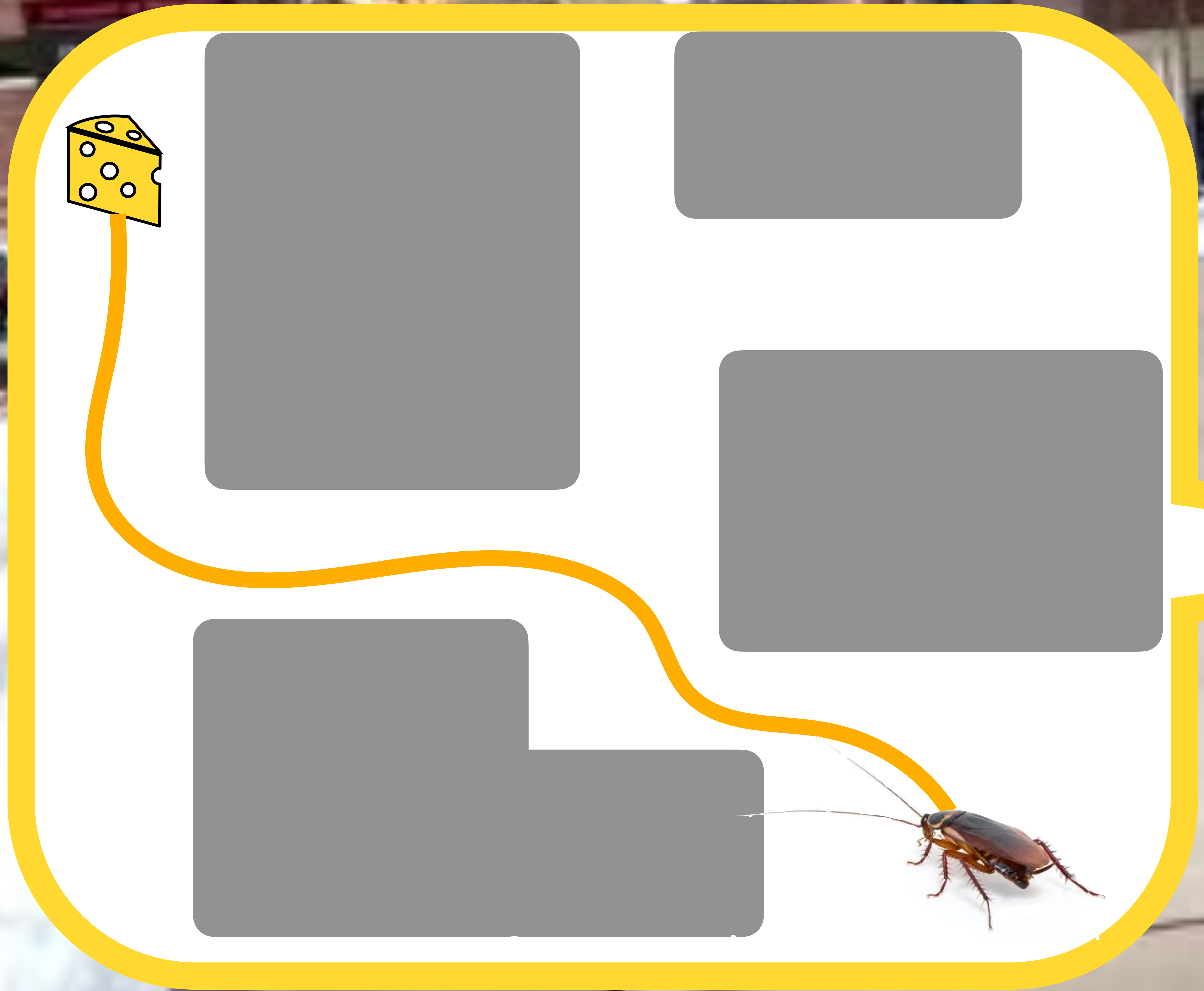
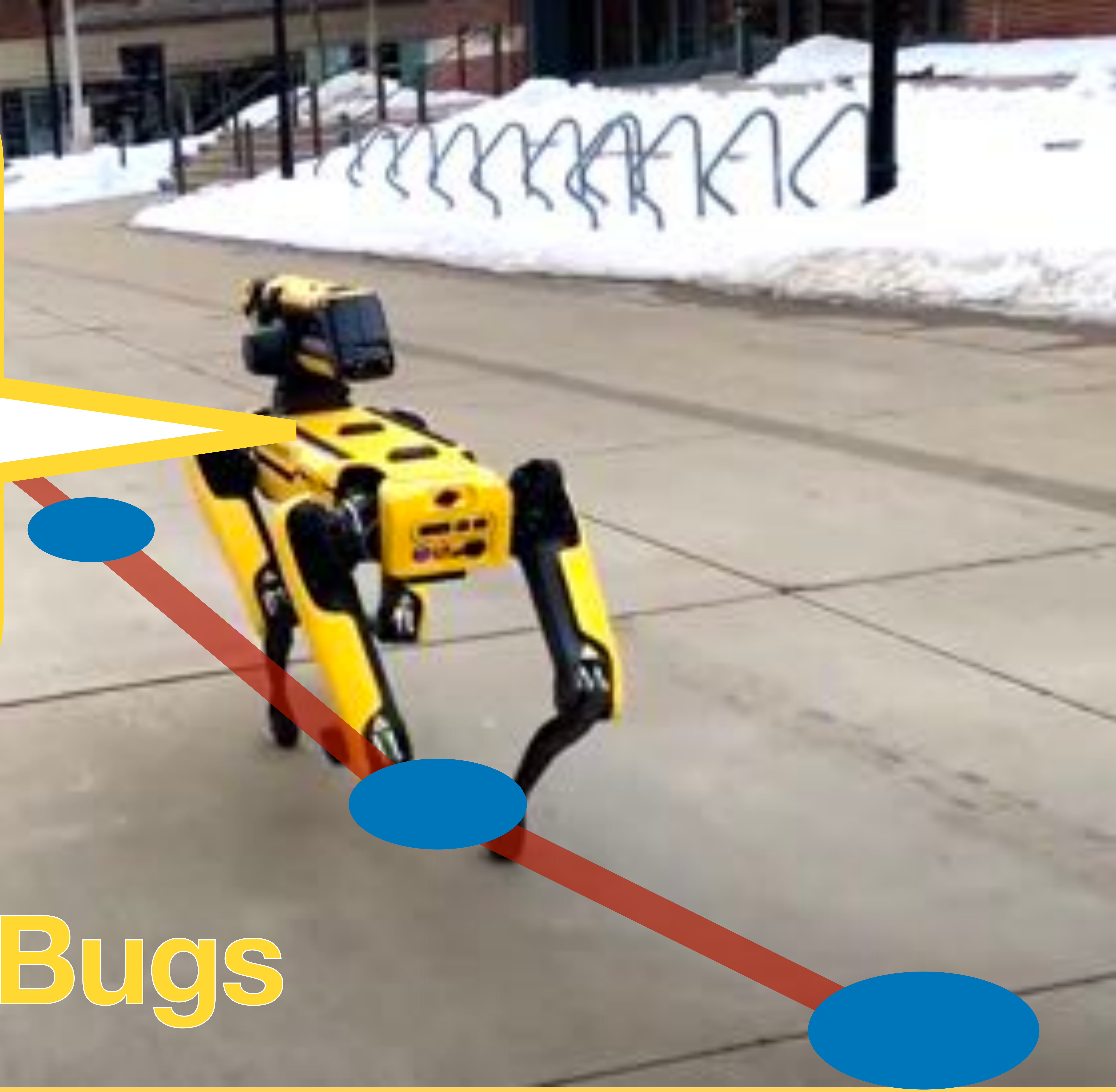




Lecture 10

Planning - II - Bugs

Course Logistics

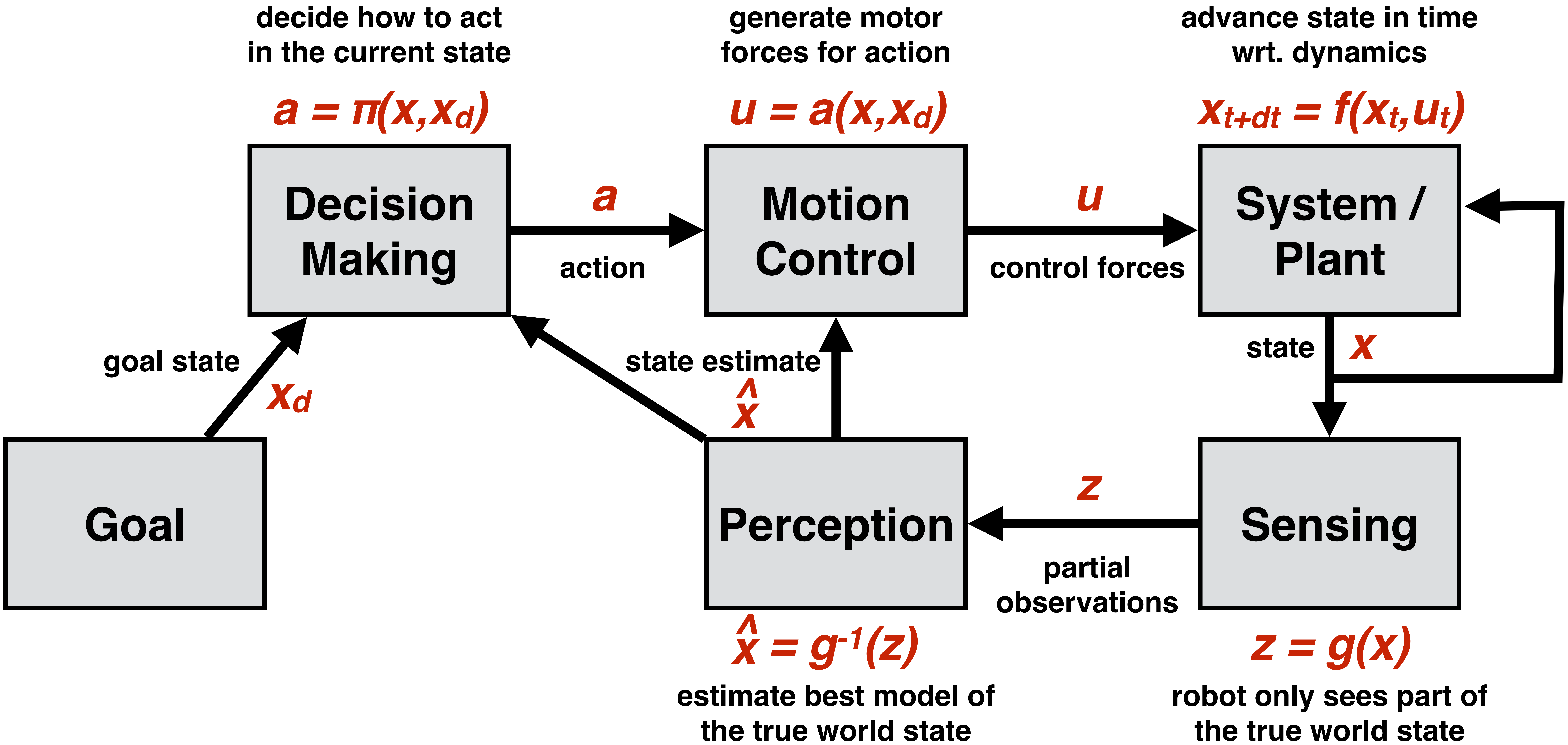
- **Quiz 8 was posted today and was due before the lecture.**
- Project 2 is posted on 10/02 and will be due 10/11.

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

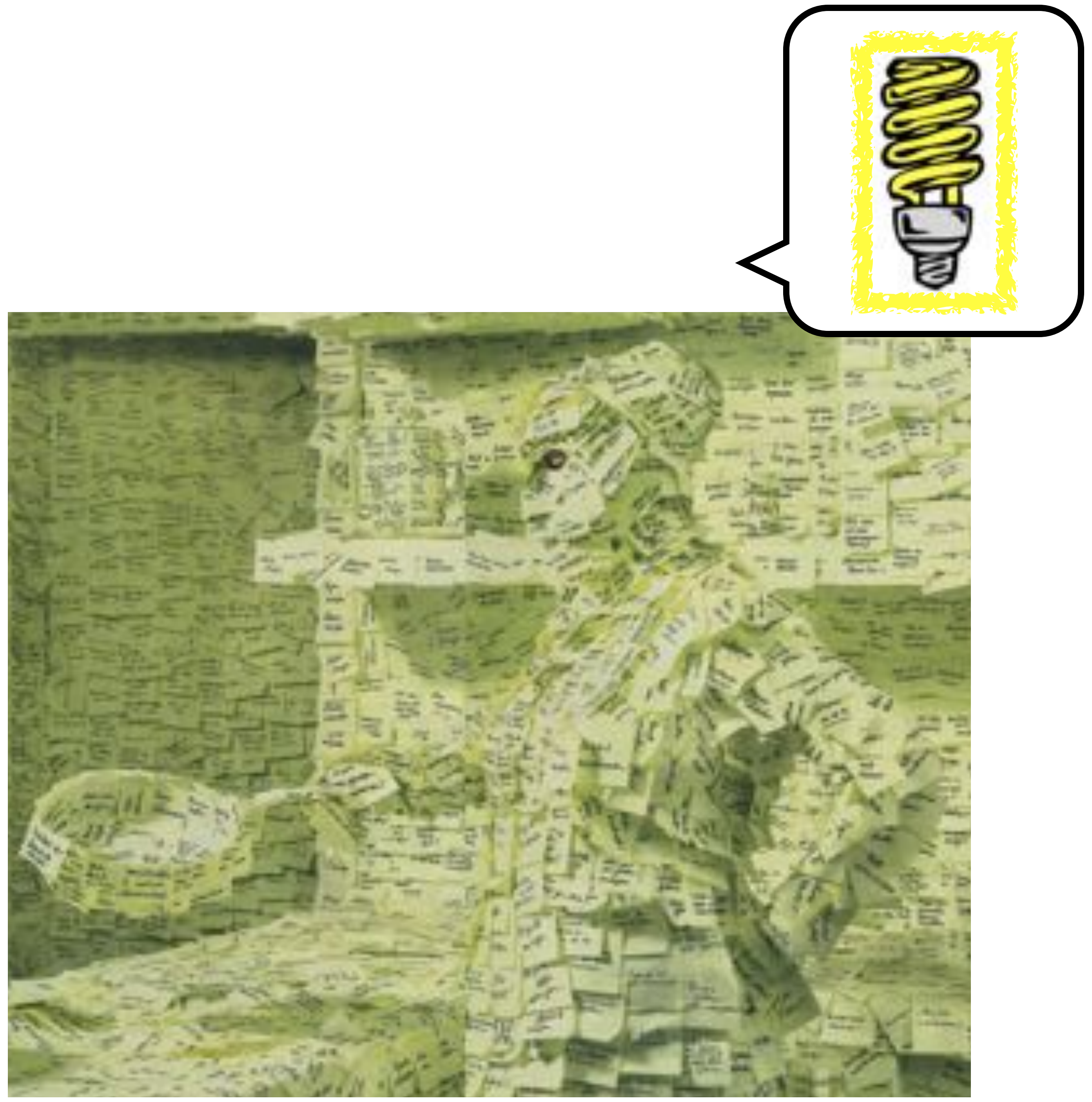
Slide borrowed from Michigan Robotics autorob.org

Robot Control Loop



Should your robot's decision making

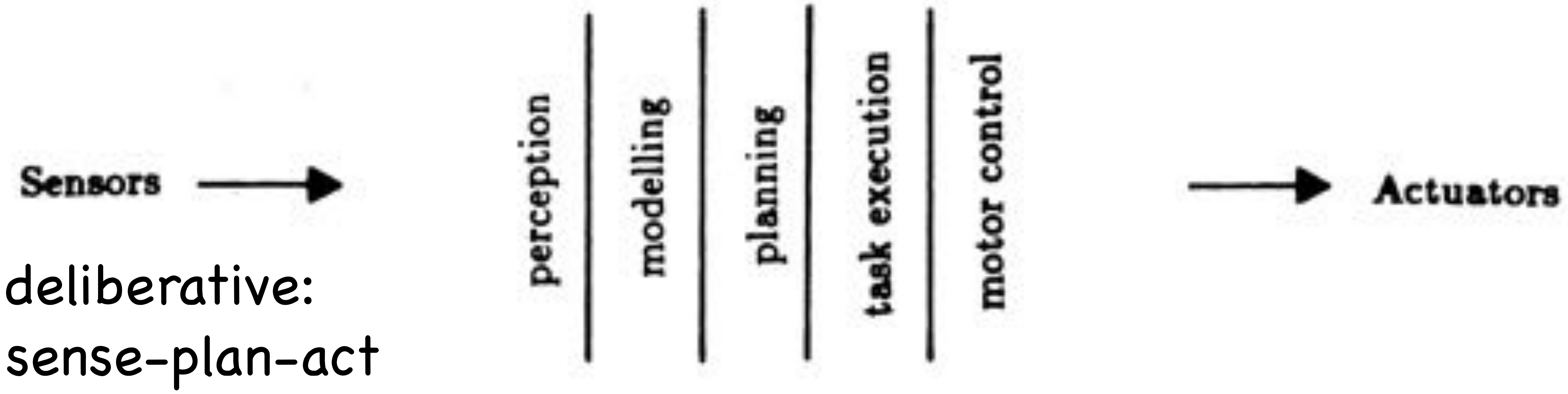
OR



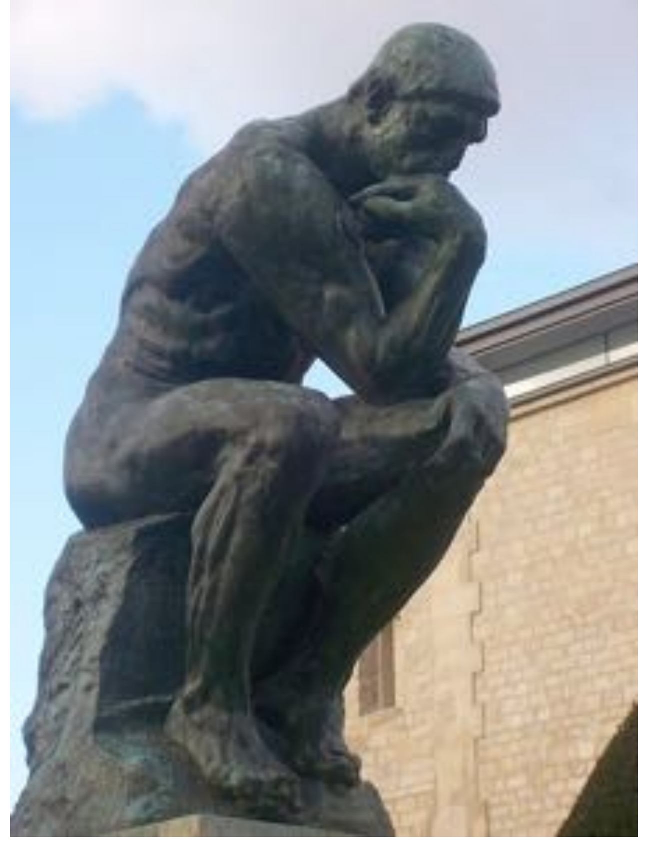
fully think through solving a problem?

react quickly to changes in its world?

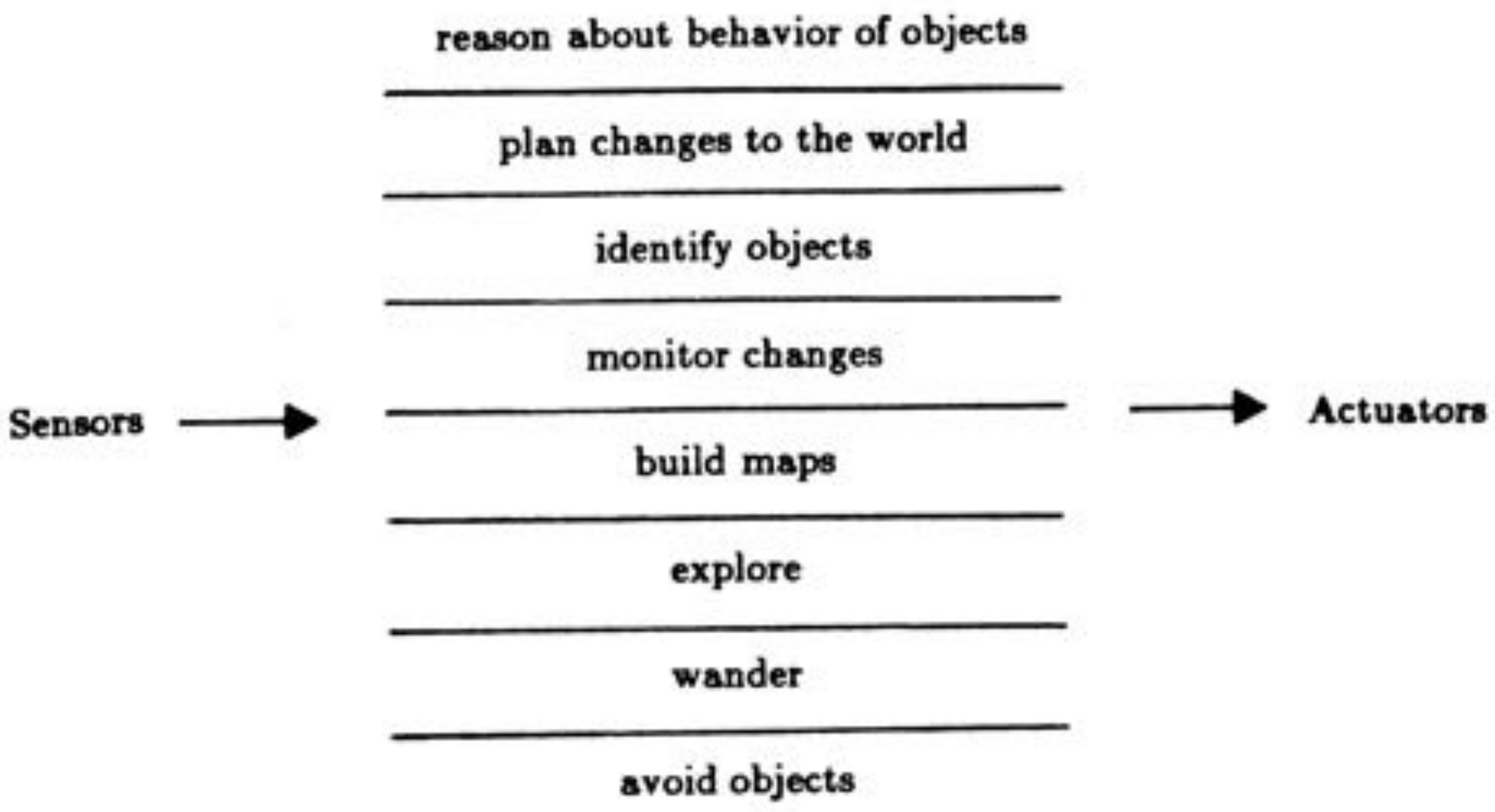
Deliberation v. Reaction



deliberative:
sense-plan-act

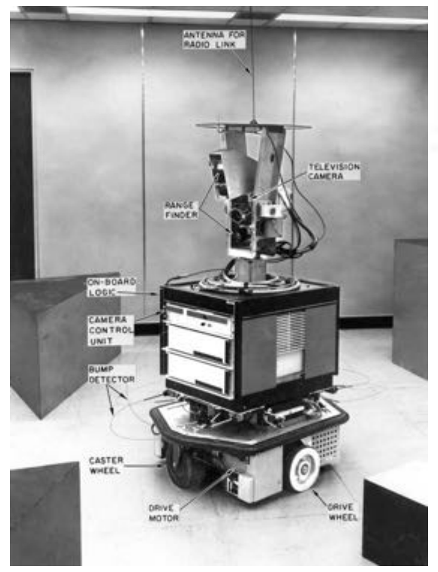


reaction: subsumption,
Finite State Machine
controllers act in parallel



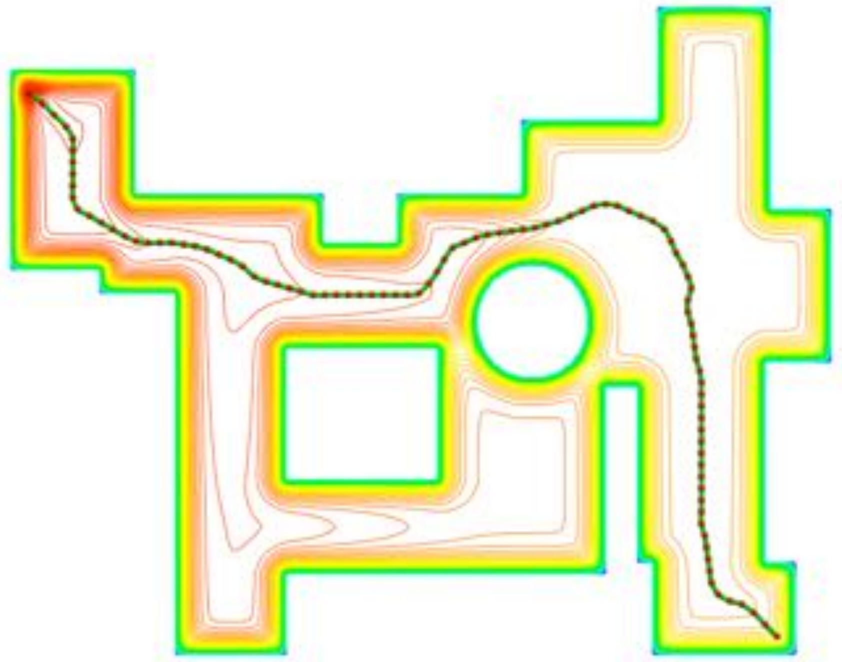
Slide borrowed from Michigan Robotics autorob.org

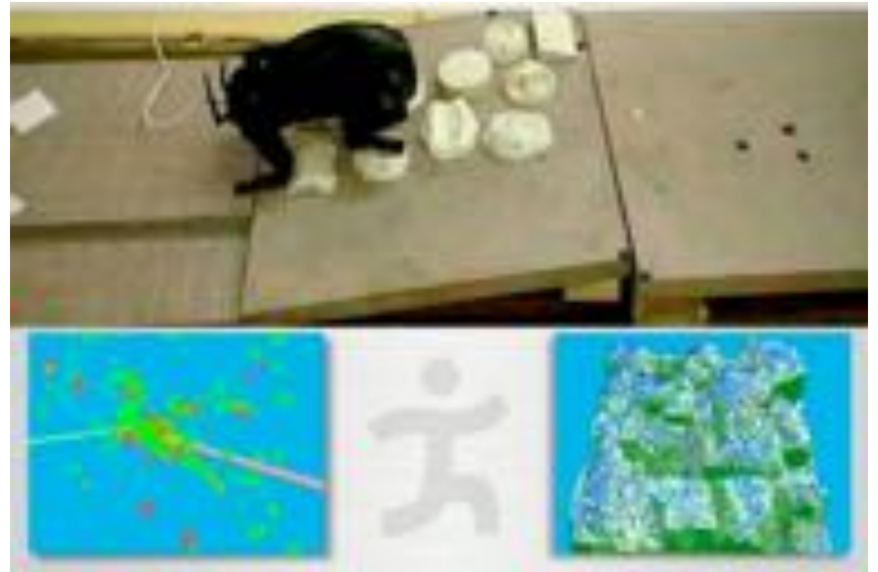
Deliberation



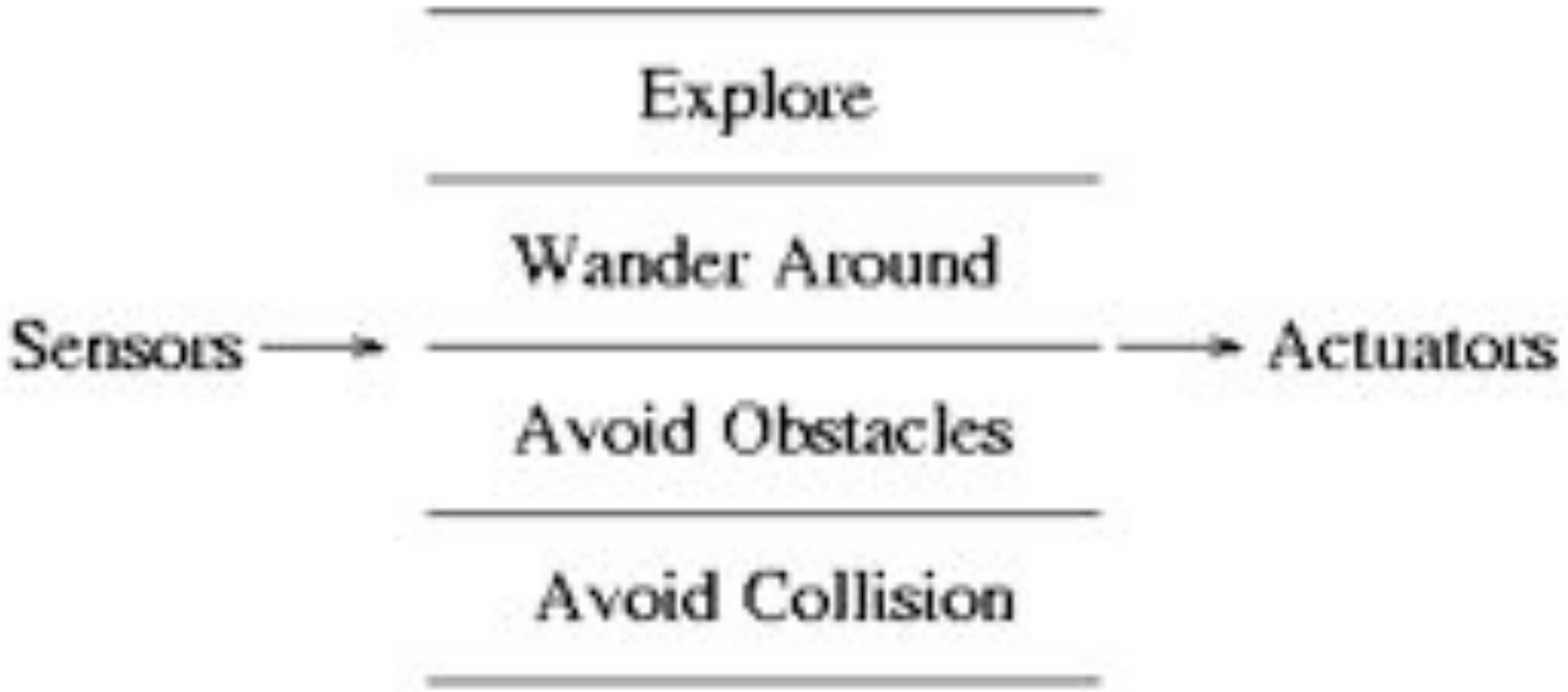
<u>"Sense-Plan-Act" paradigm</u>

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





Reaction

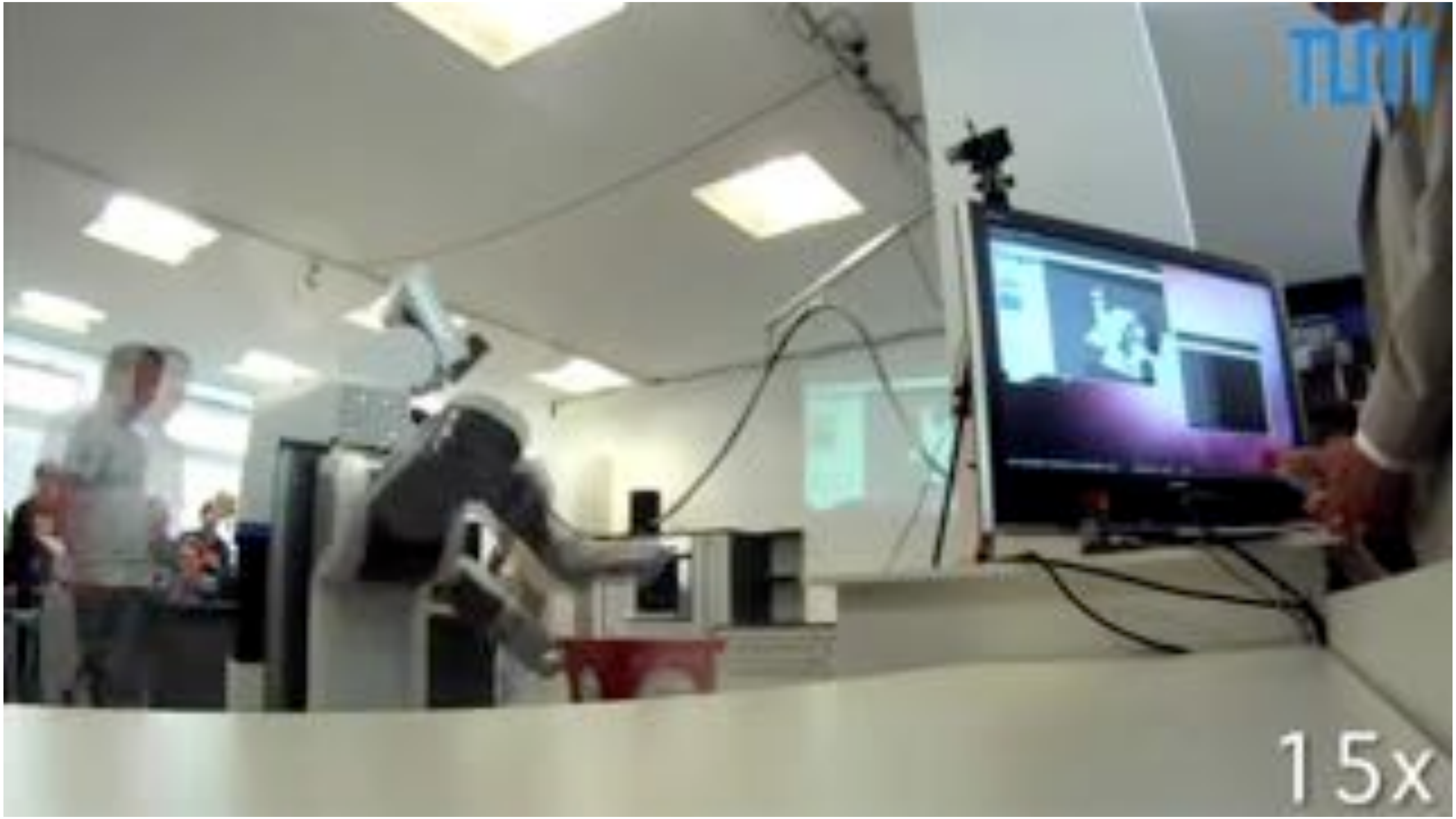


- No representation of state
- Typically, fast hardcoded rules
- Embodied intelligence
  - behavior := control + embodiment
  - ant analogy, stigmergy
- Subsumption architecture
  - prioritized reactive policies
- Ghengis hexpod video

Slide borrowed from Michigan Robotics autorob.org

MIT Genghis

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



Robots have to make lots of decisions

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

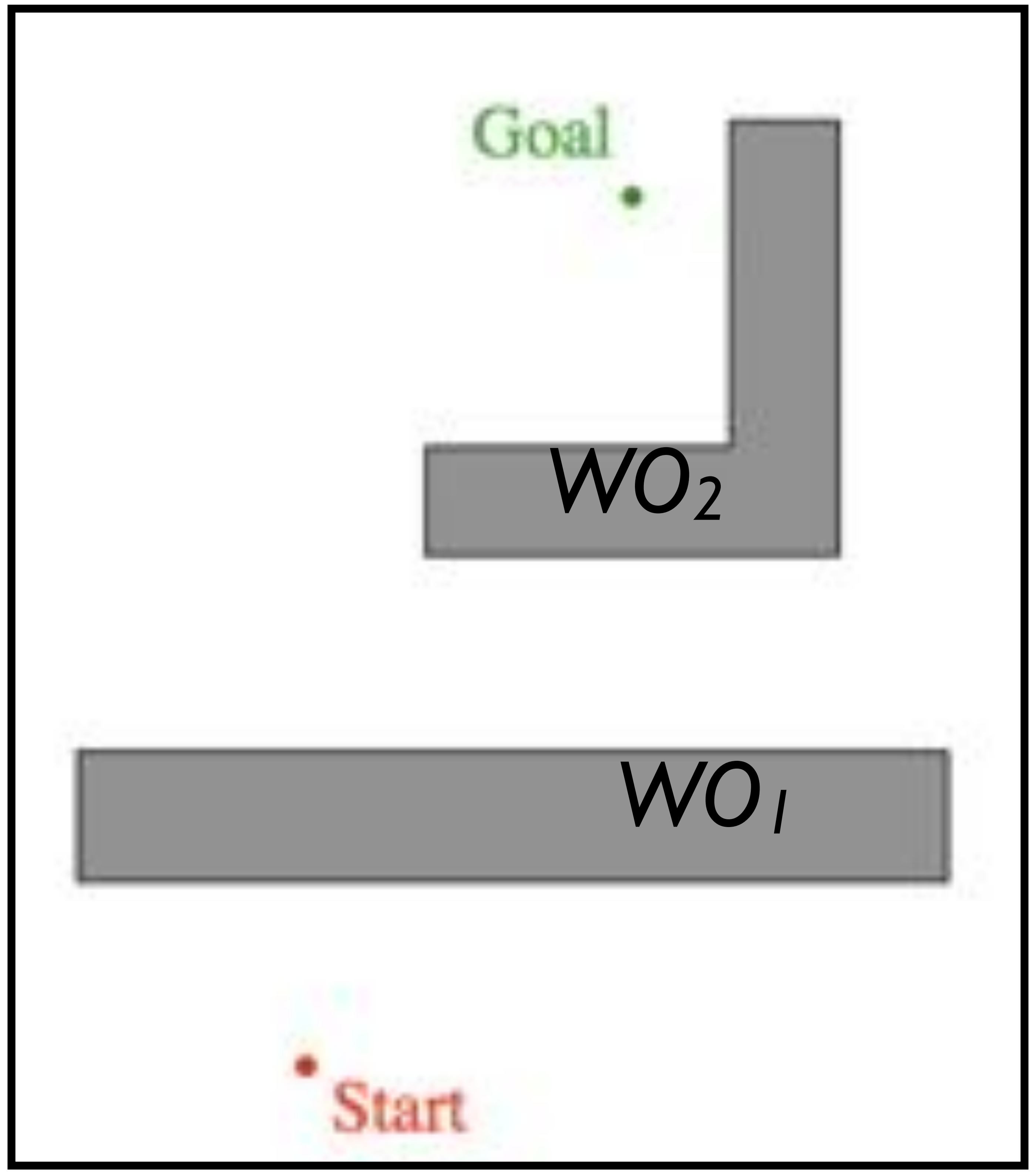
Lisa Miller, http://www.youtube.com/watch?v=VBzXDrz8rMI

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?**

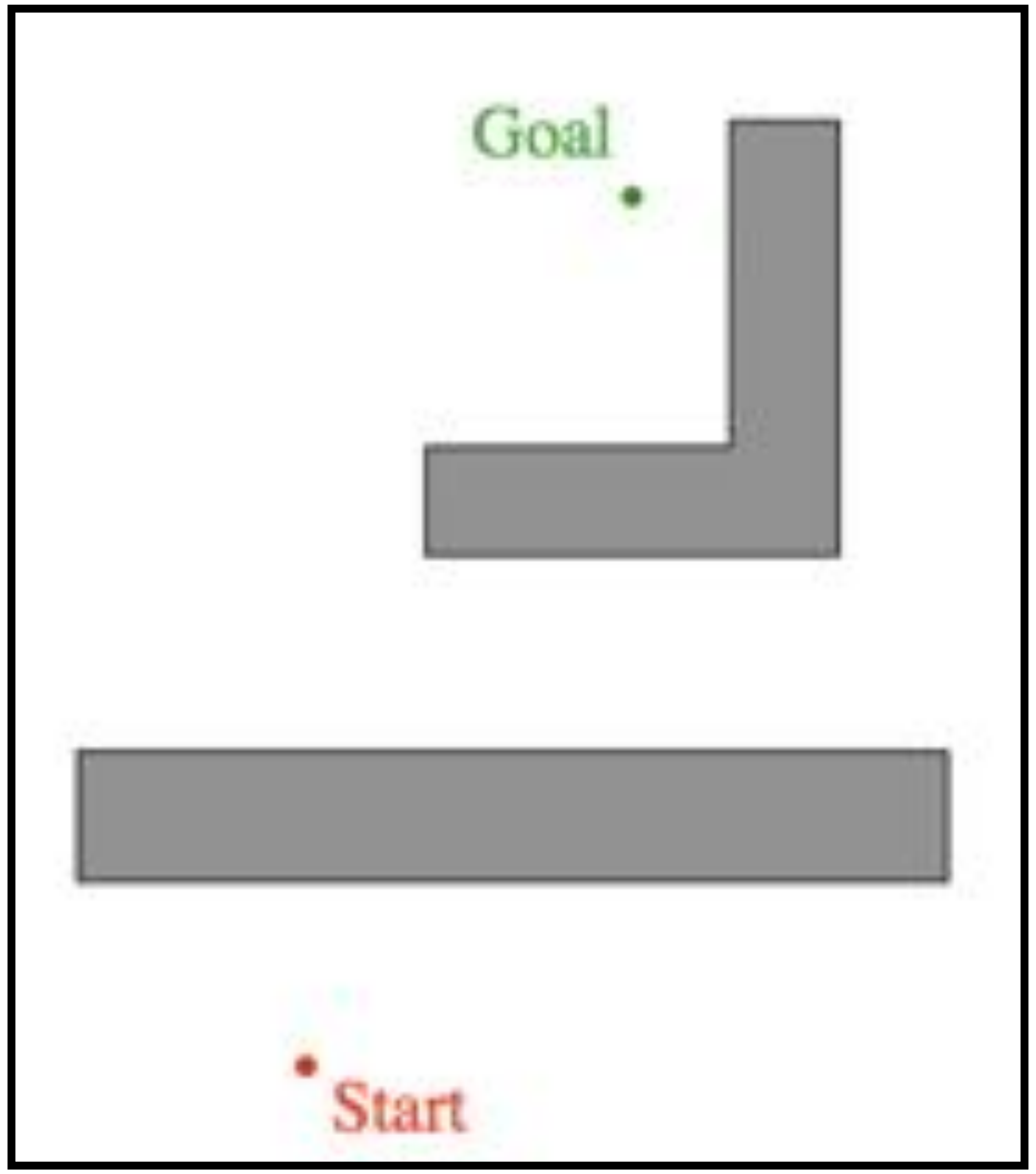
Bug Algorithms

- 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



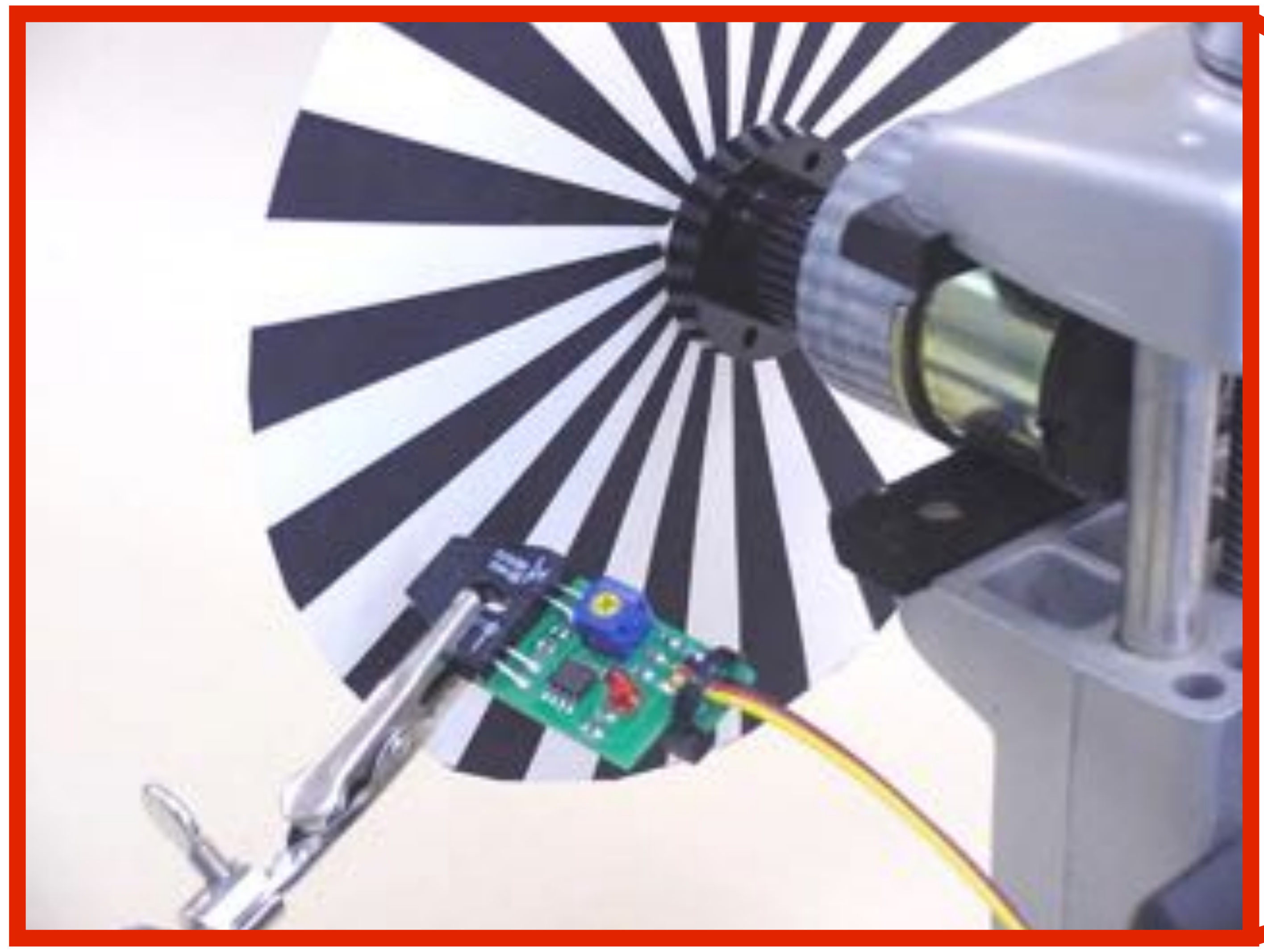
≈

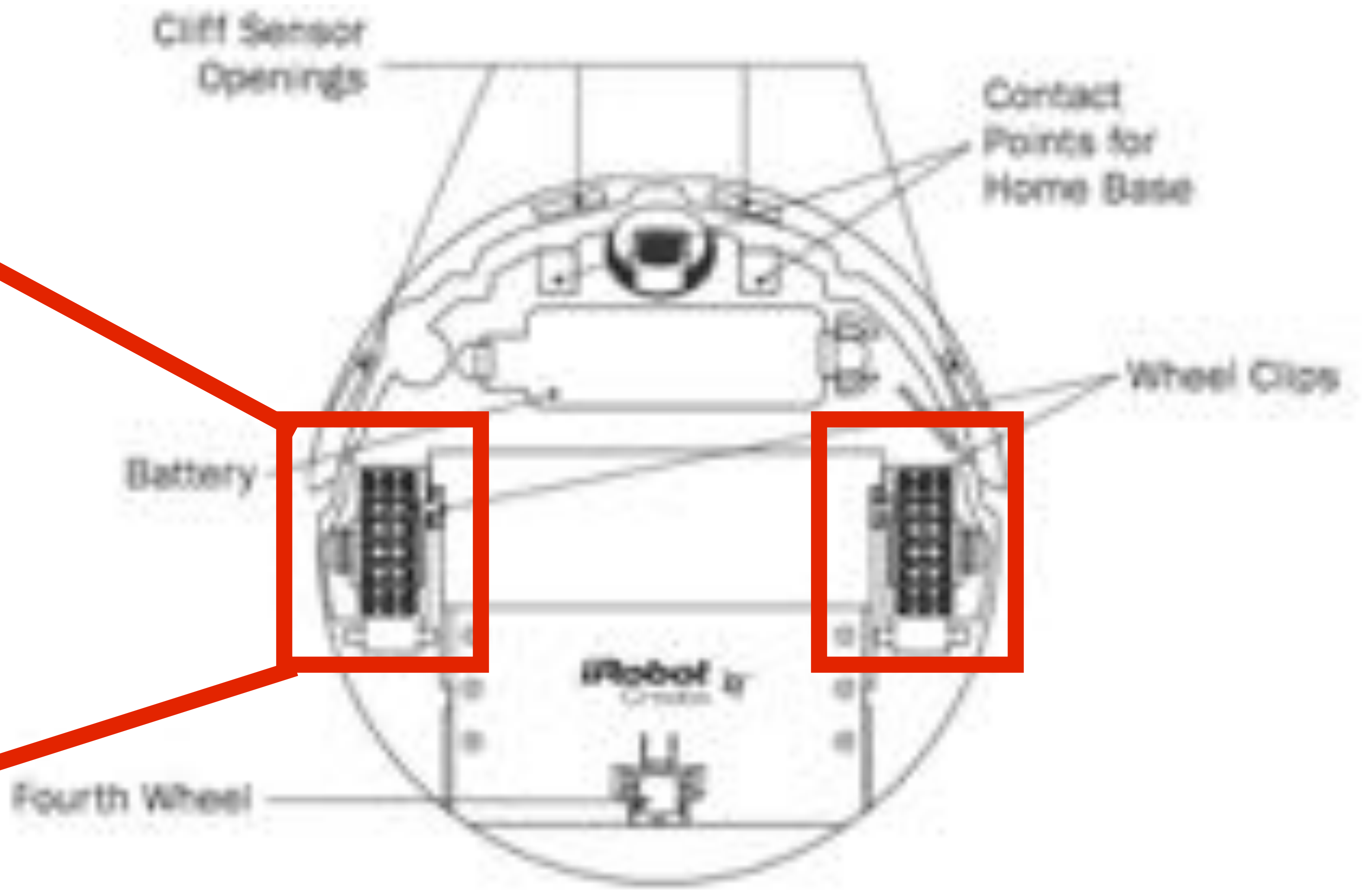
bumper is essentially an on/off button

Slide borrowed from Michigan Robotics autorob.org

Bug A

Optical encoders



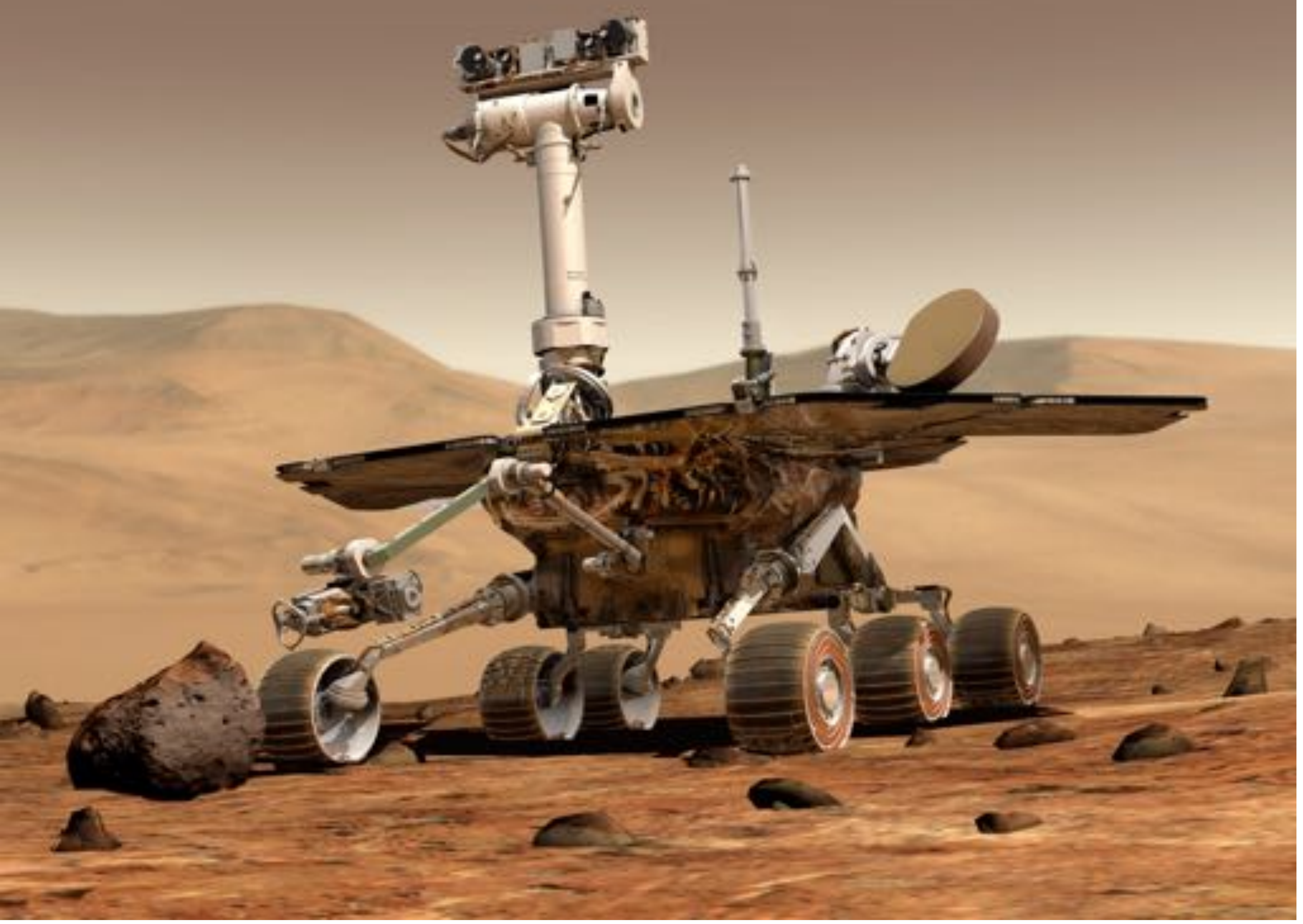


- Assume
- Known:
  - measur
- Unknow
- Local sensing
  - bump
  - odometry

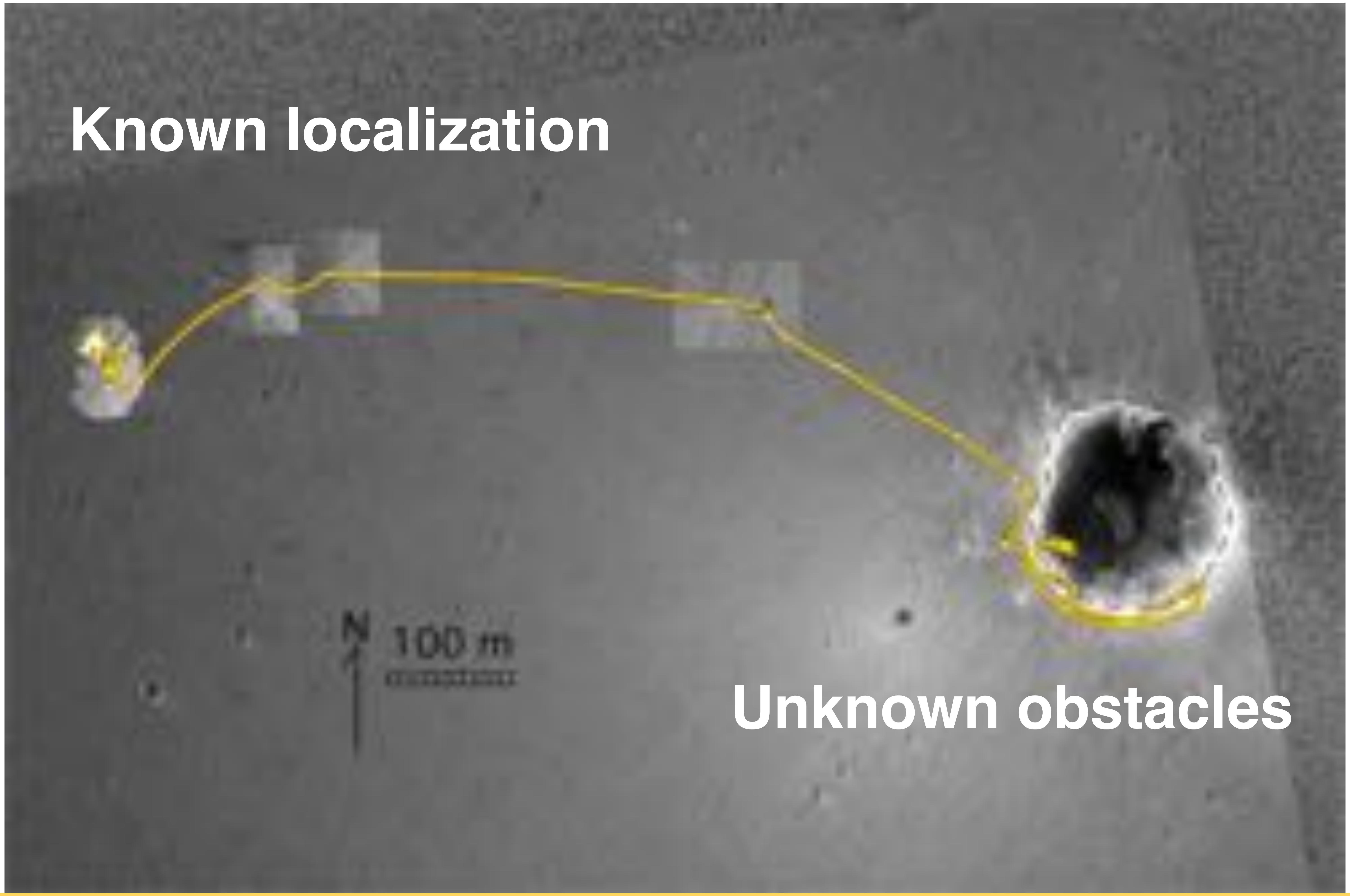
≈

Interesting application of Bug algorithms ?

Mars Exploration Rover



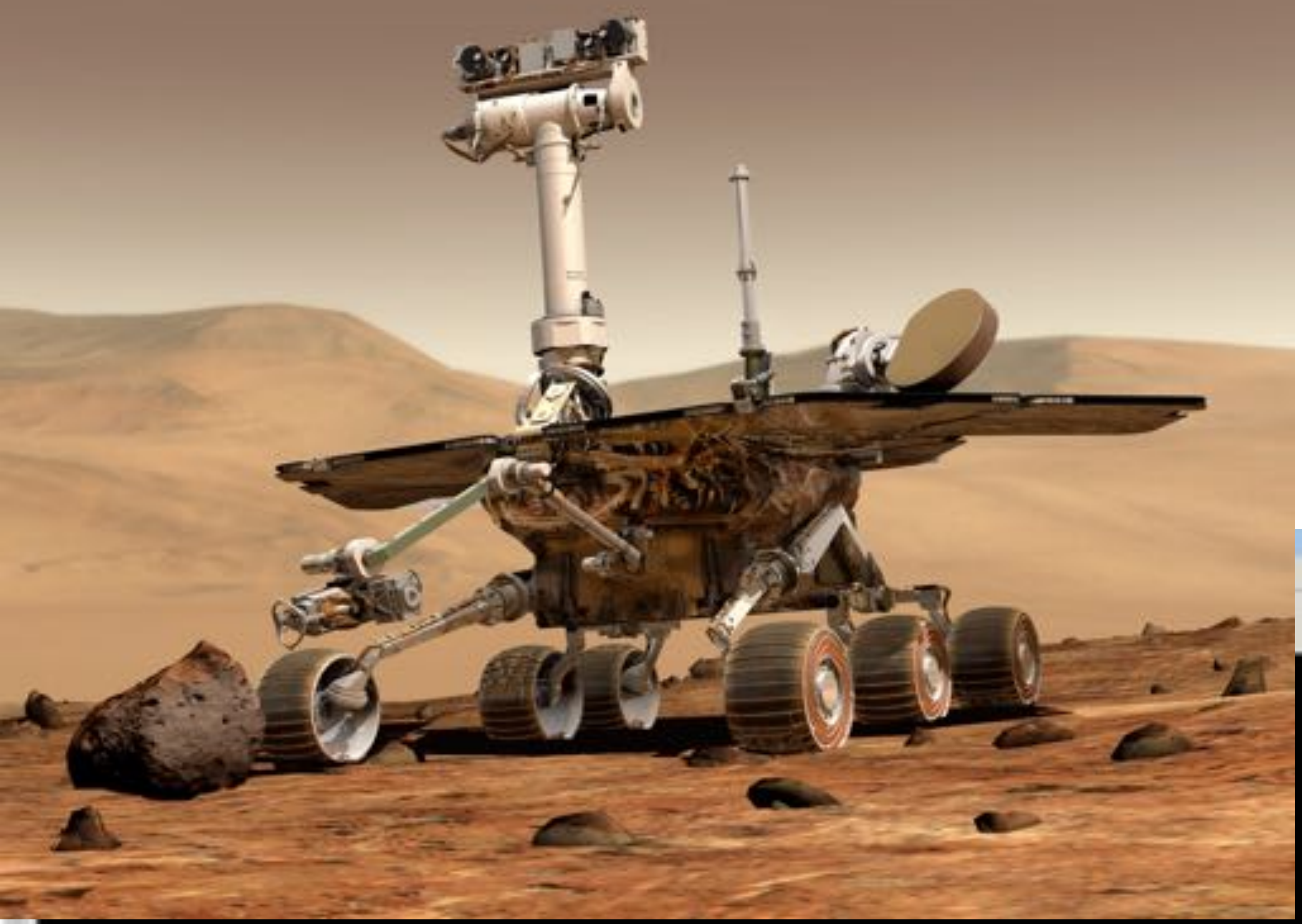
http://mars.nasa.gov/mer/gallery/press/opportunity/20040921a.html

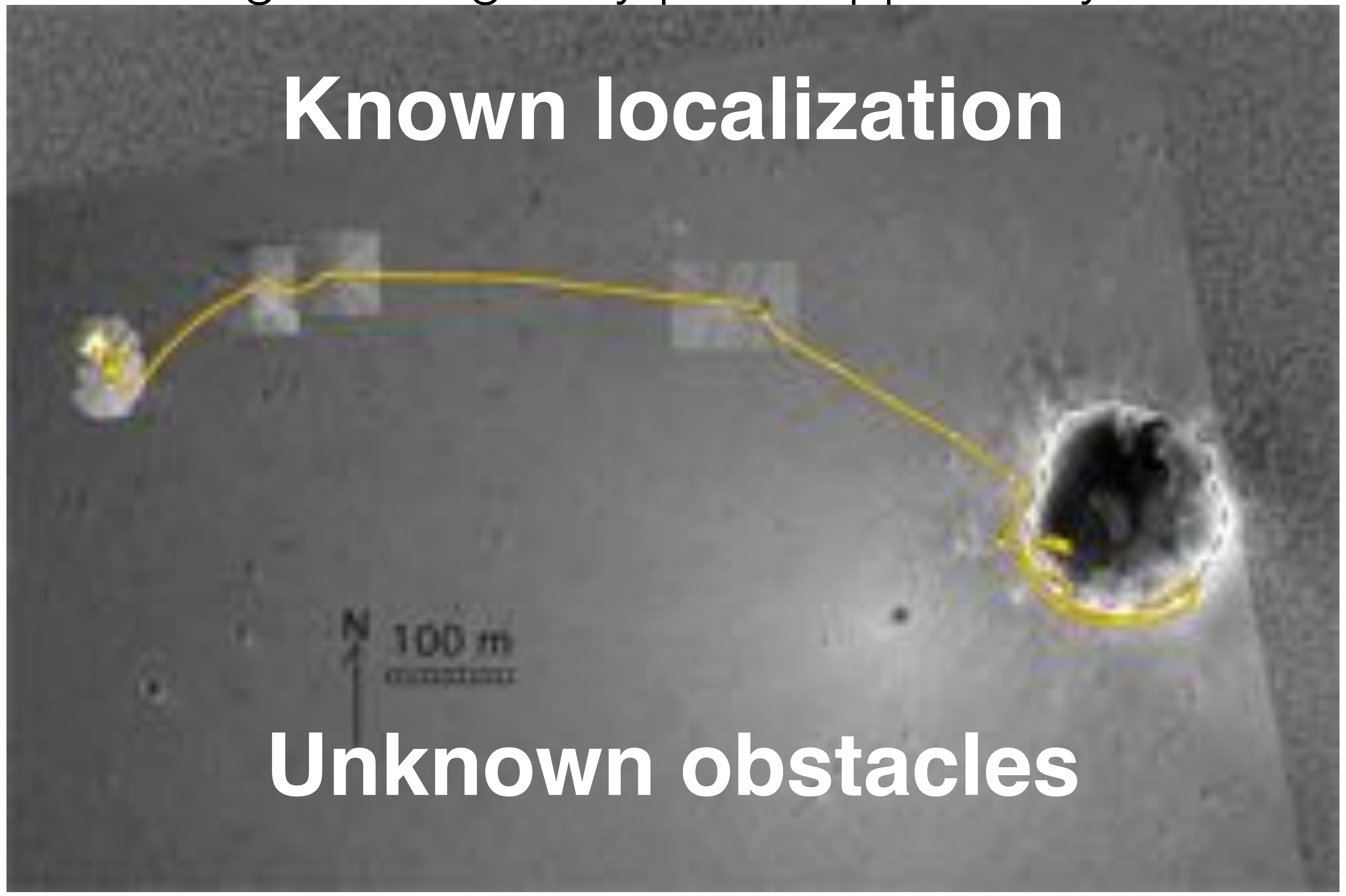


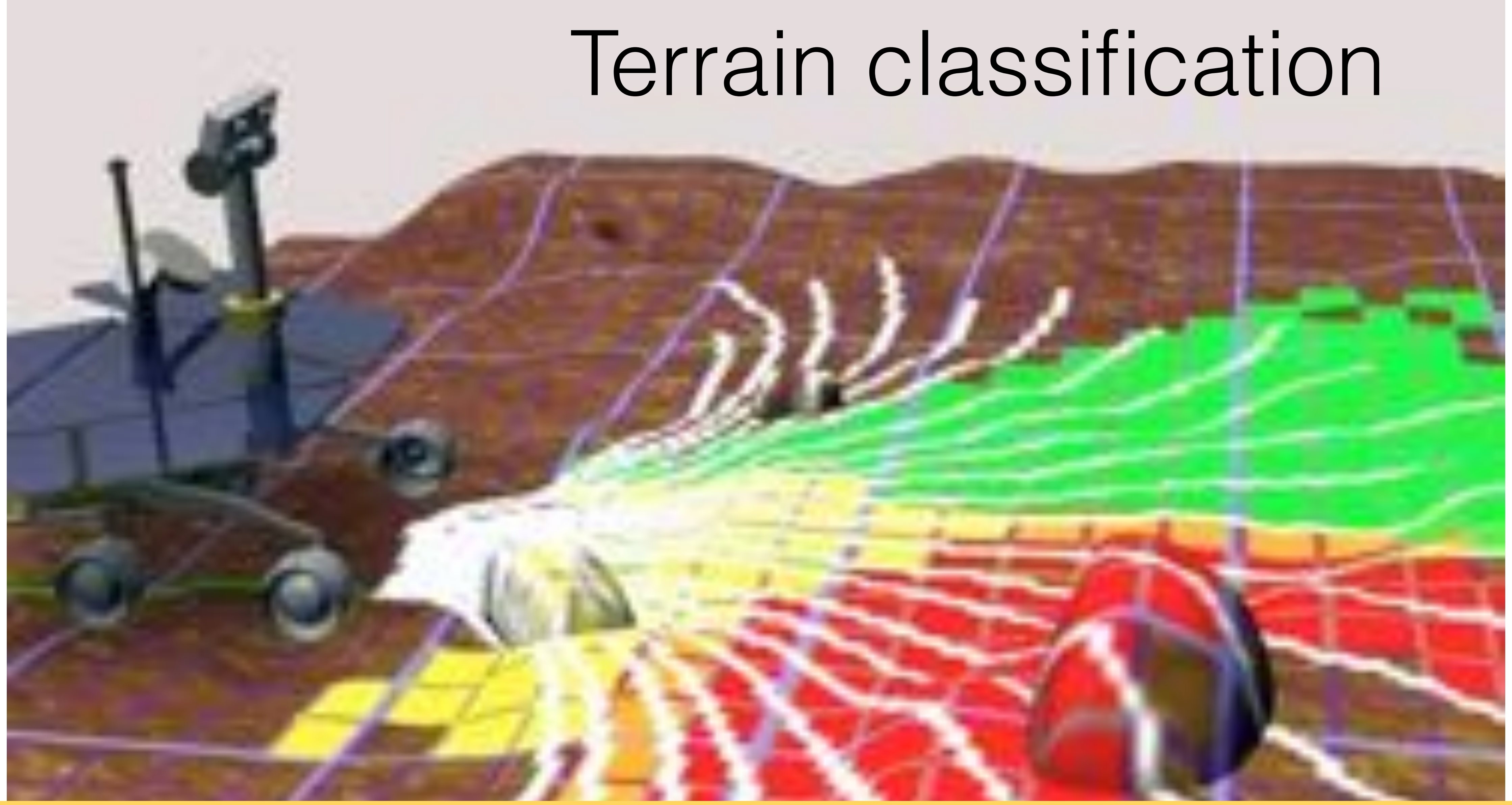
Slide borrowed from Michigan Robotics autorob.org

Mars Exploration Rover

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

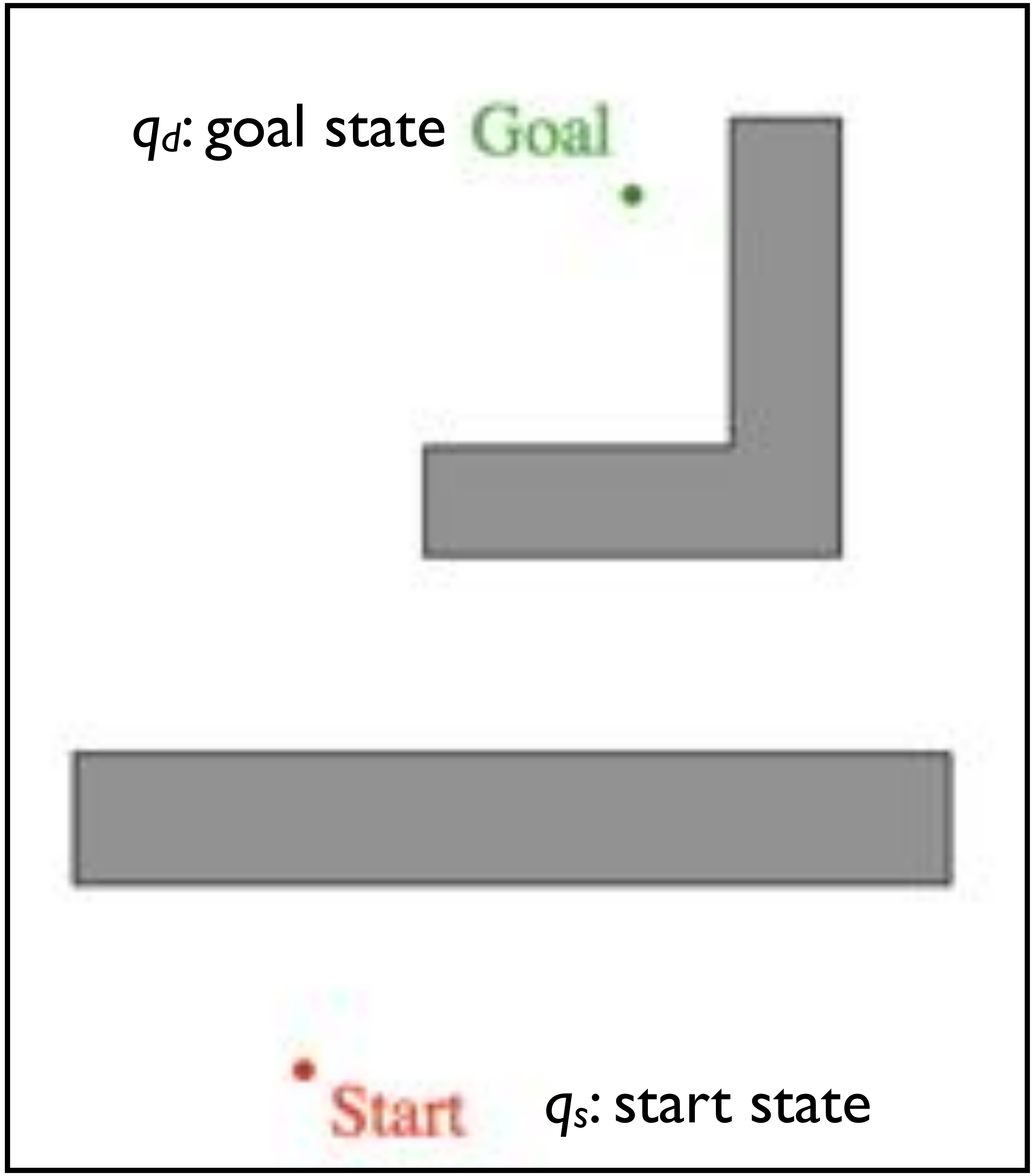






Slide borrowed from Michigan Robotics autorob.org

Bug Navigation

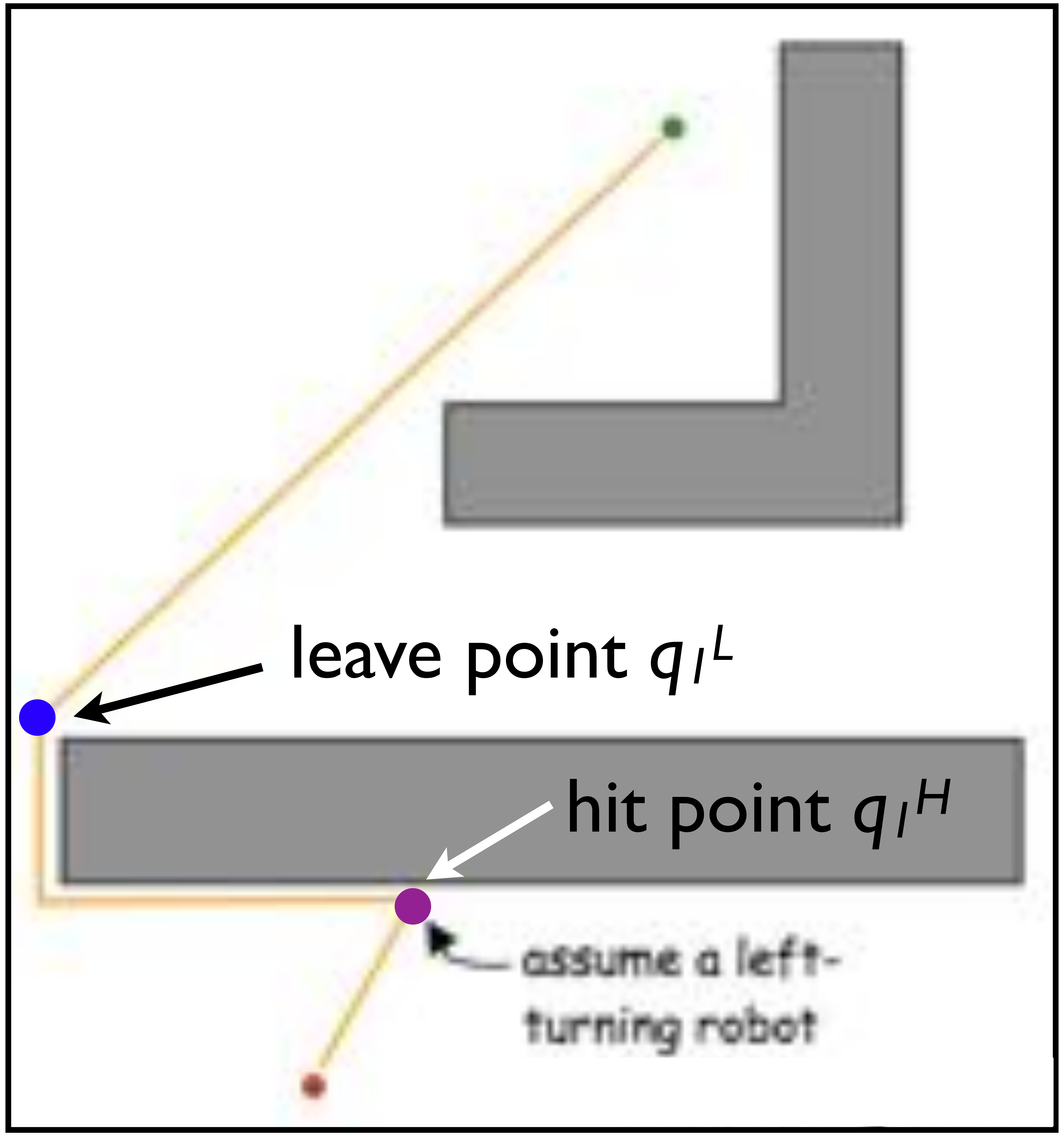


Plan navigation path from start q_s to goal q_d

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)

Wall following

follow wall

One approach:

a) move forward with slight turn

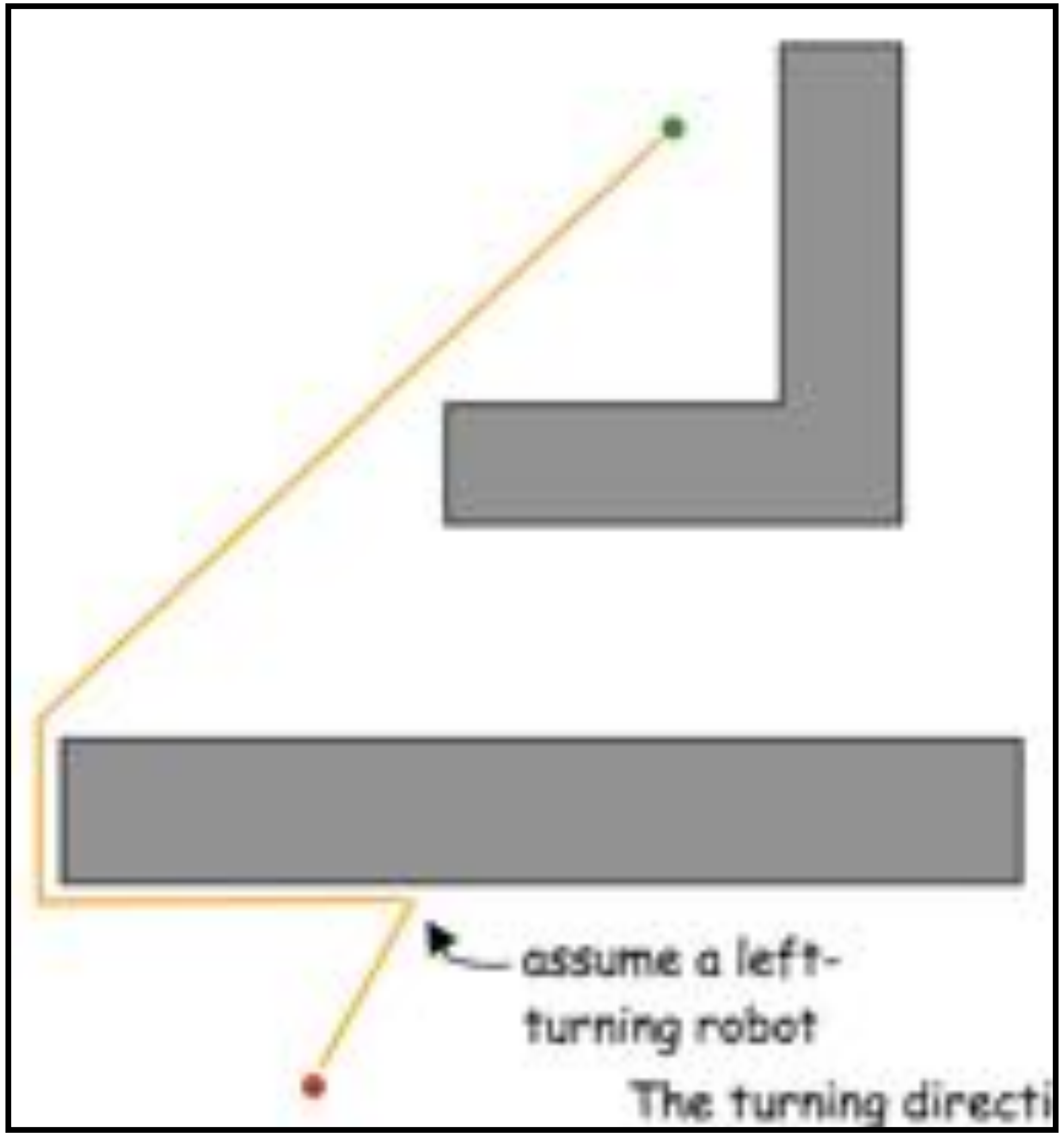
b) when bumped, turn opposite direction

c) goto (*a*)

Trevor Jay

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)

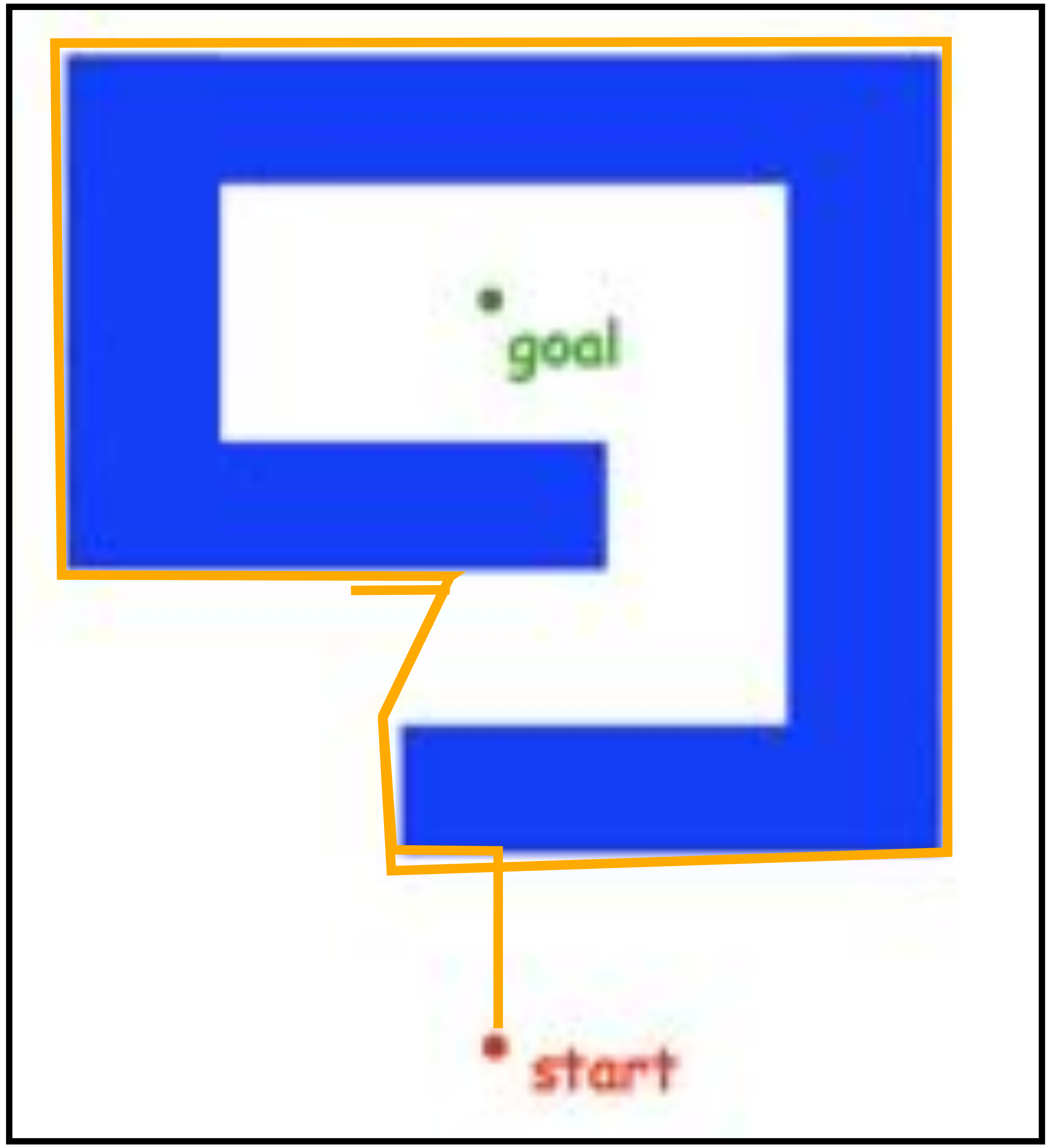
Bug 0

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?
Can we make a better bug? How?



Bug 0

1) Head towards goal

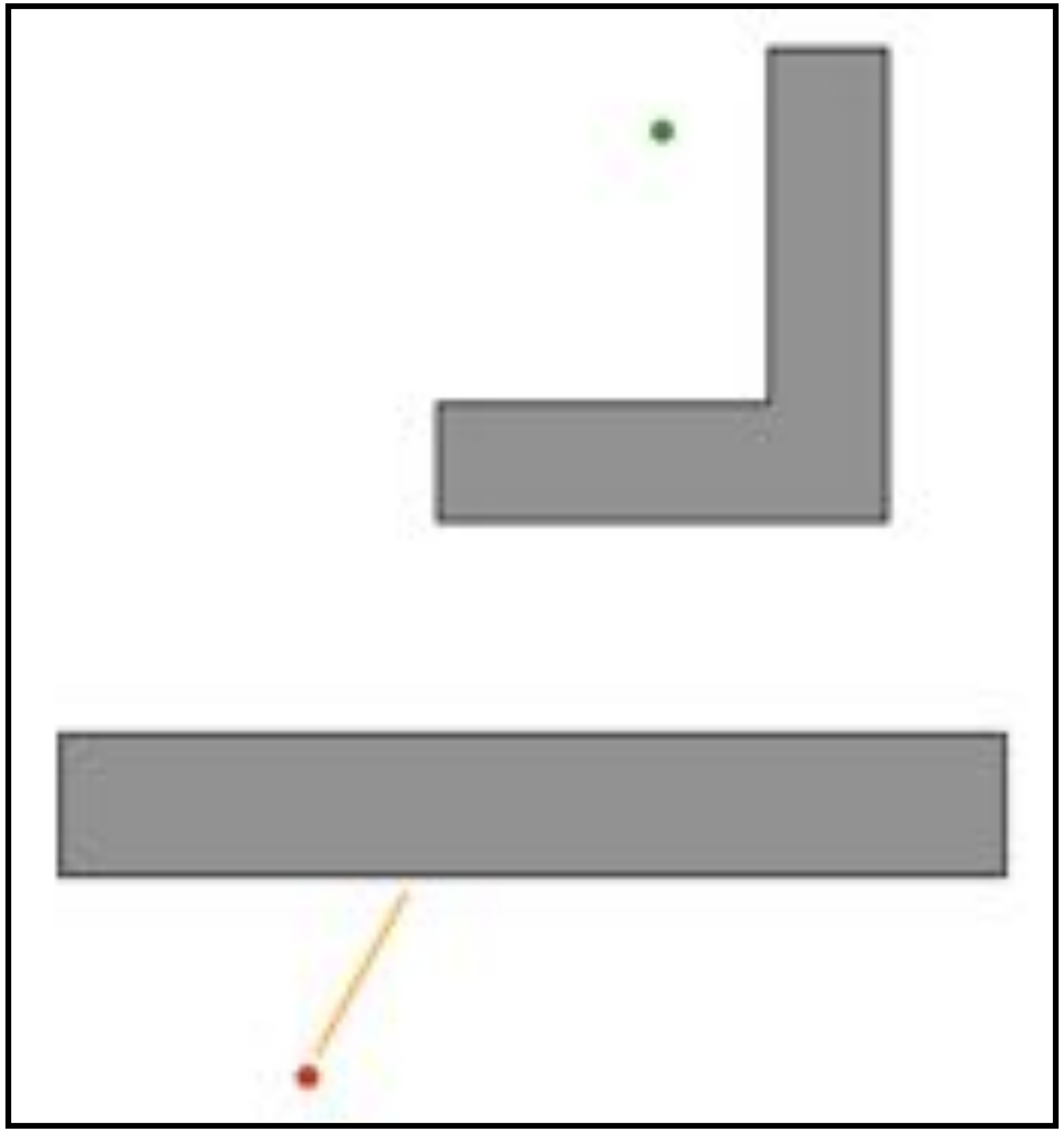
2) When hit point set, follow wall, until you can move towards goal again (leave point)

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



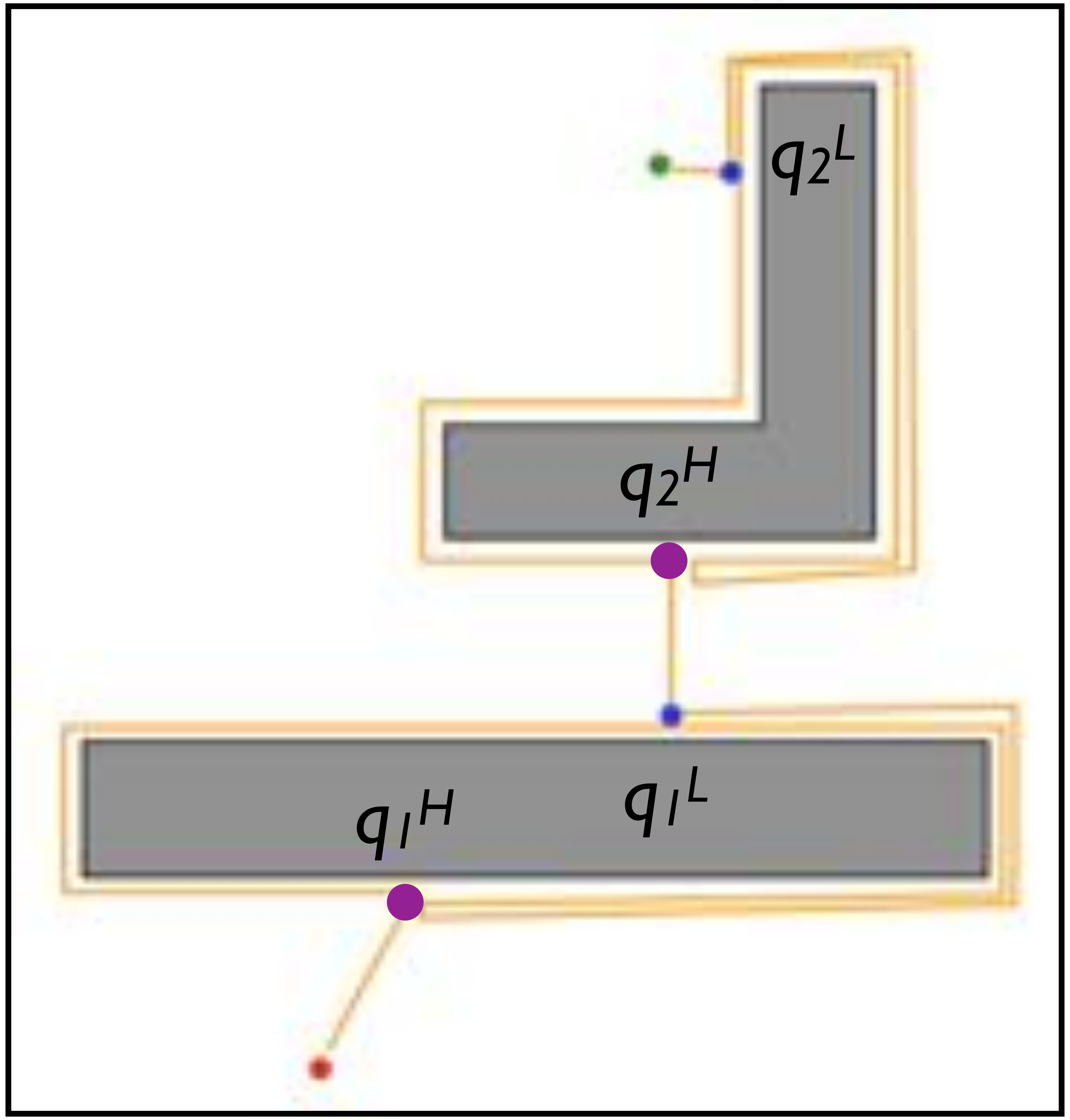
1) Head towards goal

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

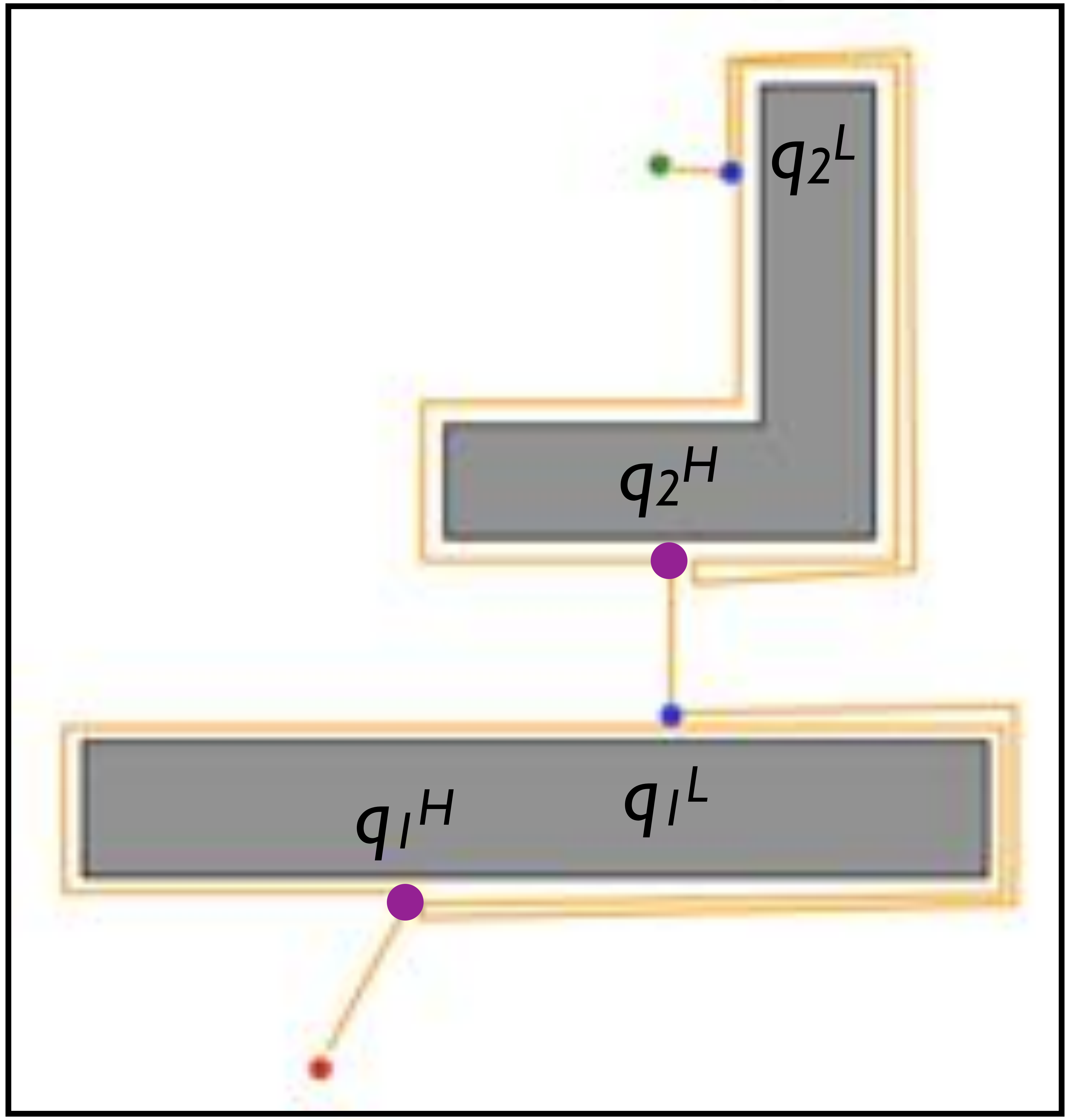
2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

3) return to leave point

4) continue from (1)

Bug 1

What map would foil Bug 1?



1) Head towards goal

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

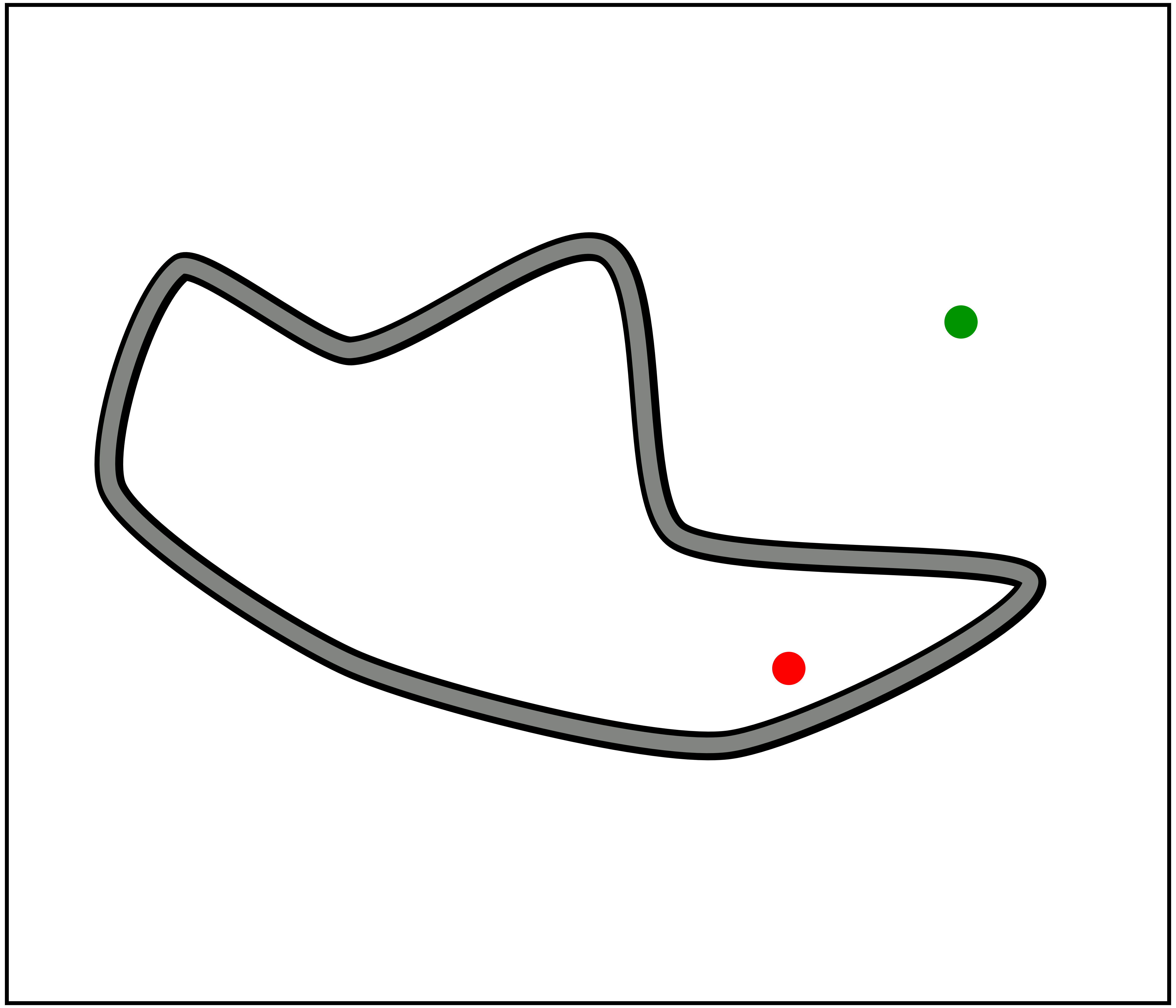
3) return to leave point

4) continue from (1)

Slide borrowed from Michigan Robotics autorob.org

Bug 1

What map would foil Bug 1?



1) Head towards goal

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

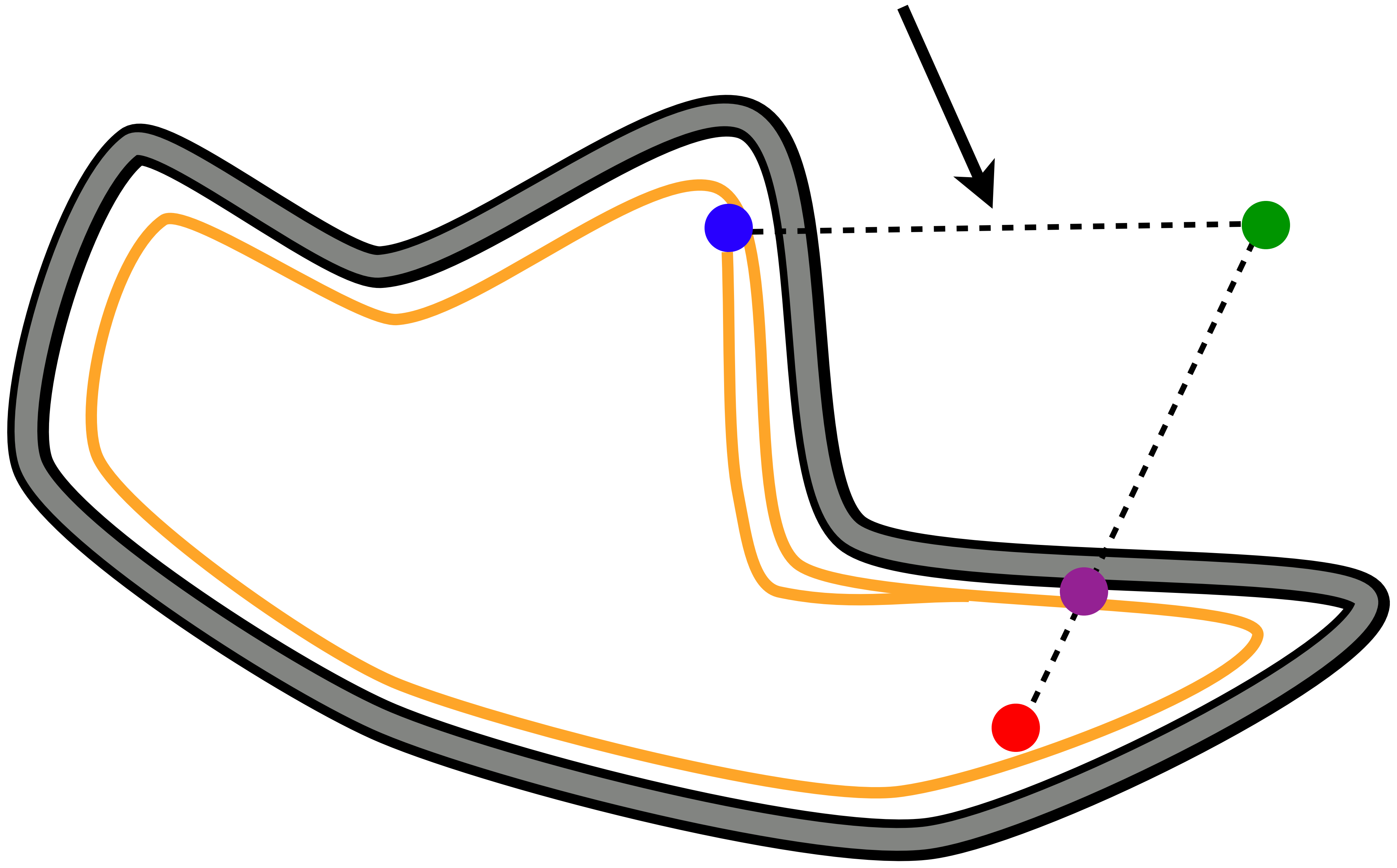
3) return to leave point

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

What map would foil Bug 1?

Bug 1: Detecting Failure

no path exists: line(q_l^L,q_d) intersects current obstacle



failure bump occurs immediately

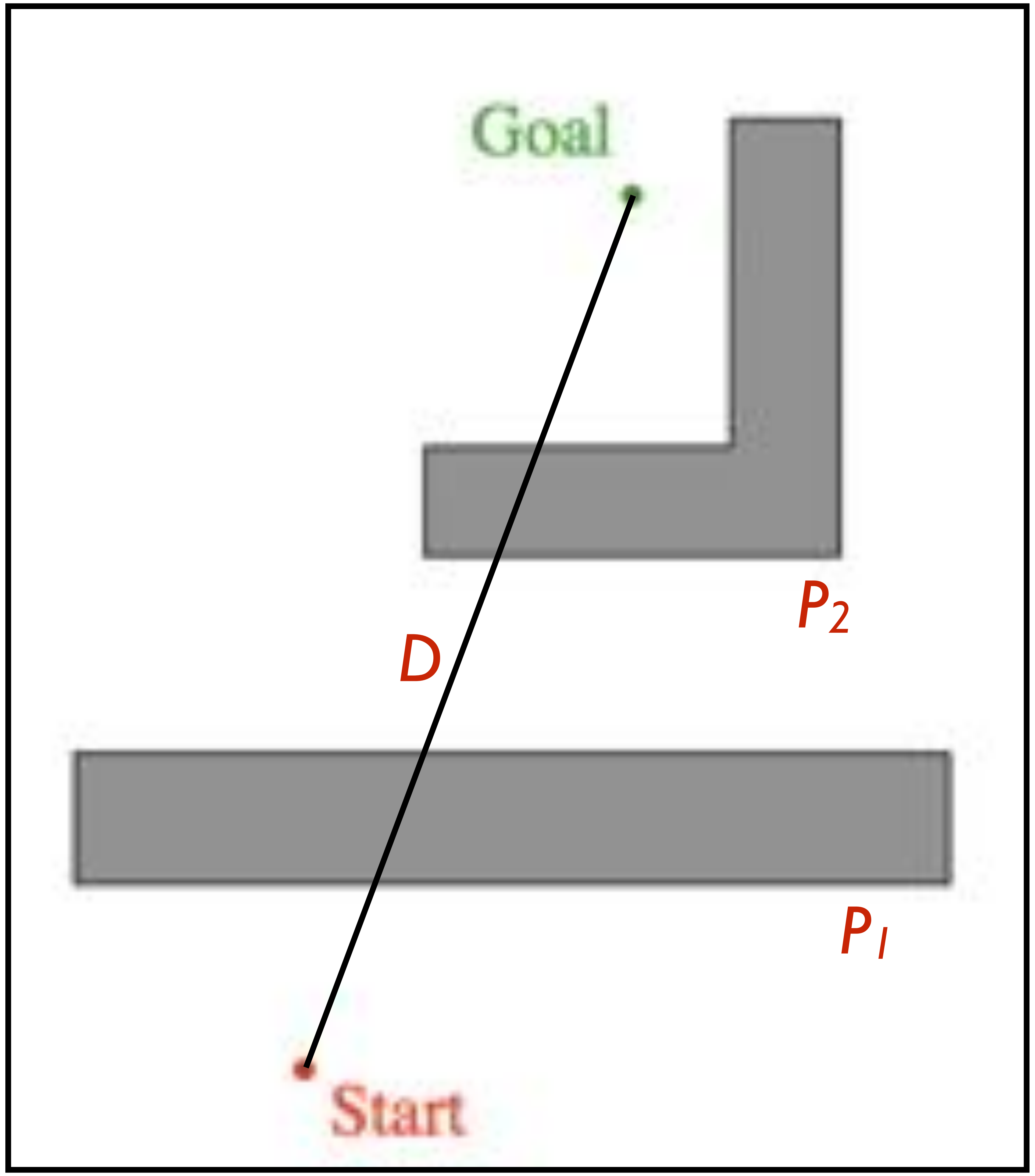
1) Head towards goal

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

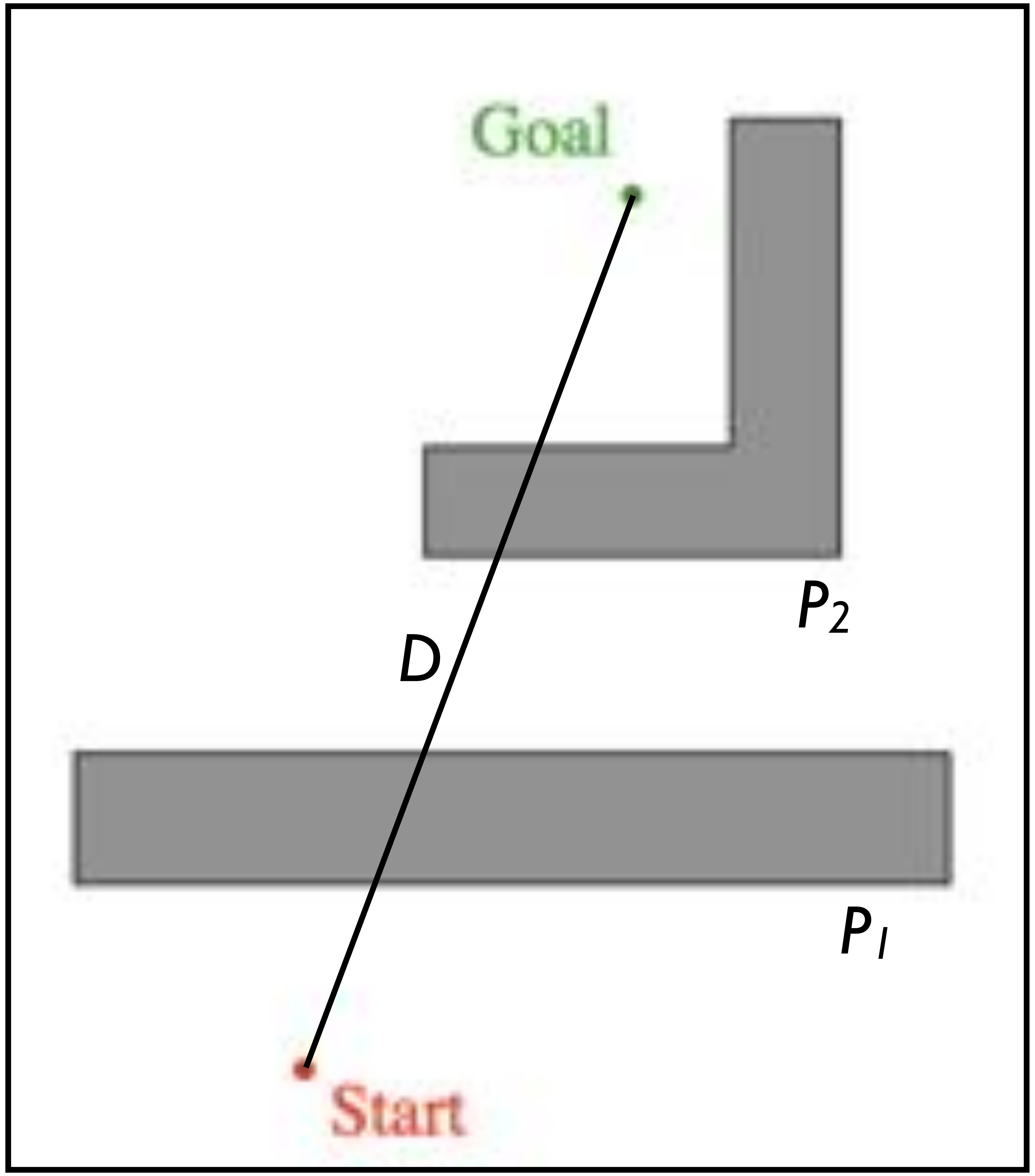


Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: [redacted]

Worst case: [redacted]

Bug 1: Search Bounds

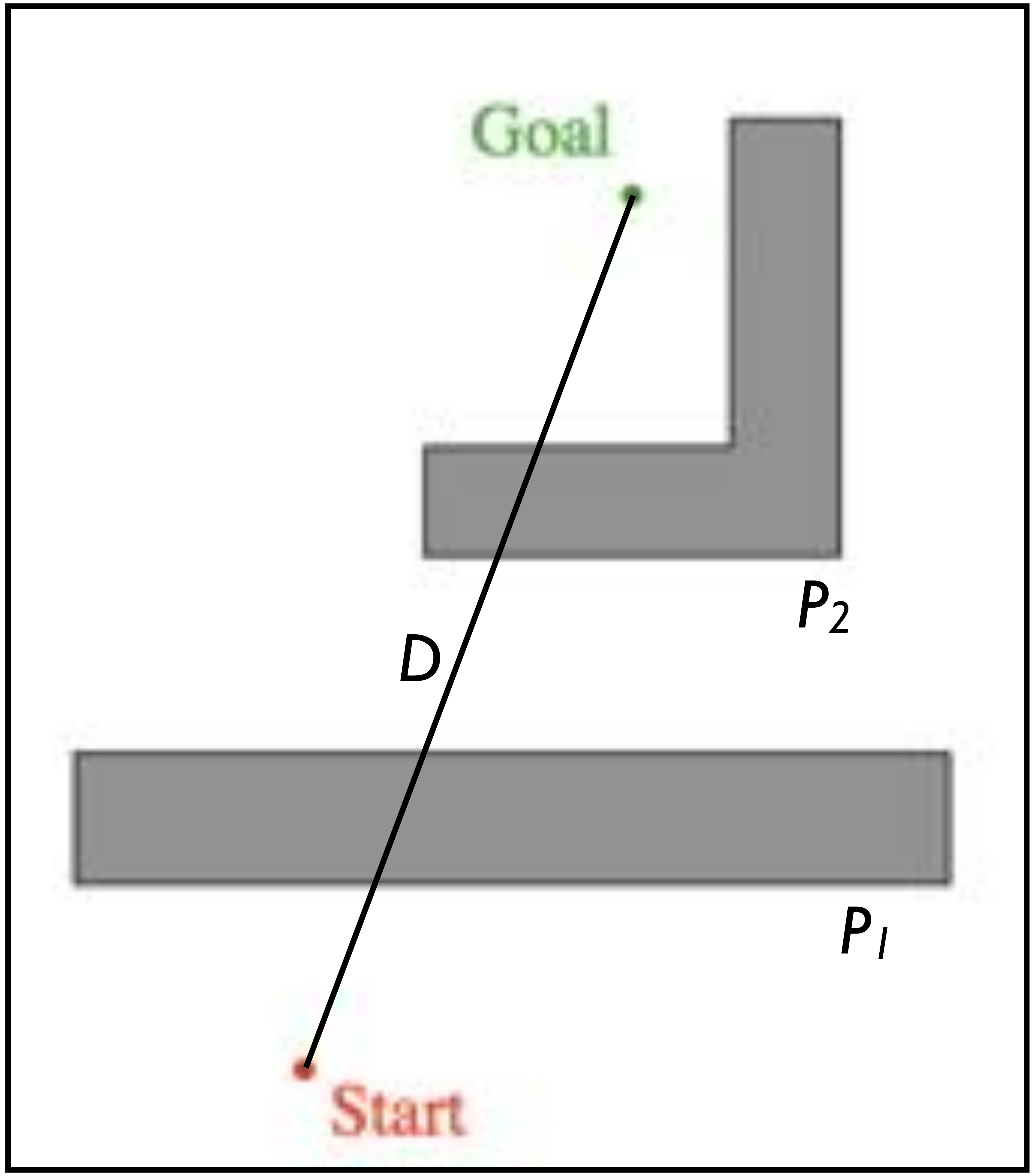


Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: D

Worst case: [redacted]

Bug 1: Search Bounds



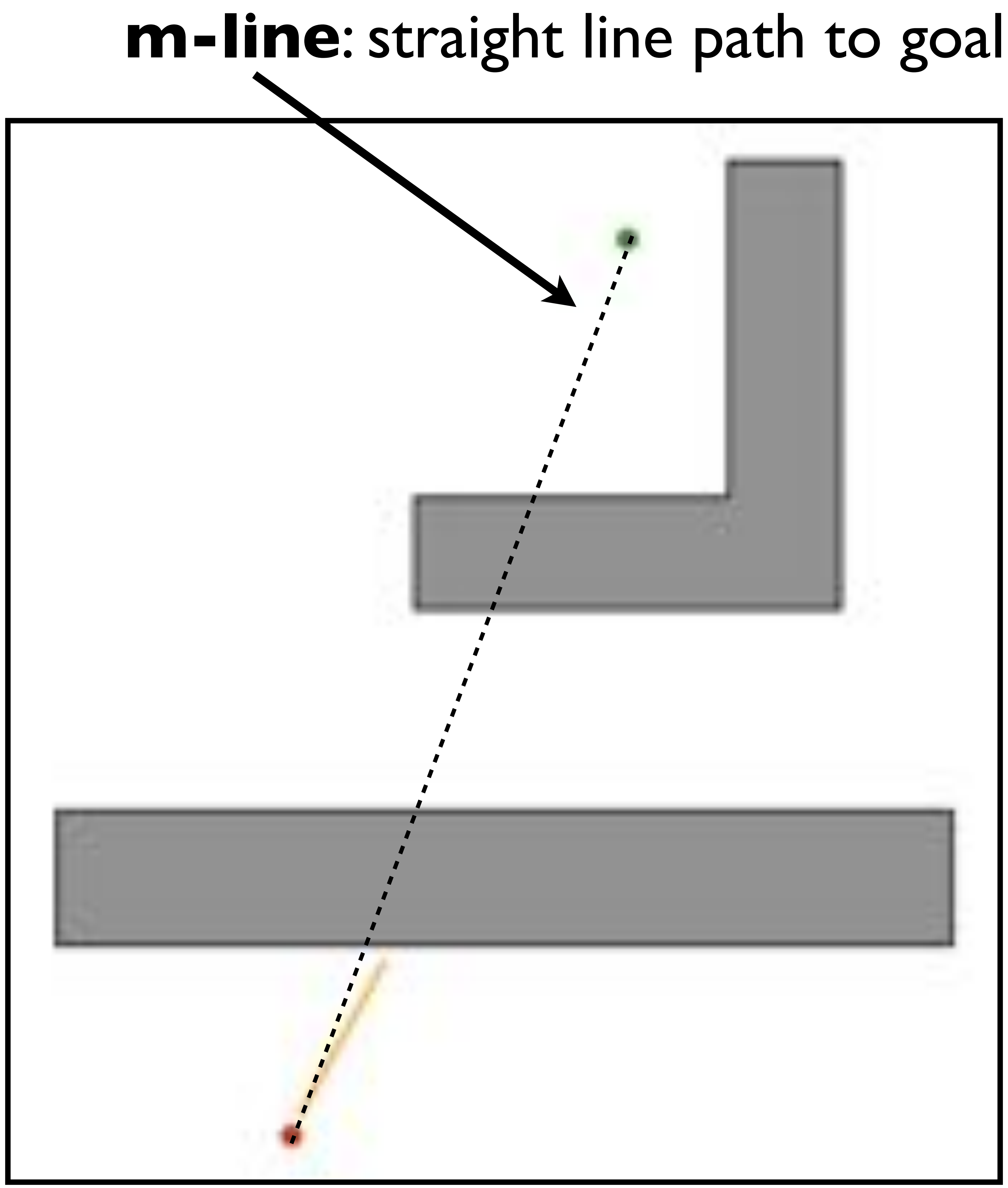
Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: D

Worst case: $D + 1.5\sum_i P_i$

Is there a faster bug?

Bug 2



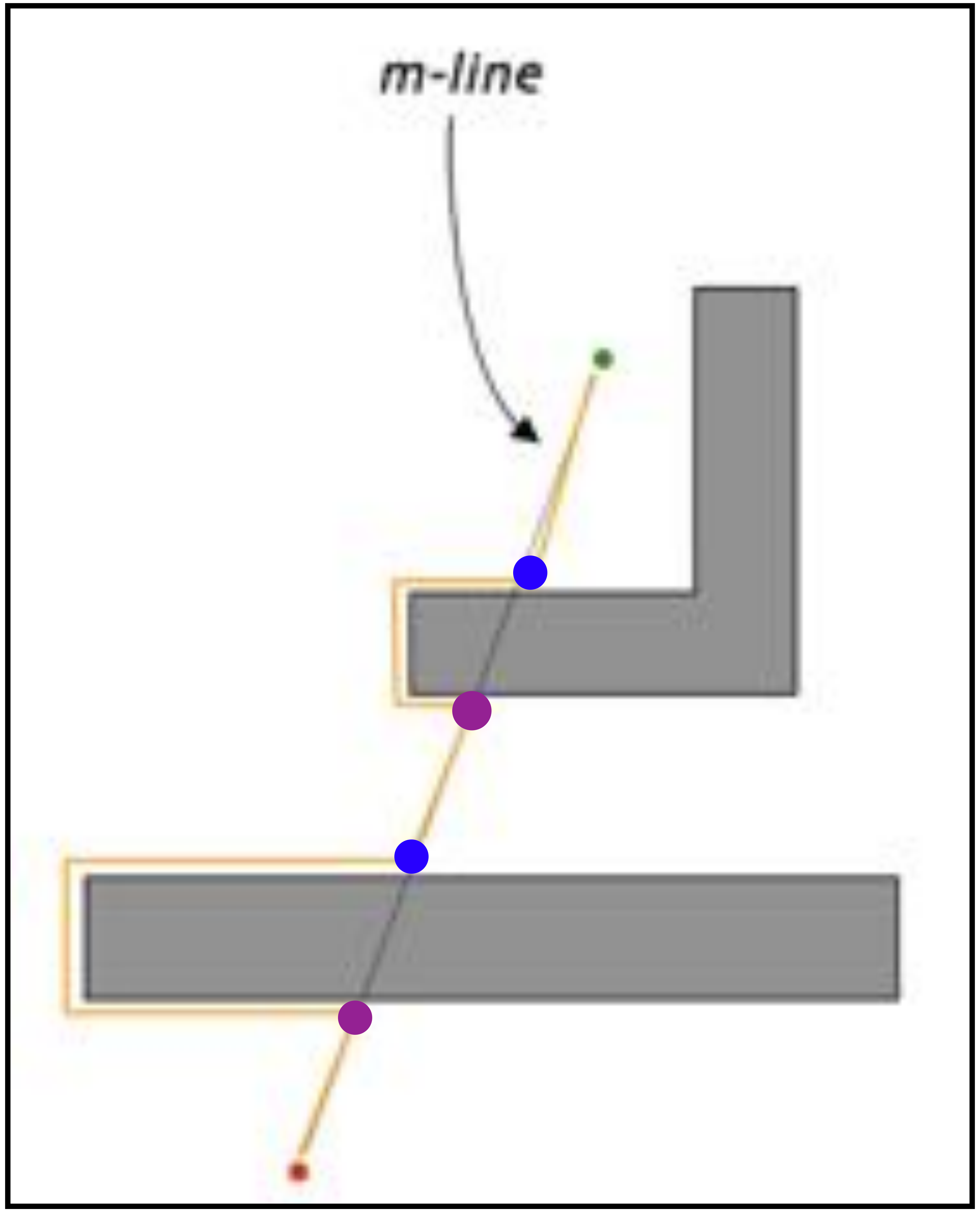
1) Head towards goal on m-line

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

3) set leave point and exit obstacle

4) continue from (1)

Bug 2



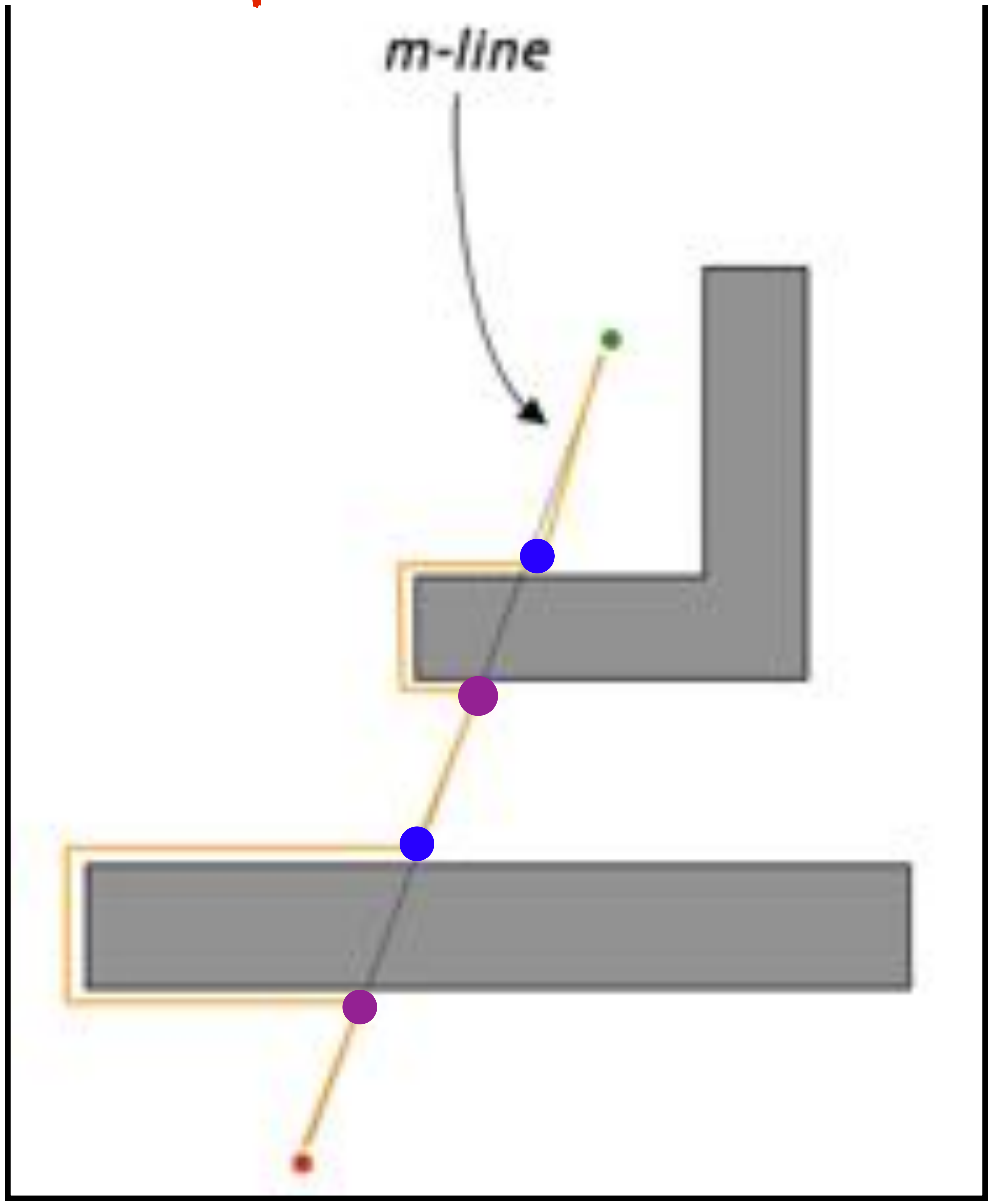
1) Head towards goal on m-line

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

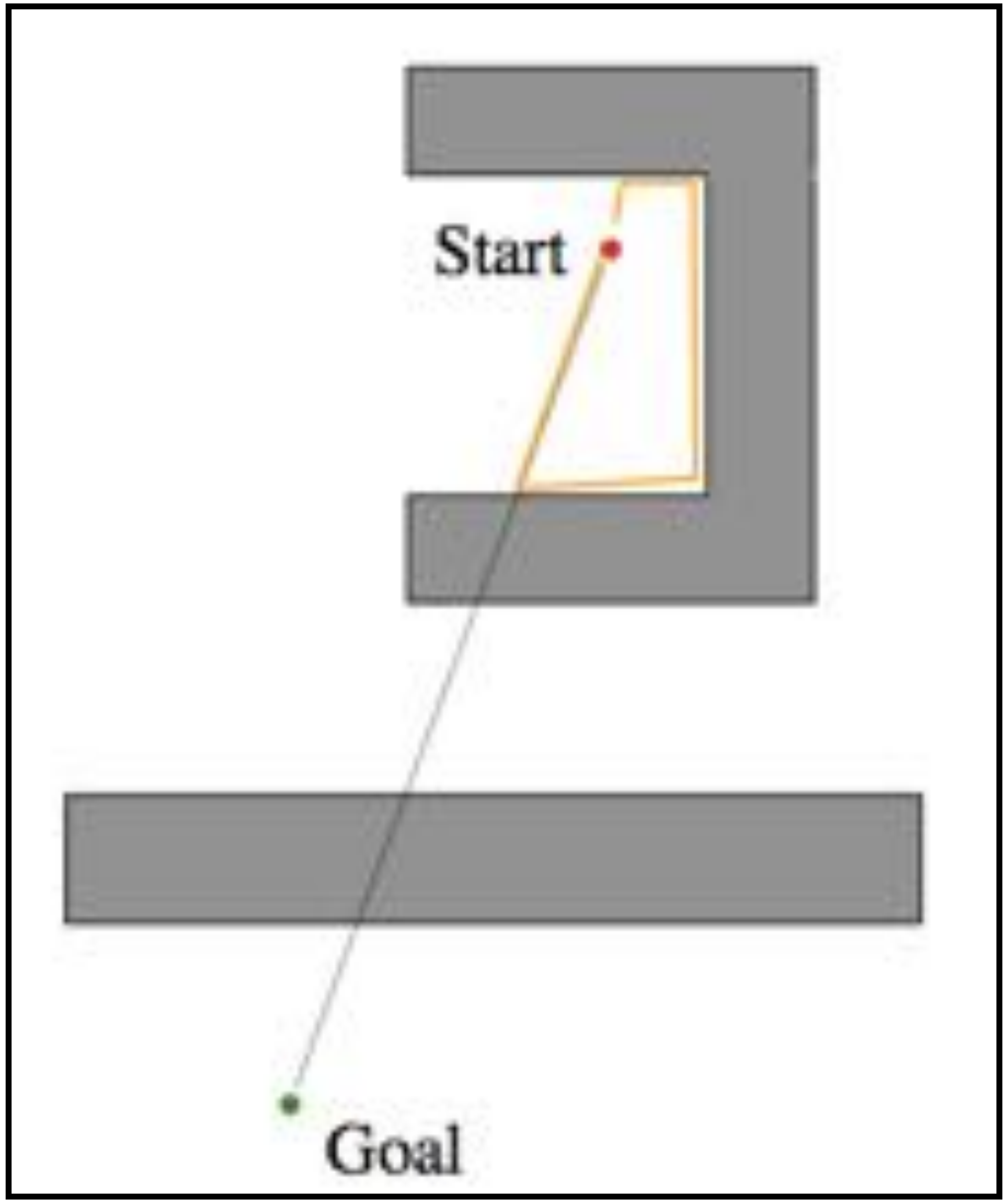
1) Head towards goal on m-line

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

3) set leave point and exit obstacle

4) continue from (1)

Bug 2



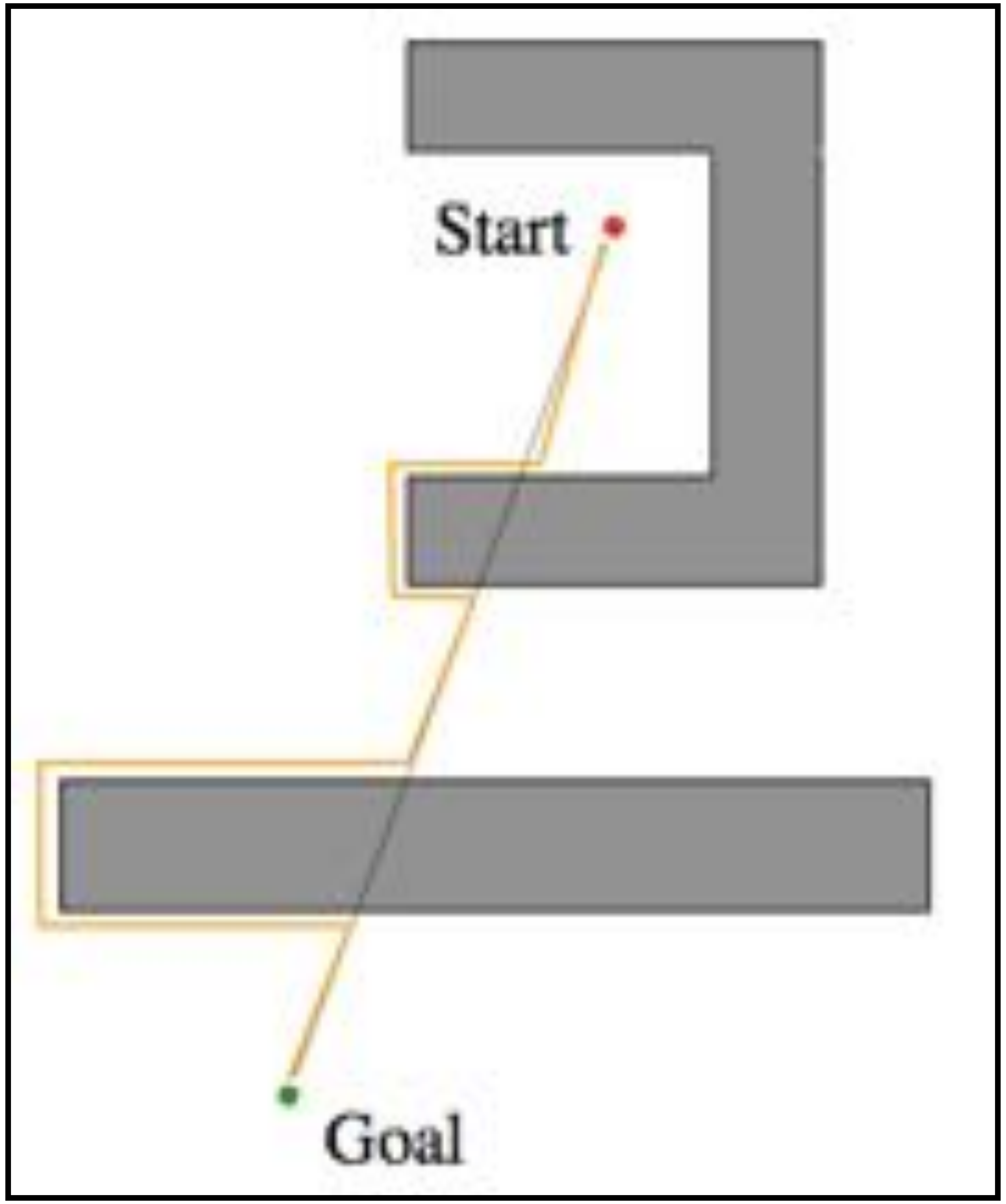
1) Head towards goal on m-line

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



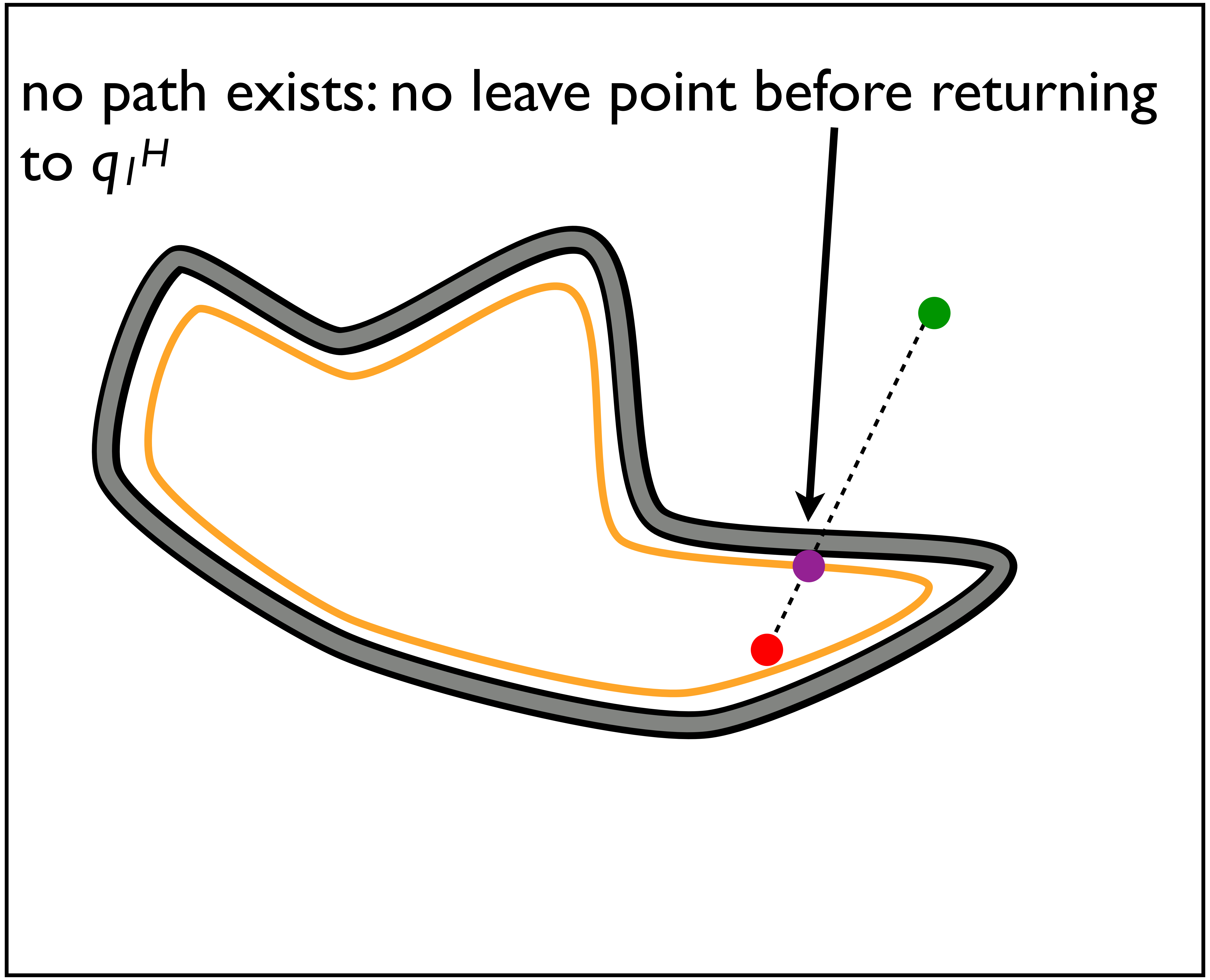
1) Head towards goal on m-line

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

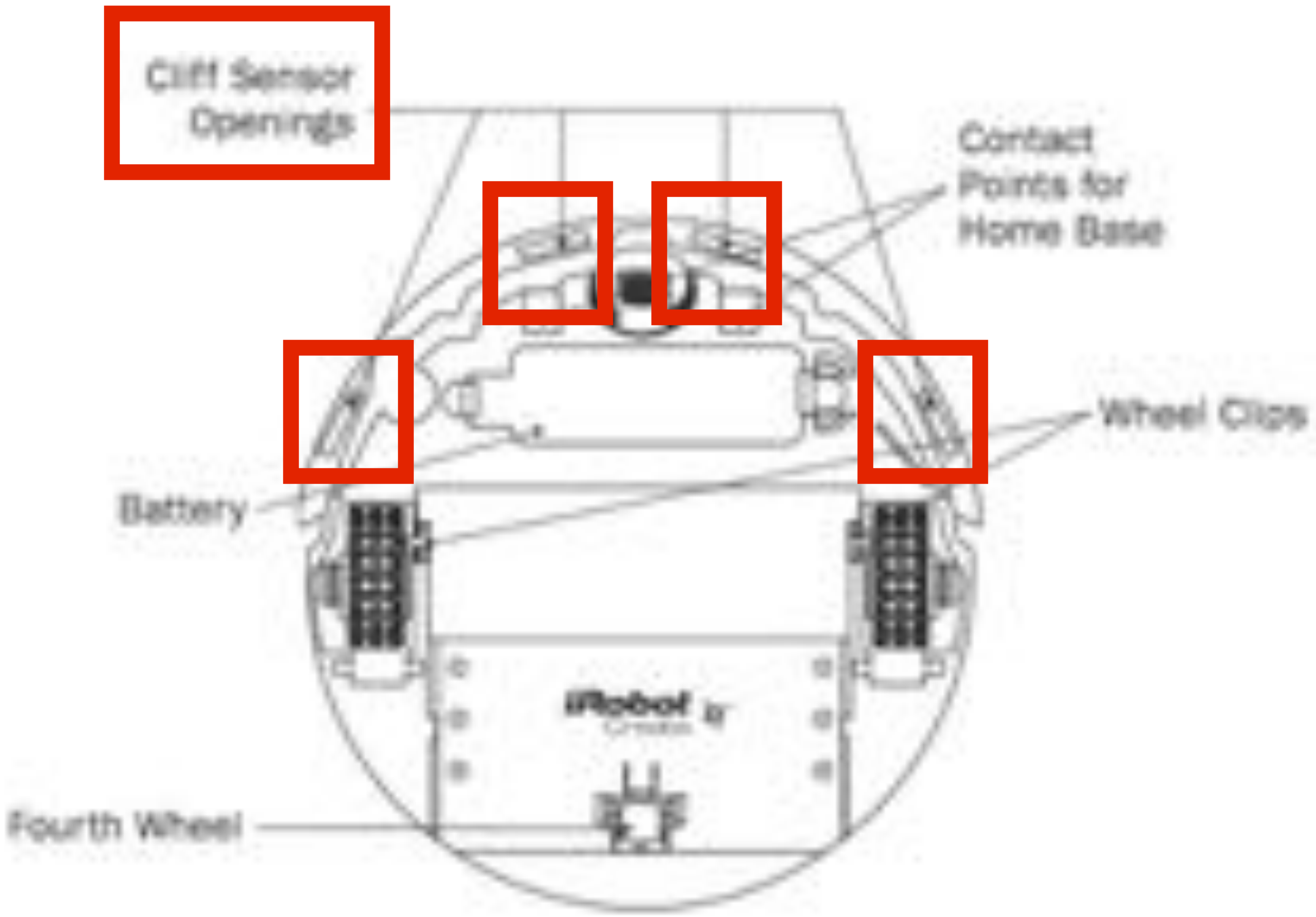
3) **if not i^{th} hit point,** set leave pt. and exit

4) continue from (1)

Slide borrowed from Michigan Robotics autorob.org

Bug 2 in action

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



Kayle Gishen

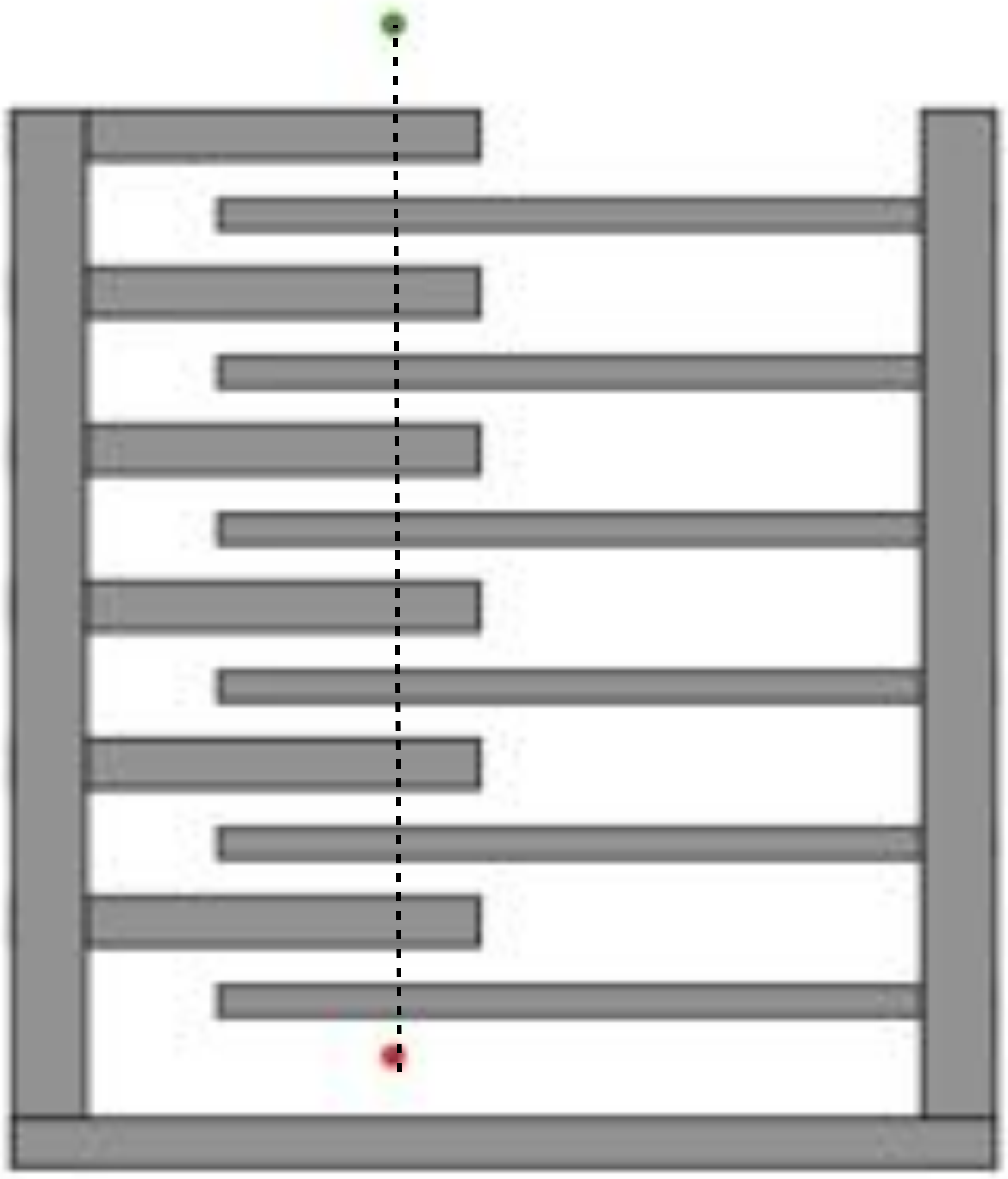
Is Bug2 better than Bug1?

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!



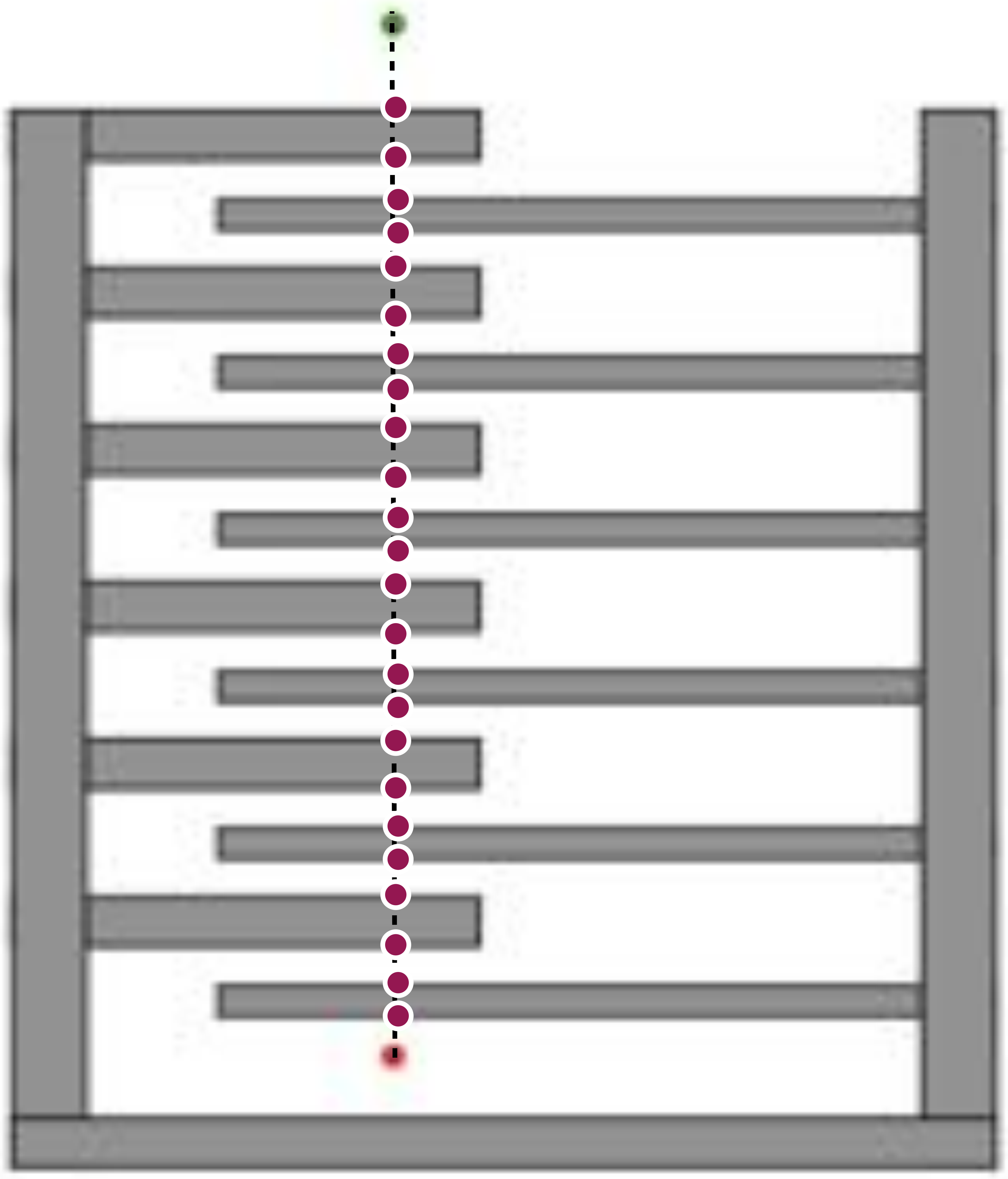
Bug 2: Search Bounds

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]



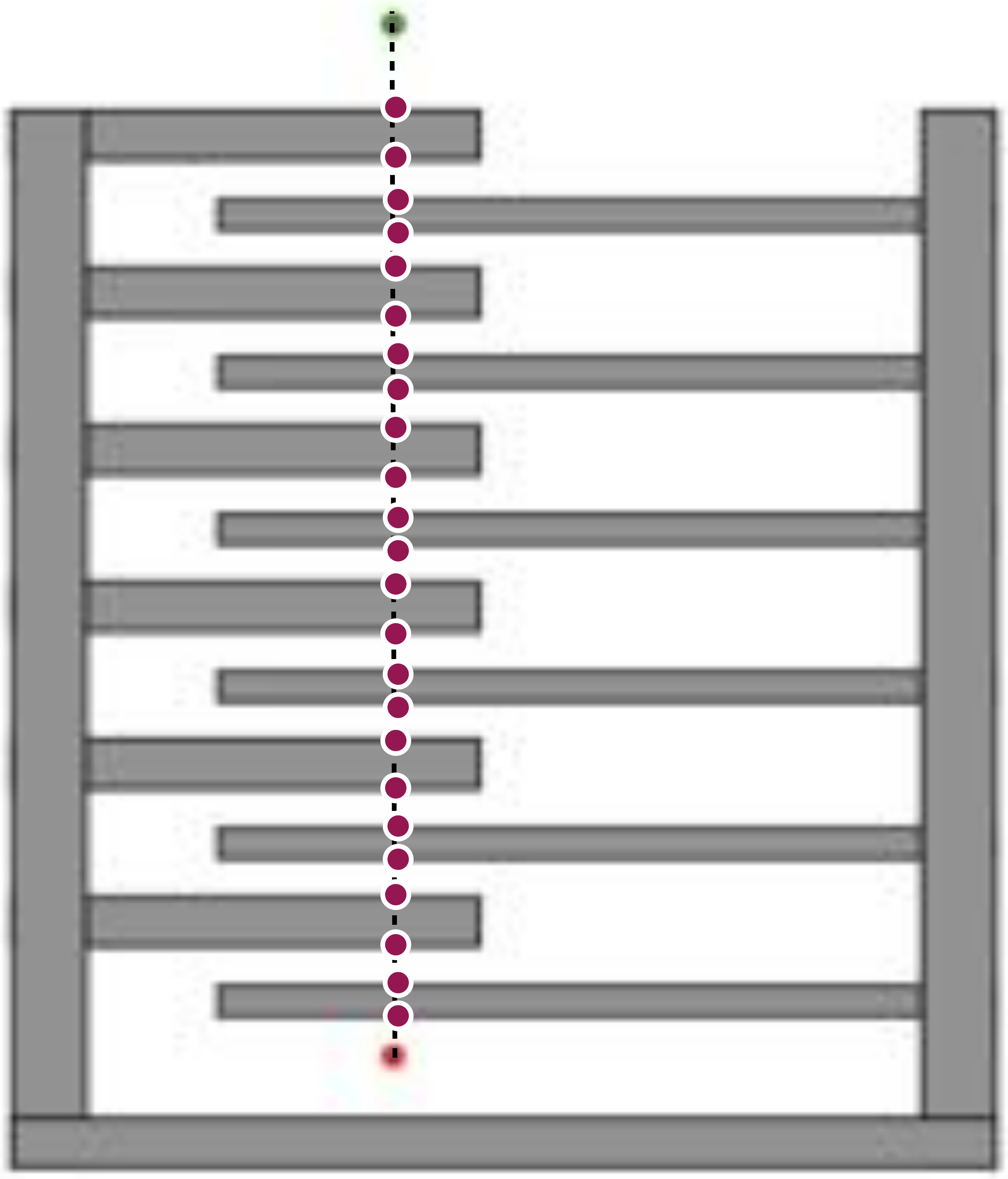
Bug 2: Search Bounds

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?

Bug 2: Search Bounds

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

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

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

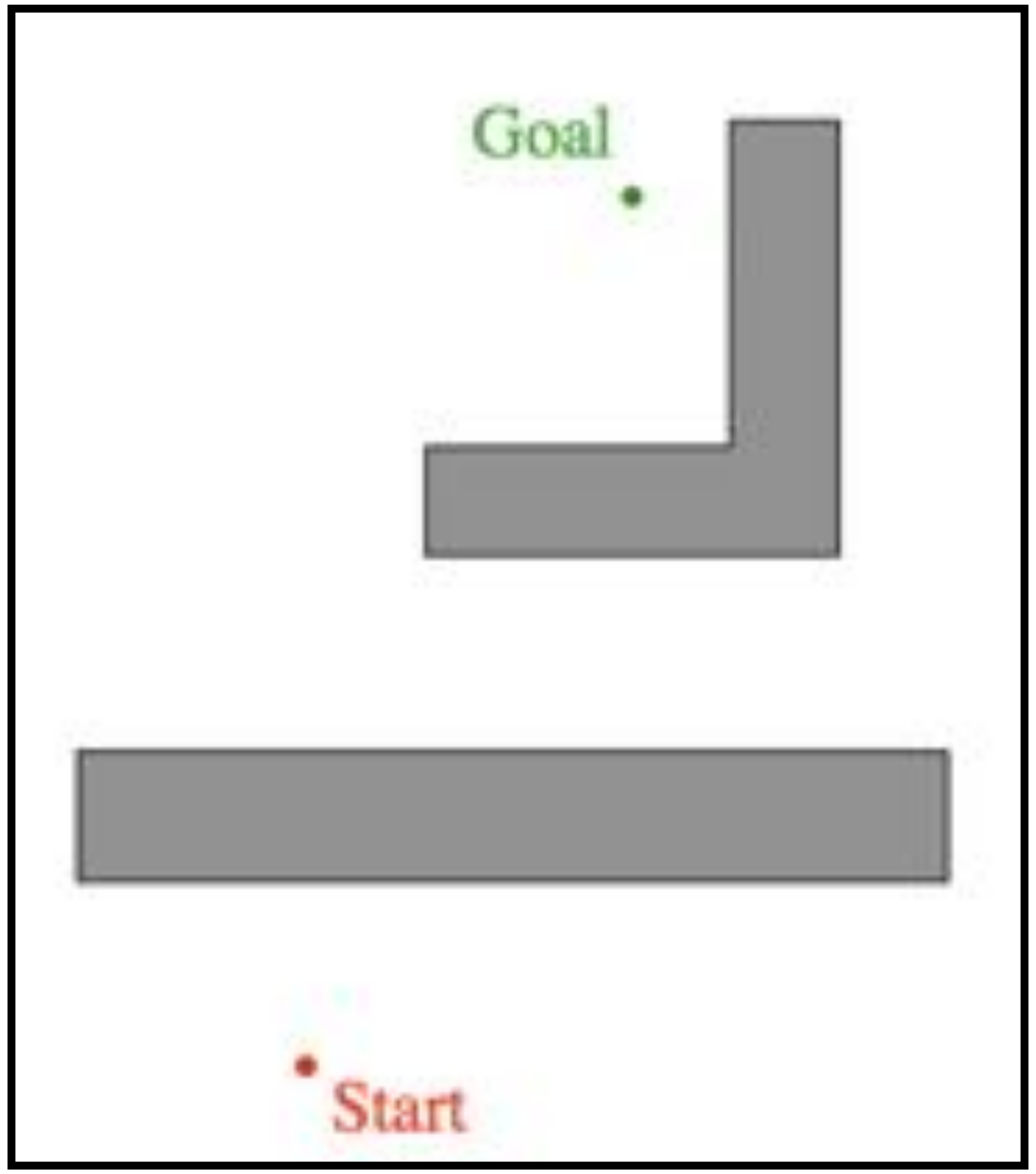
Worst case: $D + \sum_i (n_i/2)P_i$

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



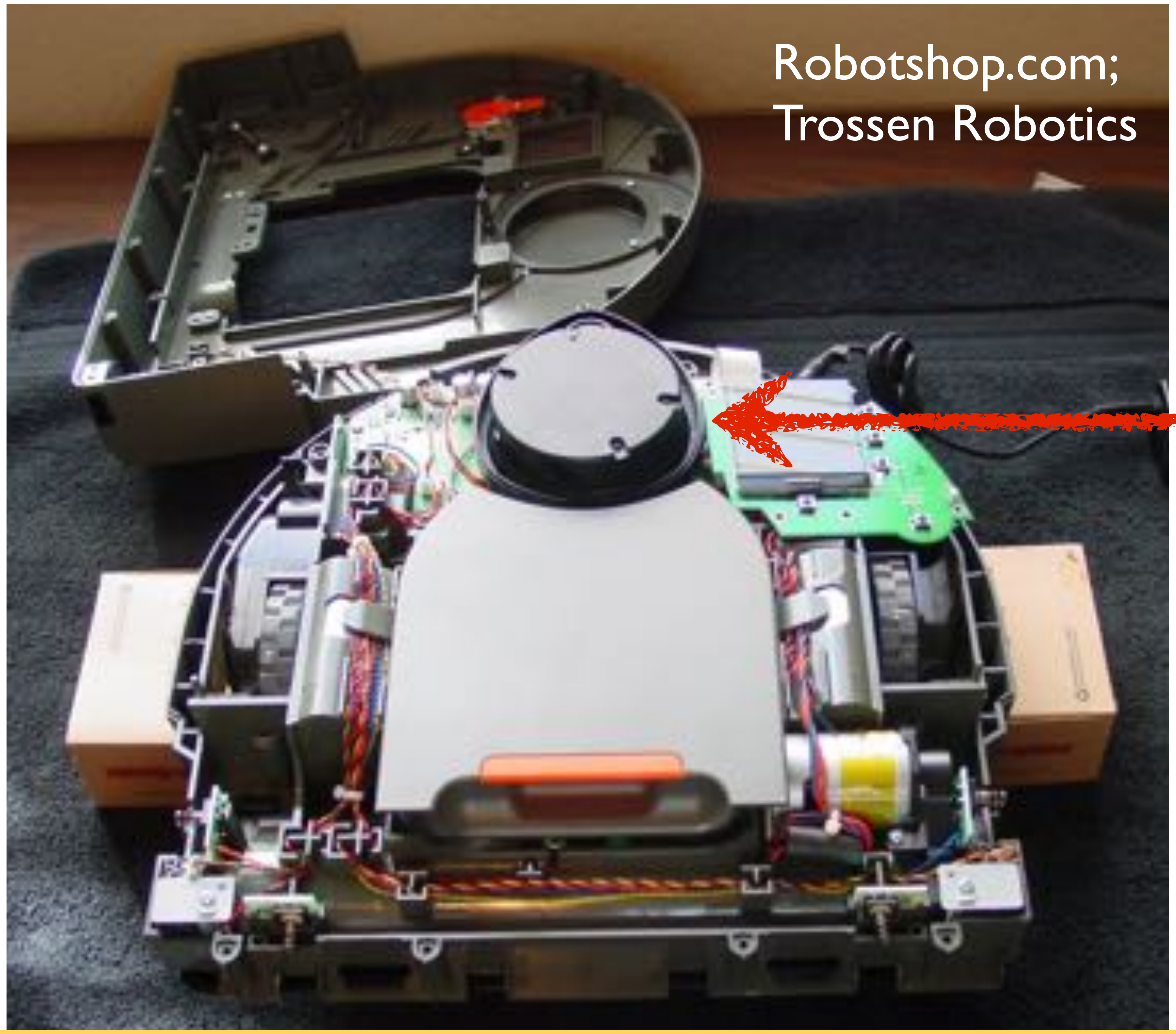
>

Laser Rangefinding
(briefly)

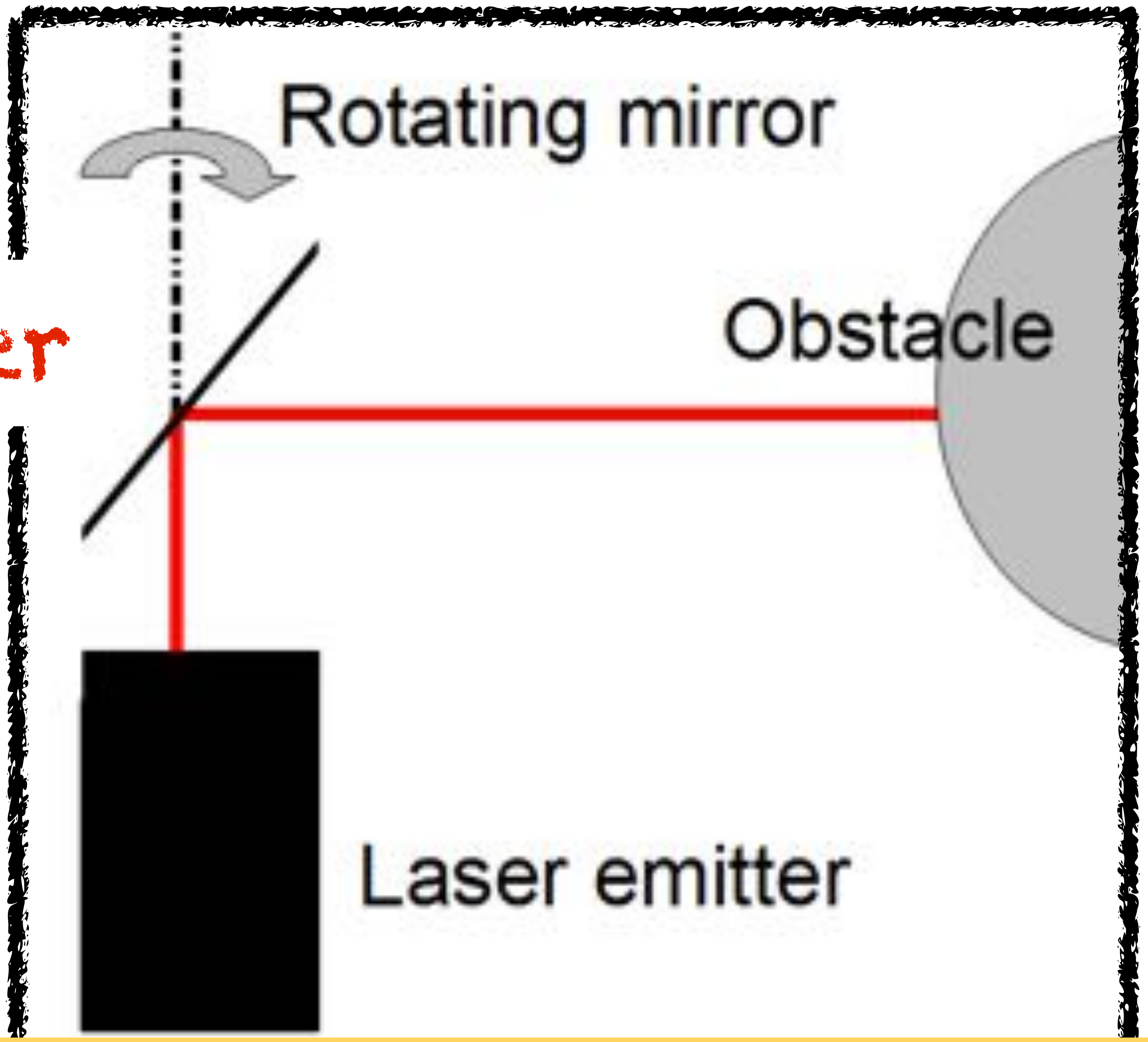
Emit laser beam in a direction

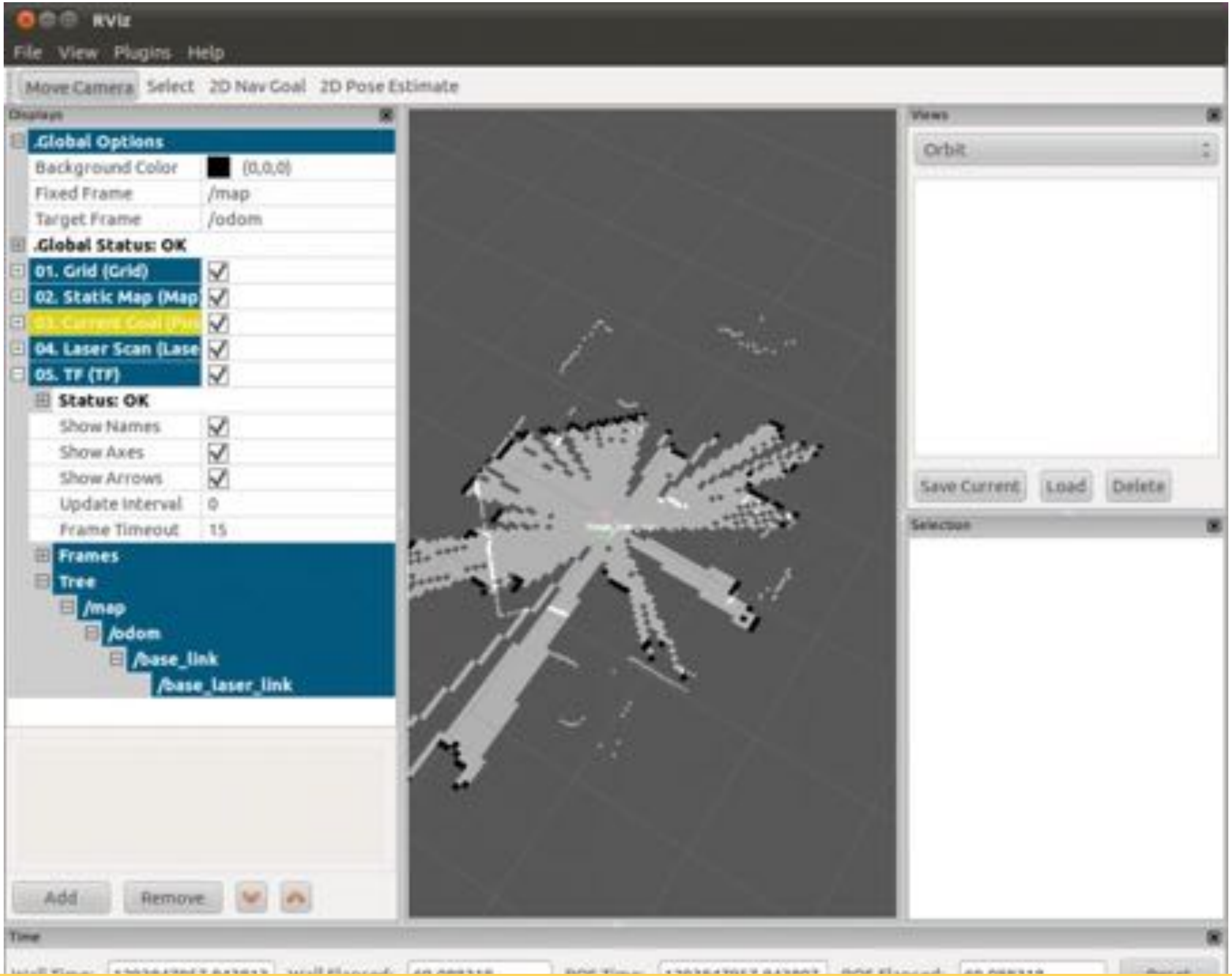
Distance to nearest object related to time from emission to sensing of beam
(assumes speed of light is known)

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



range finder





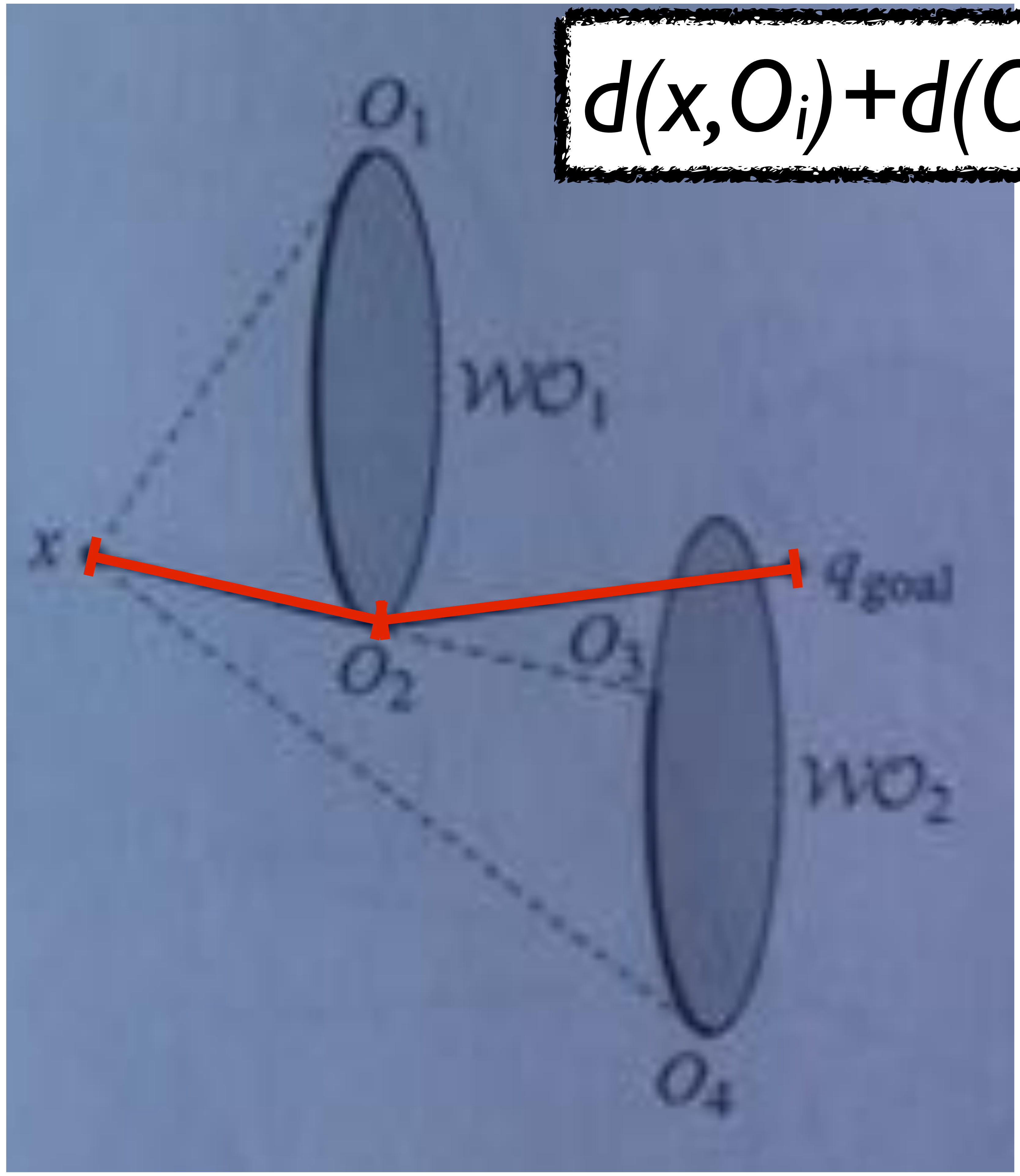
ROS LaserScan definition

LaserScan message:

```
Header header
    uint32 seq
    time stamp
    string frame_id
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```


Slide borrowed from Michigan Robotics autorob.org

Tangent Bug: Heuristic Distance-to-Goal



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

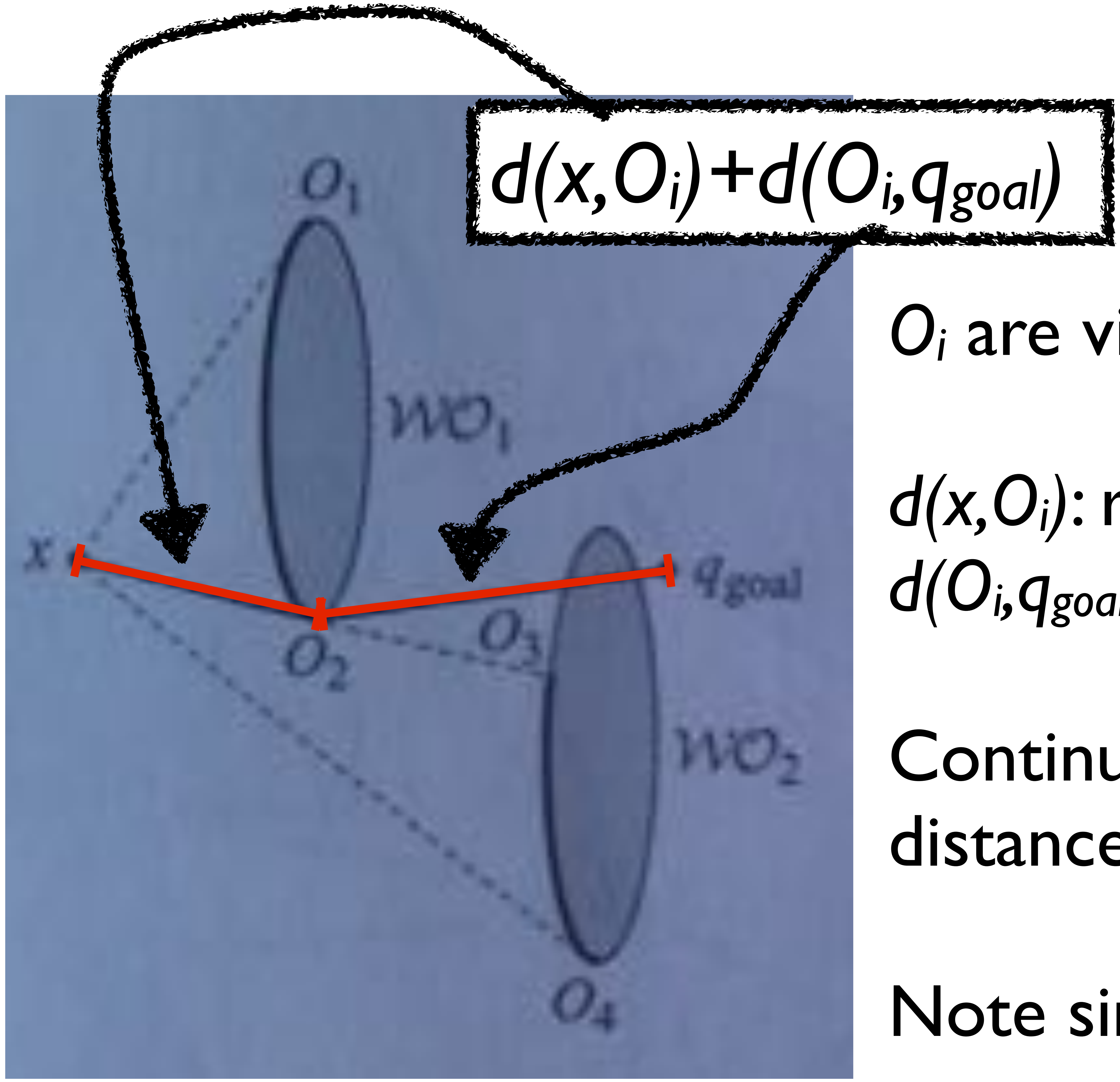
O_i are visible obstacle extents

$d(x,O_i)$: 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

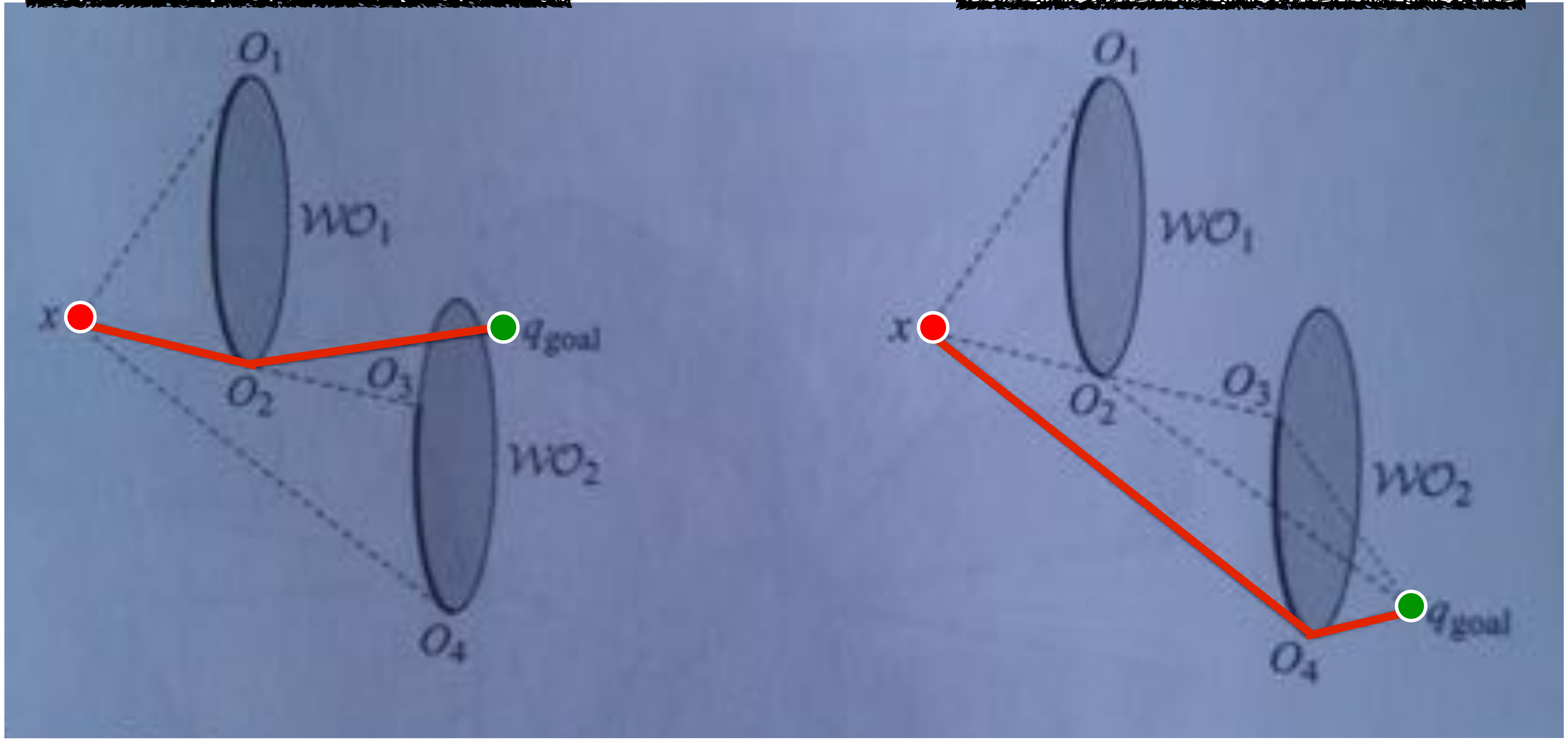
$d(x,O_i)$: robot can see
$d(O_i,q_{goal})$: best path robot cannot see

Continually move robot such that distance to goal is decreased

Note similarity to A* search heuristic

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

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



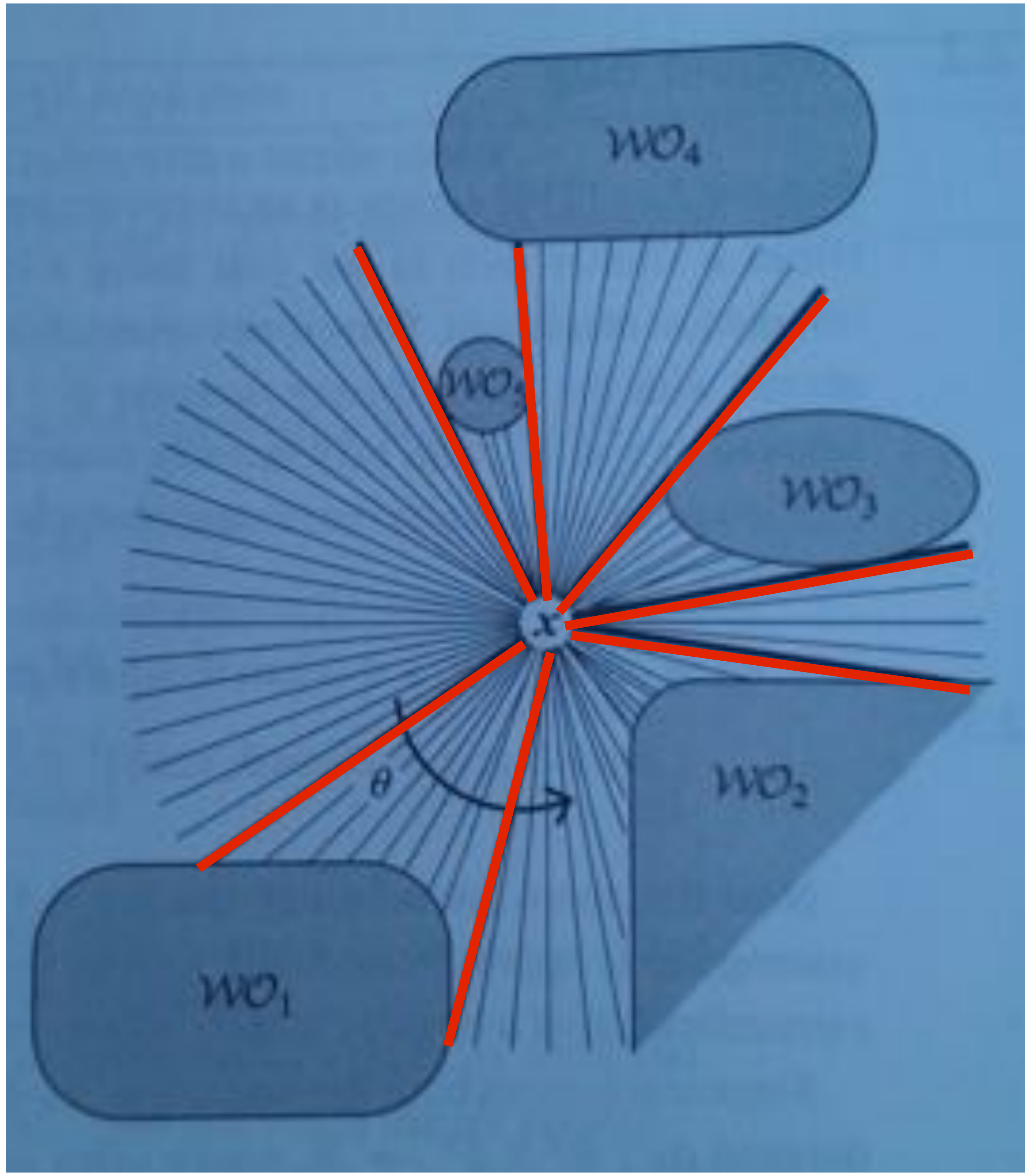
Slide borrowed from Michigan Robotics autorob.org

Range Segmentation

range scan $\rho(x,\Theta)$: 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

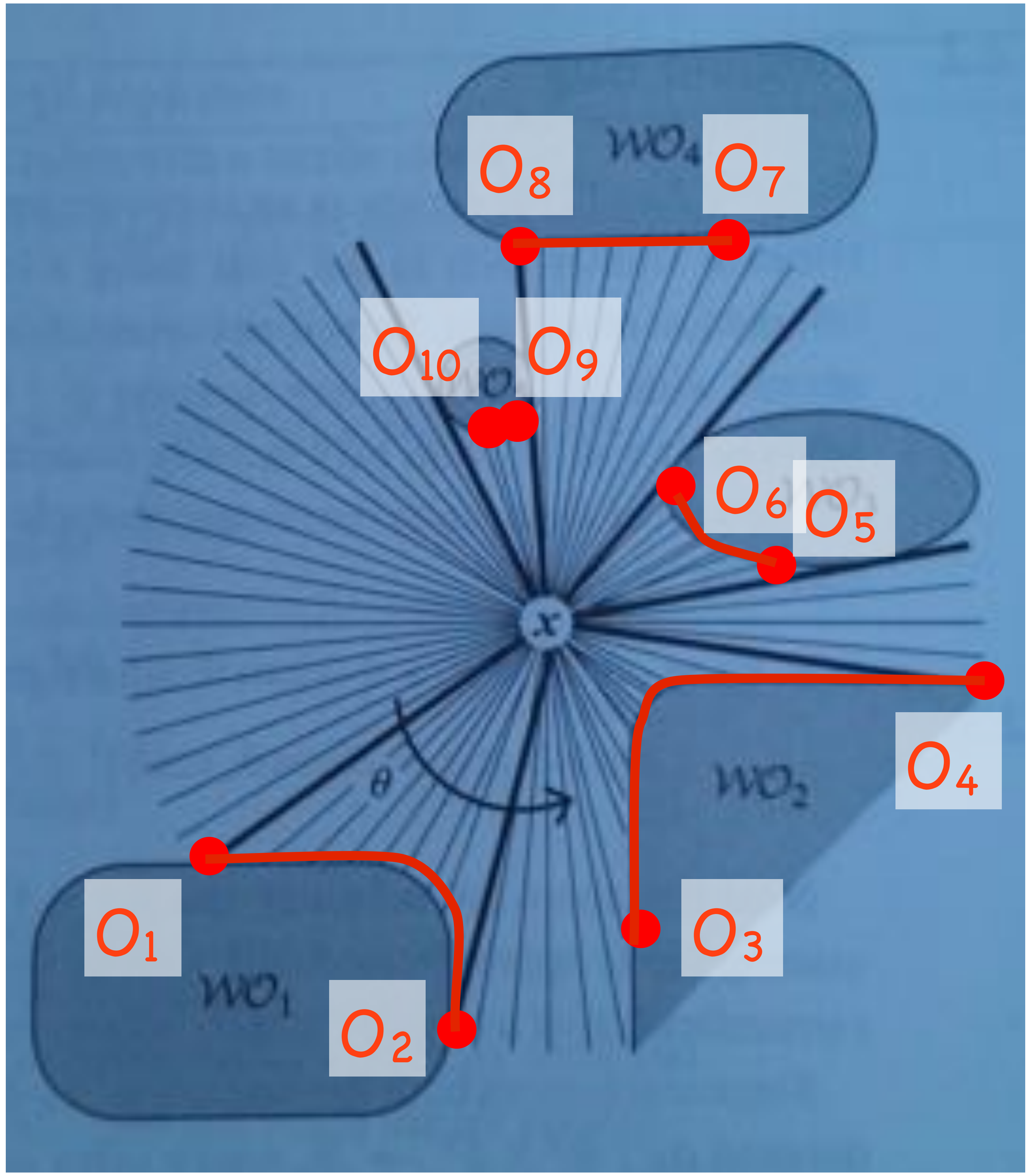


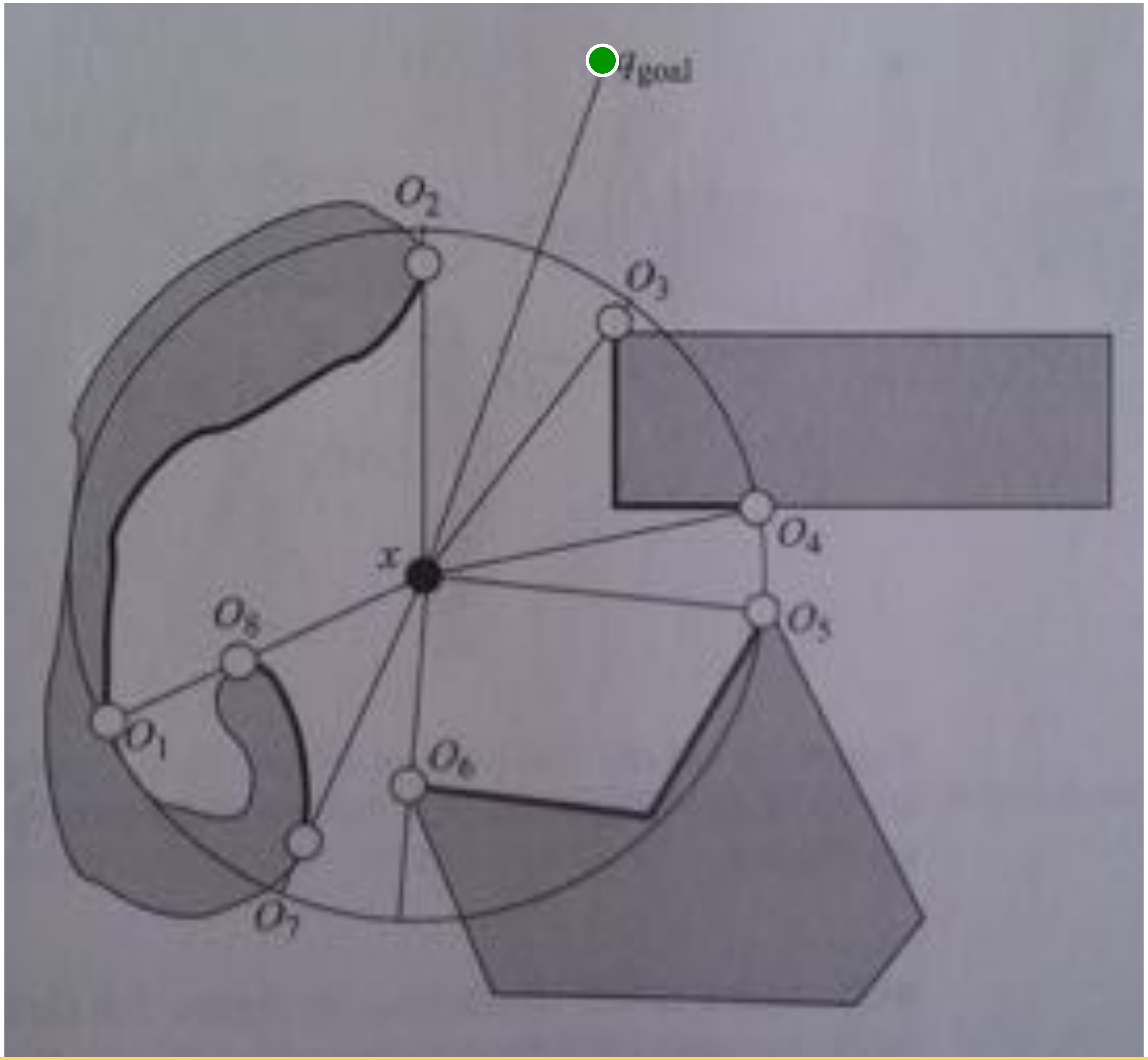
Range Segmentation

range scan $\rho(x,\Theta)$: 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

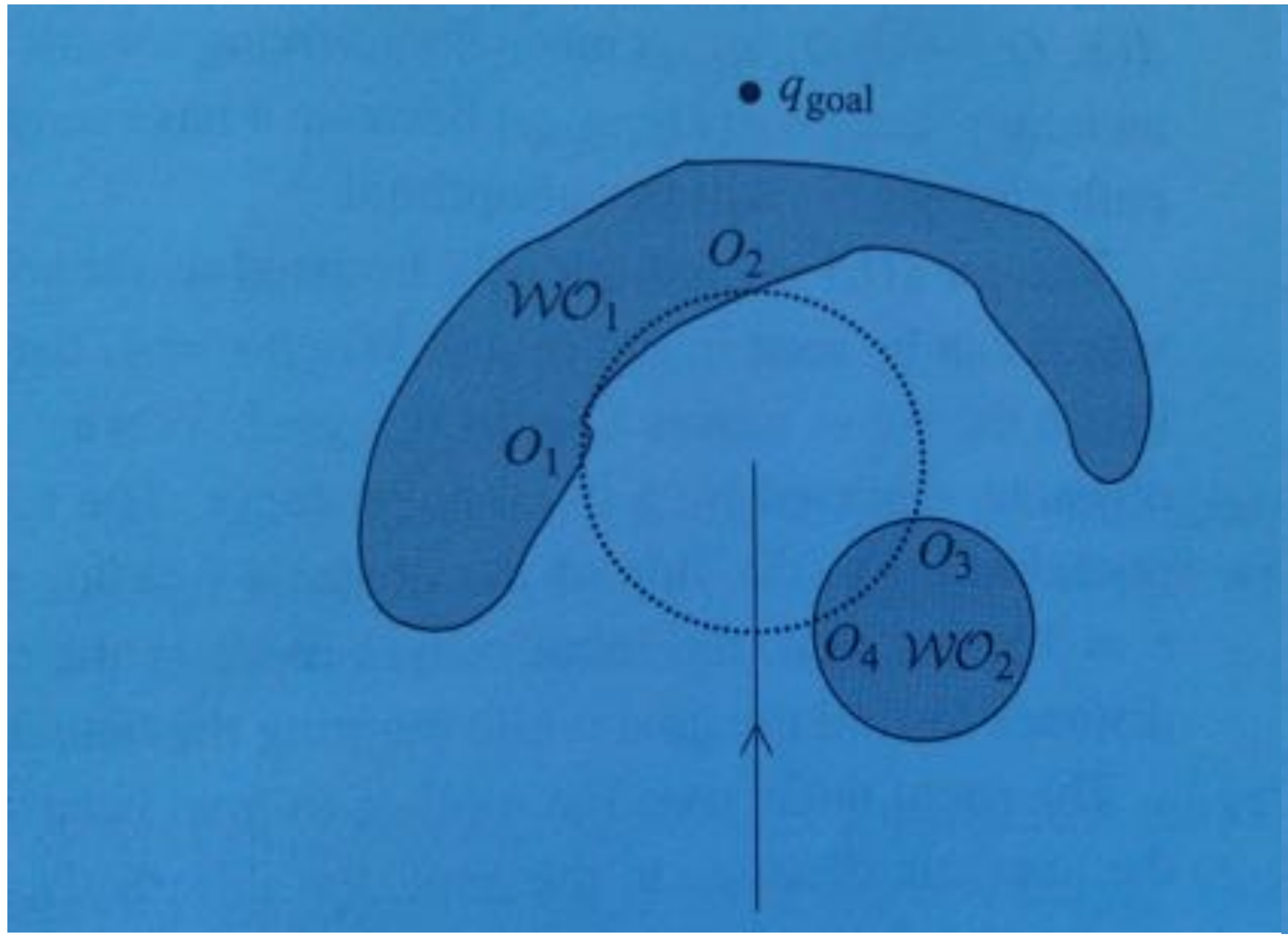




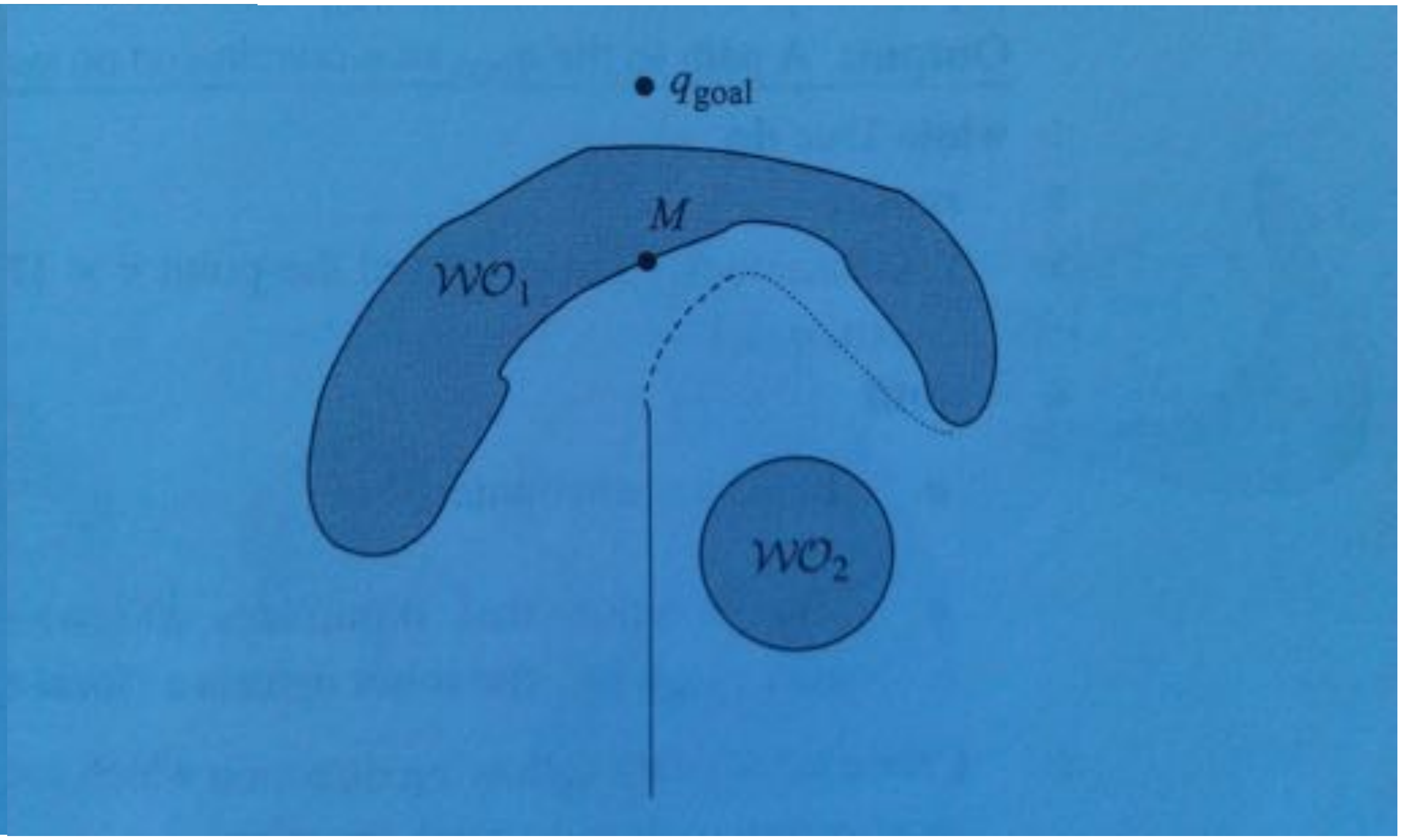
q_{goal}
O_2
O_3
O_4
x
O_8
O_5
O_1
O_6
O_7

Tangent Bug Behaviors

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



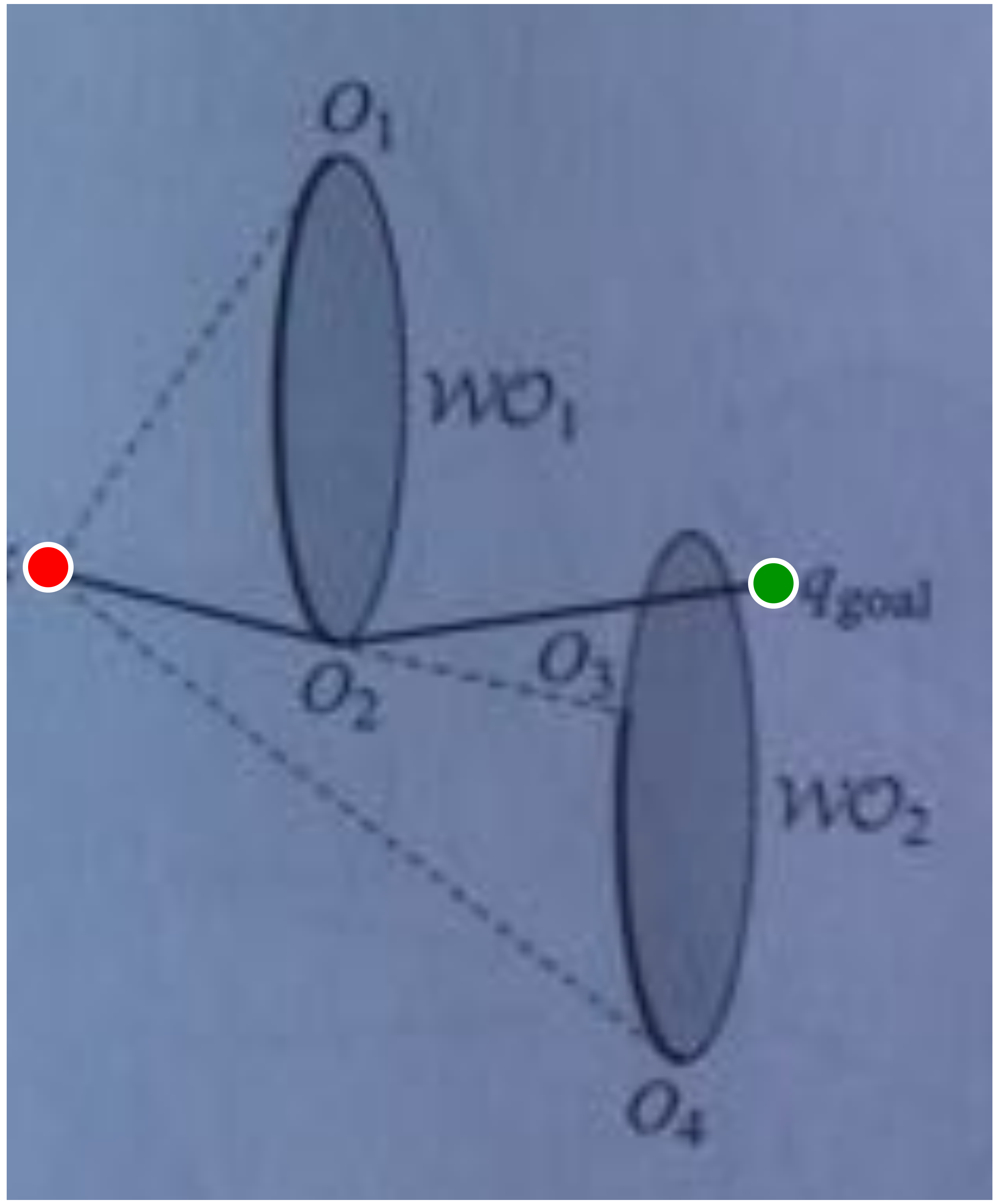
motion-to-goal



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 $d(q_{goal}$, {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**)

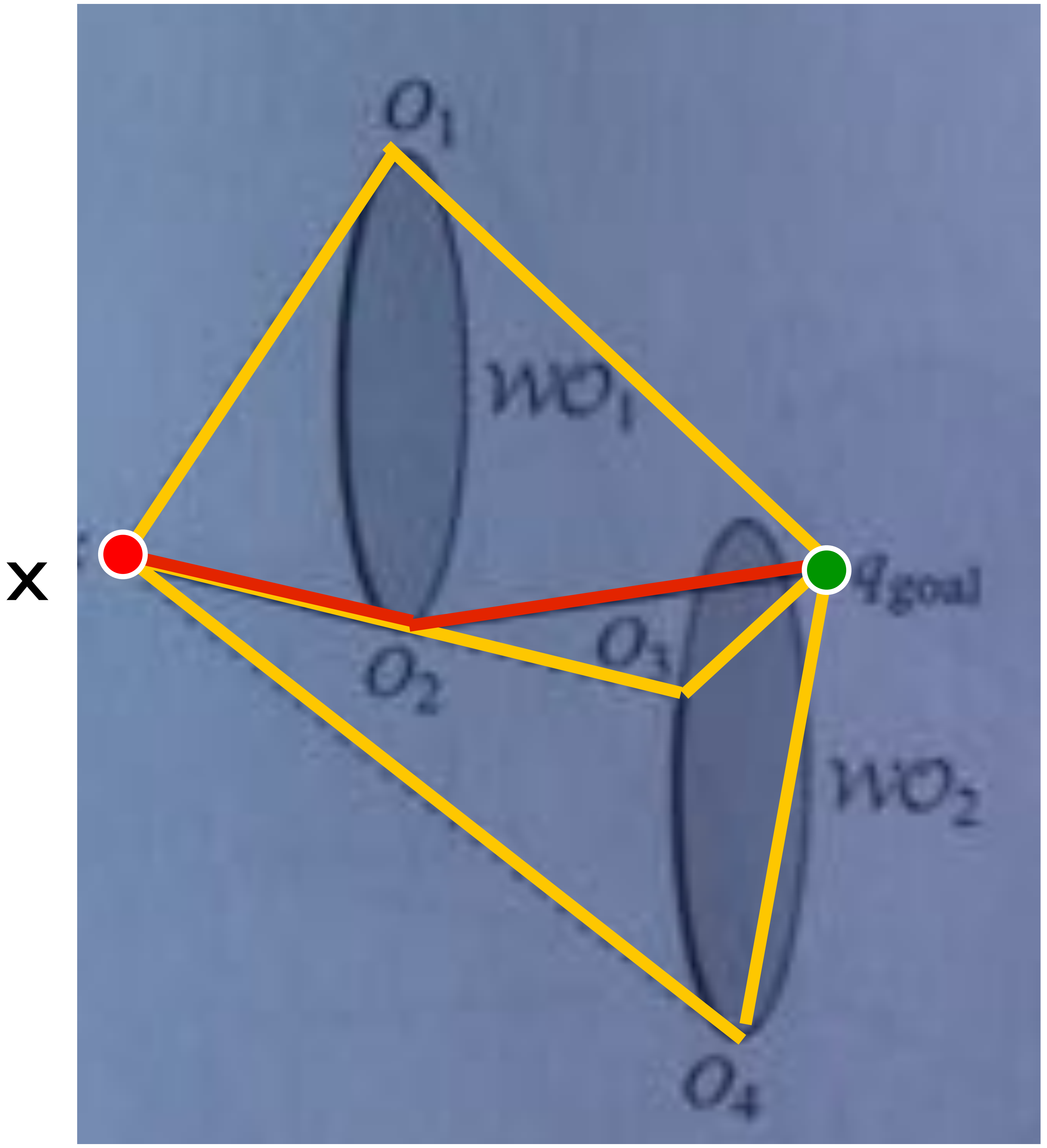
   $d_{reach} < d_{follow}$,

   (**cleared obstacle or local minima**)

3) continue from (1)

Tangent Bug

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



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 $d(q_{goal}, \{\text{visible } O_i\})$

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

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

b) until

goal reached, (**success**)

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

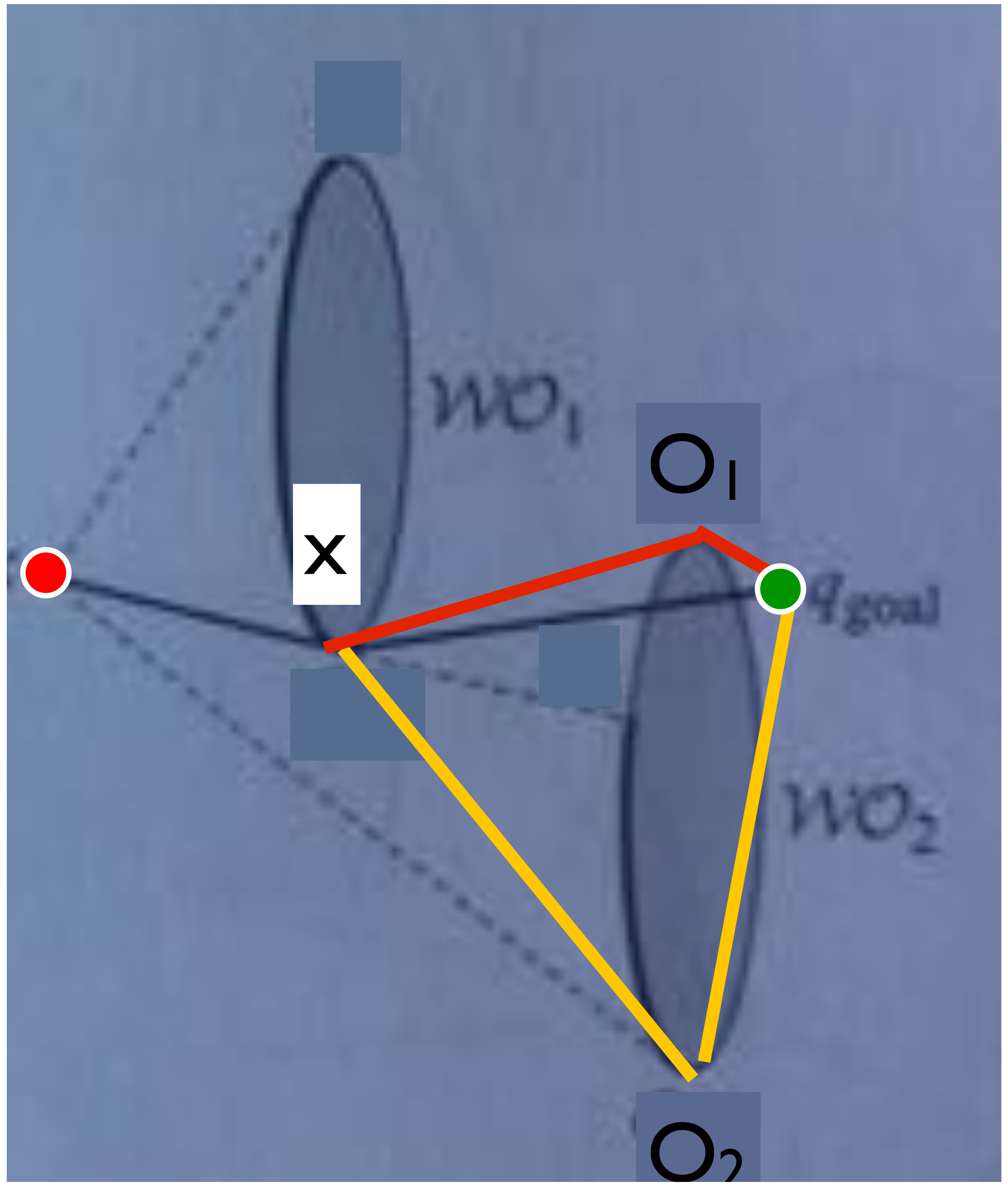
$d_{reach} < d_{follow}$,

(**cleared obstacle or local minima**)

3) continue from (1)

Tangent Bug

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



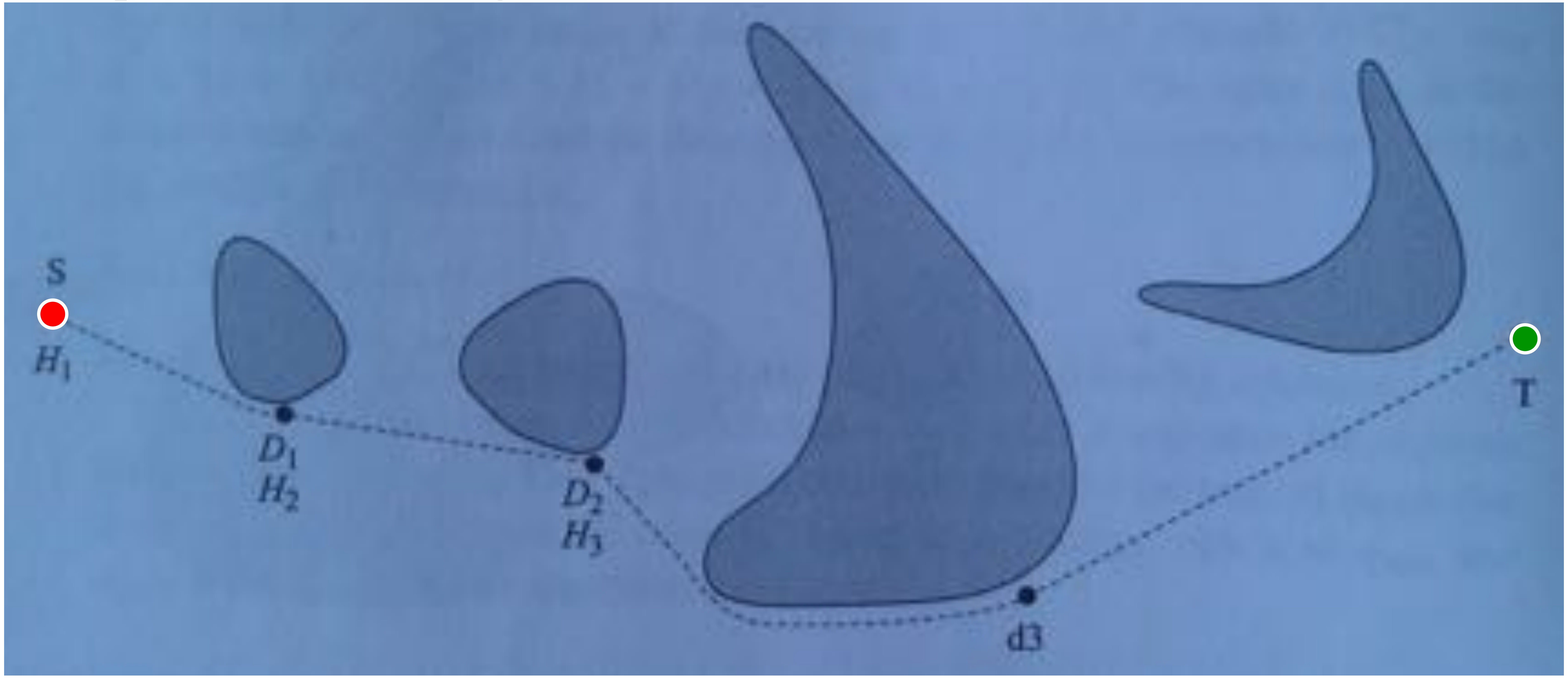
min $G(x)$ in red, others in yellow

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**)

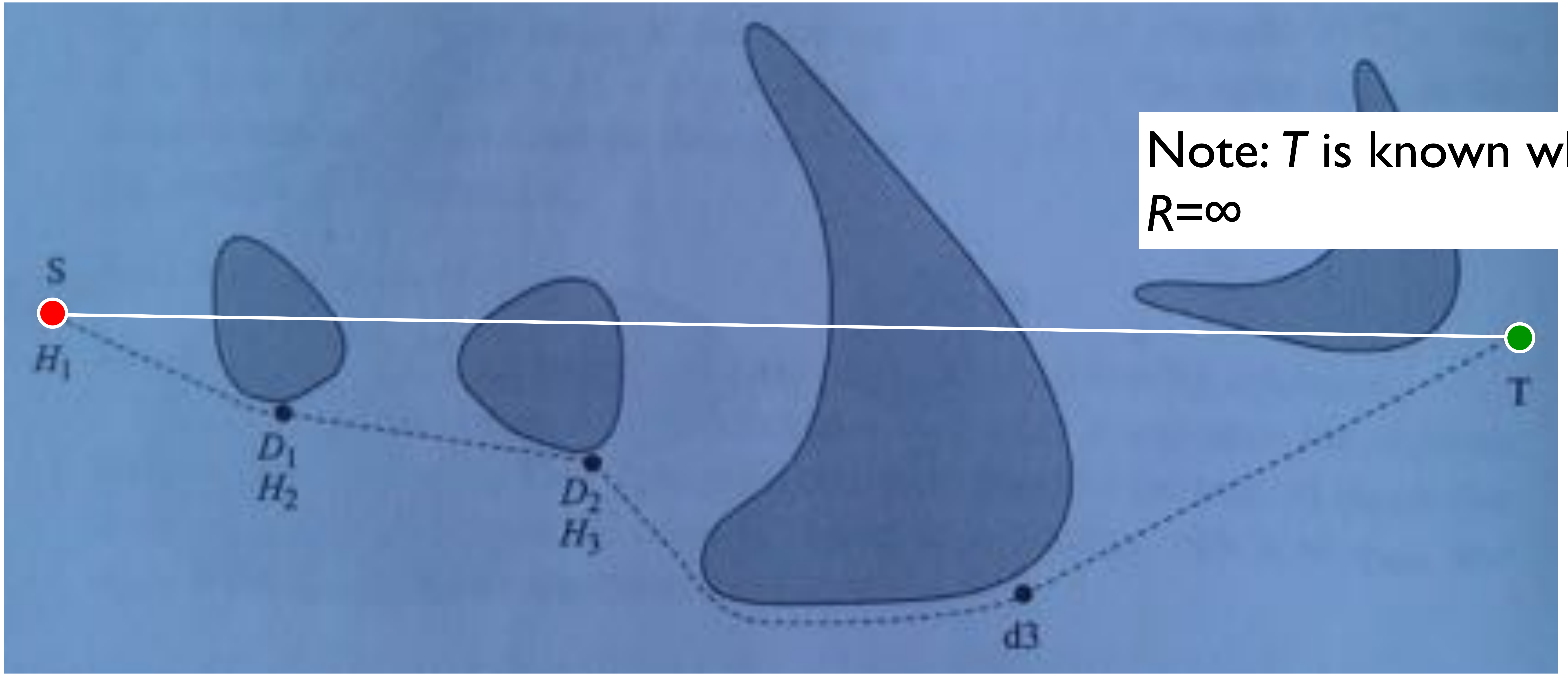
3) continue from (1)

Example: range $R=\infty$



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

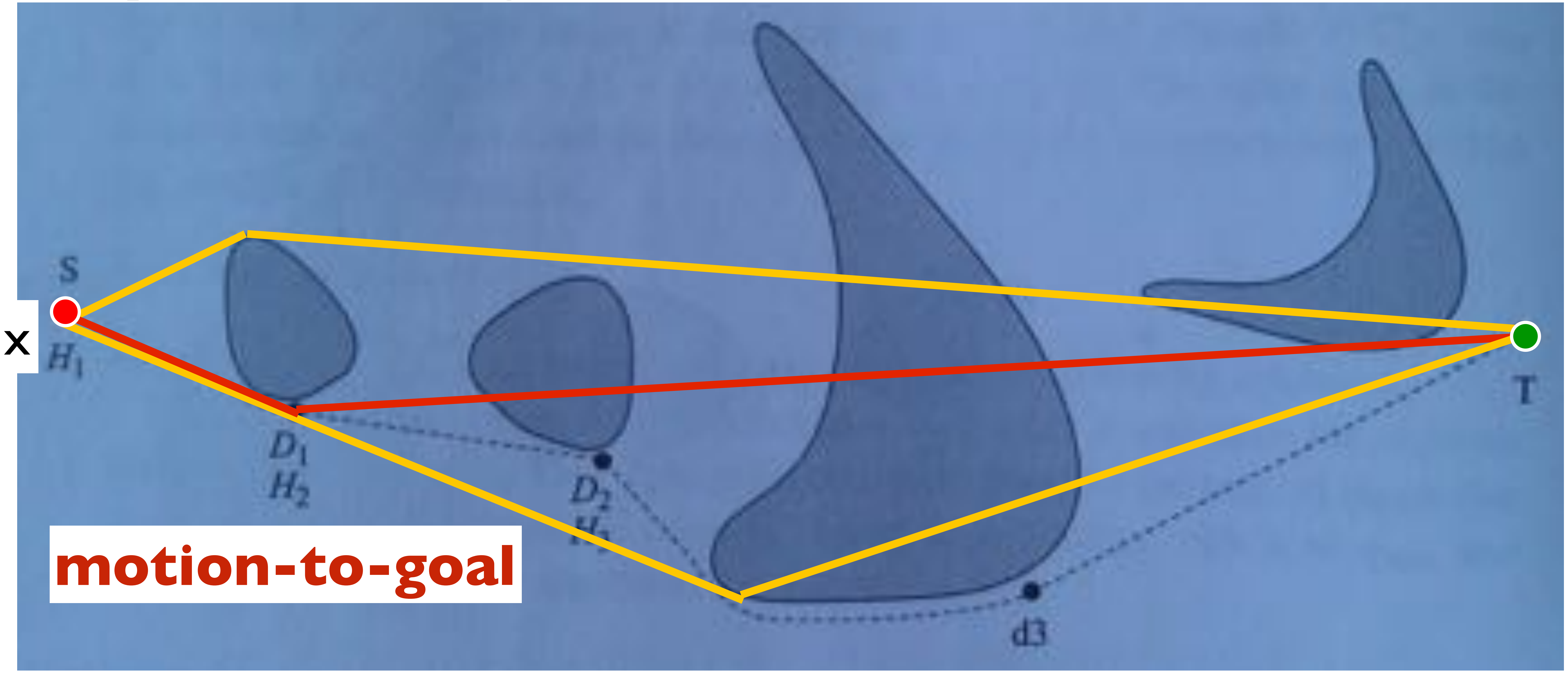
Example: range $R=\infty$



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

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

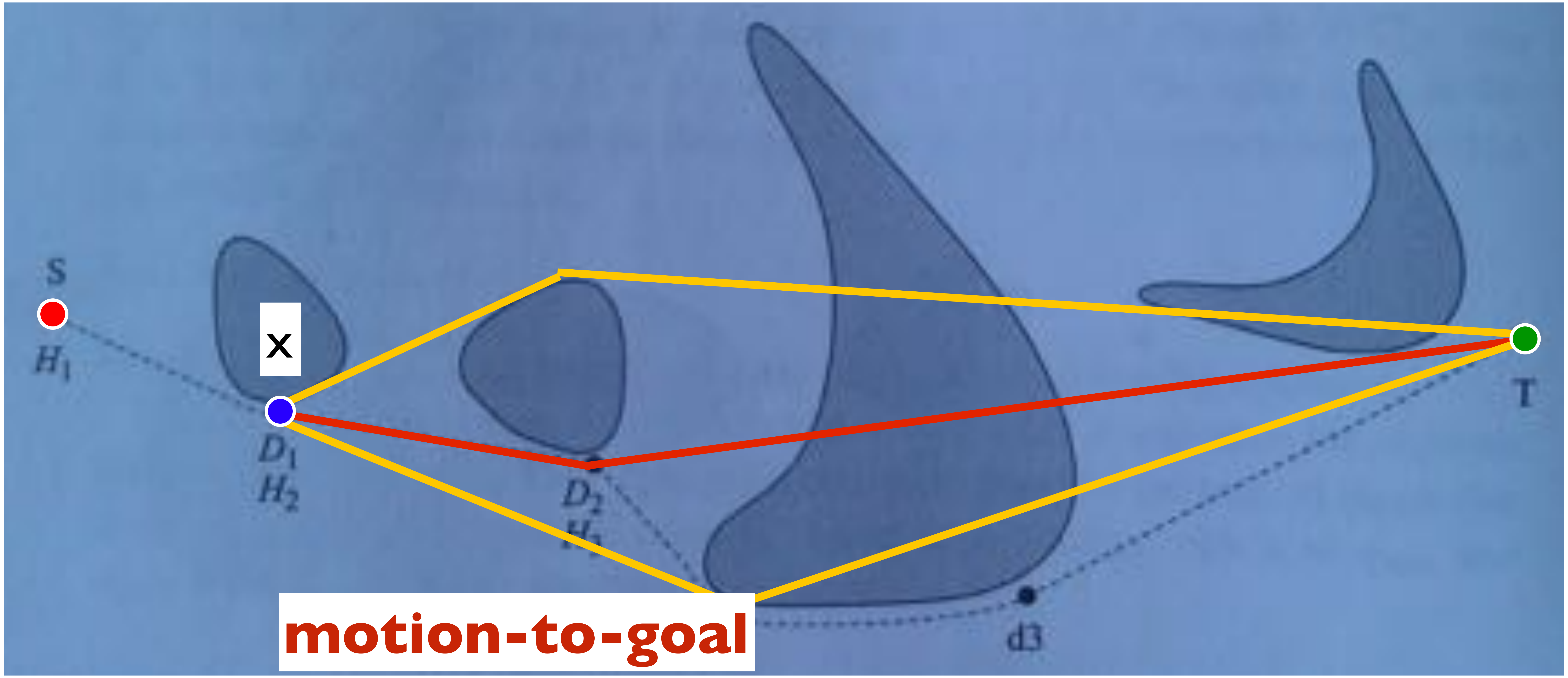
Example: range $R=\infty$



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

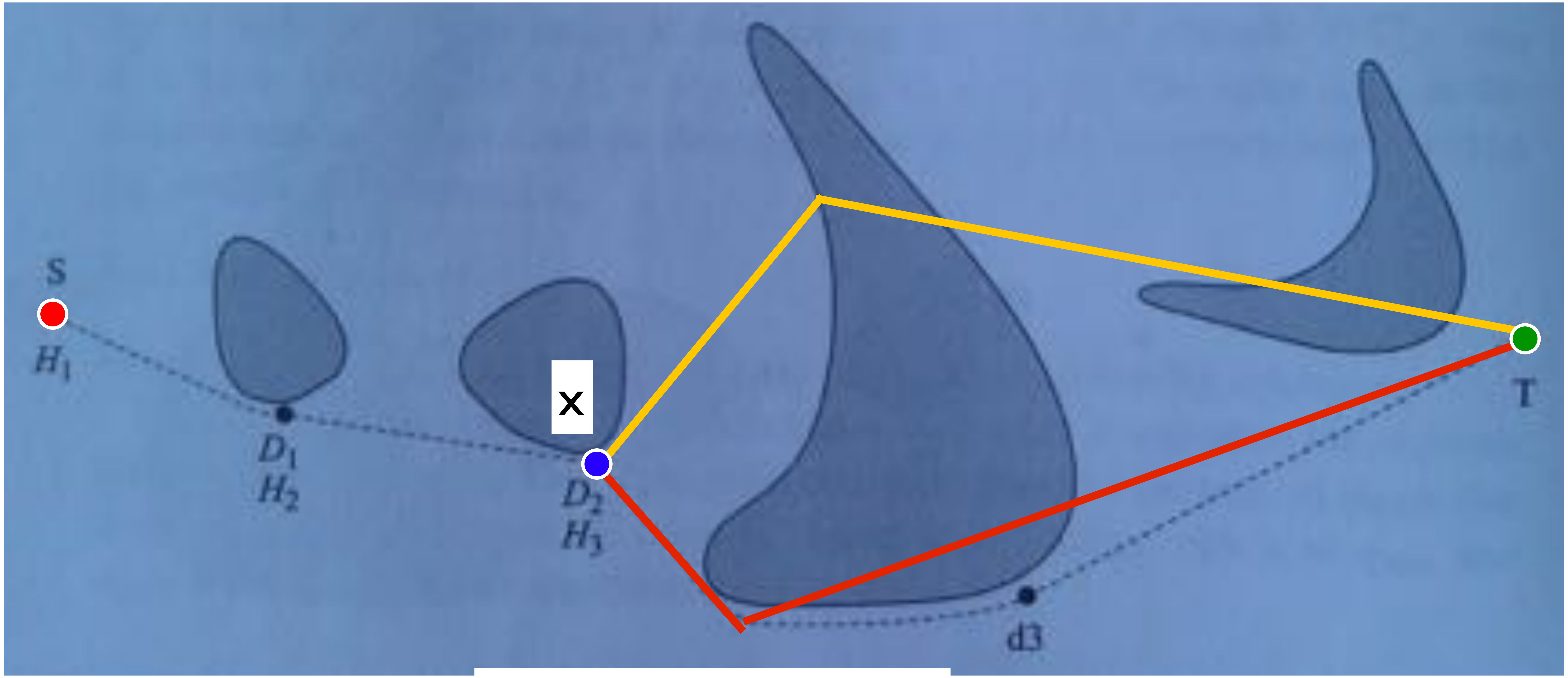
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Example: range $R=\infty$



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

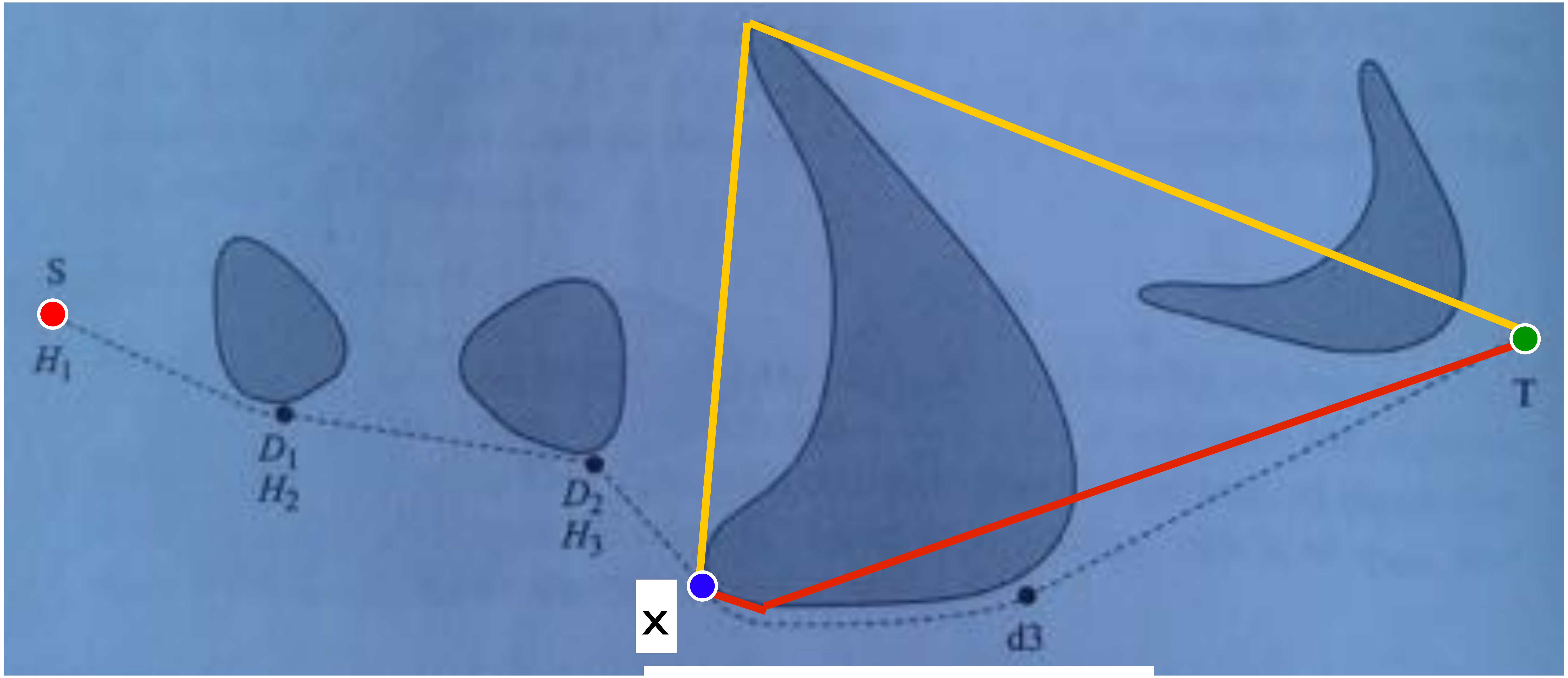
Example: range $R=\infty$



motion-to-goal

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

Example: range $R=\infty$



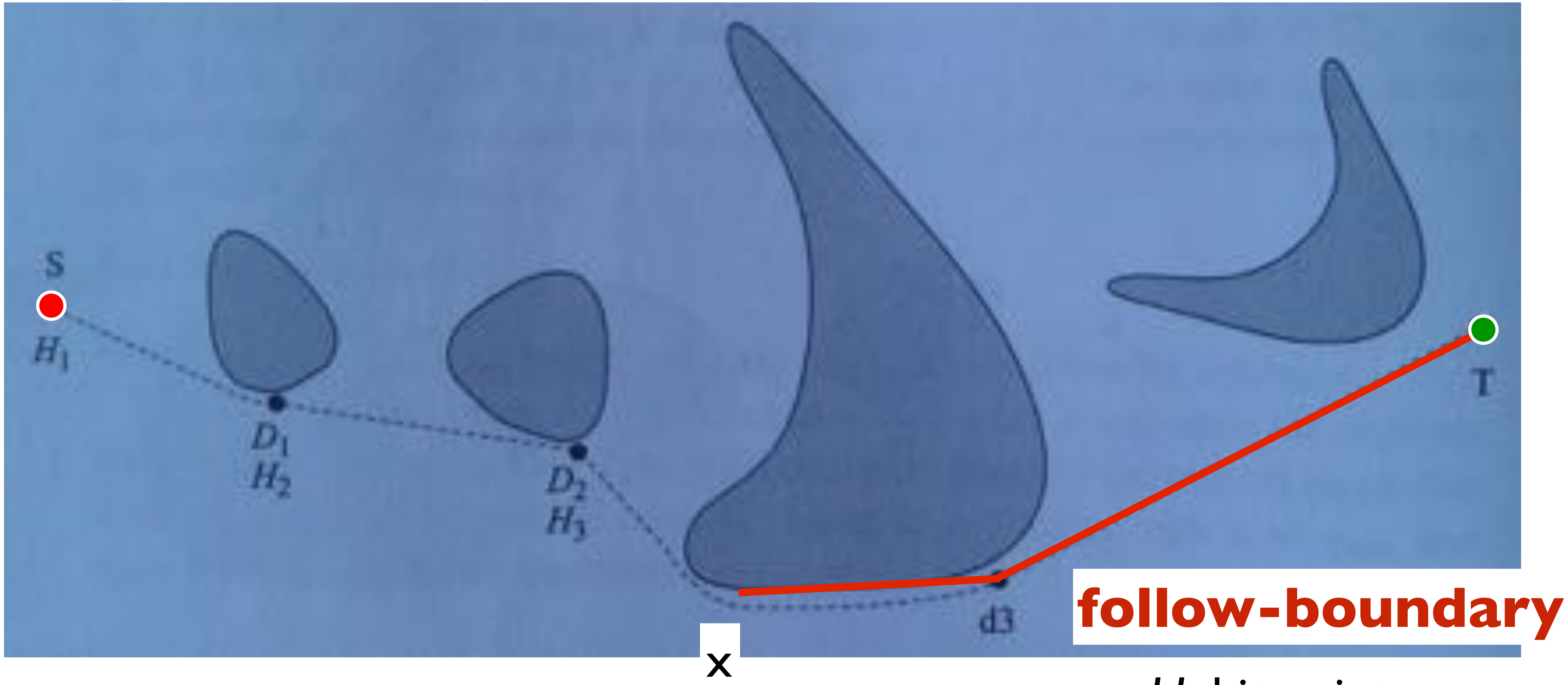
follow-boundary

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$(WO_j))$

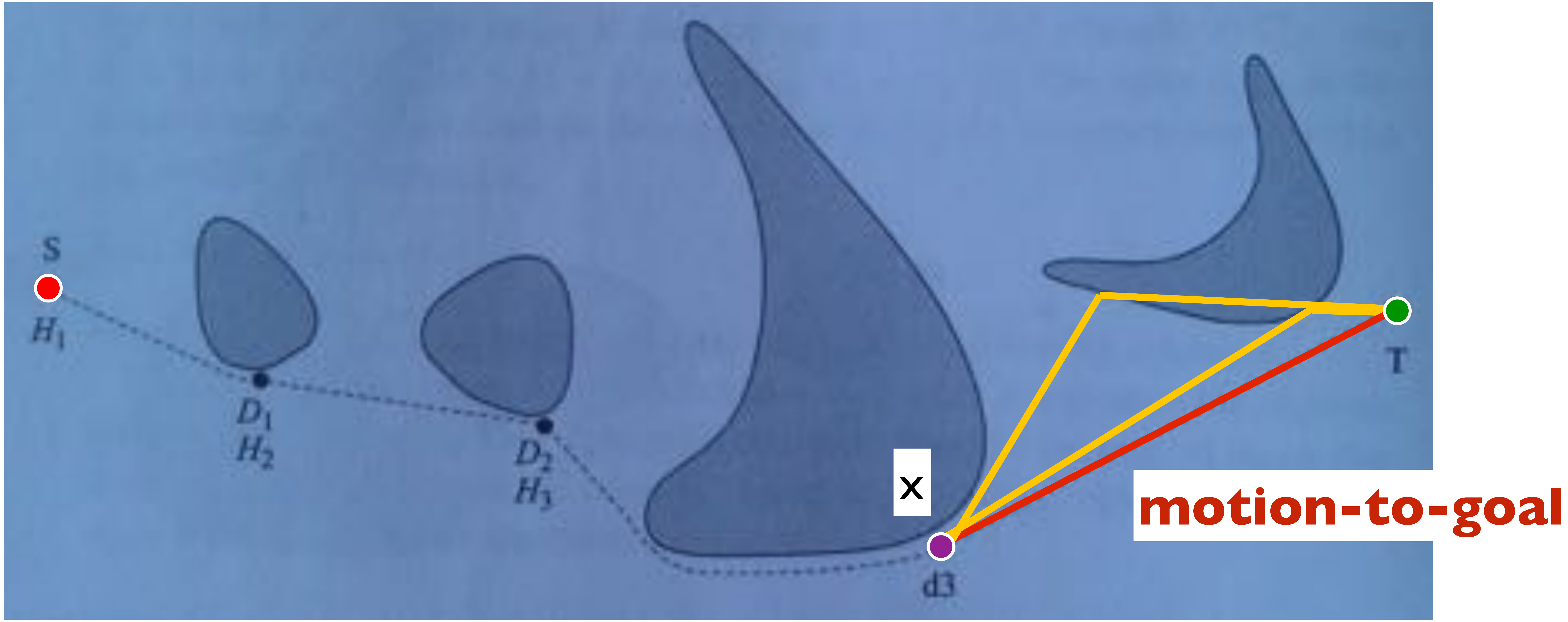
Example: range $R=\infty$



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

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

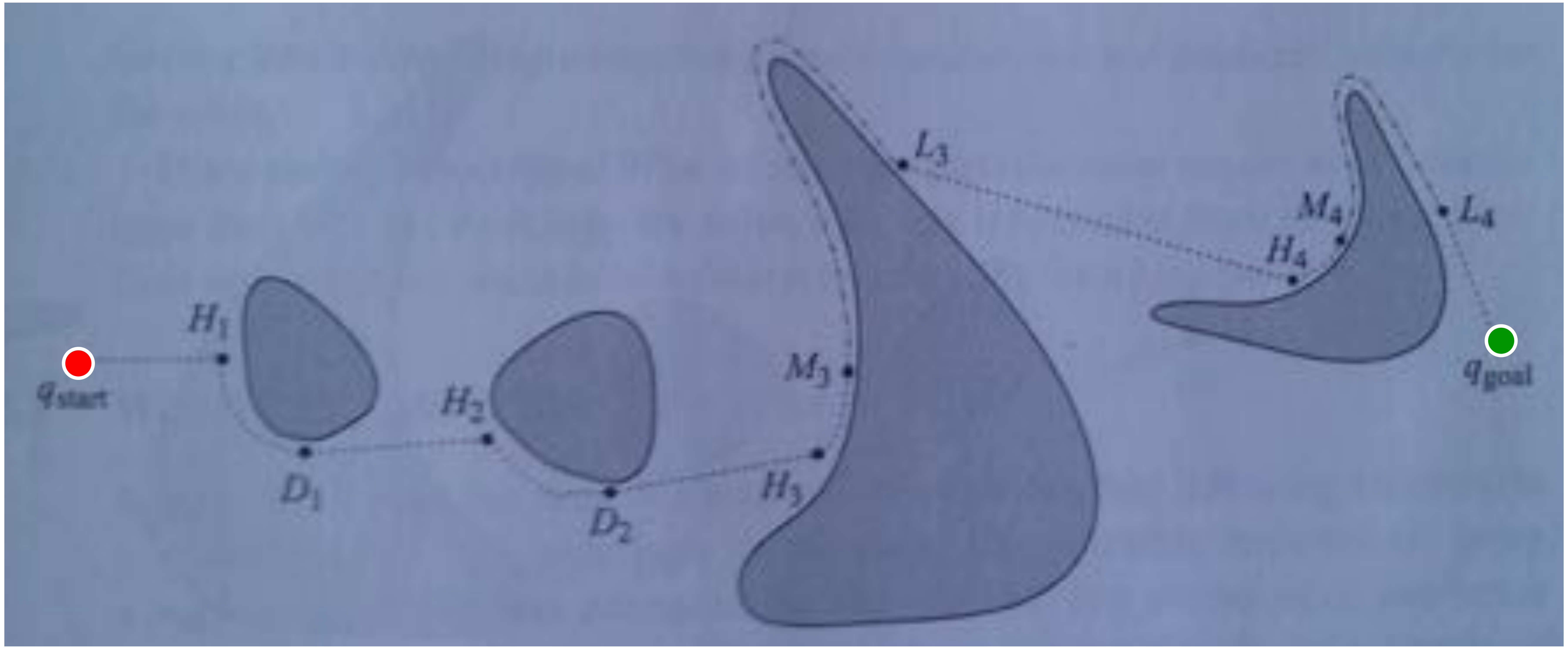
Example: range $R=\infty$



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

Example: range R=0

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

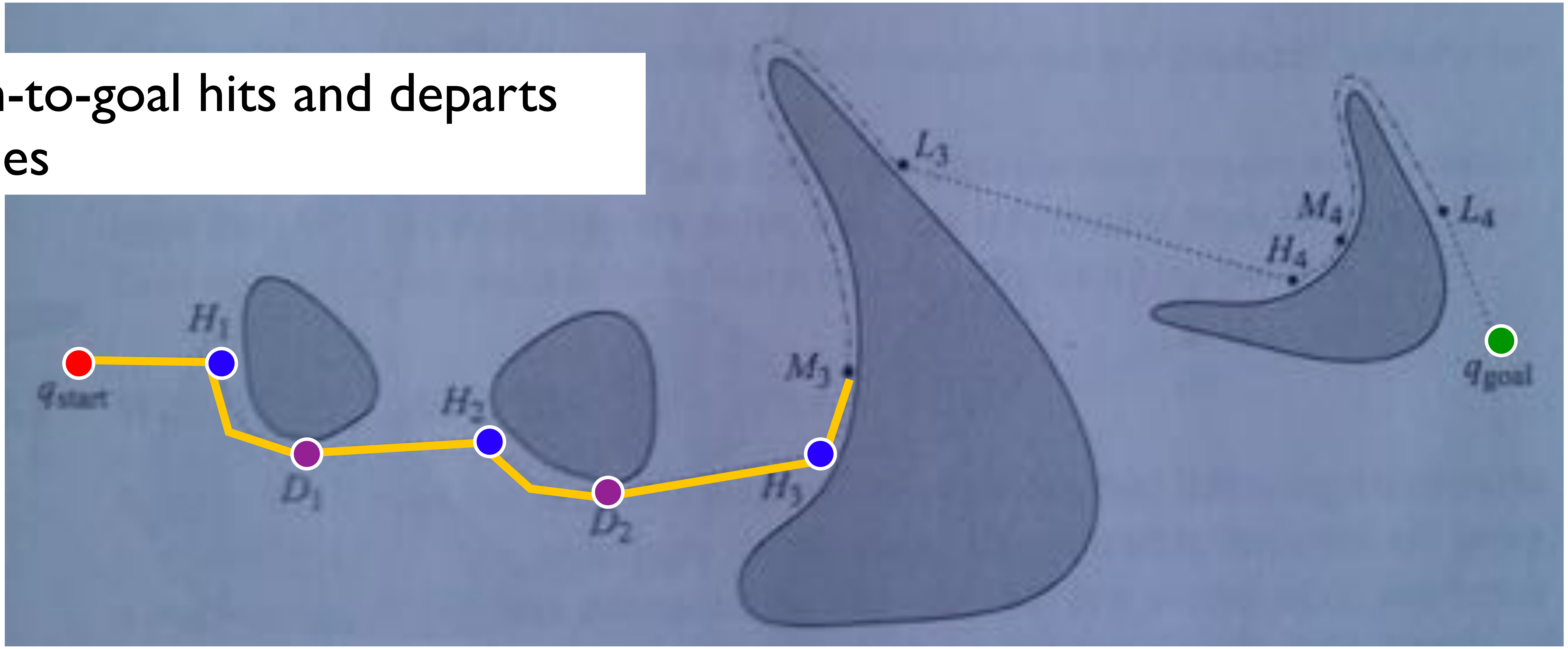


Slide borrowed from Michigan Robotics autorob.org

Example: range R=0

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

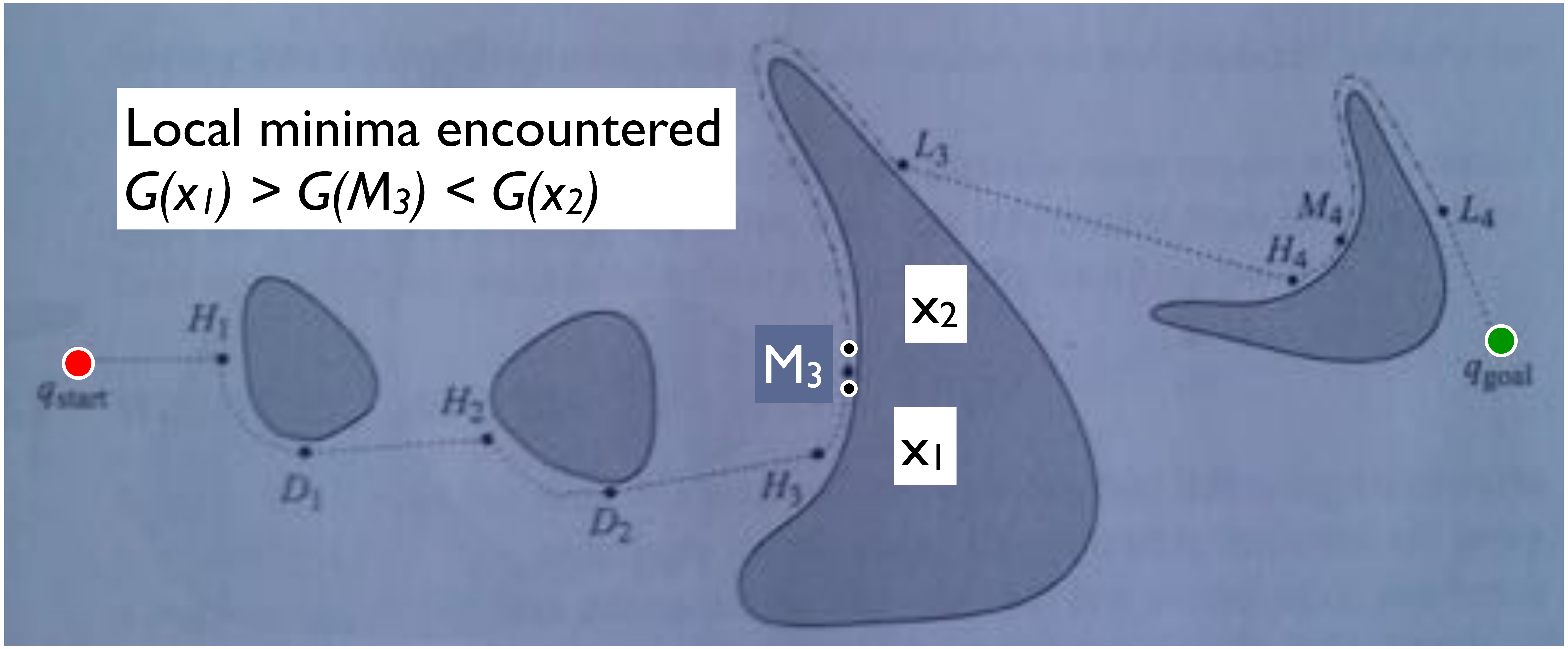
Motion-to-goal hits and departs obstacles



Slide borrowed from Michigan Robotics autorob.org

Example: range R=0

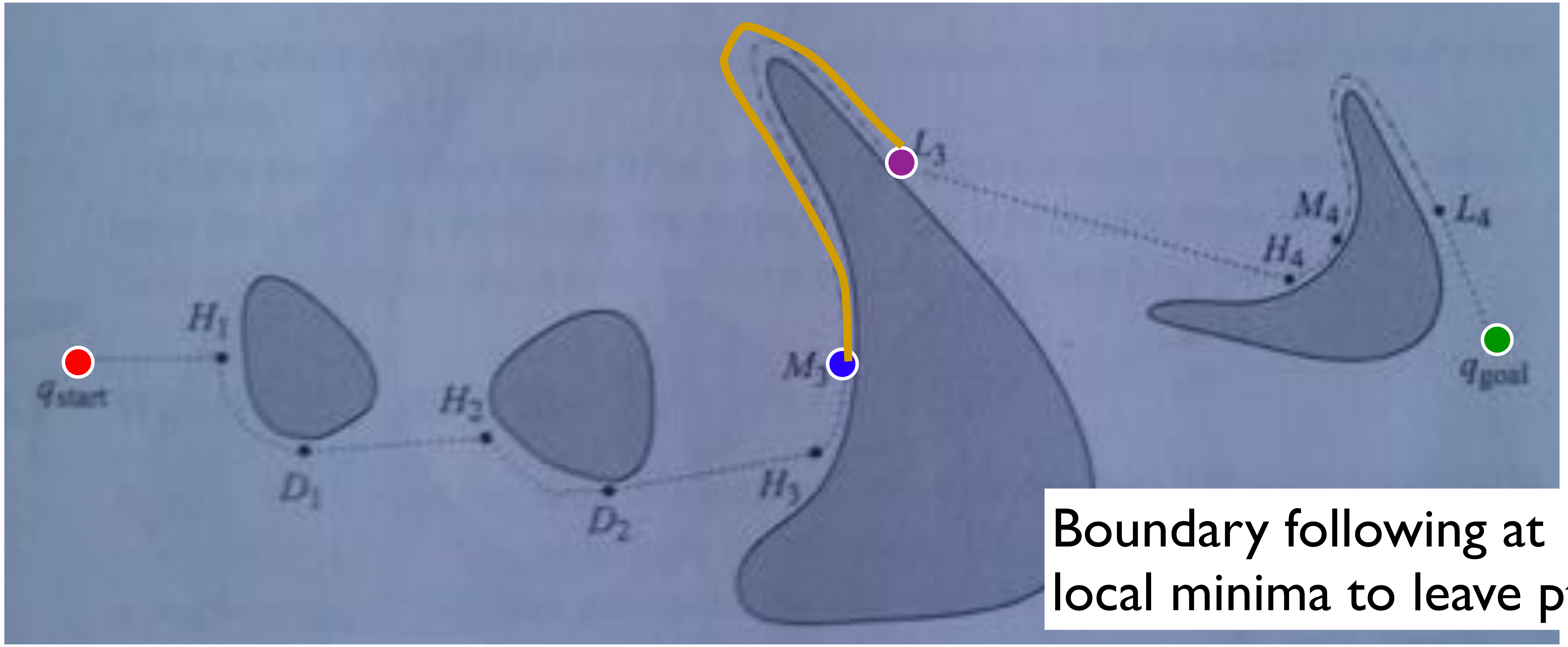
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



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

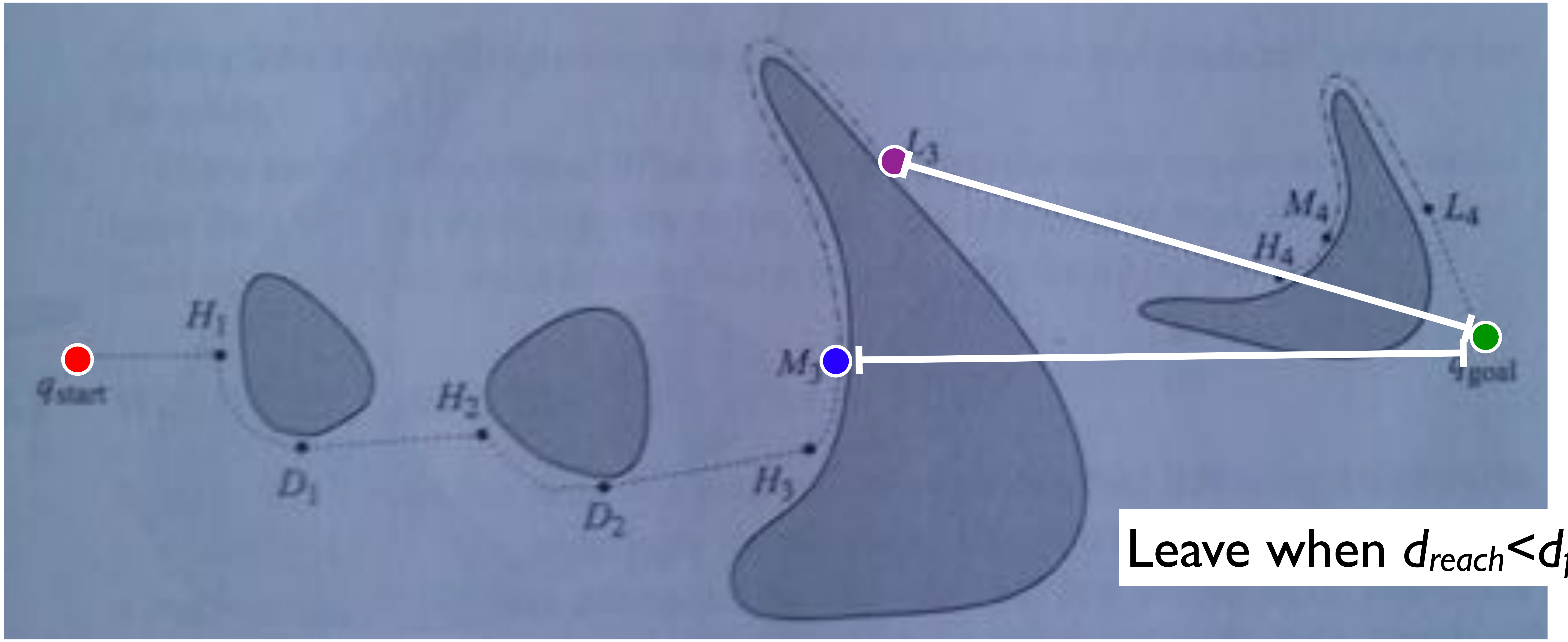


Boundary following at local minima to leave pt

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

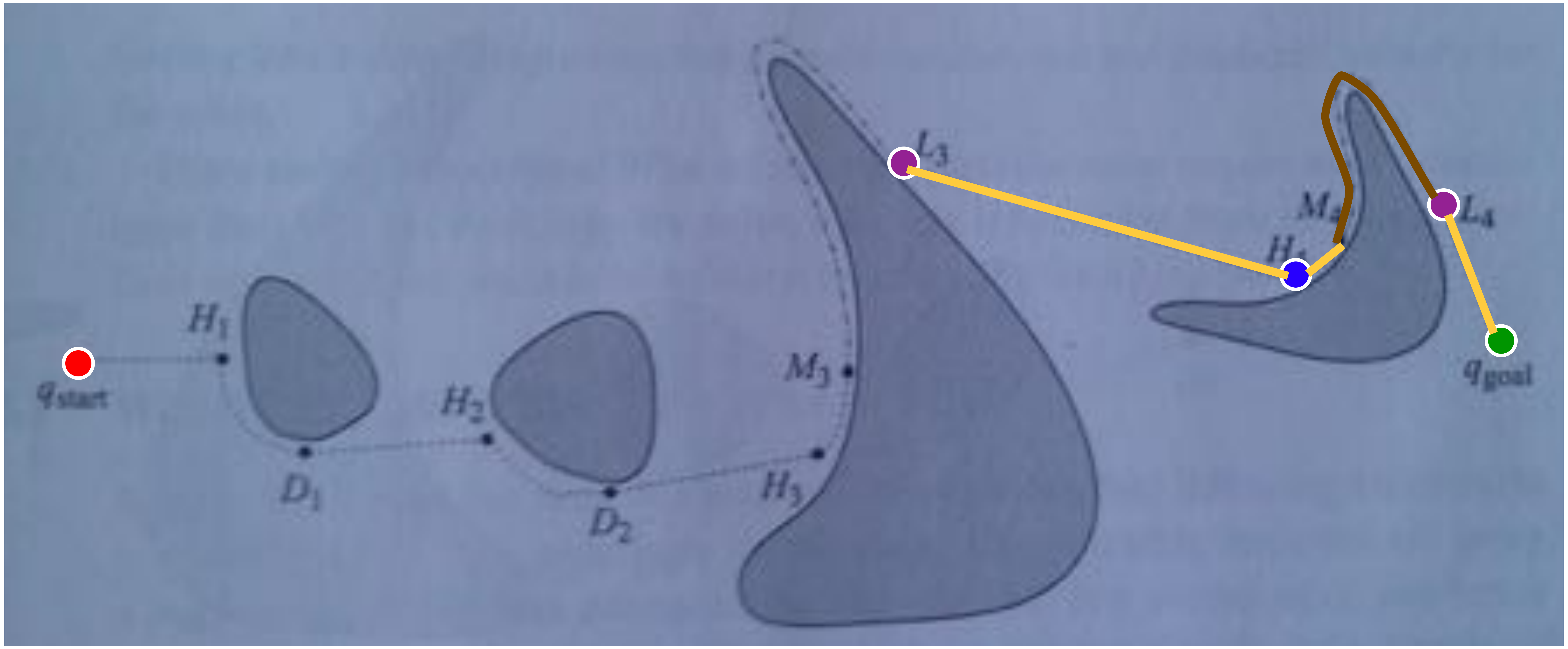


Leave when $d_{reach}<d_{follow}$

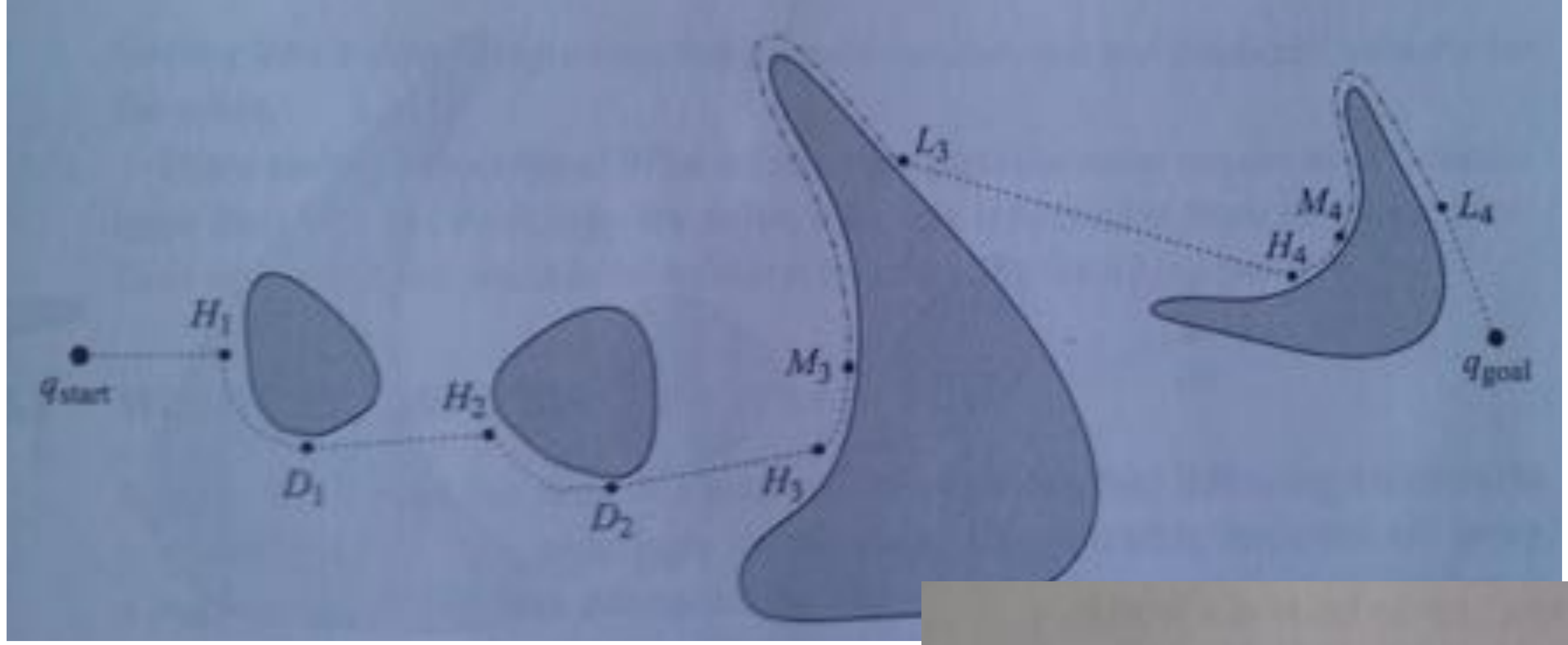
Local minima at increase of $G(x) = d(x,O_i)+d(O_i,q_{goal})$

Example: range R=0

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

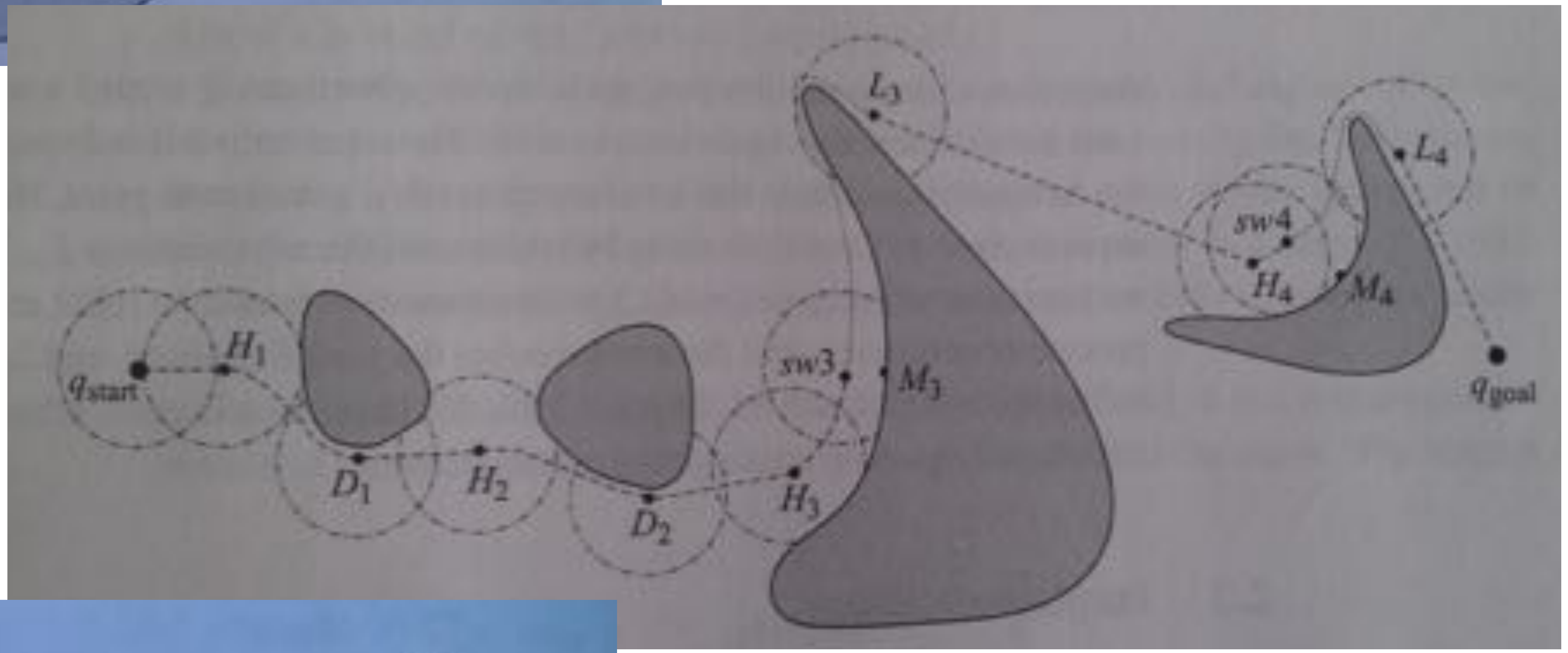


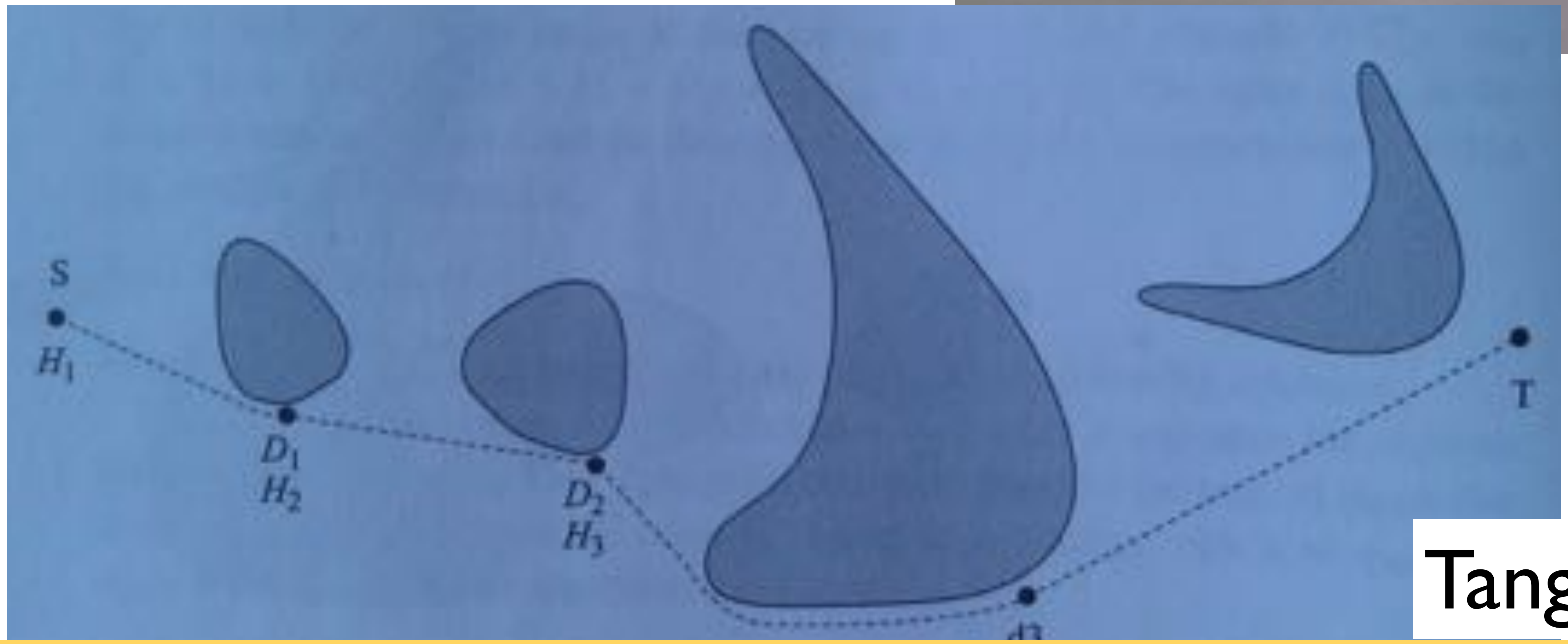
Slide borrowed from Michigan Robotics autorob.org



Tangent bug R=0

Tangent bug with limited radius





Tangent bug R=infinity

What does BugX assume that Random Walk does not?

[redacted]

What does BugX assume that Random Walk does not?

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

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?

[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

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

Next Lecture

Planning - III - Configuration Space