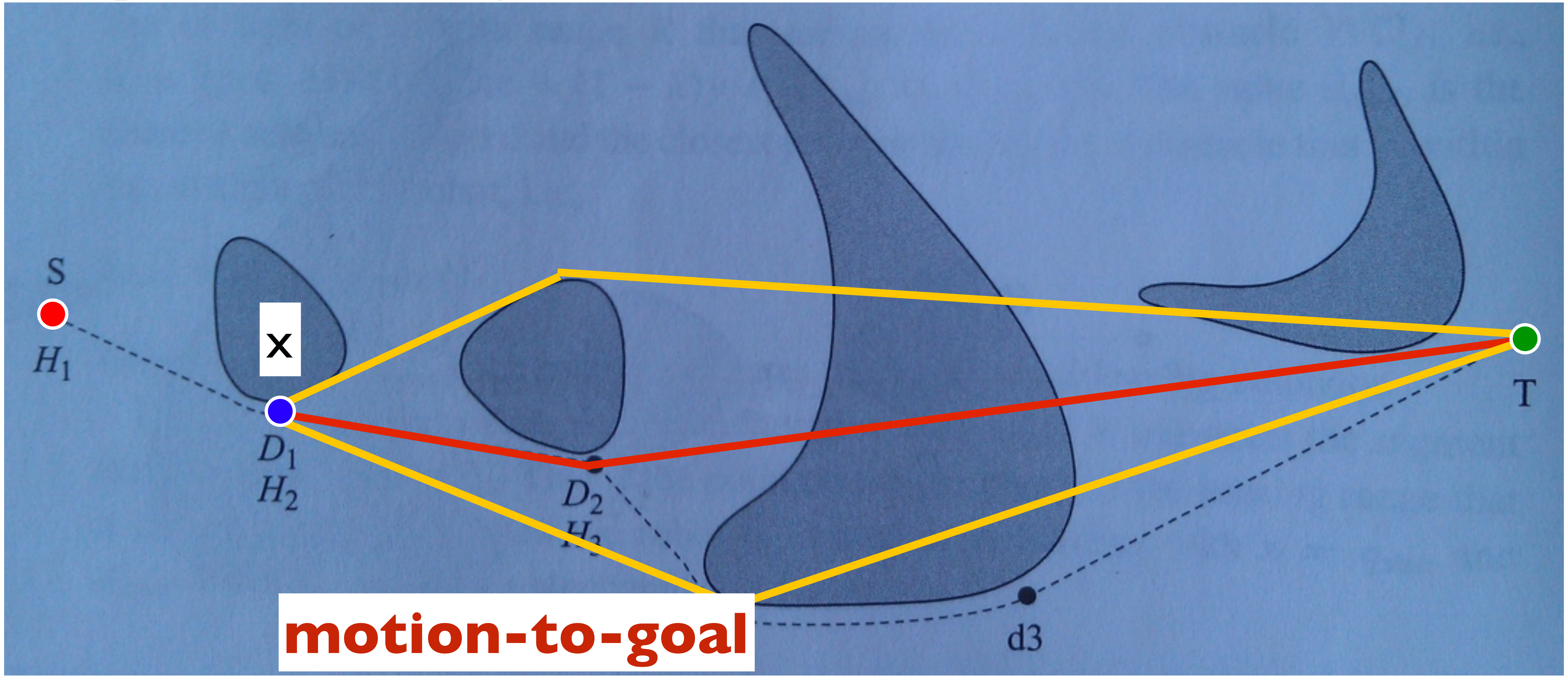
Example: range $R=\infty$



min $G(x)$ in red, others in yellow

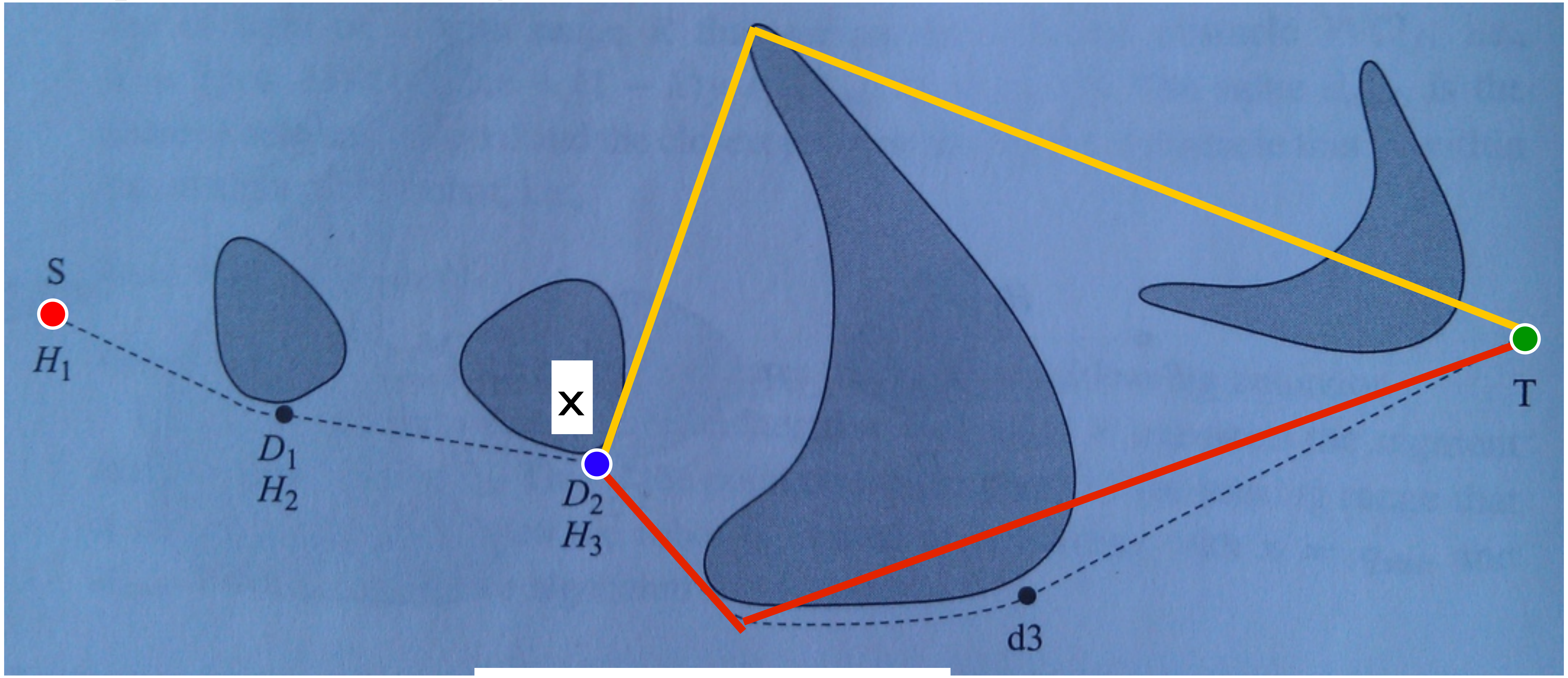
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Example: range $R=\infty$



H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

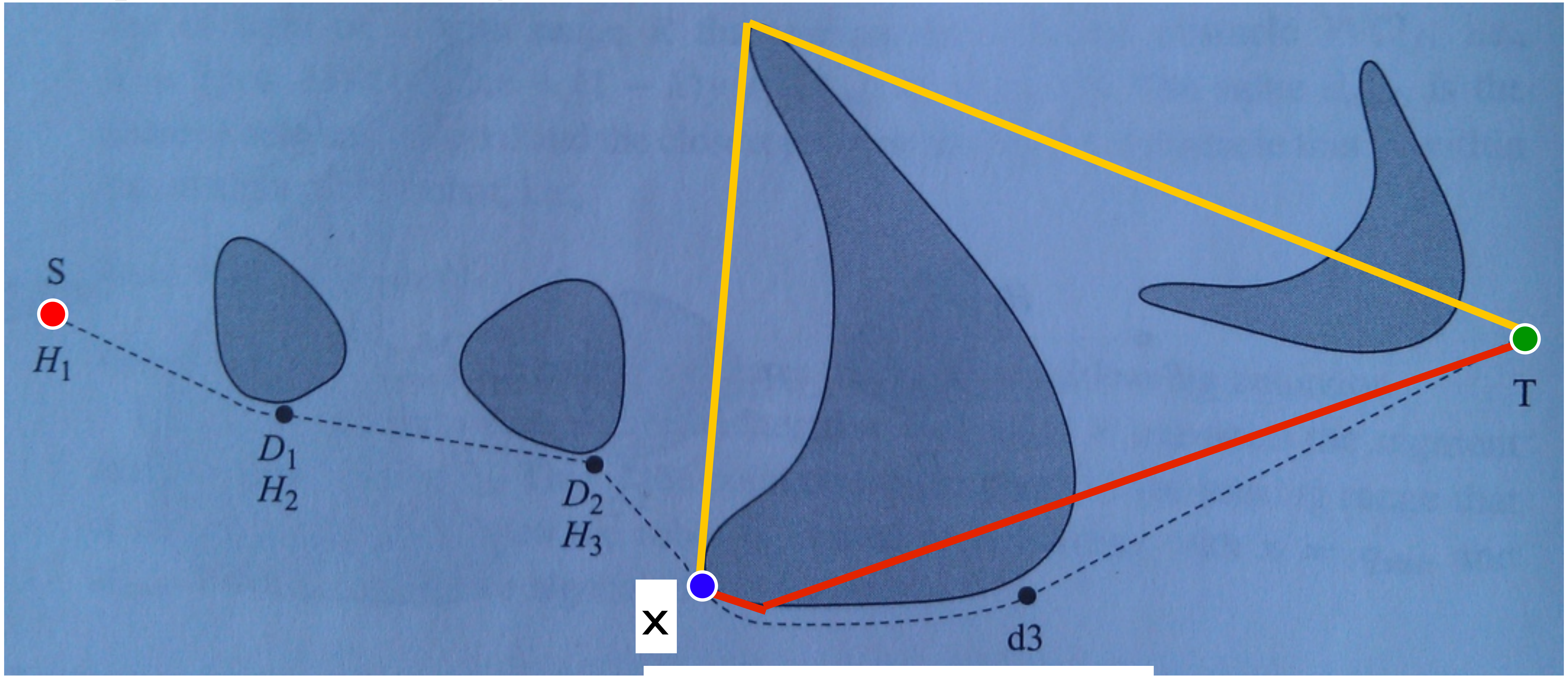
Example: range $R=\infty$



motion-to-goal

H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Example: range $R=\infty$

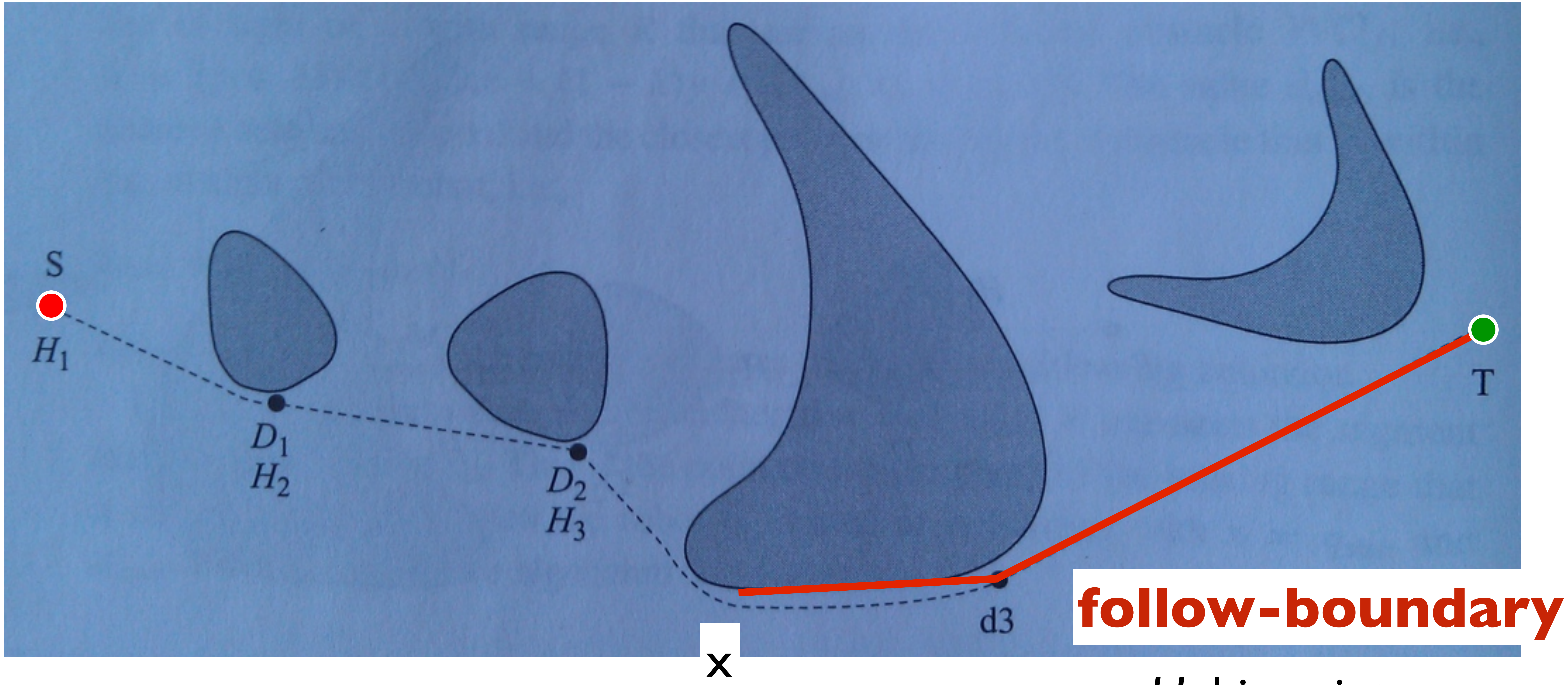


follow-boundary

H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

start following:
min d(q_{goal}, {visible O_i}) < min d(q_{goal}, sensed(WO_j))

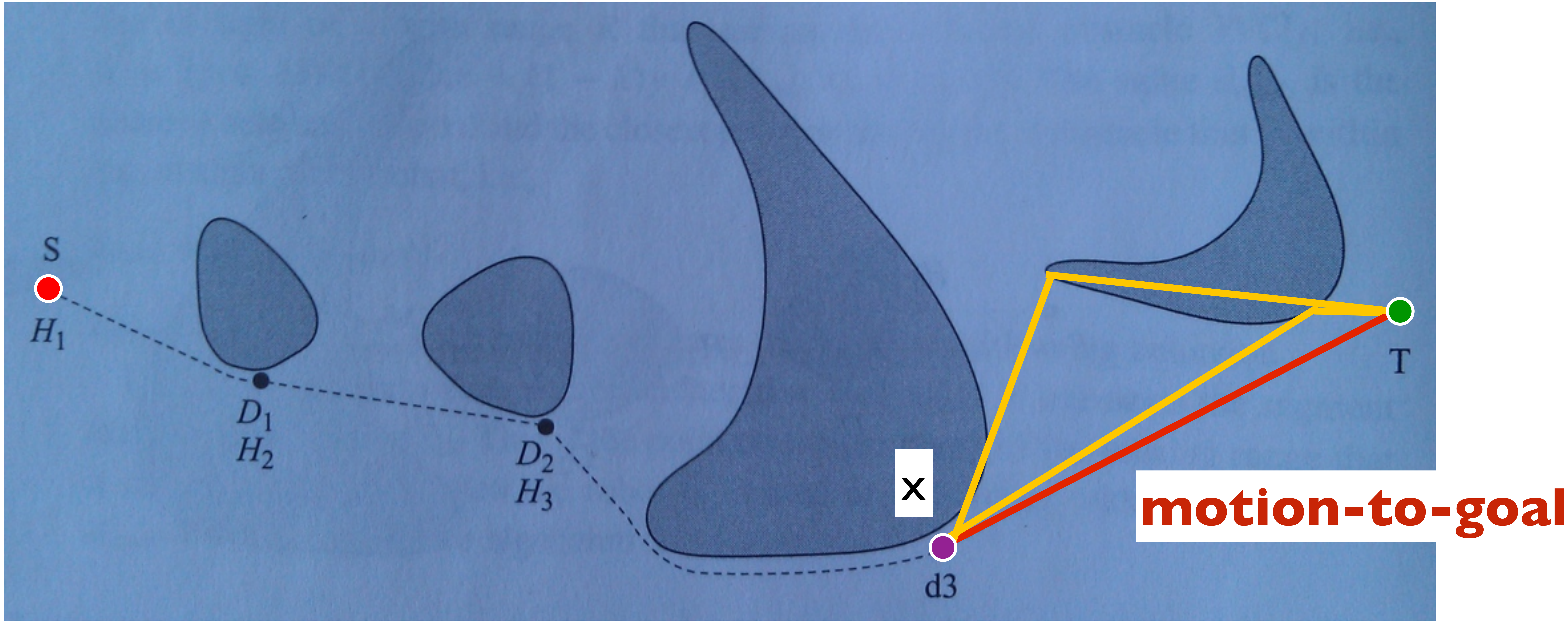
Example: range $R=\infty$



H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

end following:
$\min d(q_{goal}, \{\text{visible } O_i\}) < \min d(q_{goal}, \text{sensed}(WO_j))$

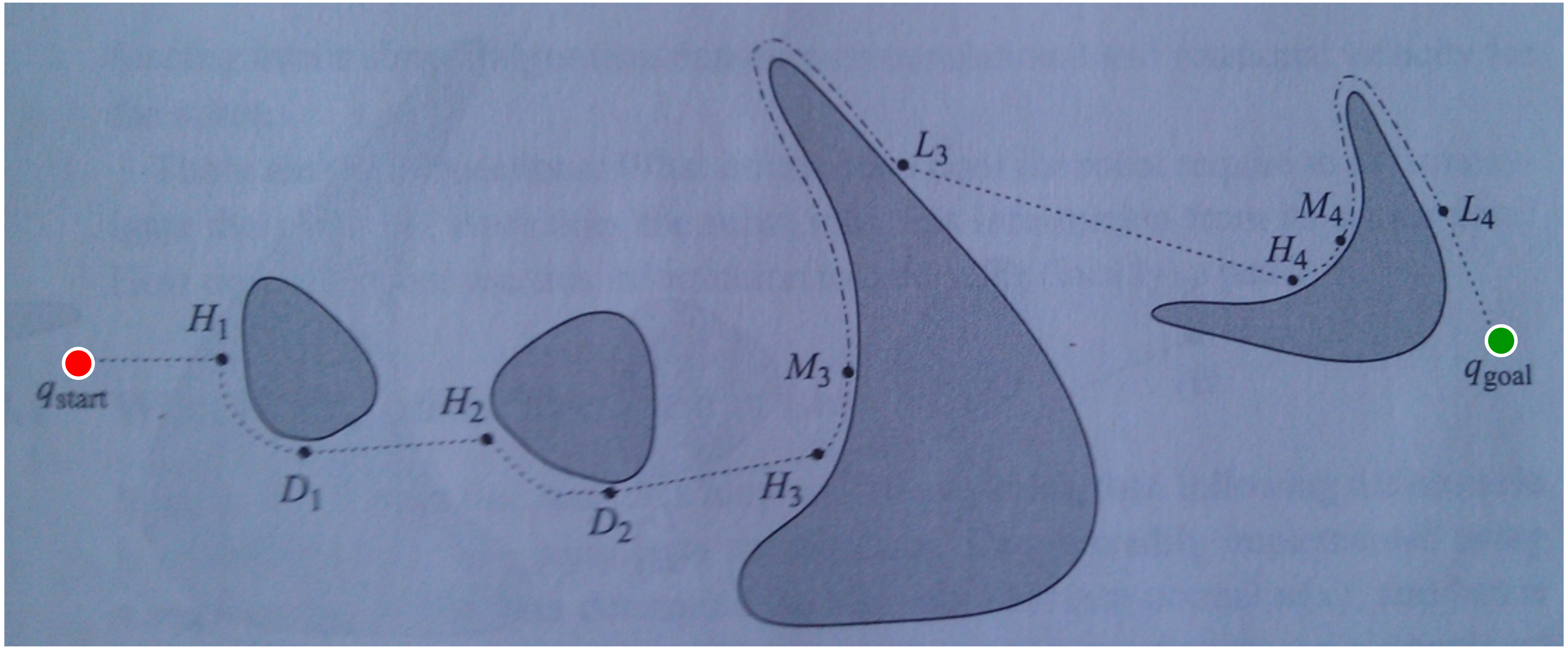
Example: range $R=\infty$



H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Example: range R=0

H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

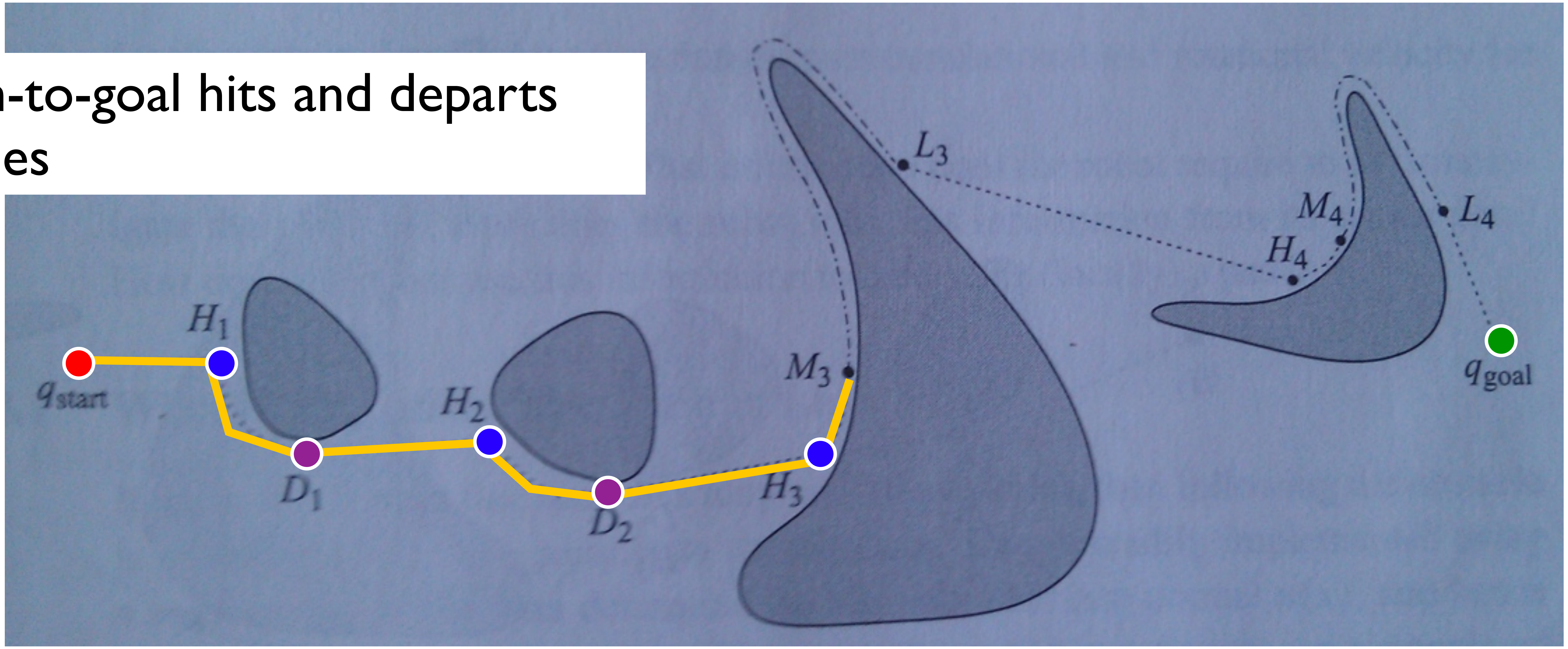


Slide borrowed from Michigan Robotics autorob.org

Example: range R=0

H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

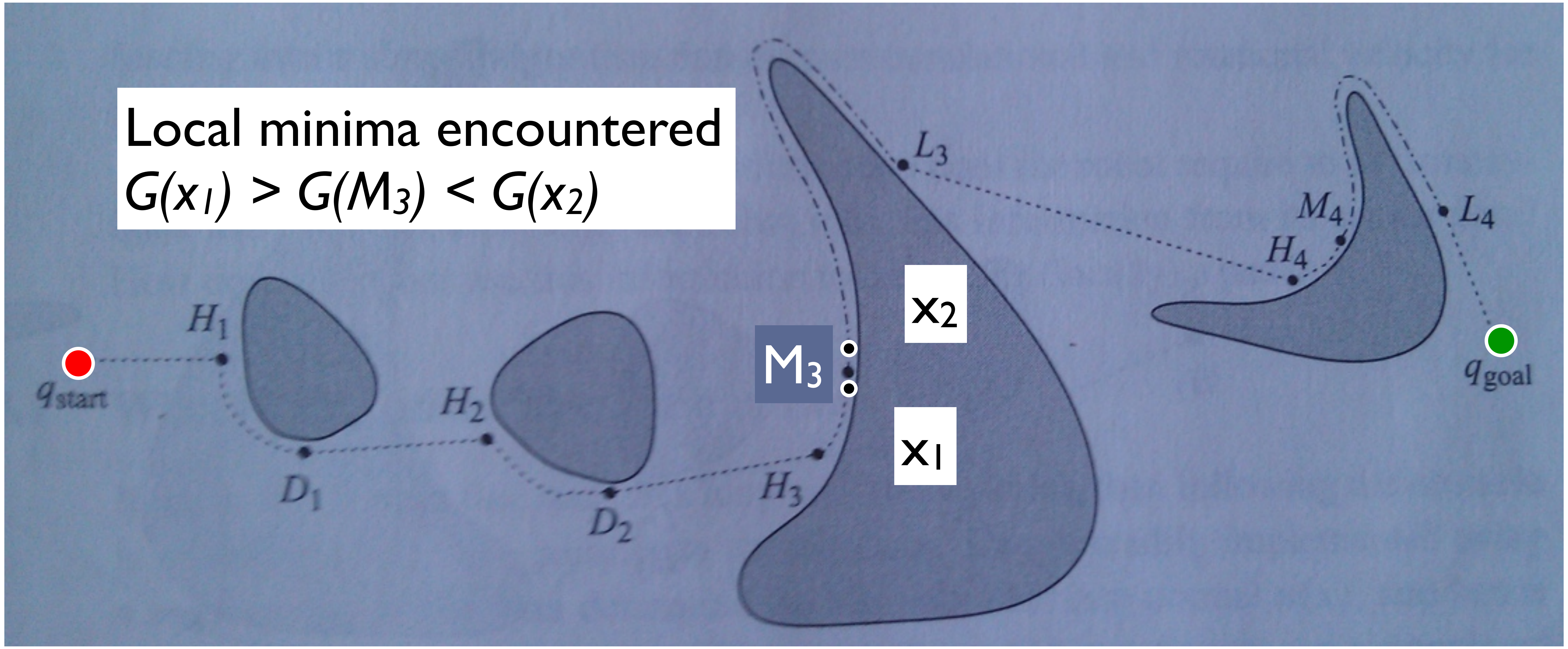
Motion-to-goal hits and departs obstacles



Slide borrowed from Michigan Robotics autorob.org

Example: range R=0

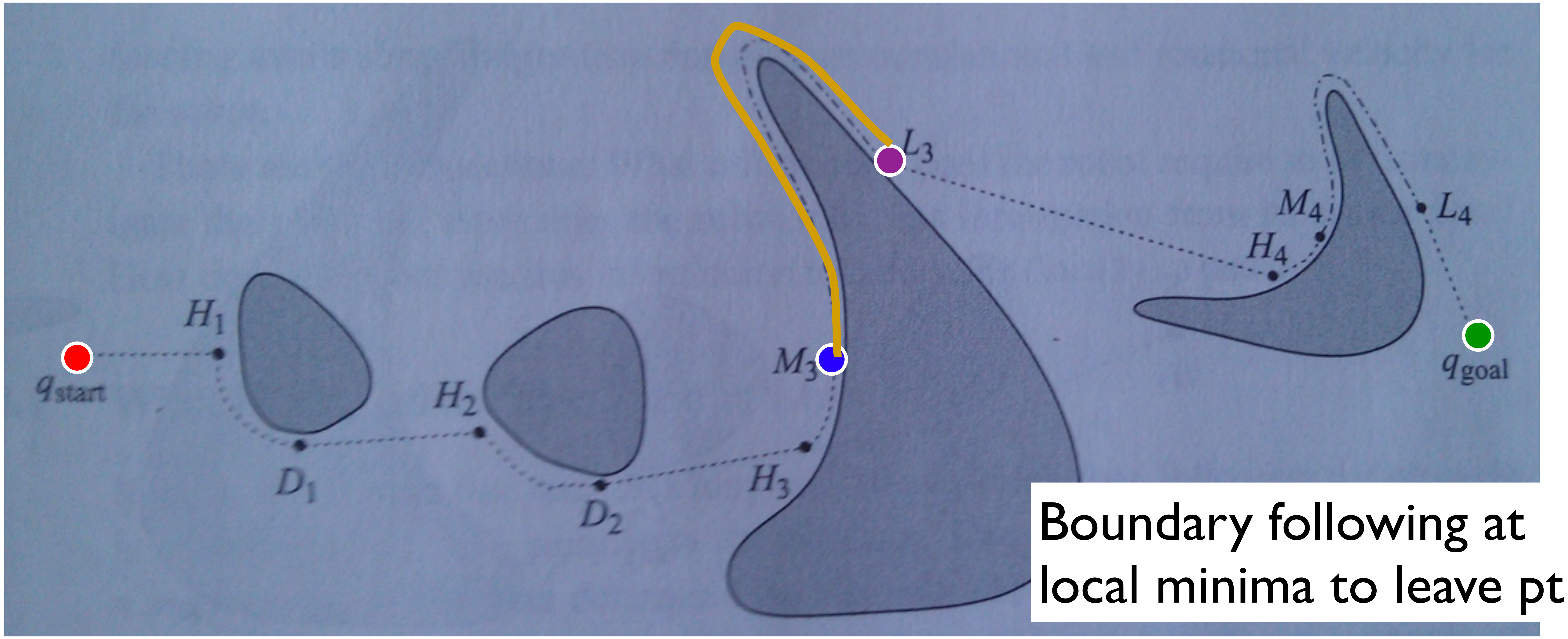
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



Local minima at increase of $G(x) = d(x,O_i)+d(O_i,q_{goal})$

Example: range R=0

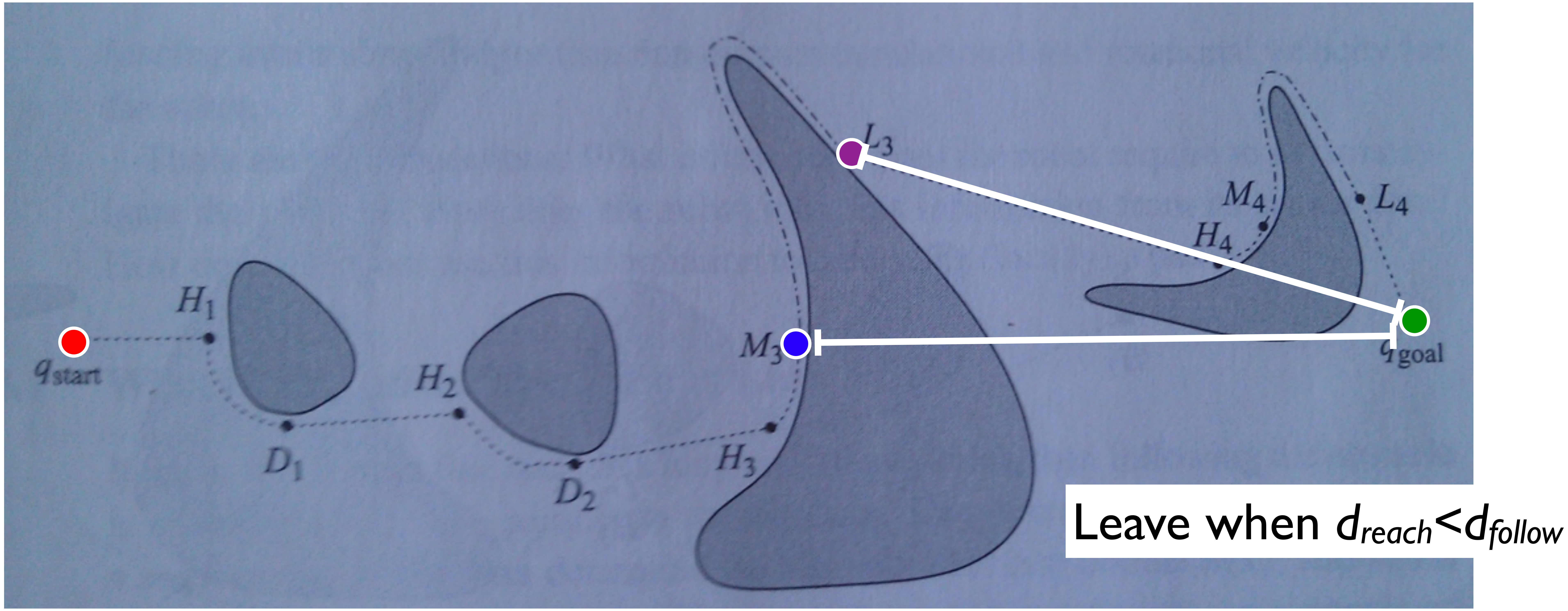
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



Local minima at increase of $G(x) = d(x,O_i)+d(O_i,q_{goal})$

Example: range R=0

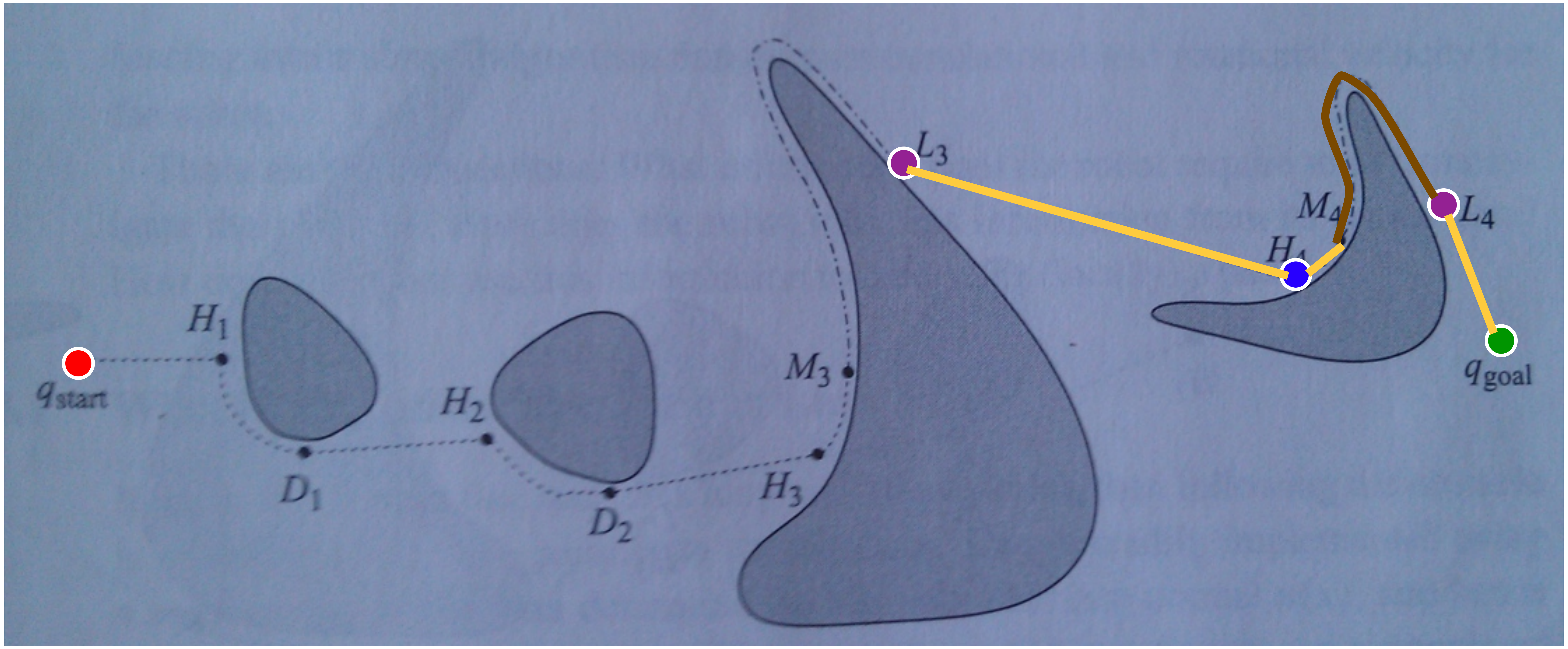
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



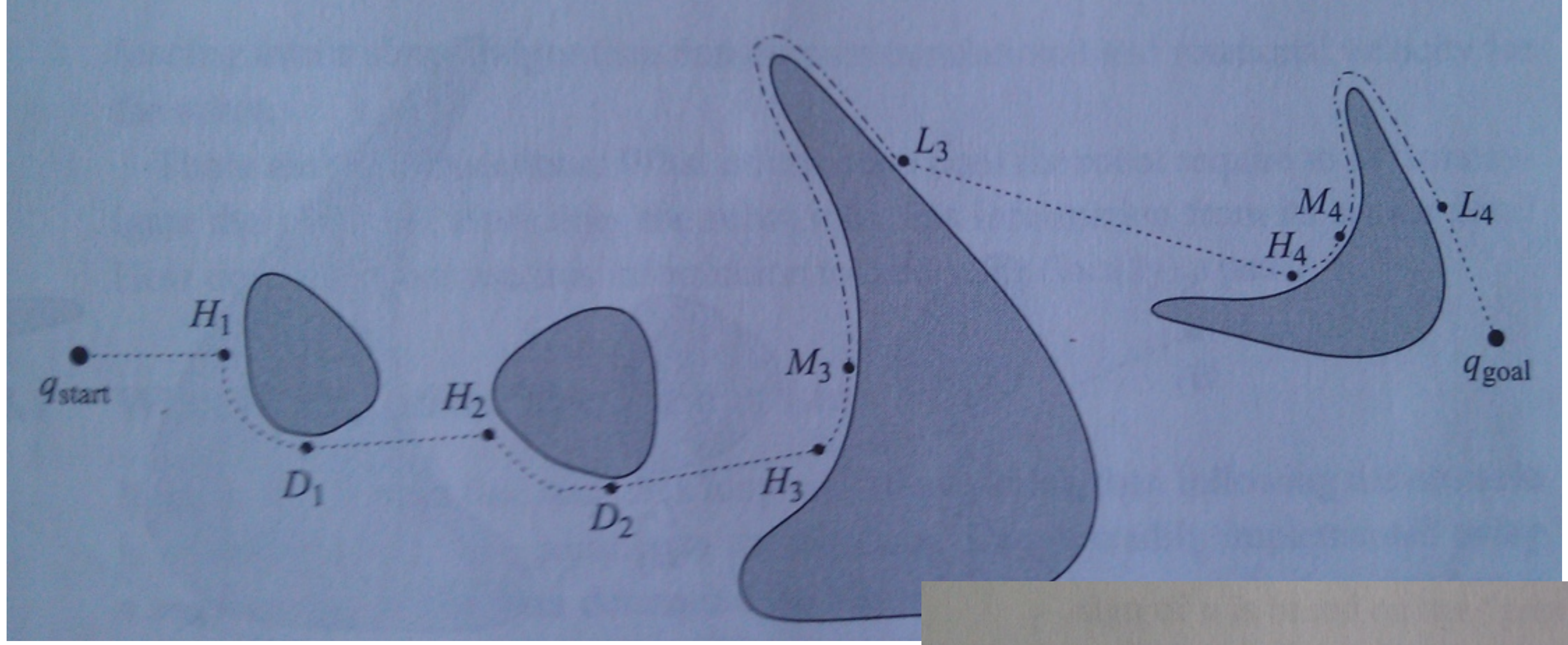
Local minima at increase of $G(x) = d(x,O_i)+d(O_i,q_{goal})$

Example: range R=0

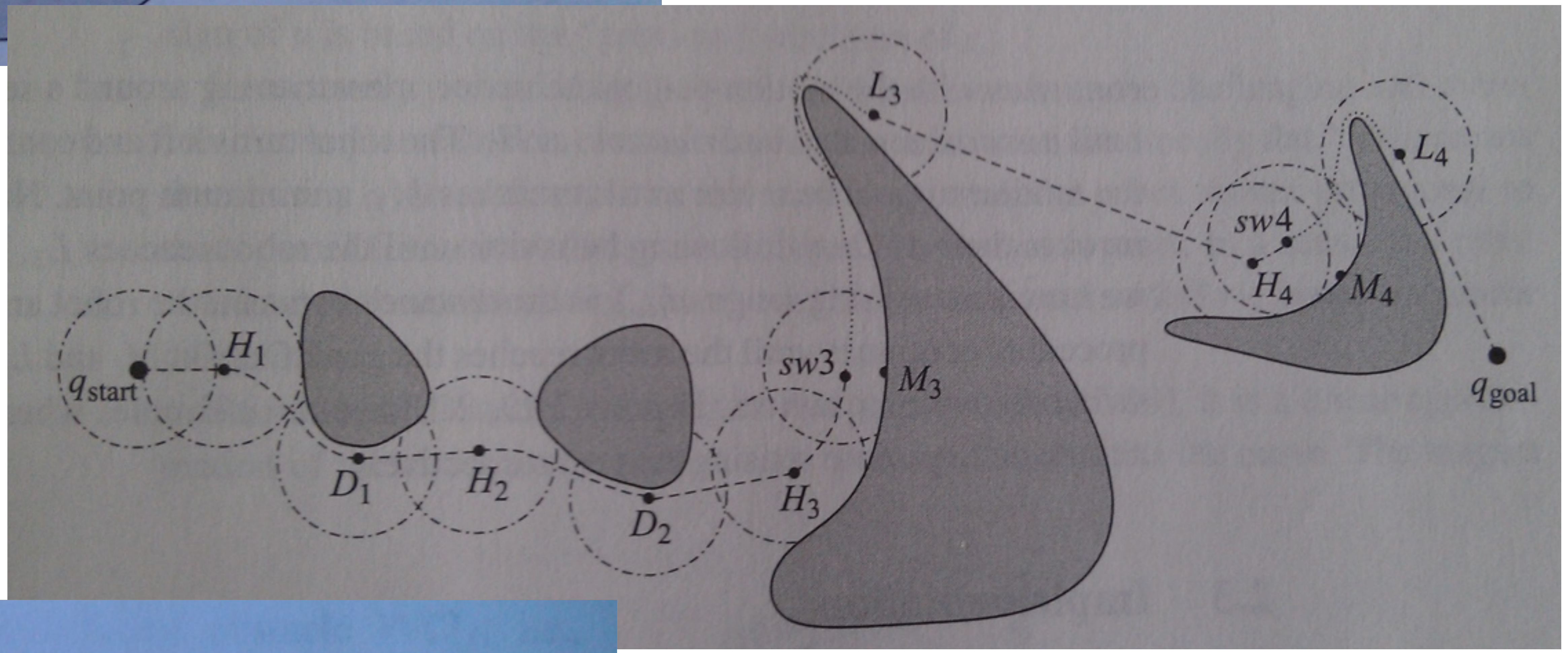
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



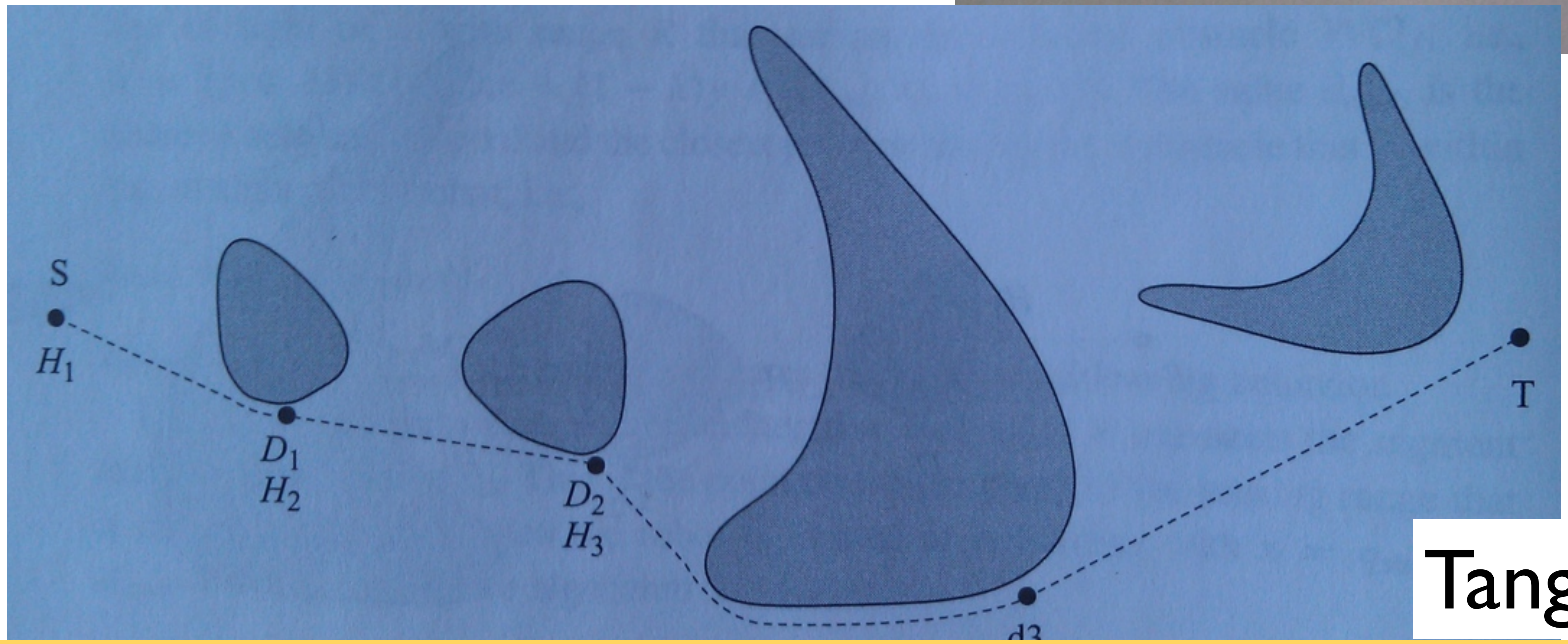
Slide borrowed from Michigan Robotics autorob.org



Tangent bug R=0



Tangent bug with limited radius



Tangent bug R=infinity

Slide borrowed from Michigan Robotics autorob.org

What does BugX assume that Random Walk does not?

[redacted]

What does BugX assume that Random Walk does not?

<u>Localization</u>: knowing the robot's location, at least wrt. distance to goal

What does BugX assume that Random Walk does not?

<u>Localization</u>: knowing the robot's location, at least wrt. distance to goal

What do graph search algorithms assume that BugX does not?

[redacted]

What does BugX assume that Random Walk does not?

<u>Localization</u>: knowing the robot's location, at least wrt. distance to goal

What do graph search algorithms assume that BugX does not?

A graph of valid locations that can be traversed

What does BugX assume that Random Walk does not?

Localization: knowing the robot's location, at least wrt. distance to goal

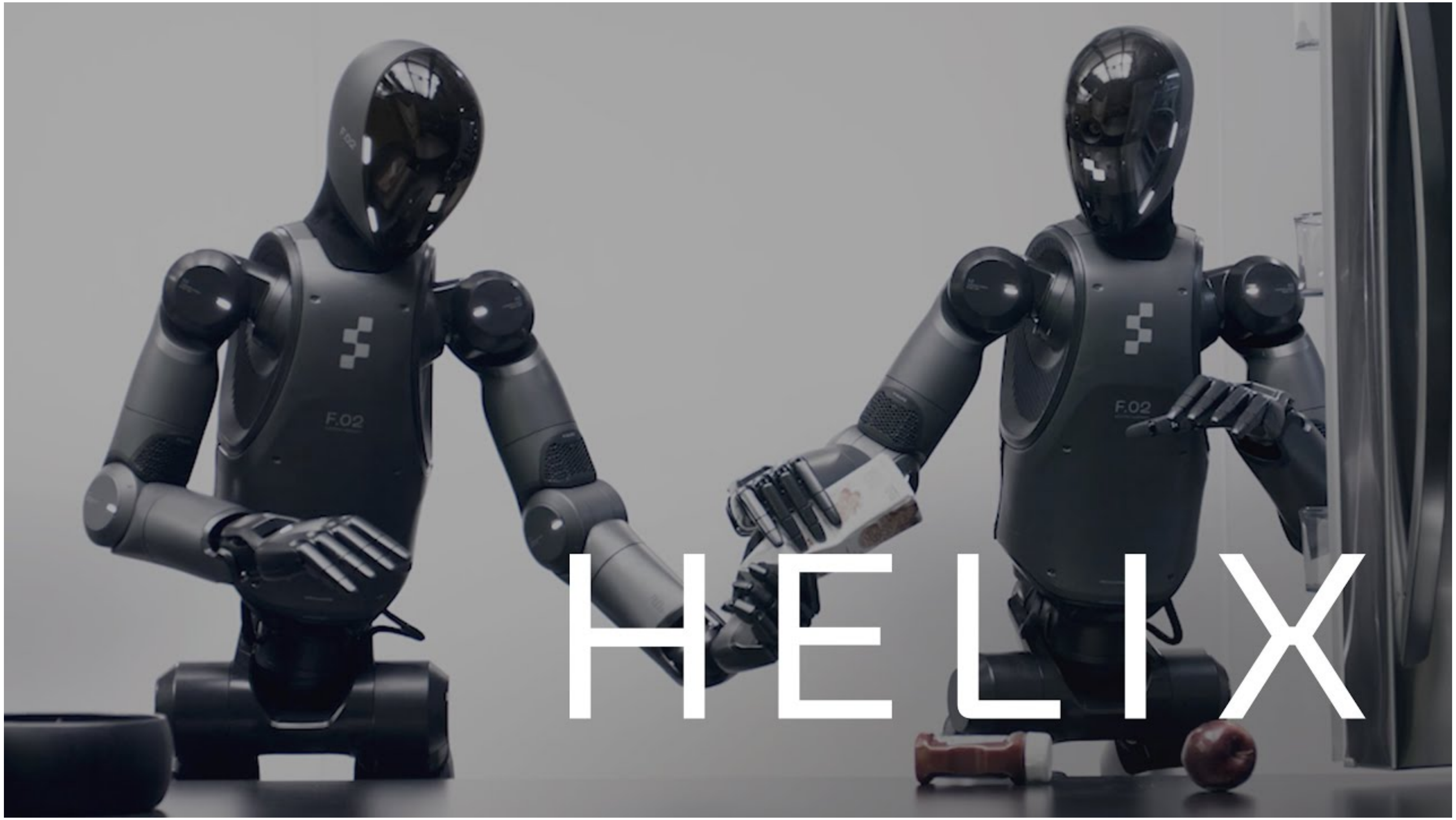
What do graph search algorithms assume that BugX does not?

A graph of valid locations that can be traversed

Suppose we have or can build such a graph...

Next Lecture

Planning - III - Configuration Space



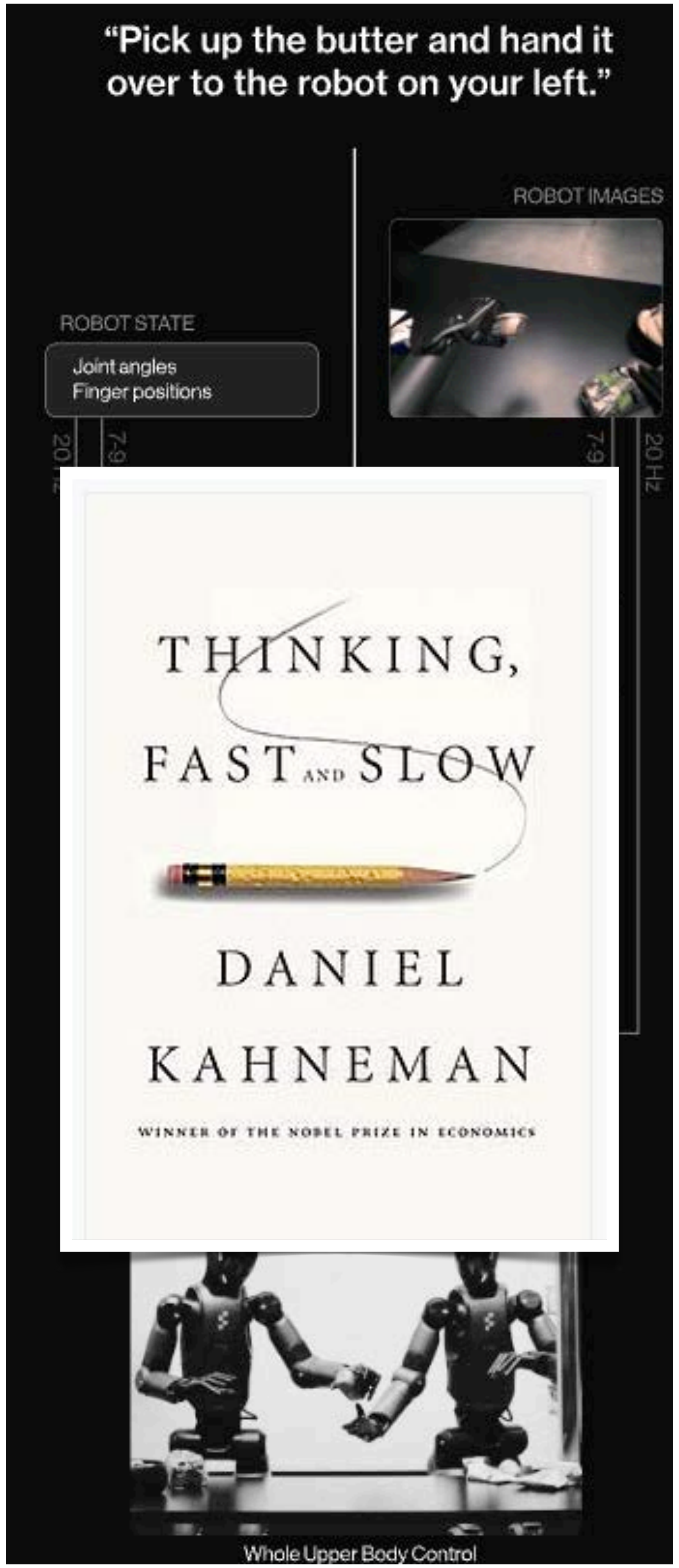


Figure - https://www.youtube.com/watch?v=Z3yQHYNXPws