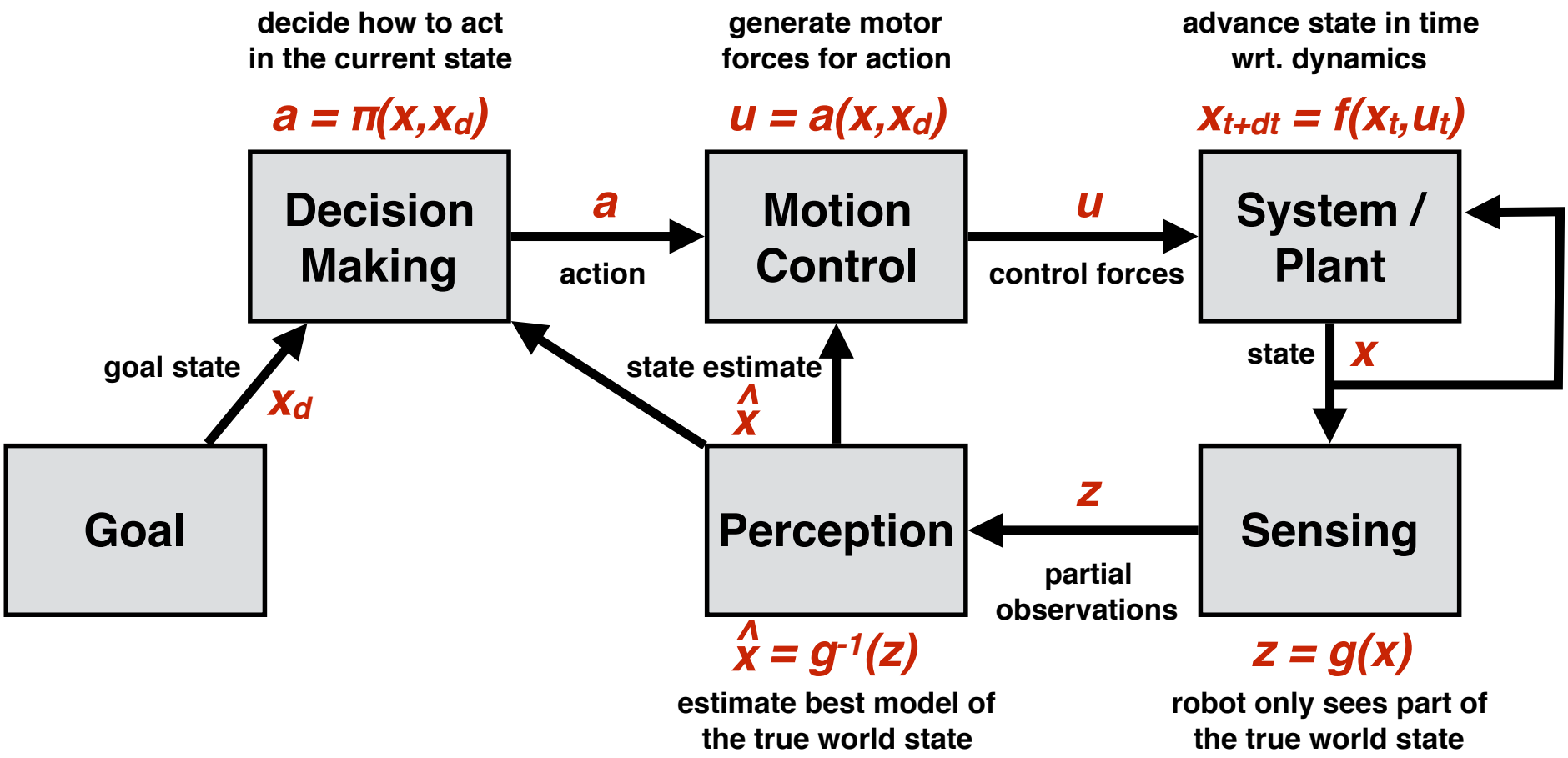
Bug Algorithms

autorob.github.io

https://mrhoracefranklinturtle.files.wordpress.com/2013/12/roach-maze.jpg

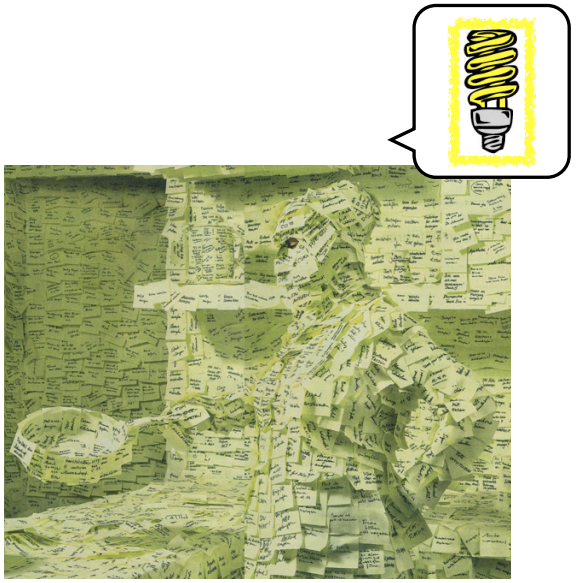
Robot Control Loop



Should your robot's decision making

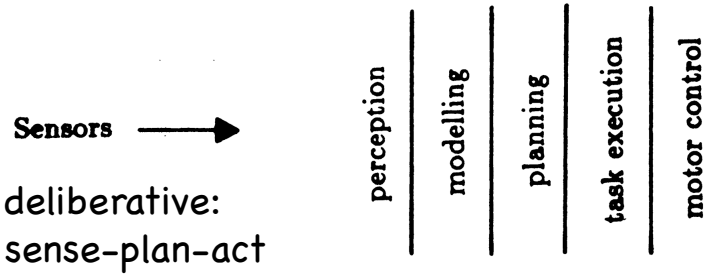
fully think through
solving a problem?

OR

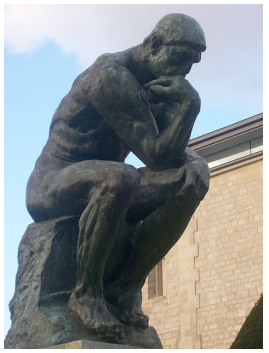


react quickly to
changes in its world?

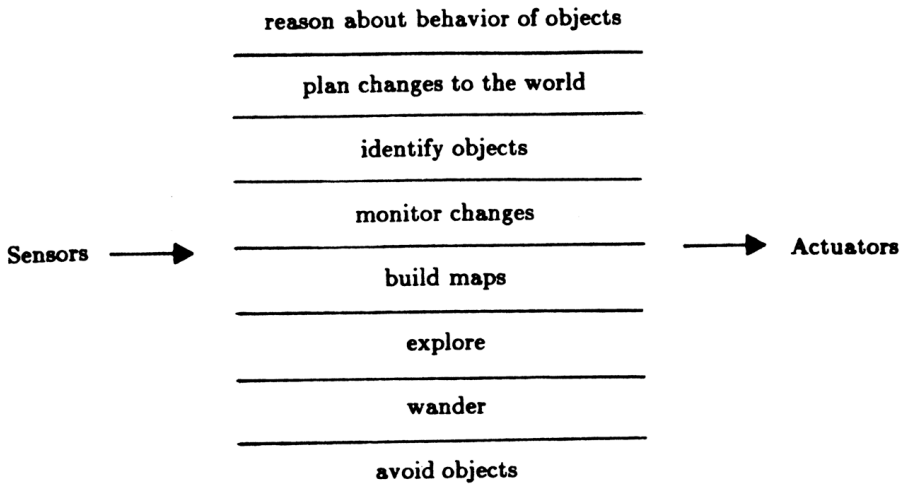
Deliberation v. Reaction



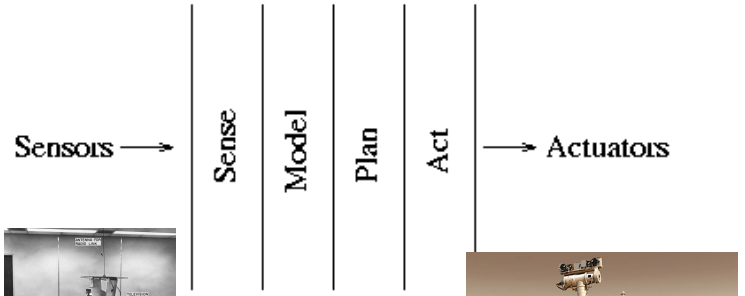
deliberative:
sense-plan-act

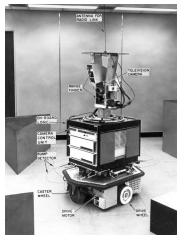


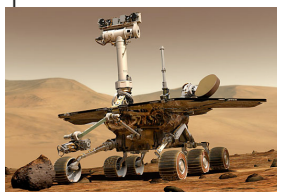
reaction: subsumption,
Finite State Machine
controllers act in parallel



Deliberation

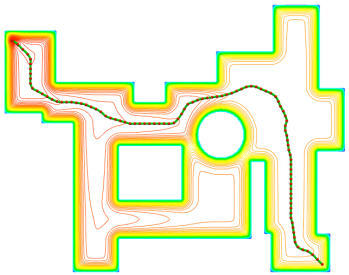


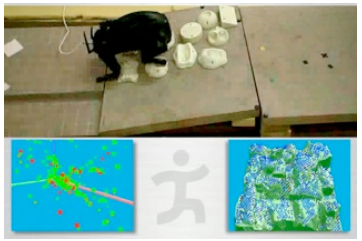




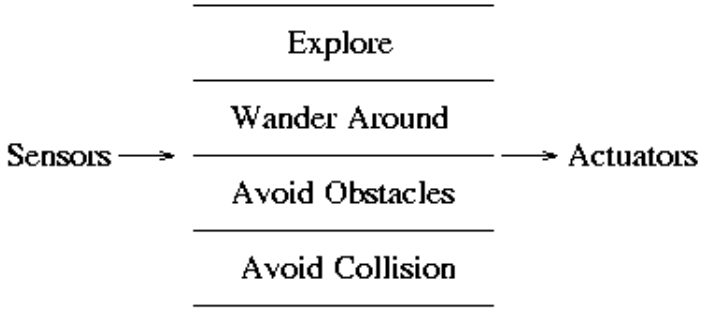
"Sense-Plan-Act" paradigm

- 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

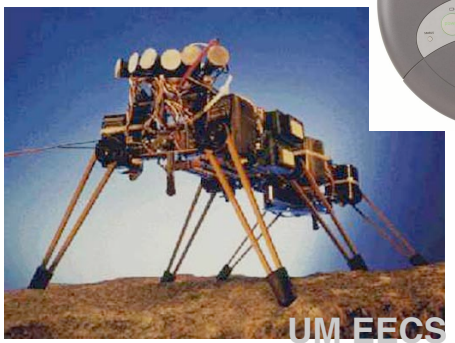


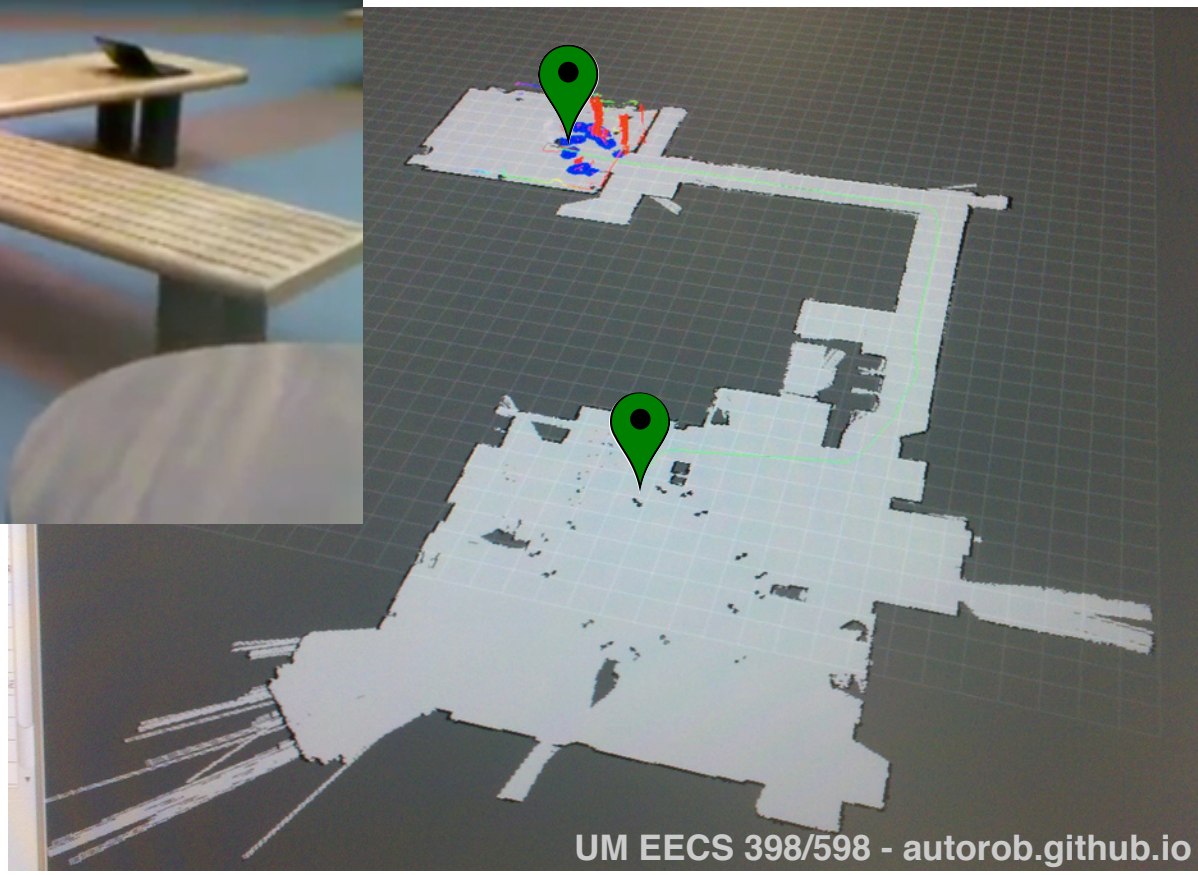


Reaction



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





Let's start with base navigation

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?**

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

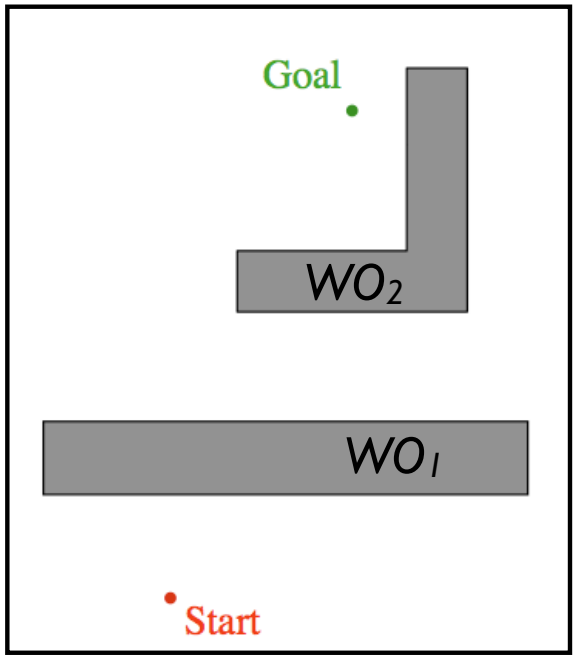
Lisa Miller, http://www.youtube.com/watch?v=VBzXDrz8rMI

Base Navigation

- How get from point A to point B
- What is the simplest policy to perform navigation?
  - random walk (2010 Enclosure Escape project)
  - reactive: embodied intelligence
- **What is a "simple" deliberative policy?**

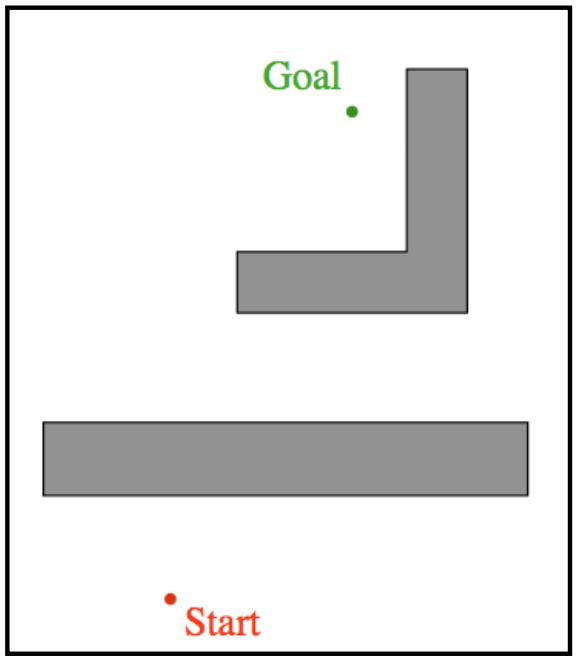
Bug Algorithms

- 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

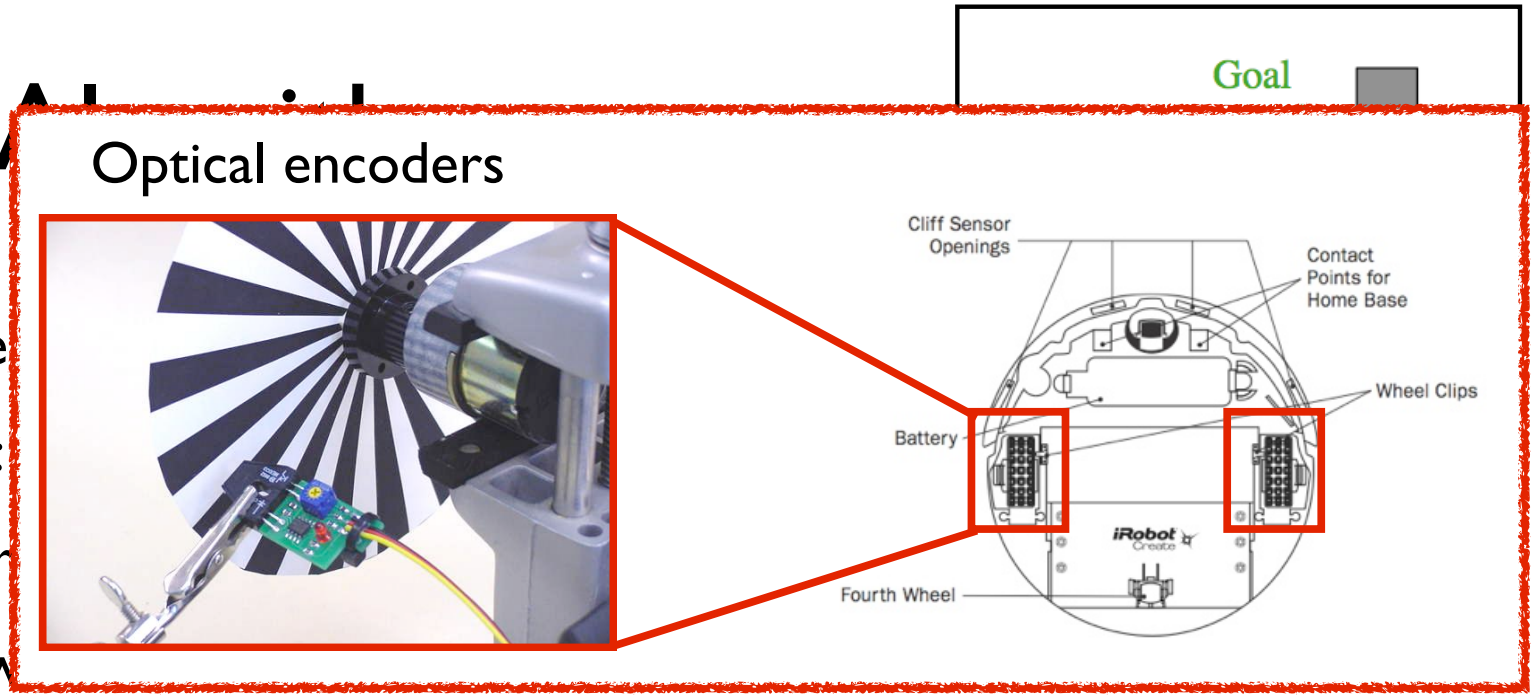


≈

bumper is essentially an on/off button

Bug A

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



≈

Interesting application of Bug algorithms ?

Secure | https://www.technologyreview.com/s/400145/honey-theyve-shrunk-the-rover/

MIT Technology Review

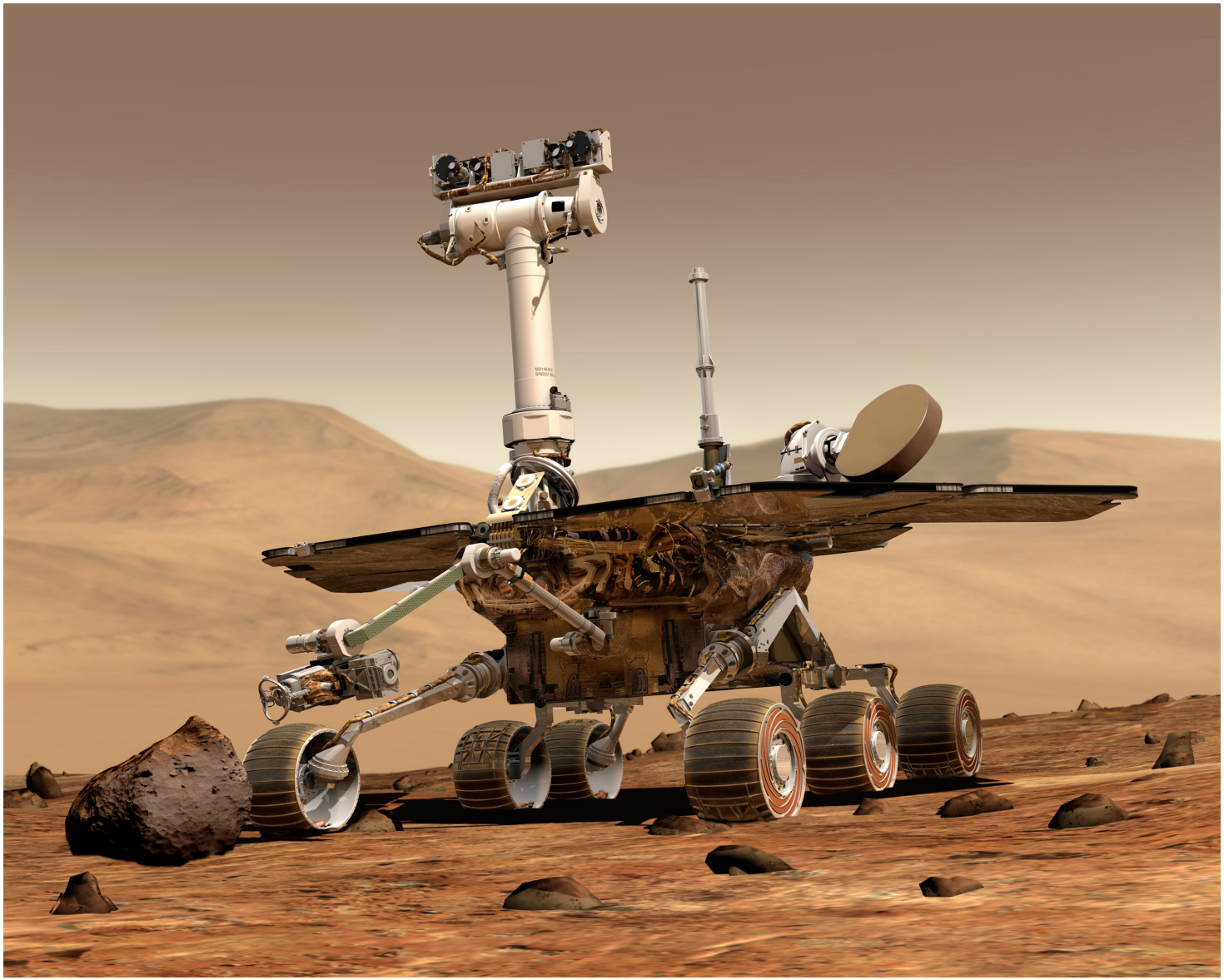
Ghengis hexapod robot

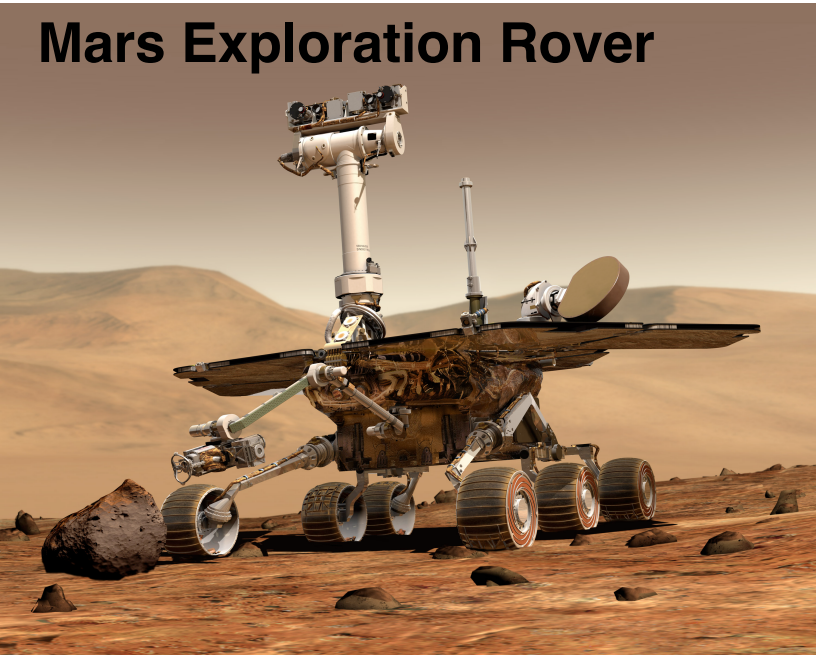
Intelligent Machines

Honey, They've Shrunk the Rover

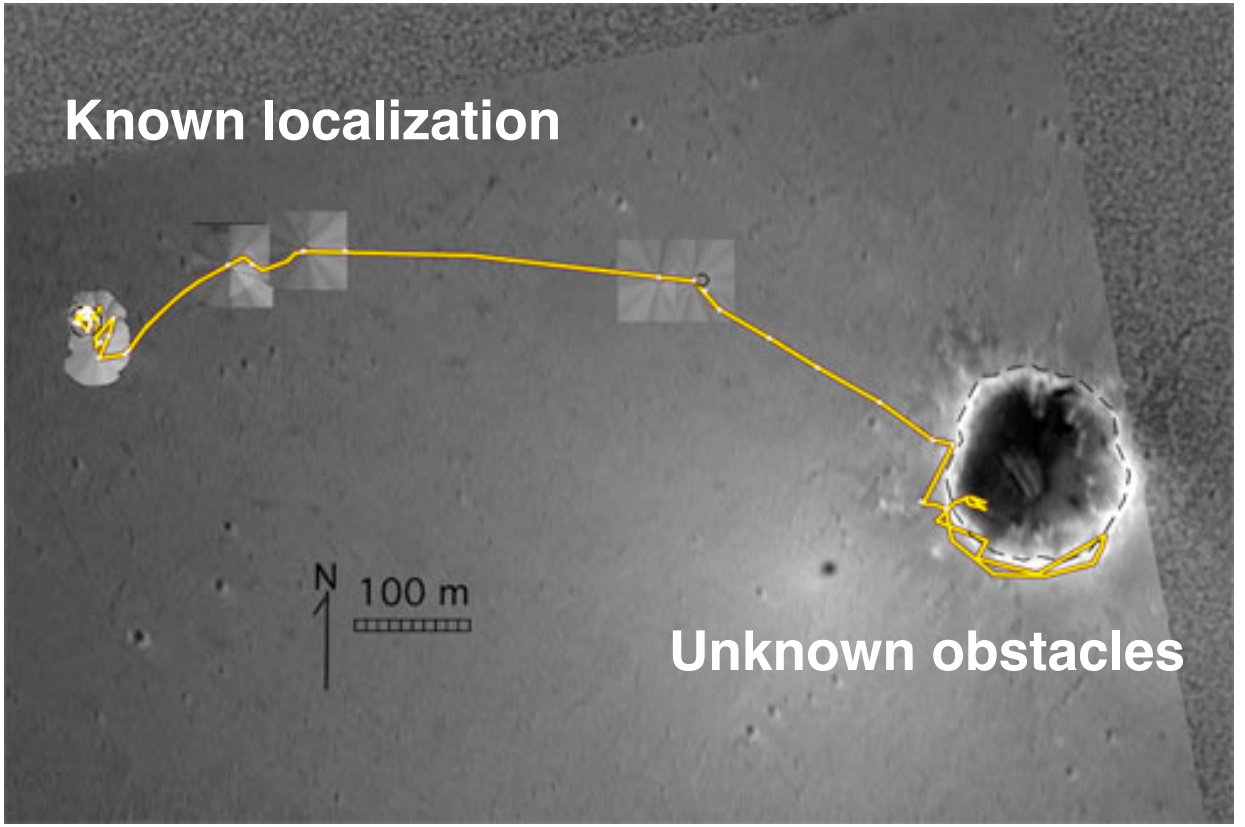
The Sojourner vehicle that trekked across the Martian surface captured the world's fancy last summer, but we haven't seen anything yet. Next will come rovers that can roam miles across Mars and "aerobots" able to survey other planets.

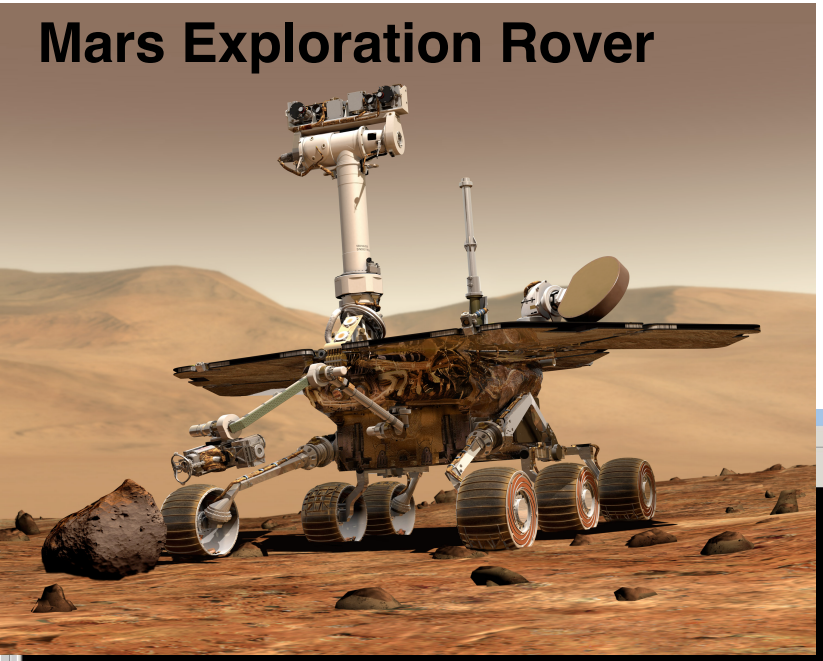
by Eric Scigliano January 1, 1998



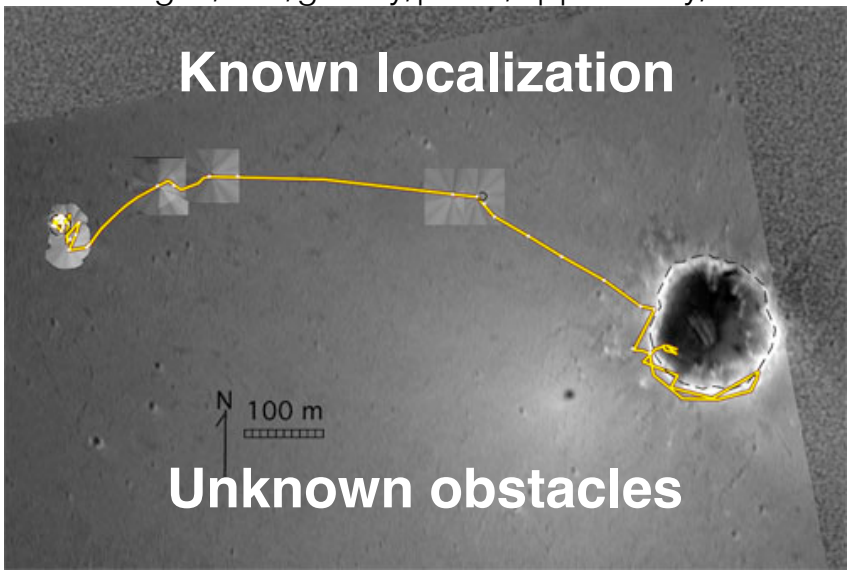


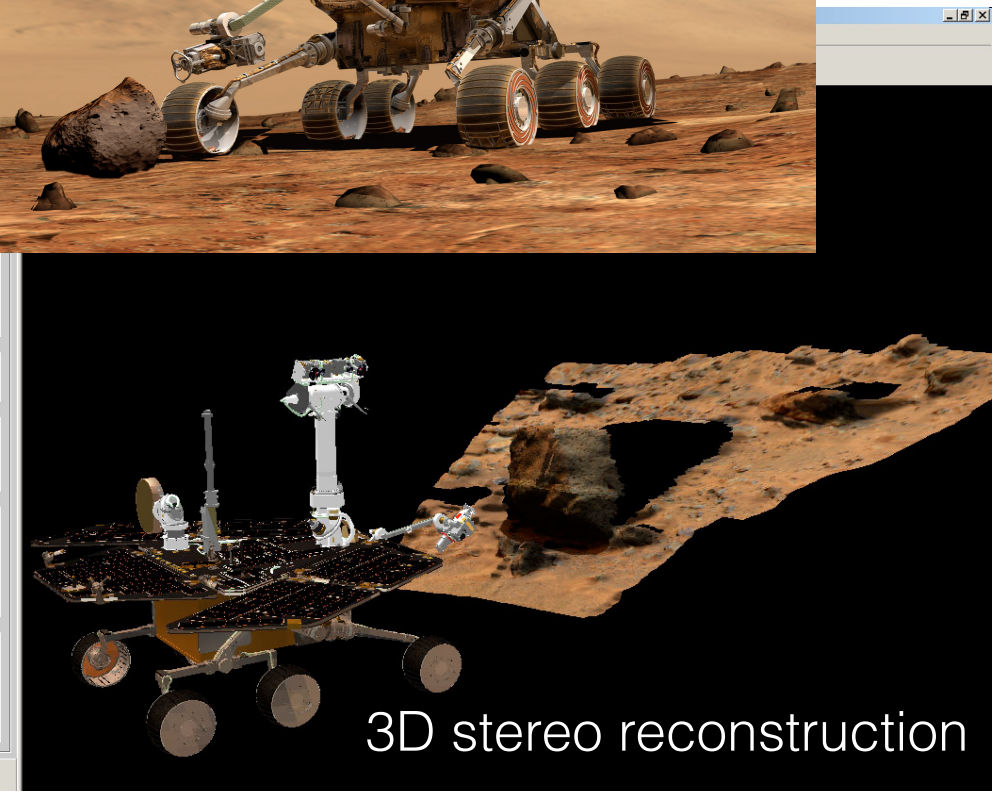
http://mars.nasa.gov/mer/gallery/press/opportunity/20040921a.html

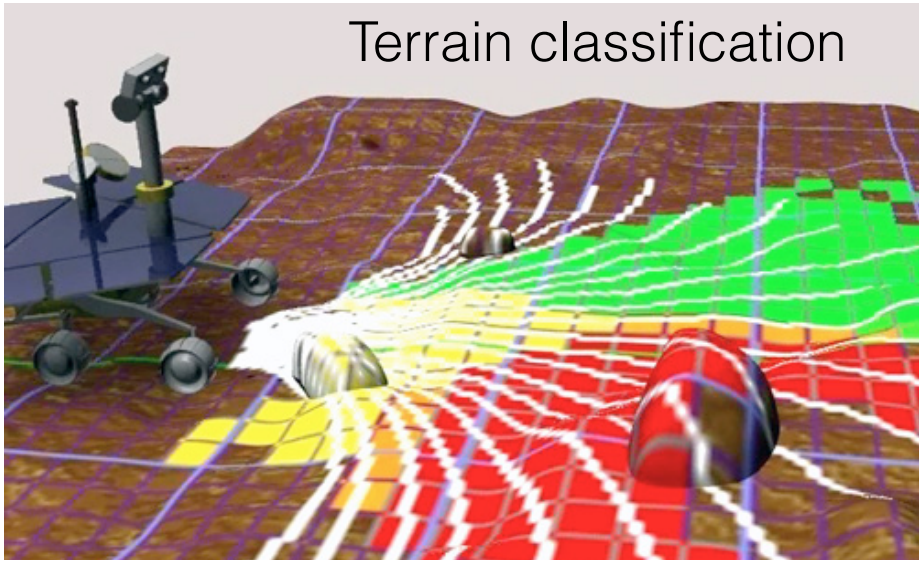




http://mars.nasa.gov/mer/gallery/press/opportunity/20040921a.html

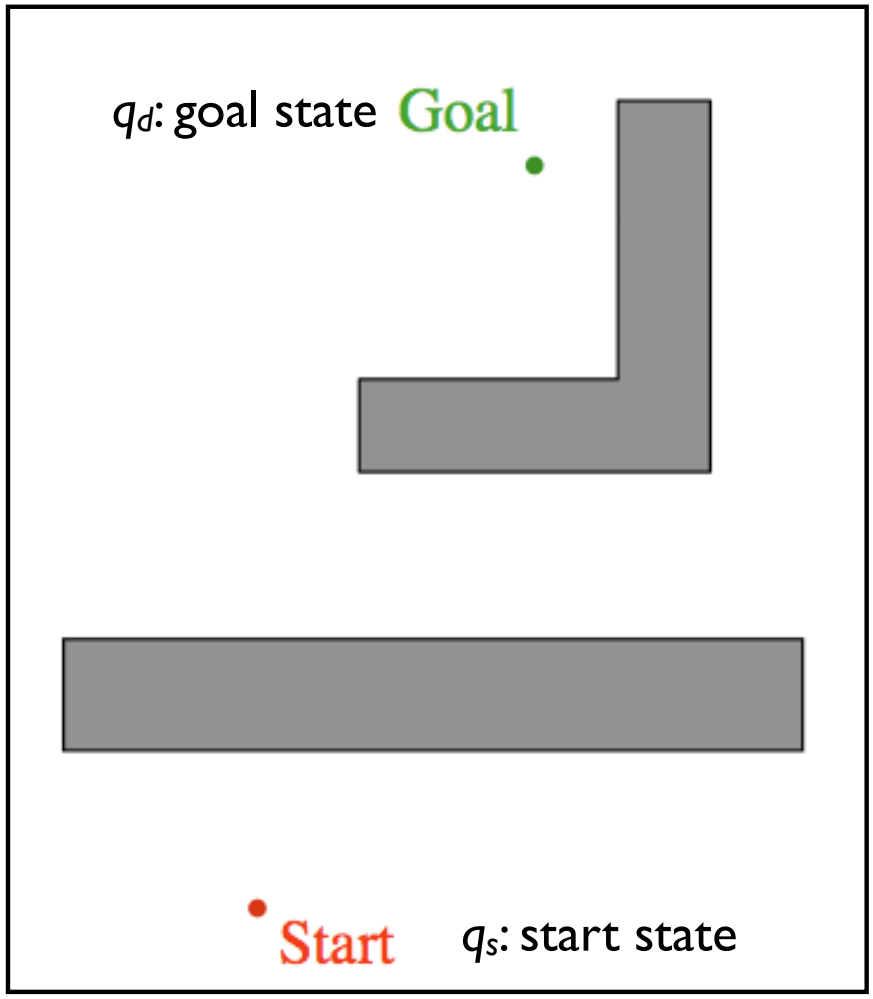






University Rover Challenge

http://urc.marssociety.org/home/photo-gallery/urc2016



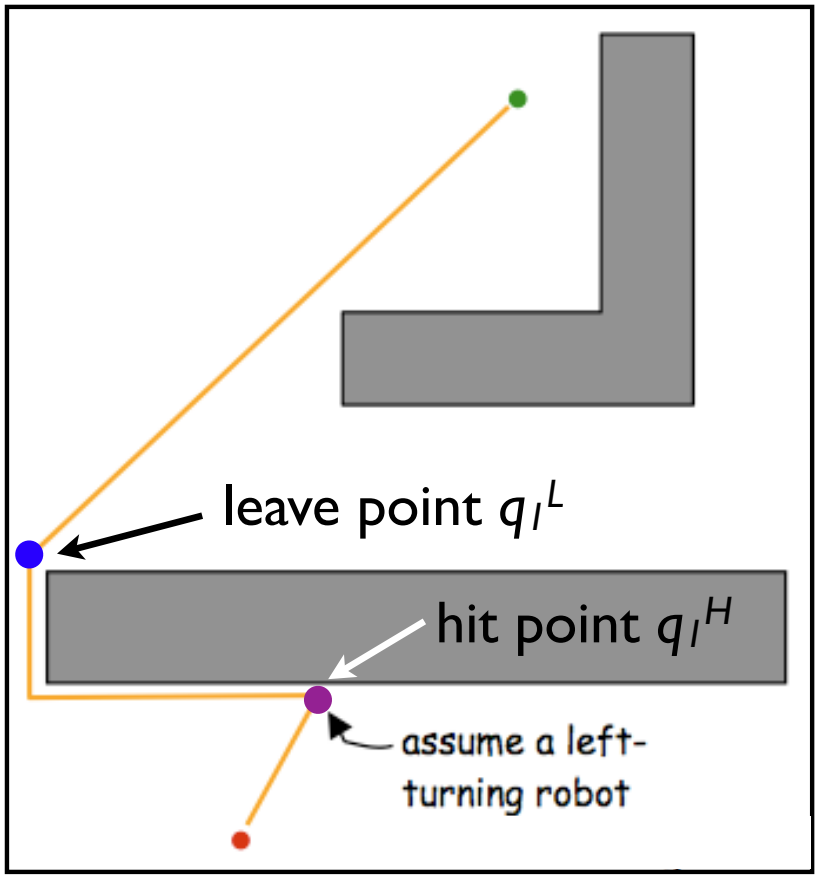
Bug Navigation

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:

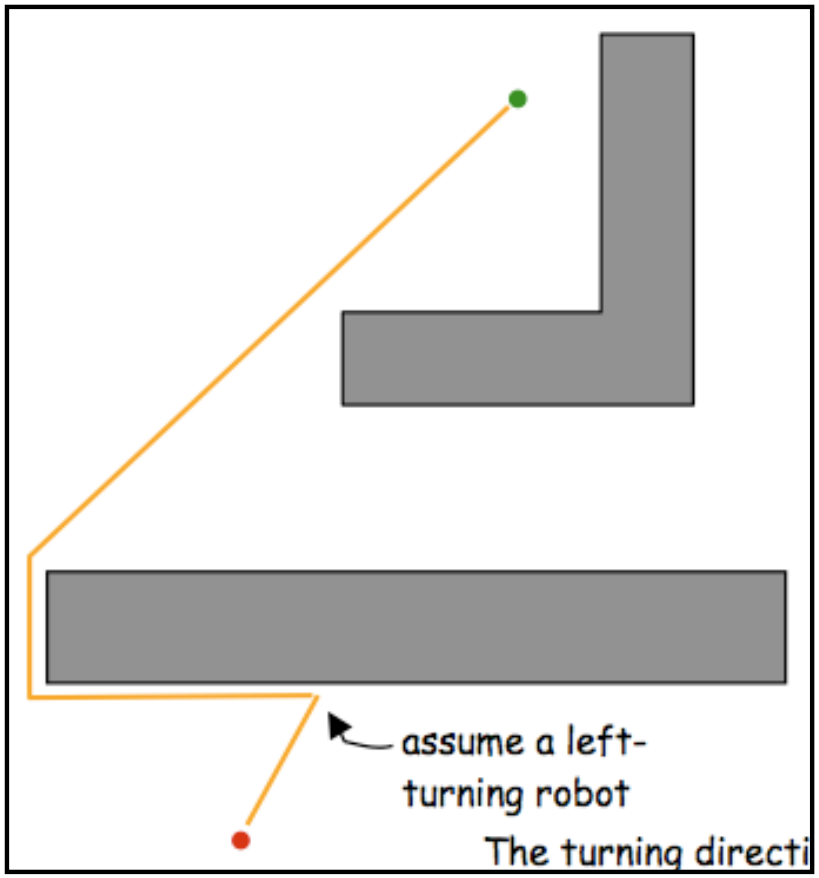
a) move forward with slight turn

b) when bumped, turn opposite direction

c) goto (*a*)

Trevor Jay

What map would foil Bug 0?

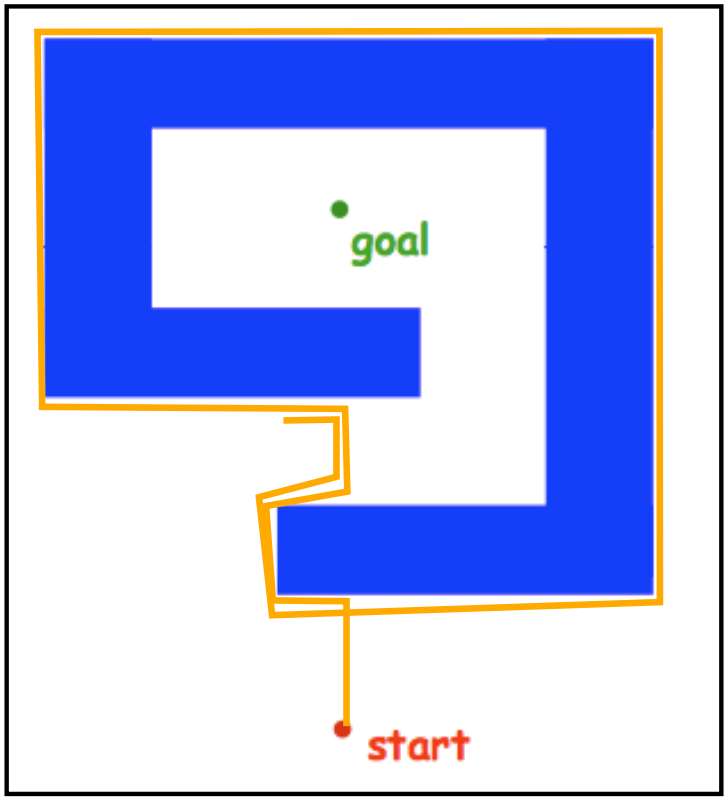


Bug 0

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

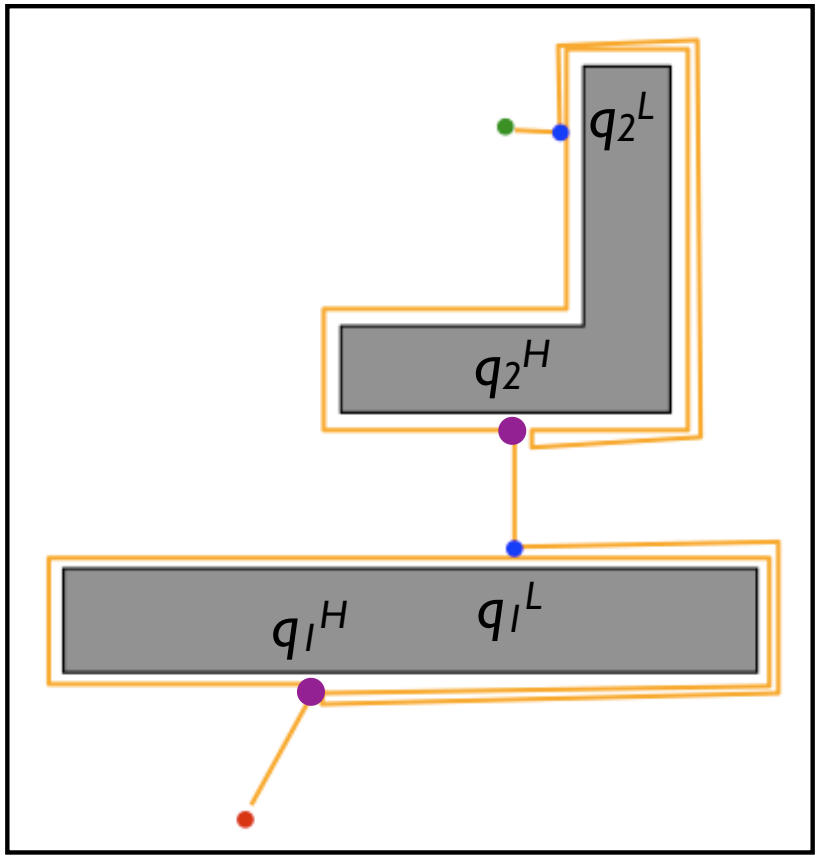
1) Head towards goal

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 1



1) Head towards goal

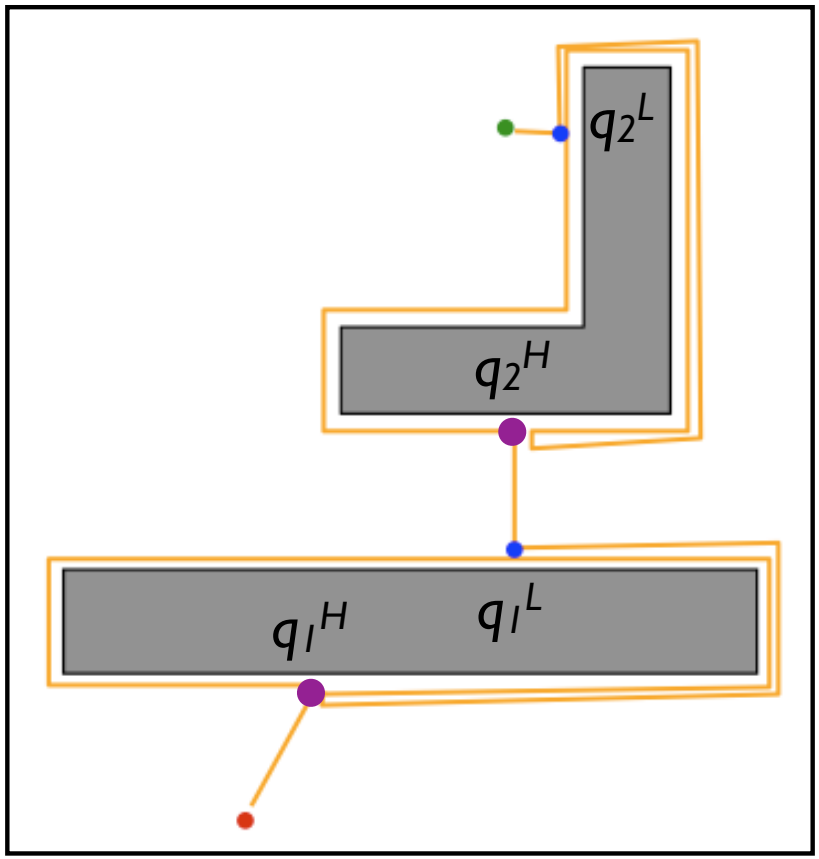
2) When hit point set, circumnavigate obstacle, setting leave point as closest to goal

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)

Bug 1: Detecting Failure

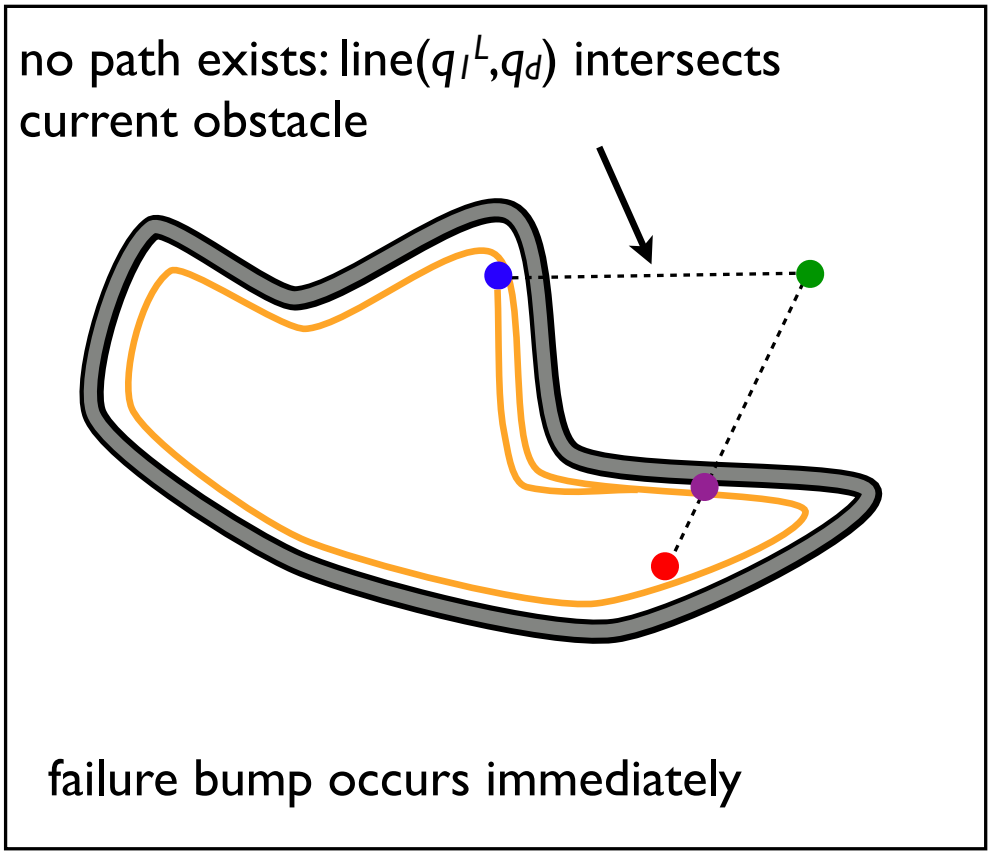
1) Head towards goal

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

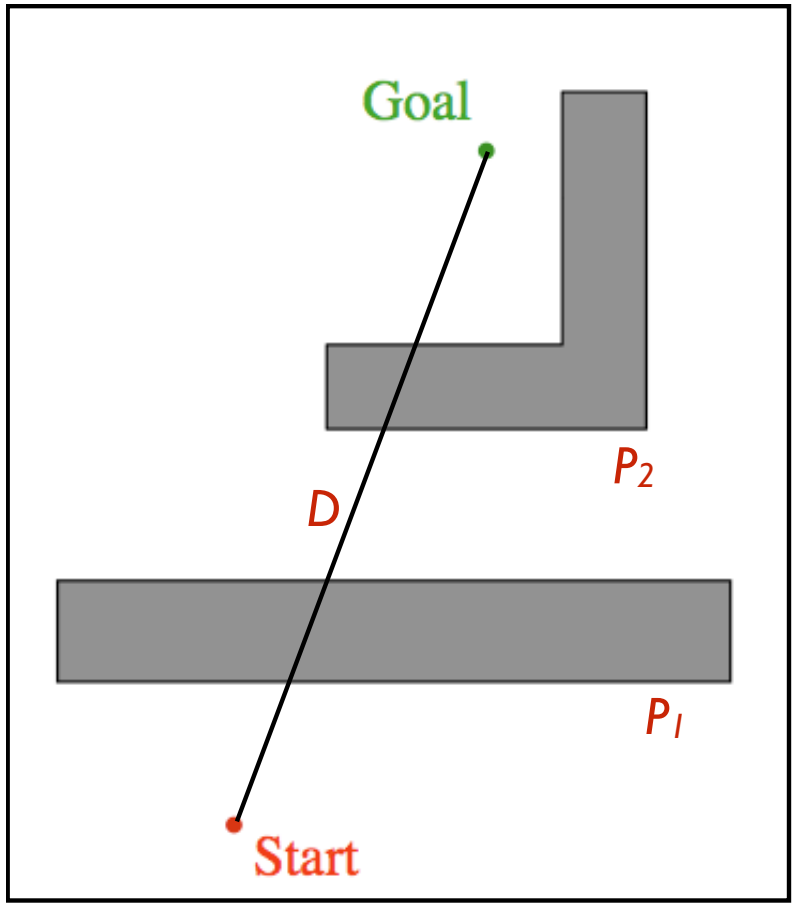
3) return to leave point

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

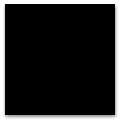
What map would foil Bug 1?



Bug 1: Search Bounds

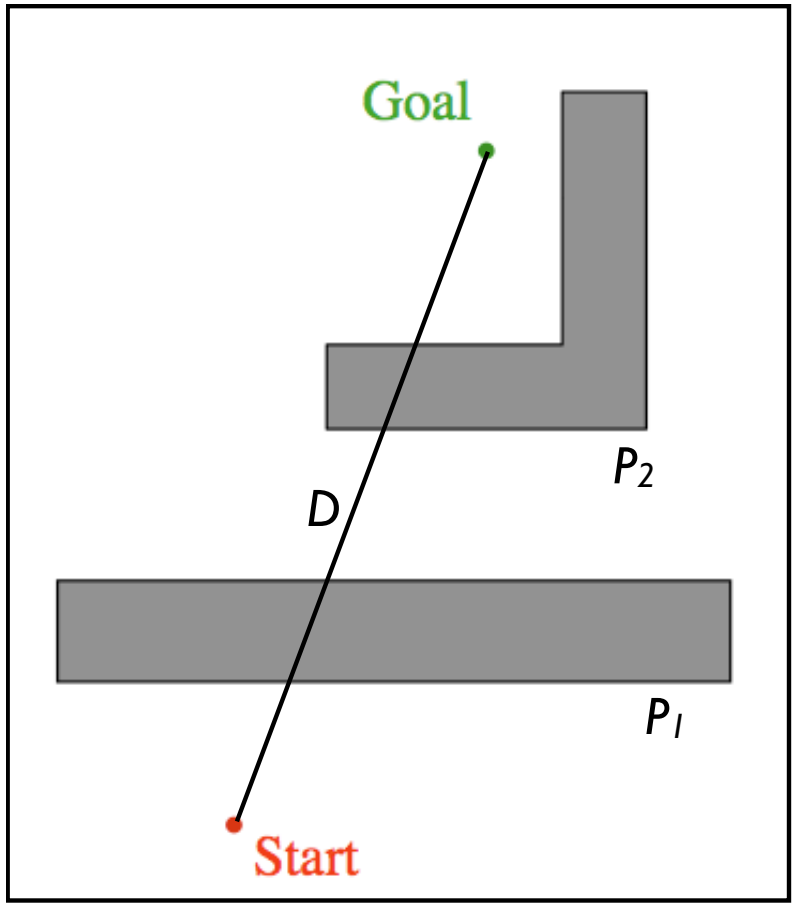


Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: 

Worst case: 

Bug 1: Search Bounds



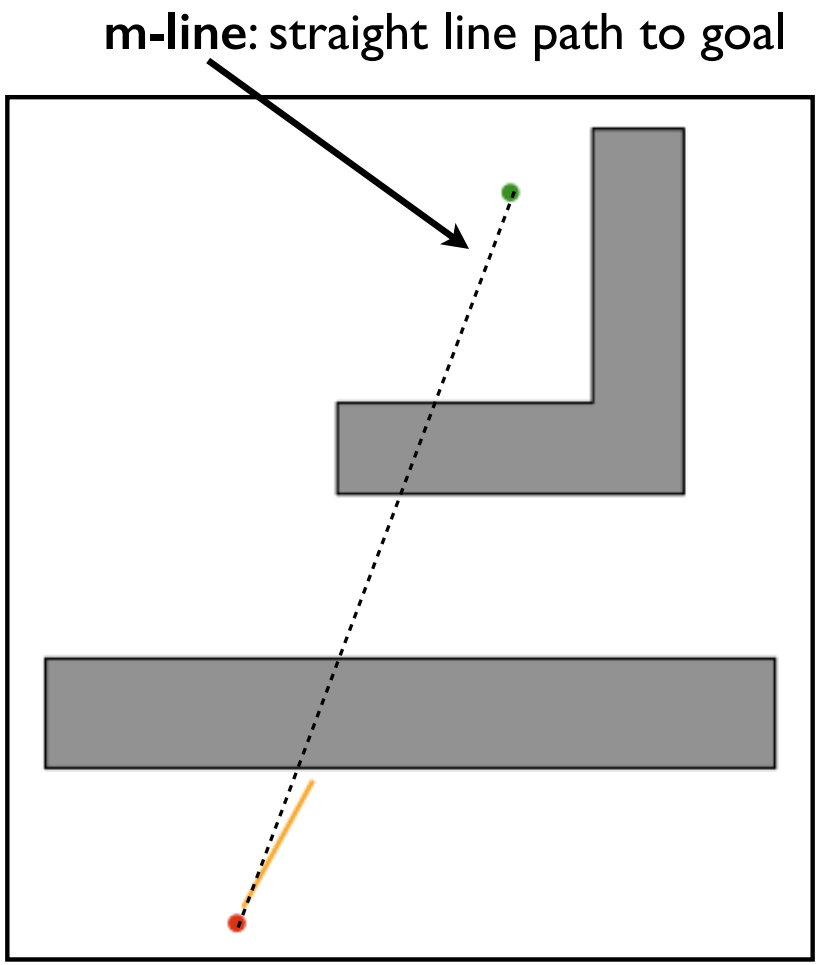
Bounds on path distance, assuming
D: distance start-to-goal
P_i: obstacle perimeter

Best case: *D*

Worst case: $D + 1.5\sum_i P_i$

Is there a faster bug?

Bug 2



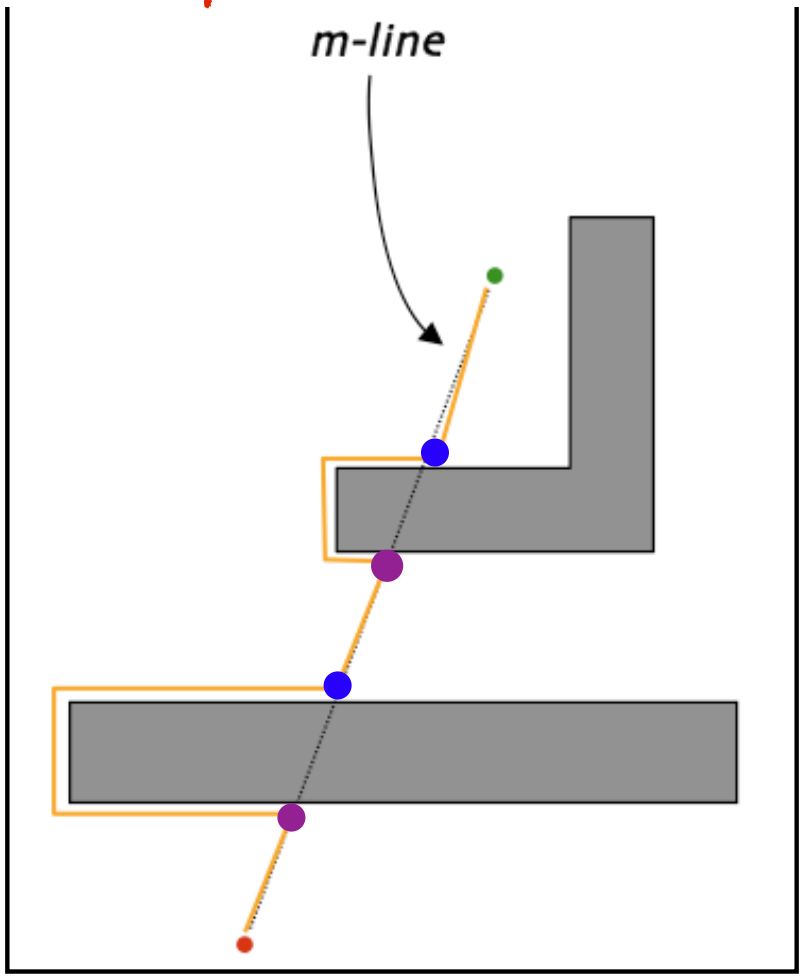
1) Head towards goal on m-line

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

3) set leave point and exit obstacle

4) continue from (1)

What map would foil Bug 2?



Bug 2

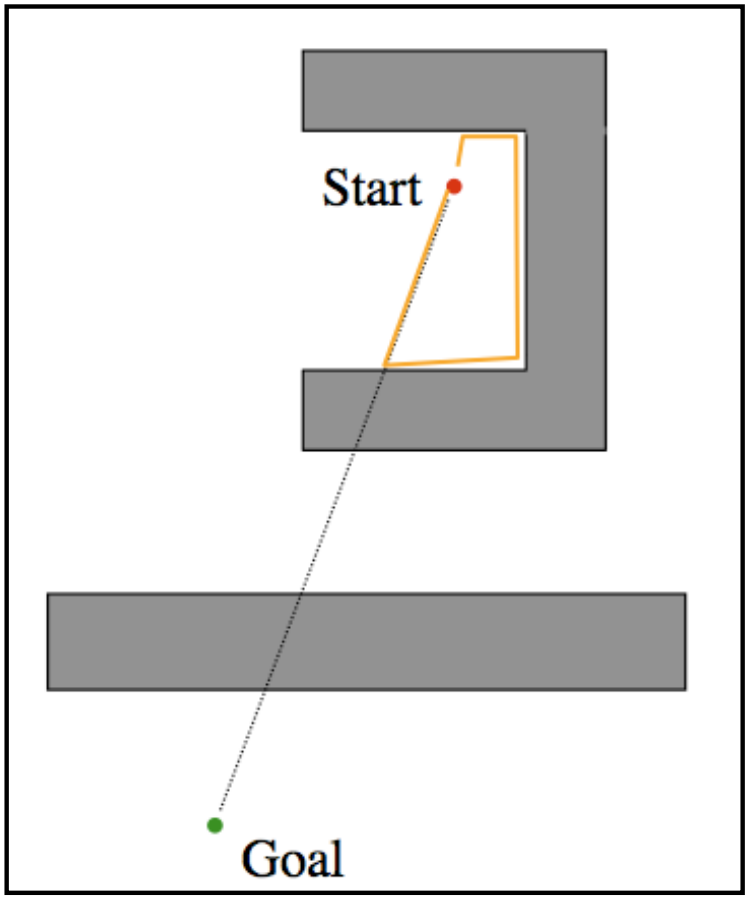
1) Head towards goal on m-line

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



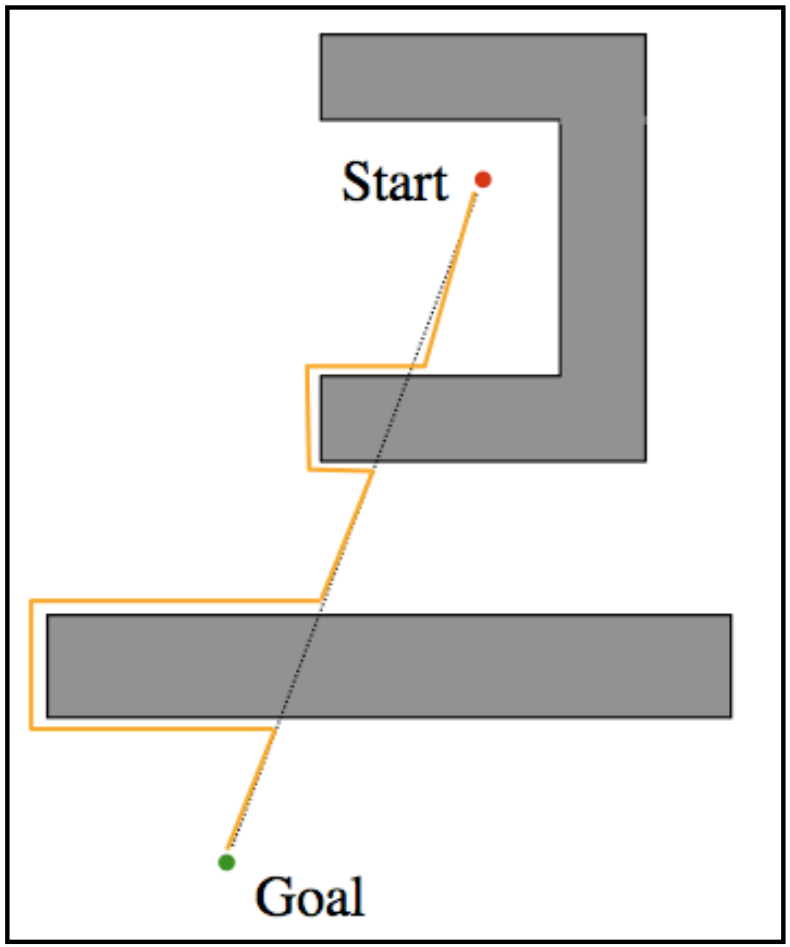
1) Head towards goal on m-line

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

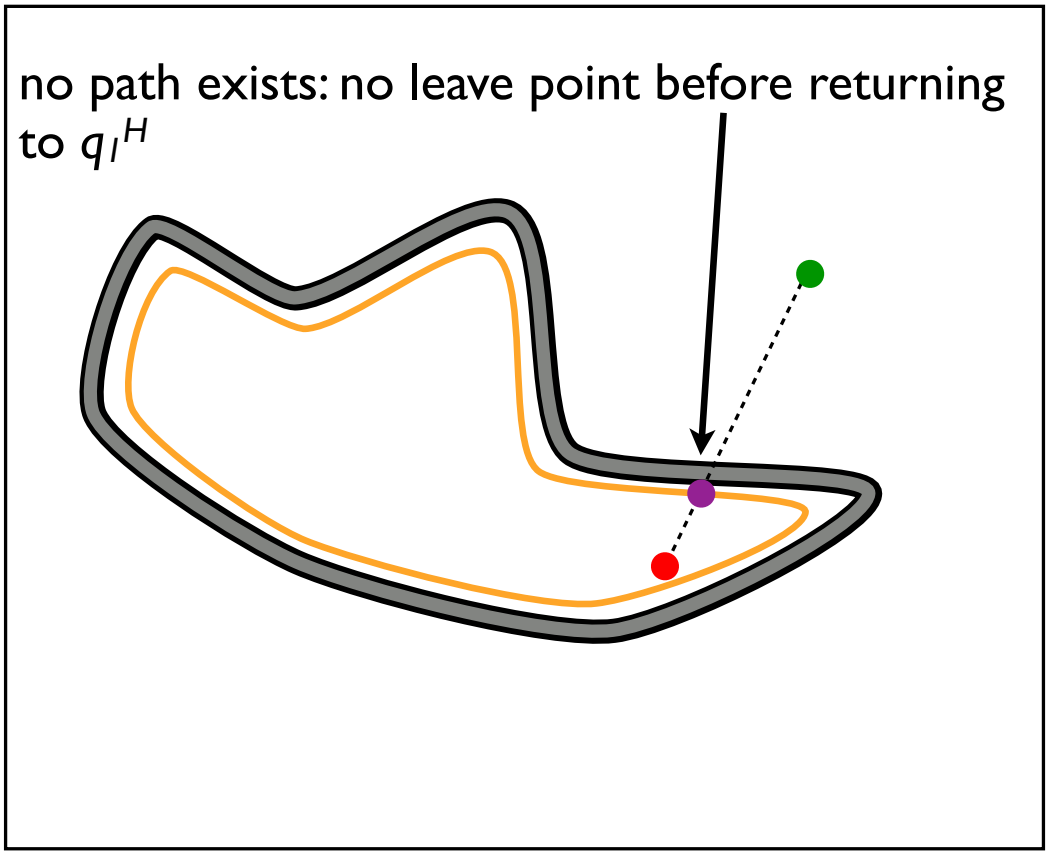


1) Head towards goal on m-line

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

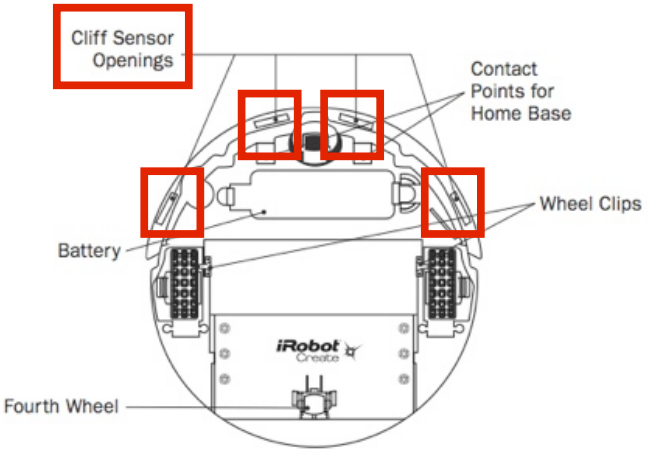
2) When hit point set, traverse obstacle until m-line is encountered & closer to the goal
or hit point reached

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

4) continue from (1)

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)

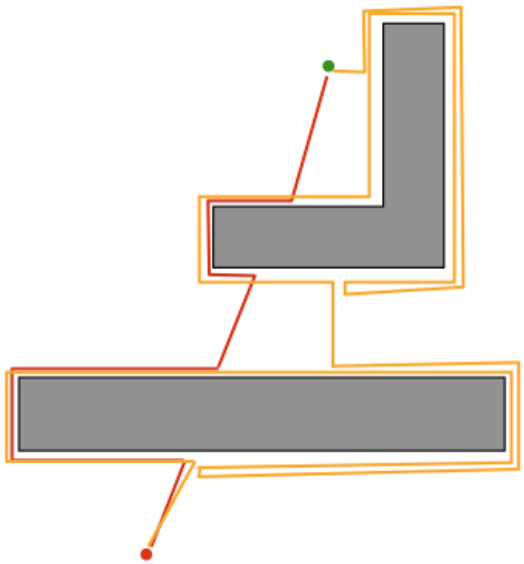
Bug 2 beats Bug 1

Bug 1 beats Bug 2

Bug 1 v. Bug 2:

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

Bug 2 beats Bug 1



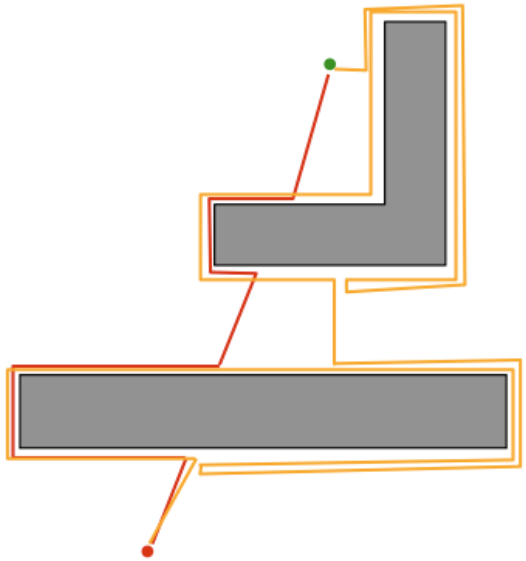
Bug 1 beats Bug 2

?

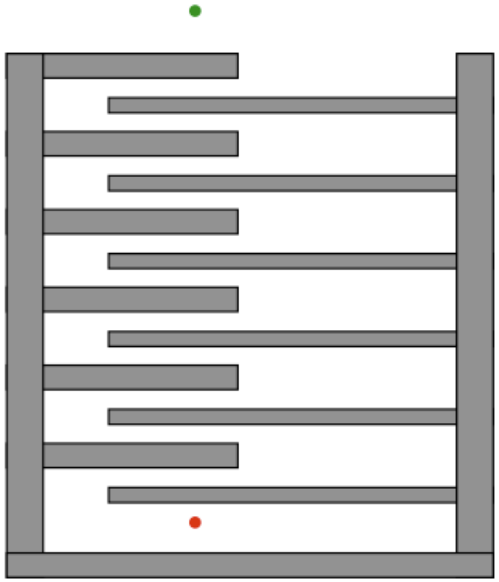
Bug 1 v. Bug 2:

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

Bug 2 beats Bug 1



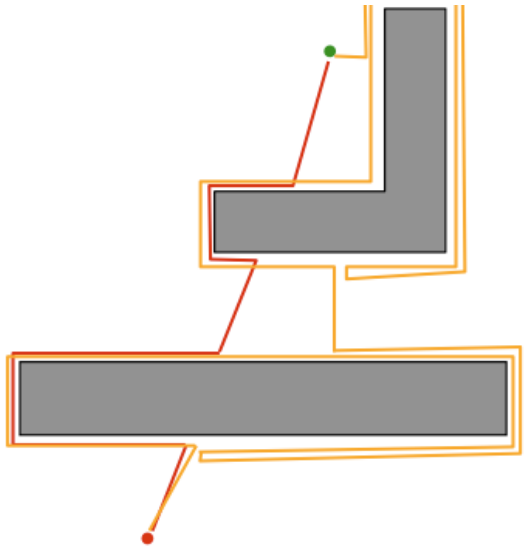
Bug 1 beats Bug 2



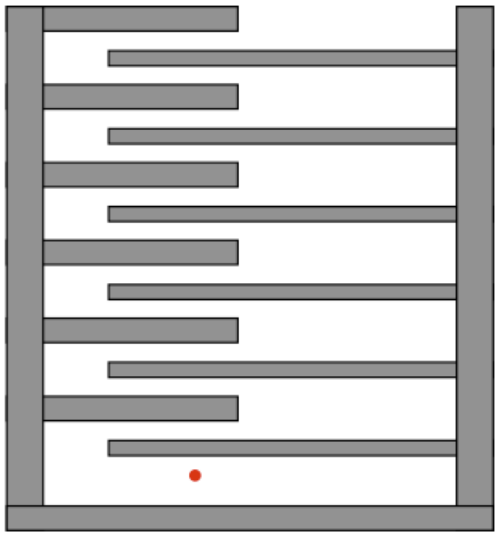
Bug 1 v. Bug 2:

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

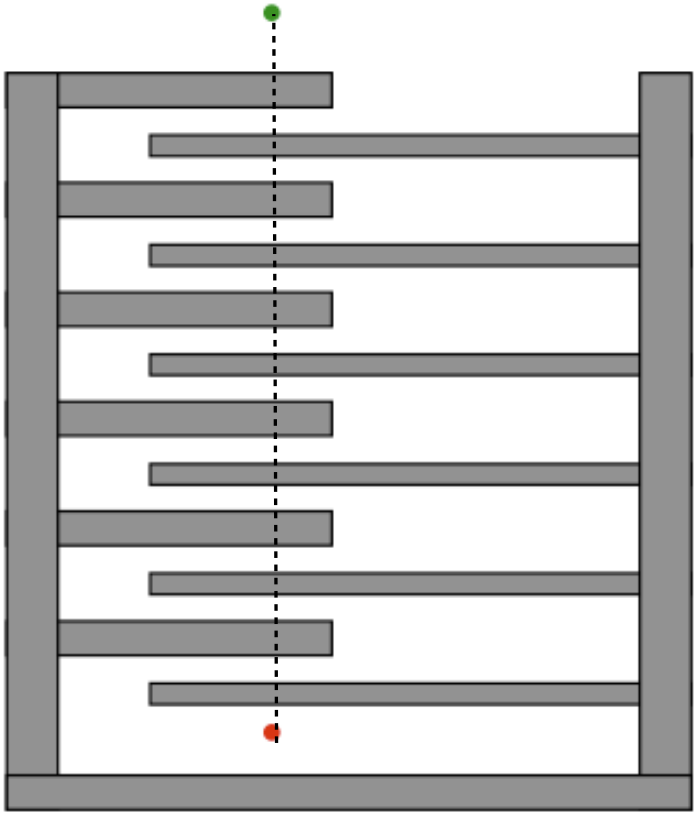
Bug 2: greedy, better typical performance



Bug 1: exhaustive search, more predictable

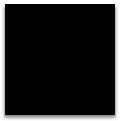


Bug 2: Search Bounds



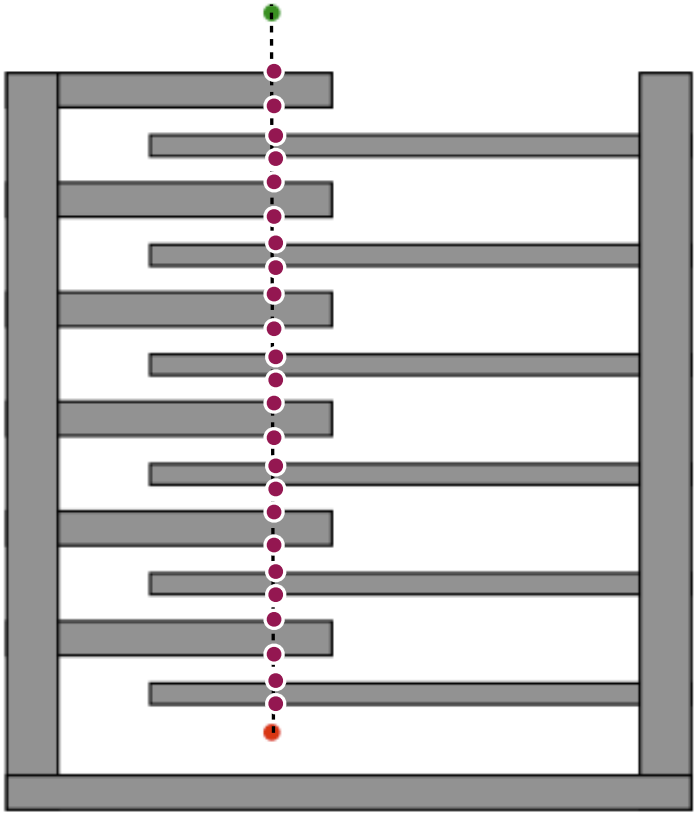
Bounds on path distance, assuming

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

Best case: 

Worst case: 

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

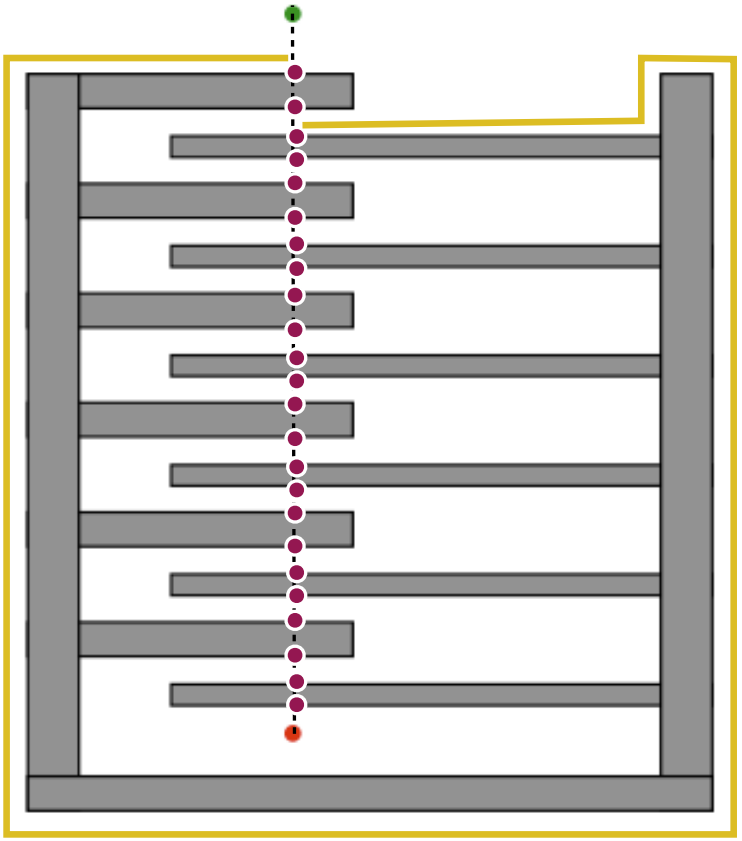
Best case: D

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

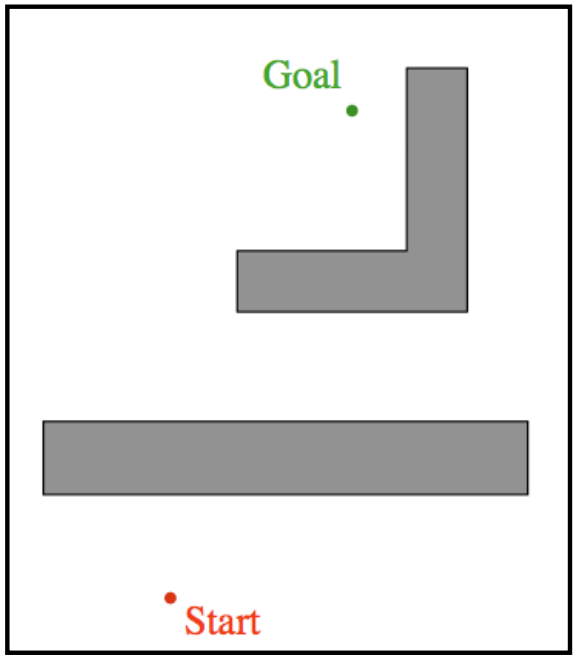
Worst case: $D + \sum_i (n_i/2)P_i$

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



>

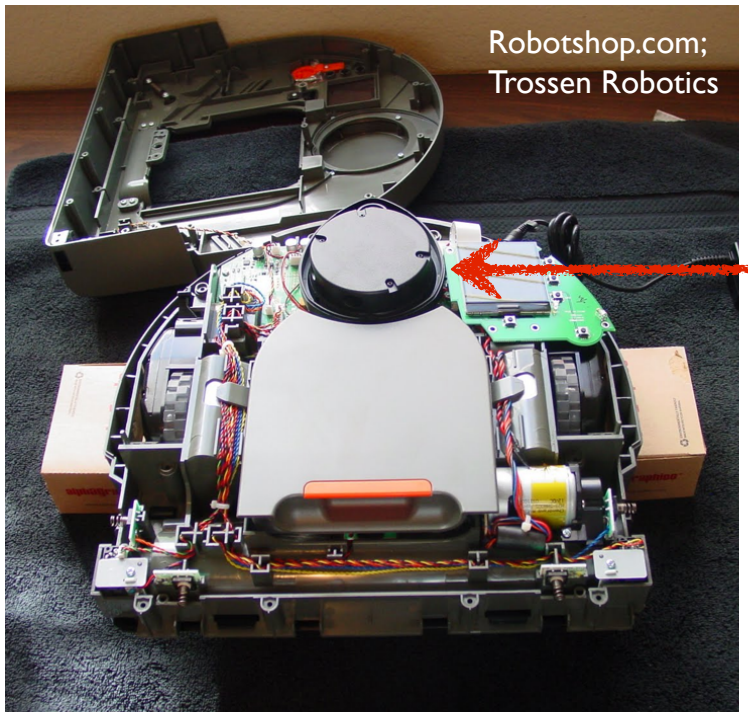
Laser Rangefinding

(briefly)

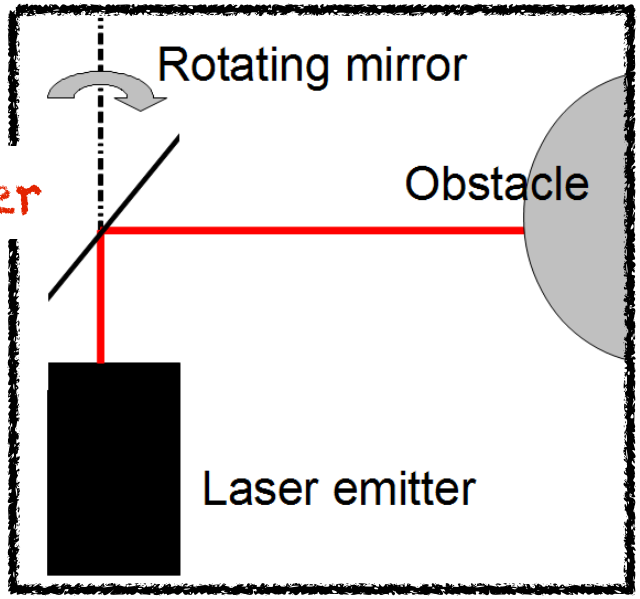
Emit laser beam in a direction

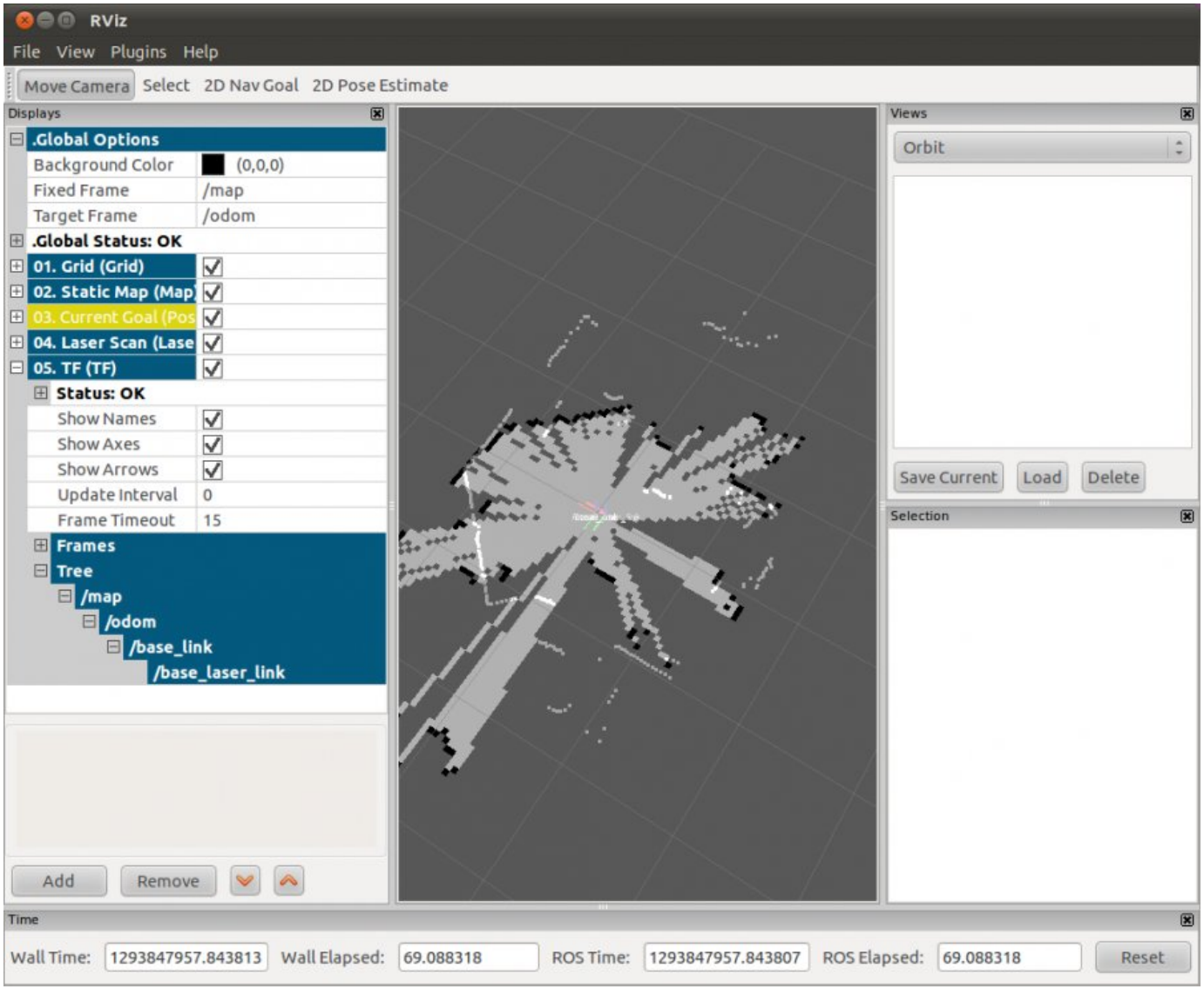
Distance to nearest object related to time from emission to sensing of beam
(assumes speed of light is known)

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



range finder



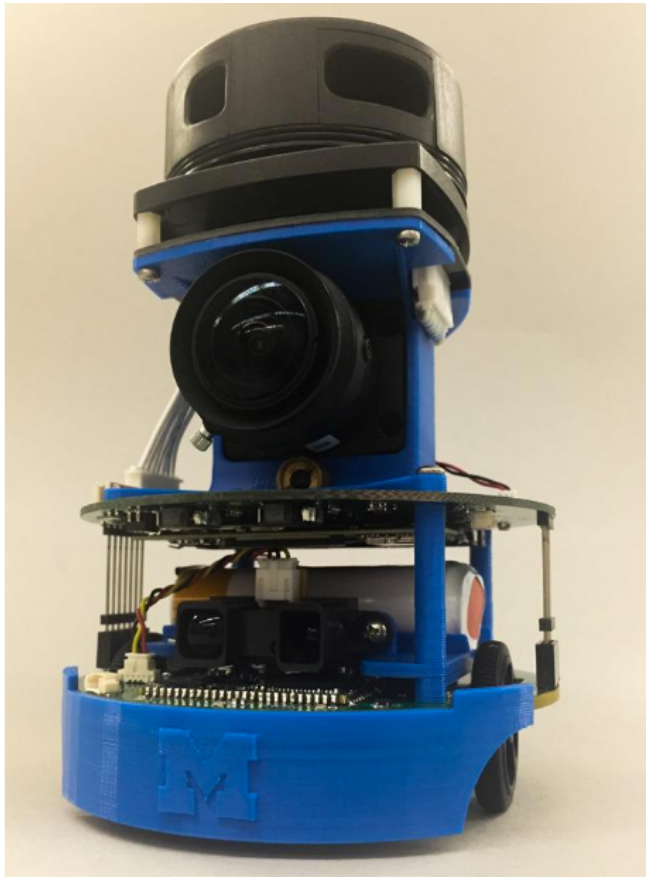


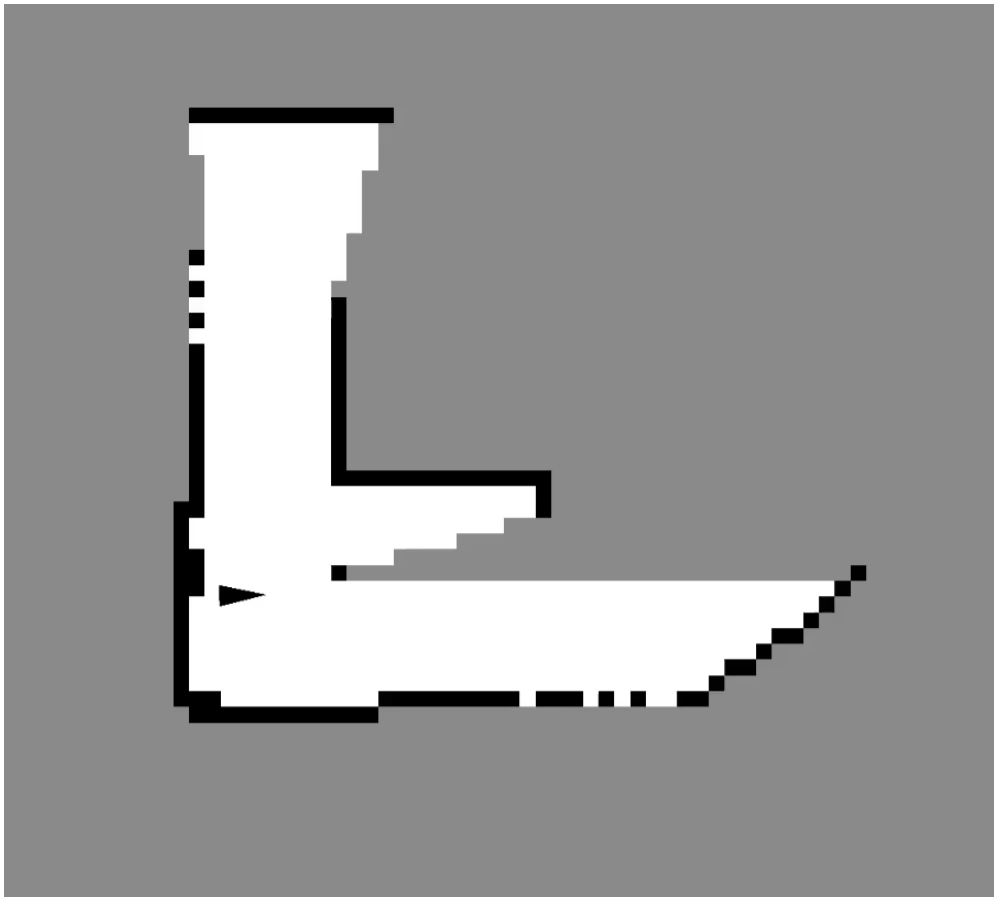
ROS LaserScan definition

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

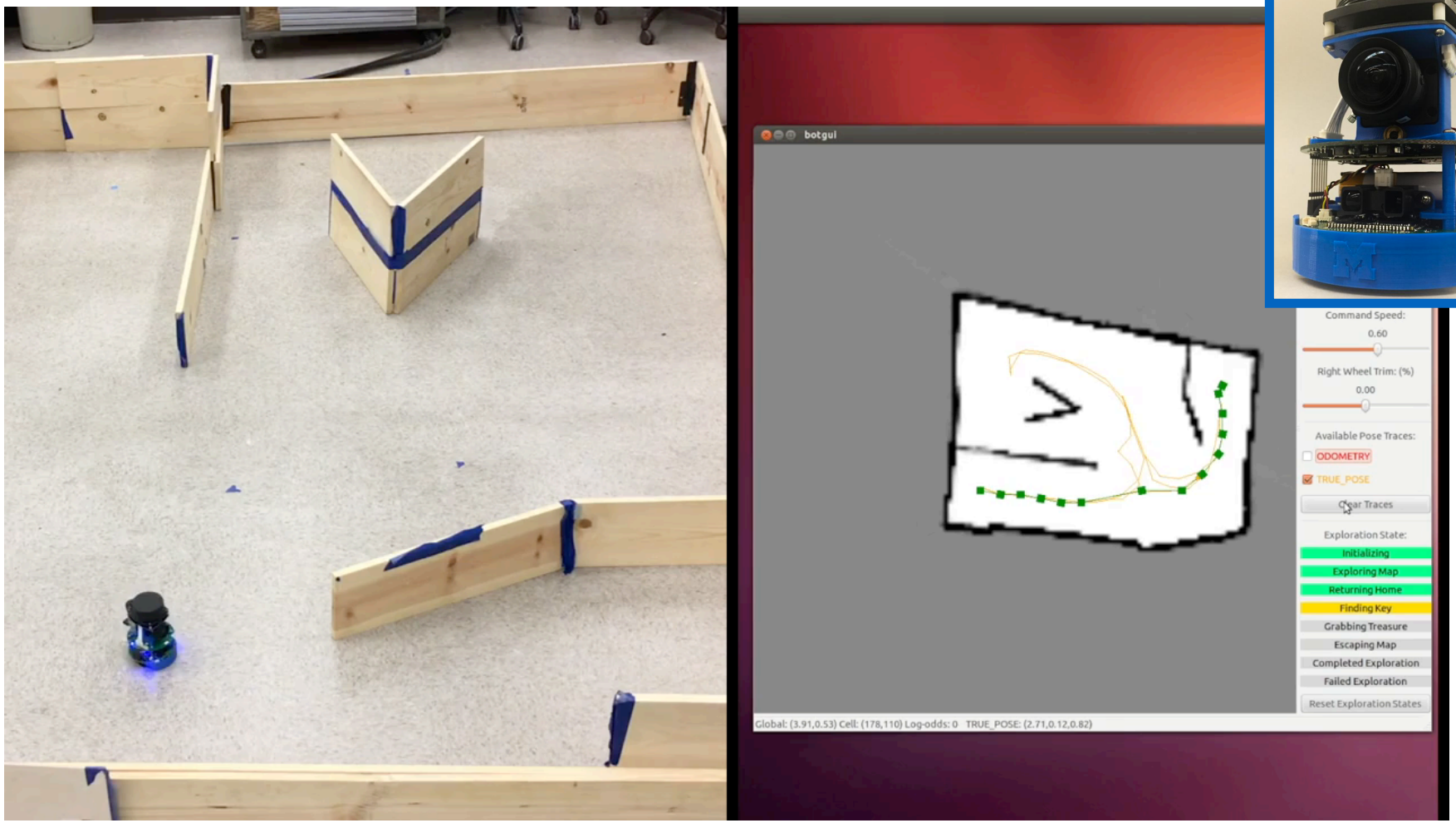
SLAM - BotLab





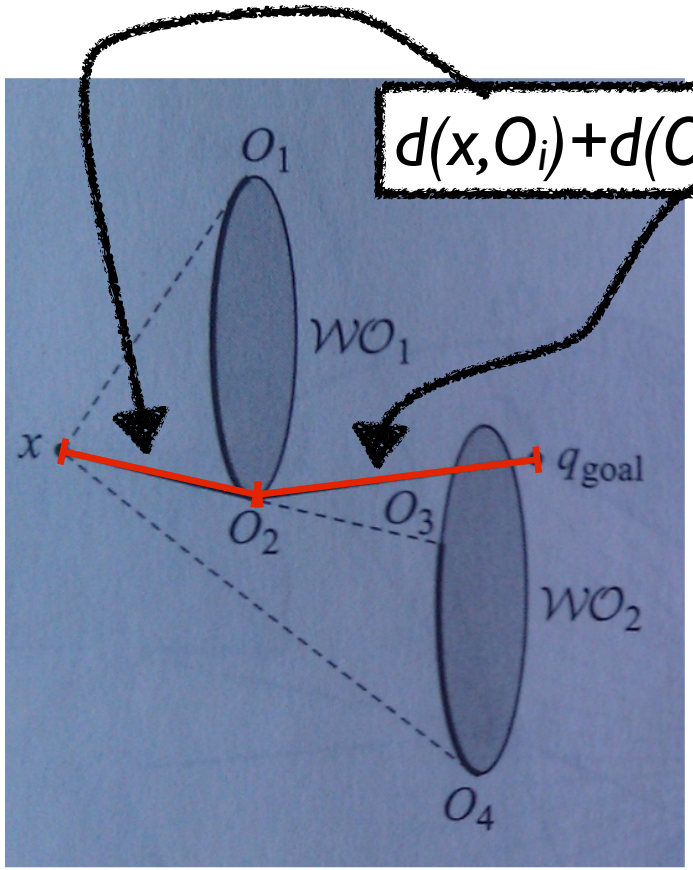
ROB 550 / EECS 467

ROB 550 BotLab / EECS 467 Escape Challenge



Jasmine Liu et al.

Tangent Bug: Heuristic Distance-to-Goal



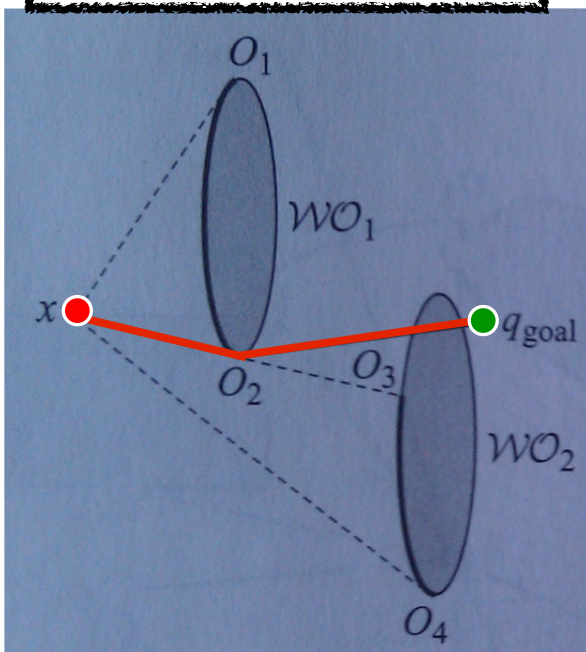
O_i are visible obstacle extents

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

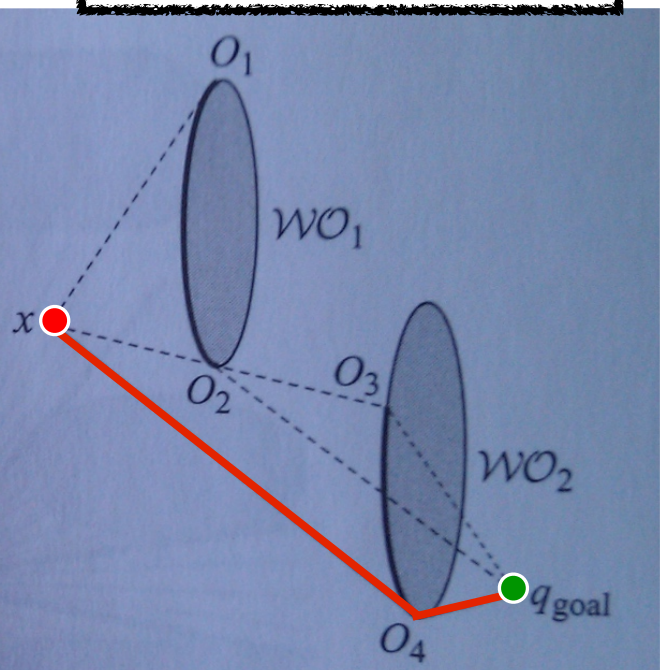
Continually move robot such that distance to goal is decreased

Note similarity to A* search heuristic

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



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

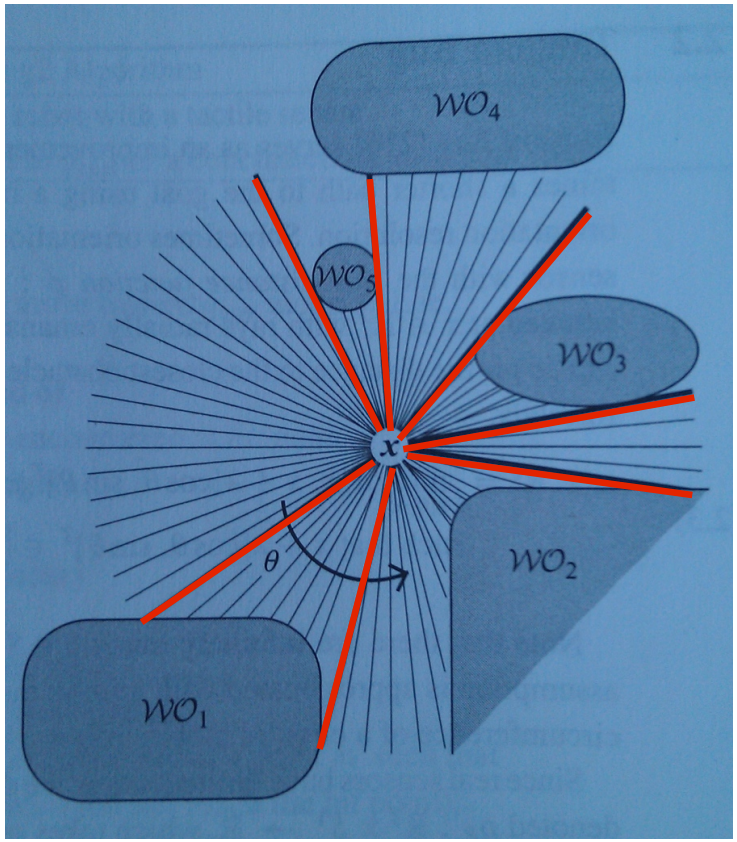


Range Segmentation

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

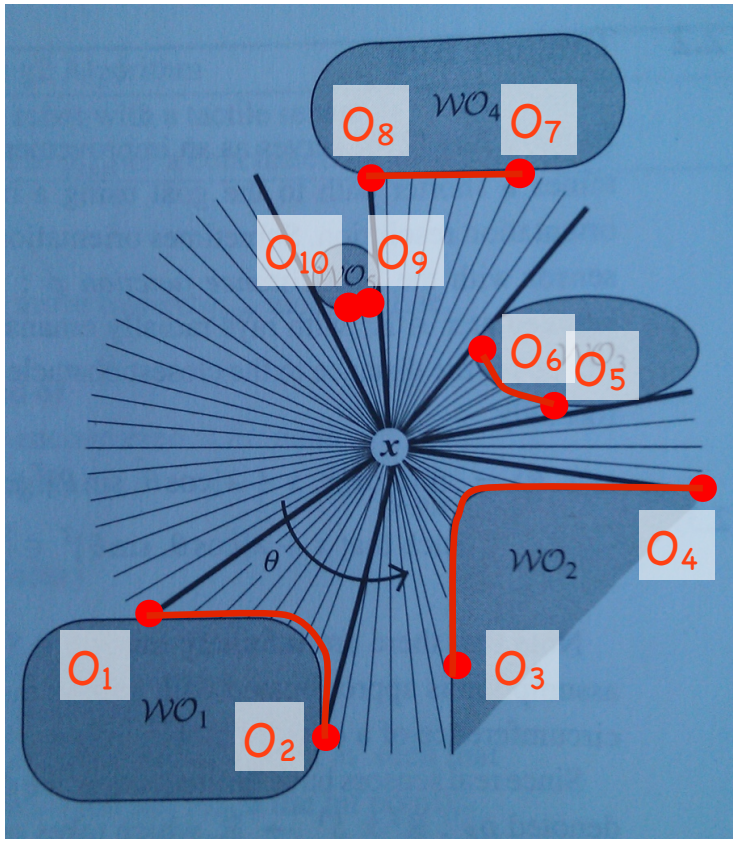


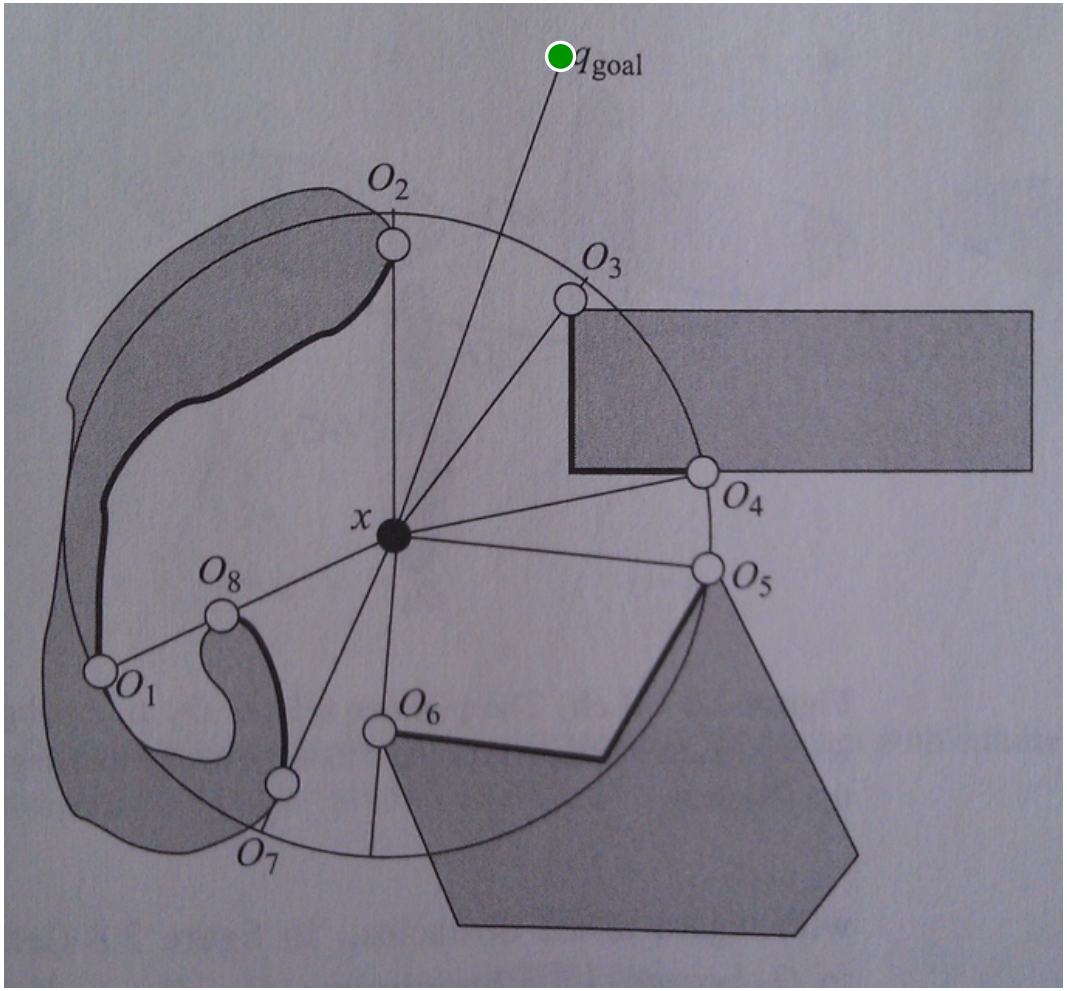
Range Segmentation

range scan $\rho(x,\Theta)$: sensed distance along ray at angle Θ within limit R

discontinuities $\{O_i\}$ in scan result from obstacles

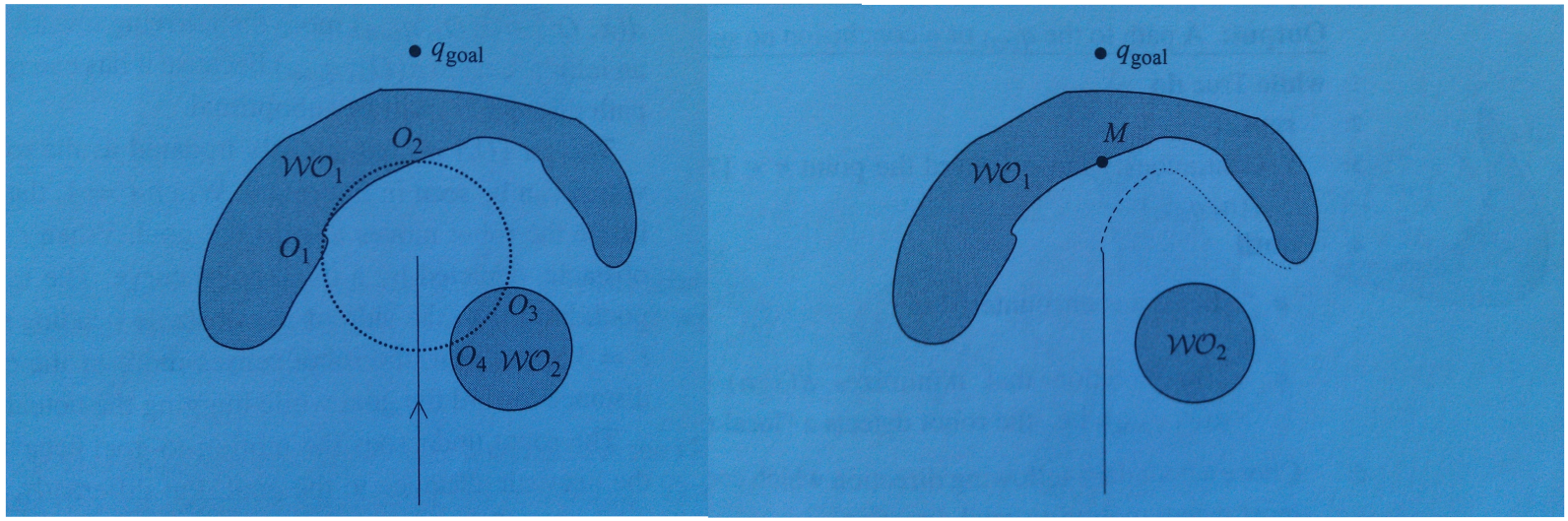
$\{O_i\}$ segments scan into intervals continuity, with obstacles and free space




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:

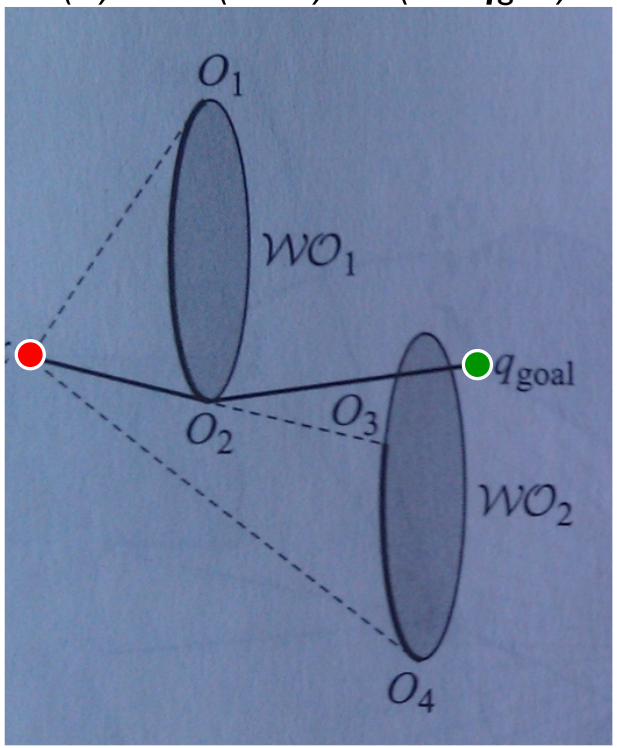


motion-to-goal

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**)

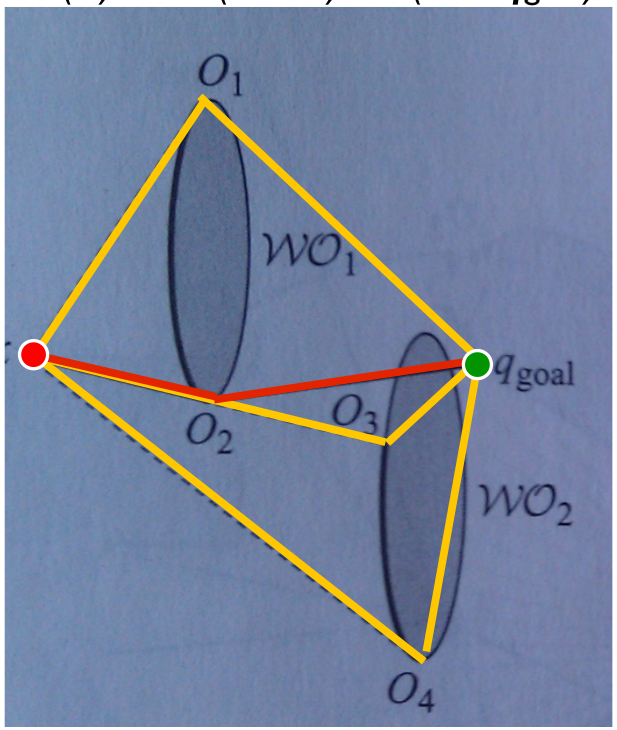
$d_{reach} < d_{follow}$,

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

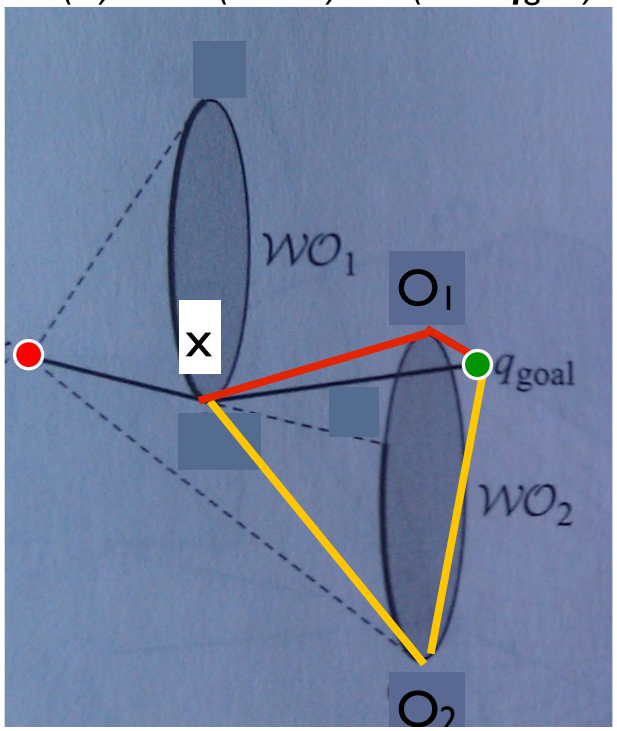
$d_{reach} < d_{follow}$,

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

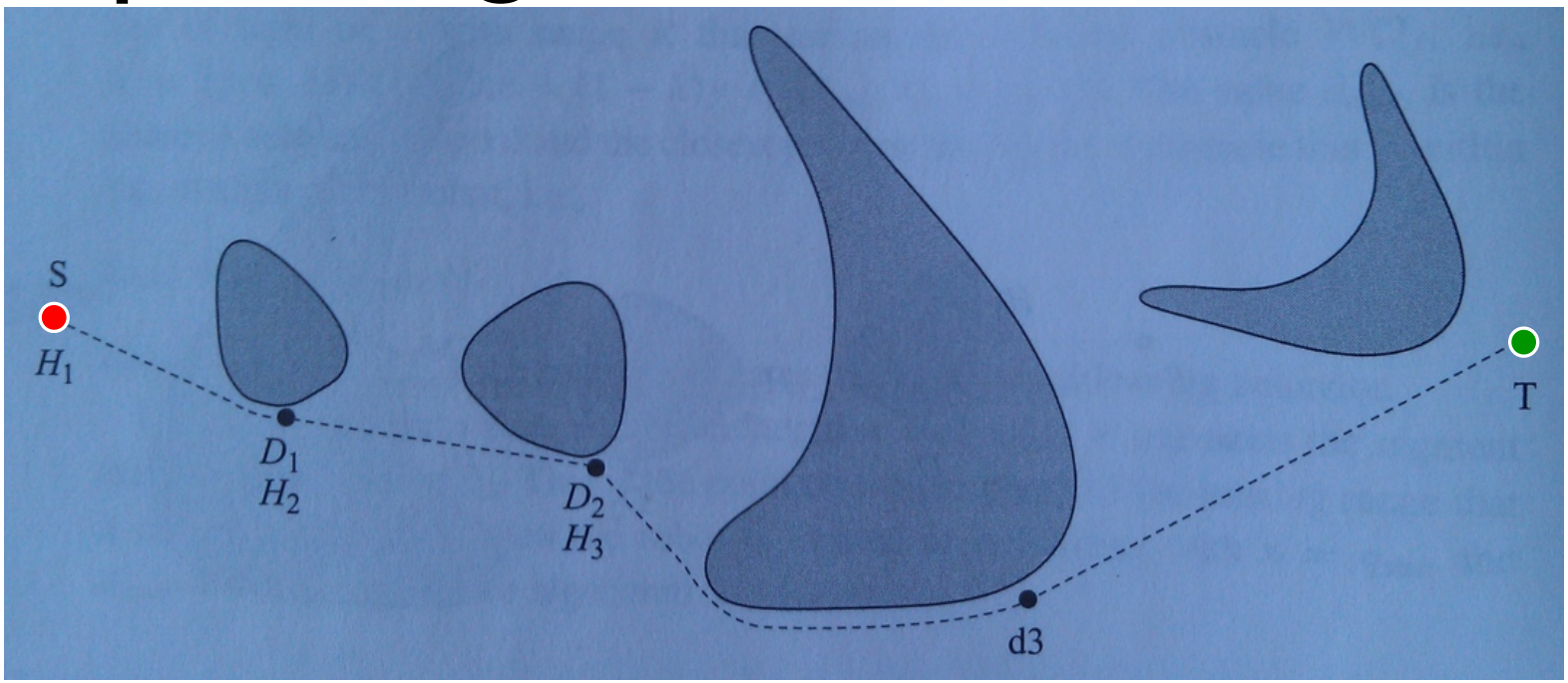
robot cycles around obstacle, (**fail**)

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

(**cleared obstacle or local minima**)

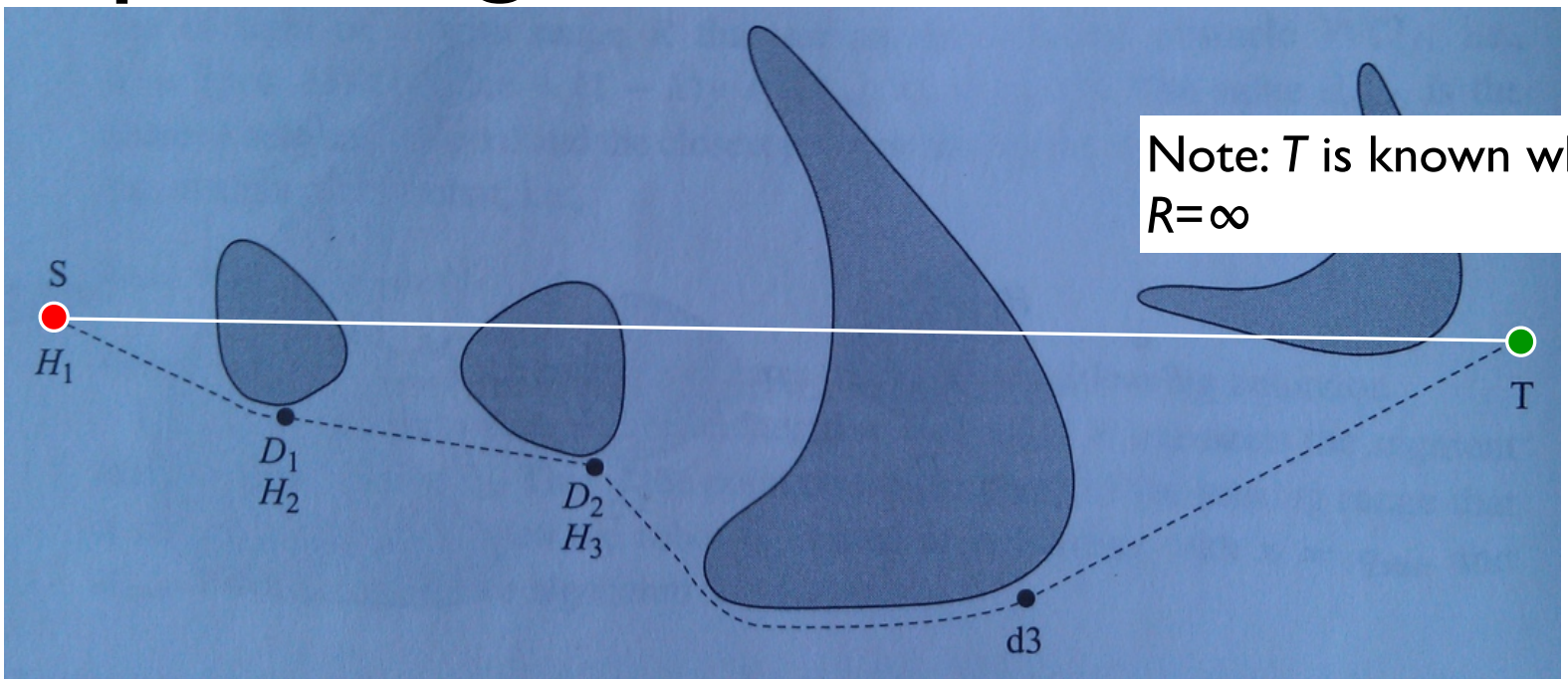
3) continue from (1)

Example: range $R=\infty$



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

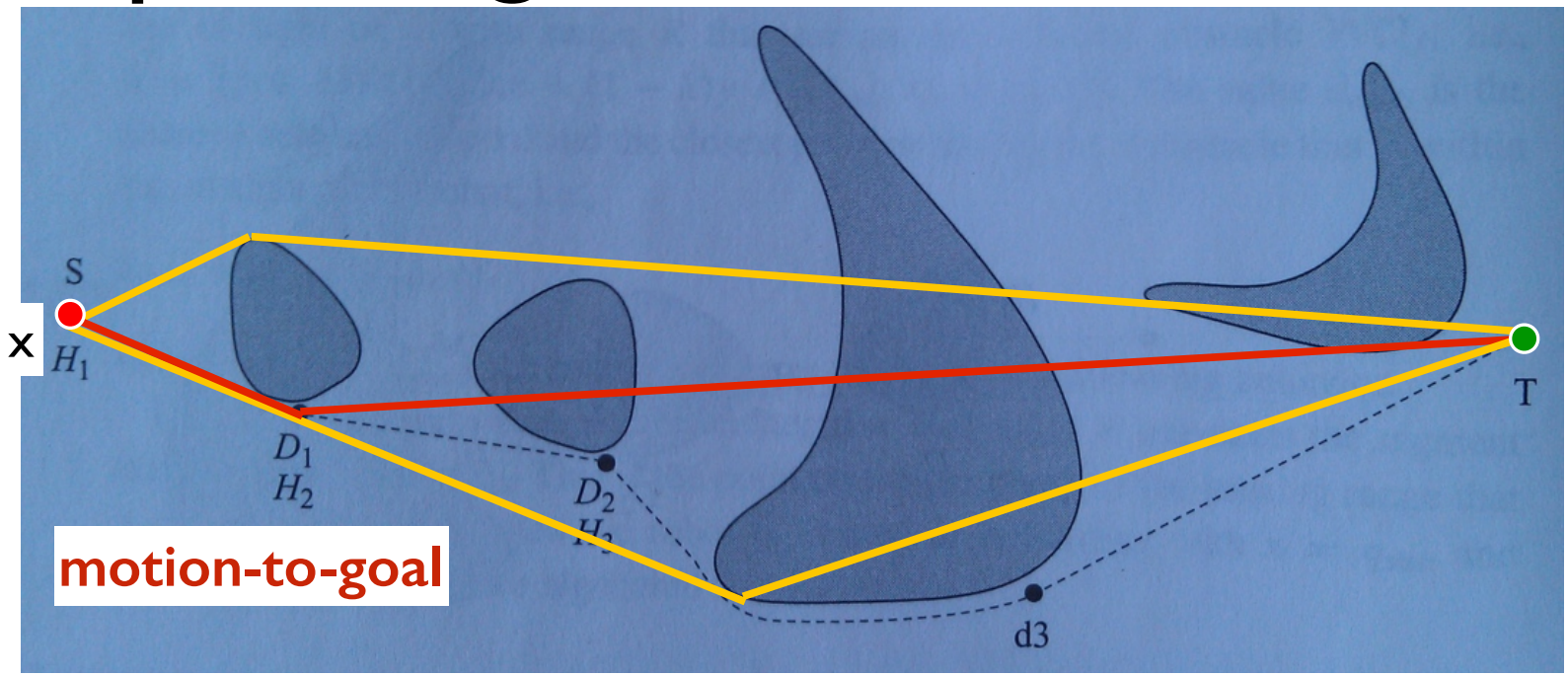
Example: range $R=\infty$



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

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

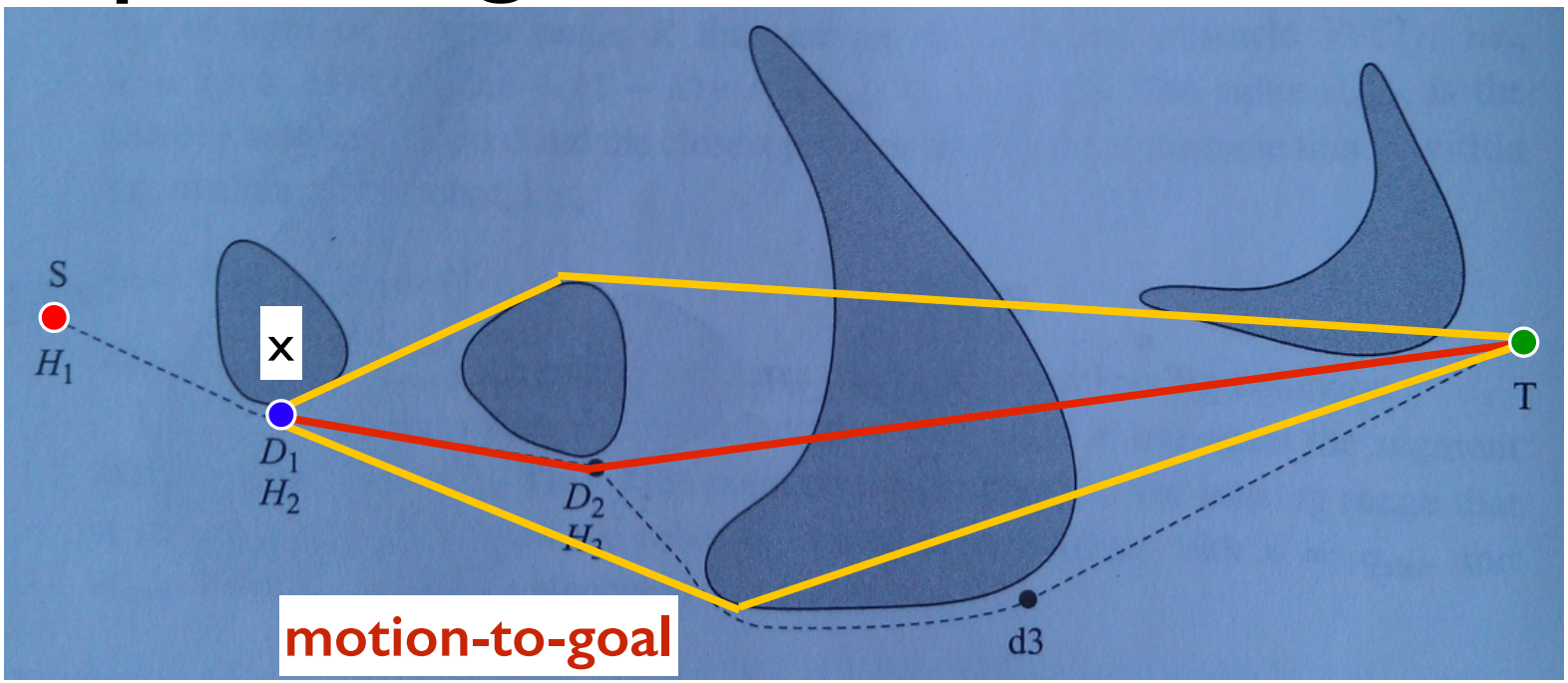
Example: range $R=\infty$



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

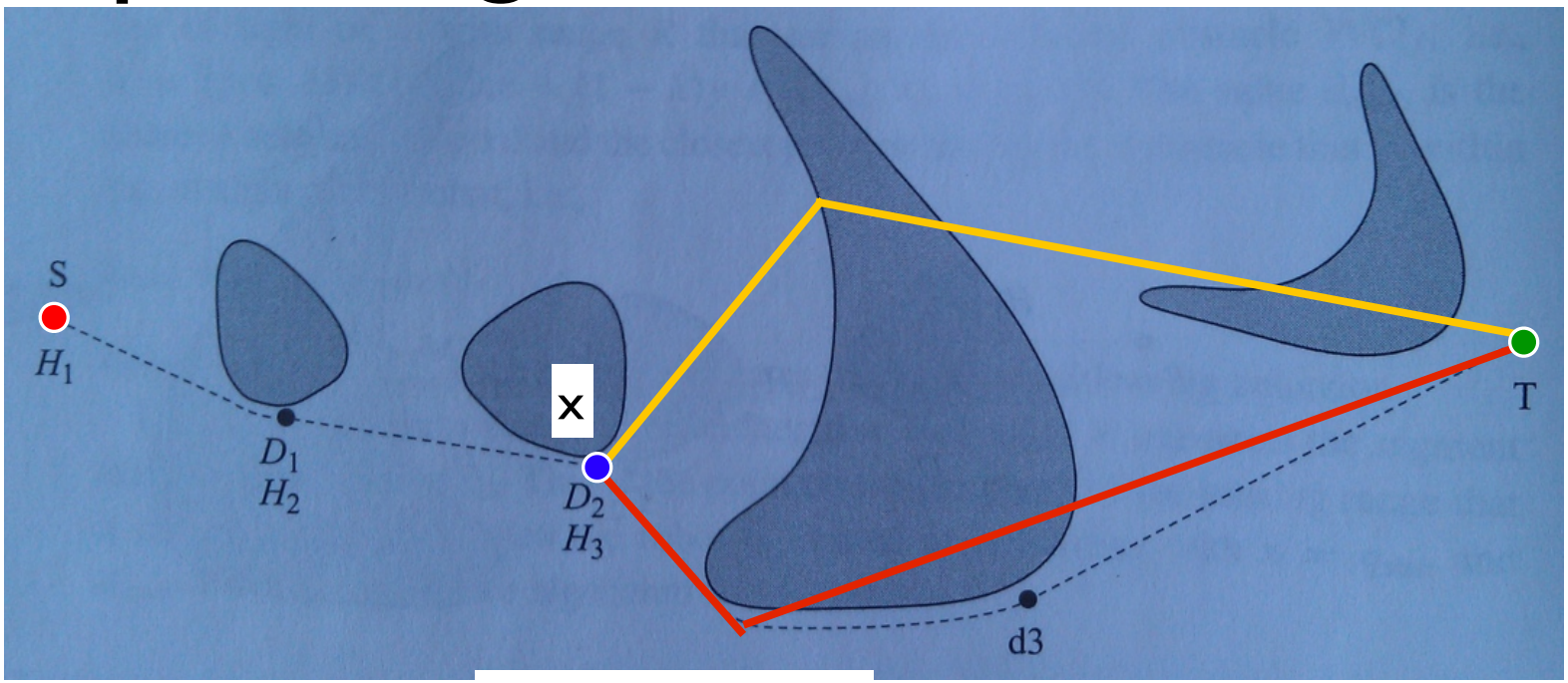
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Example: range $R=\infty$



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

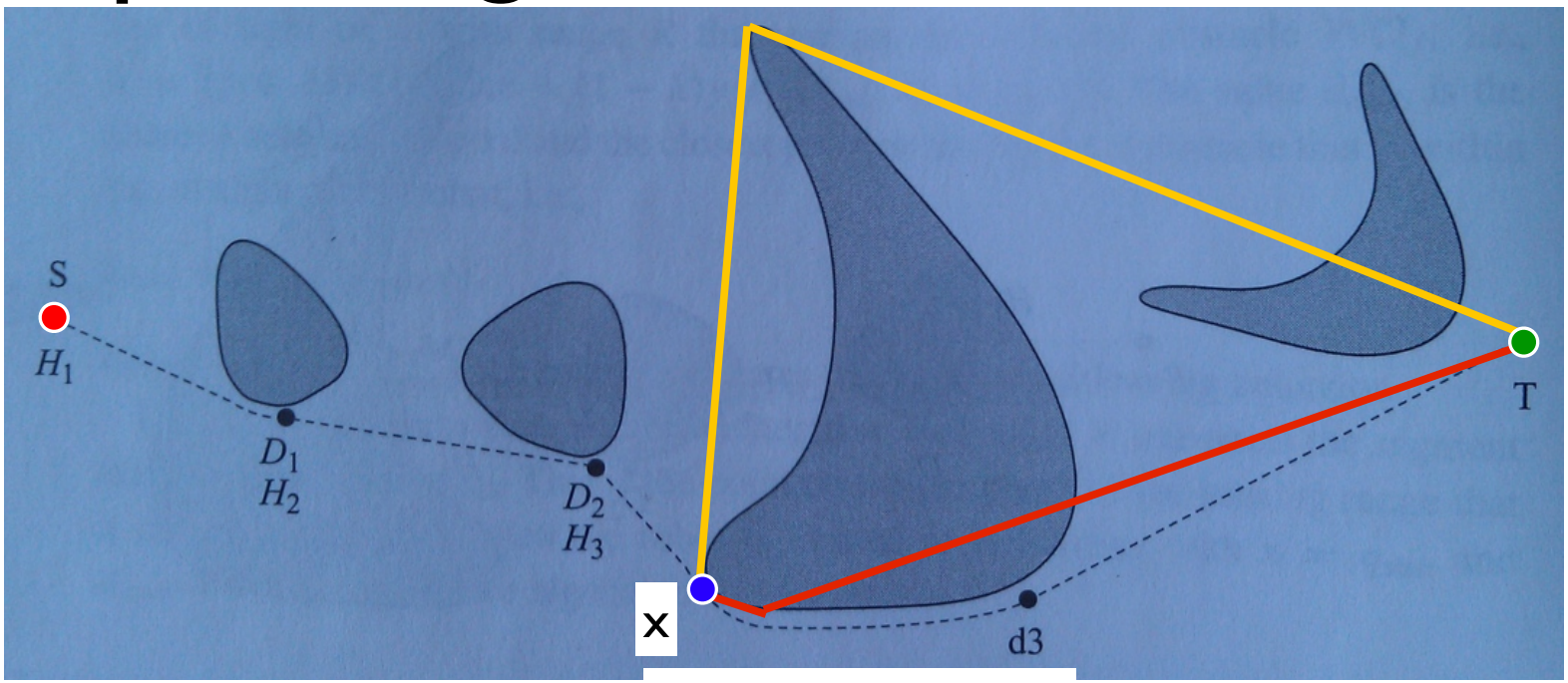
Example: range $R=\infty$



motion-to-goal

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

Example: range $R=\infty$



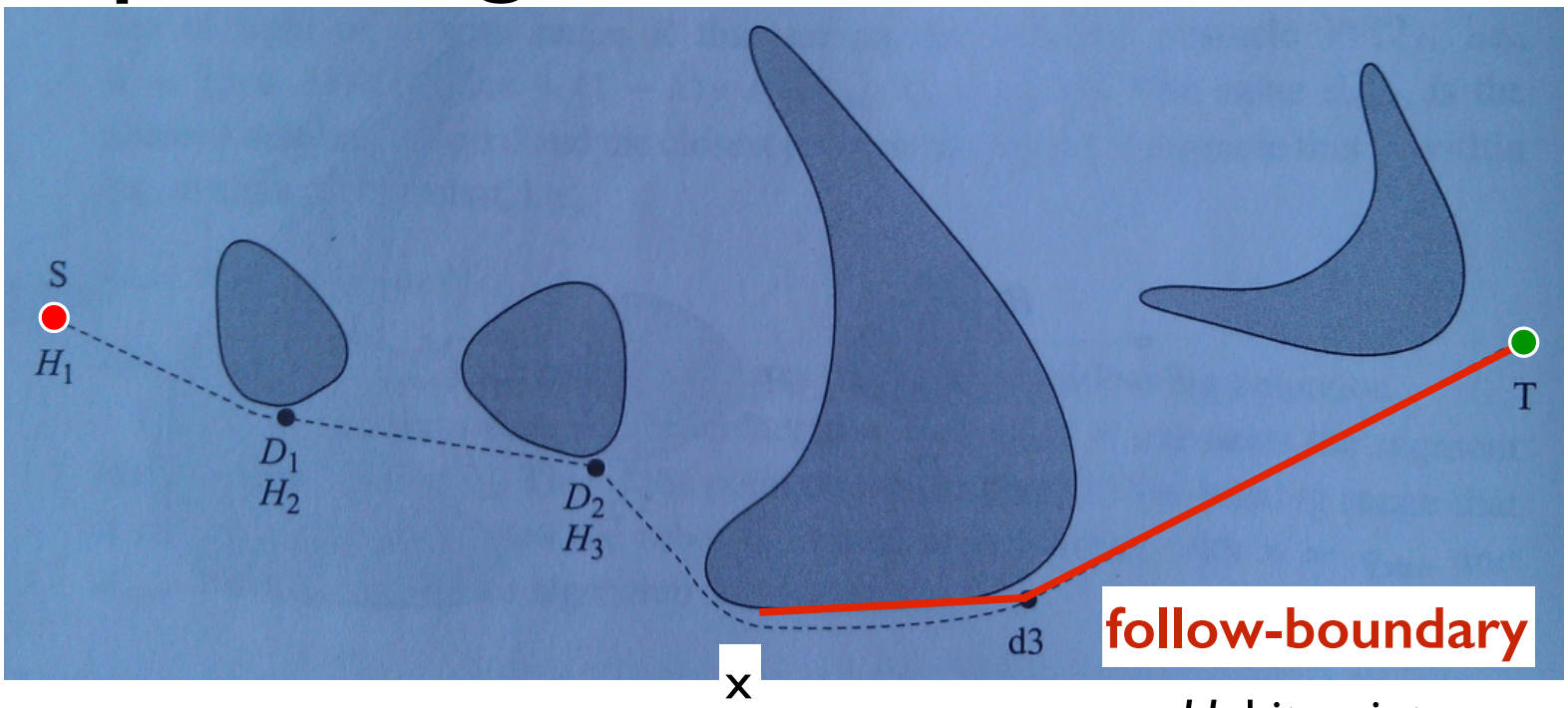
follow-boundary

start following:

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

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

Example: range $R=\infty$

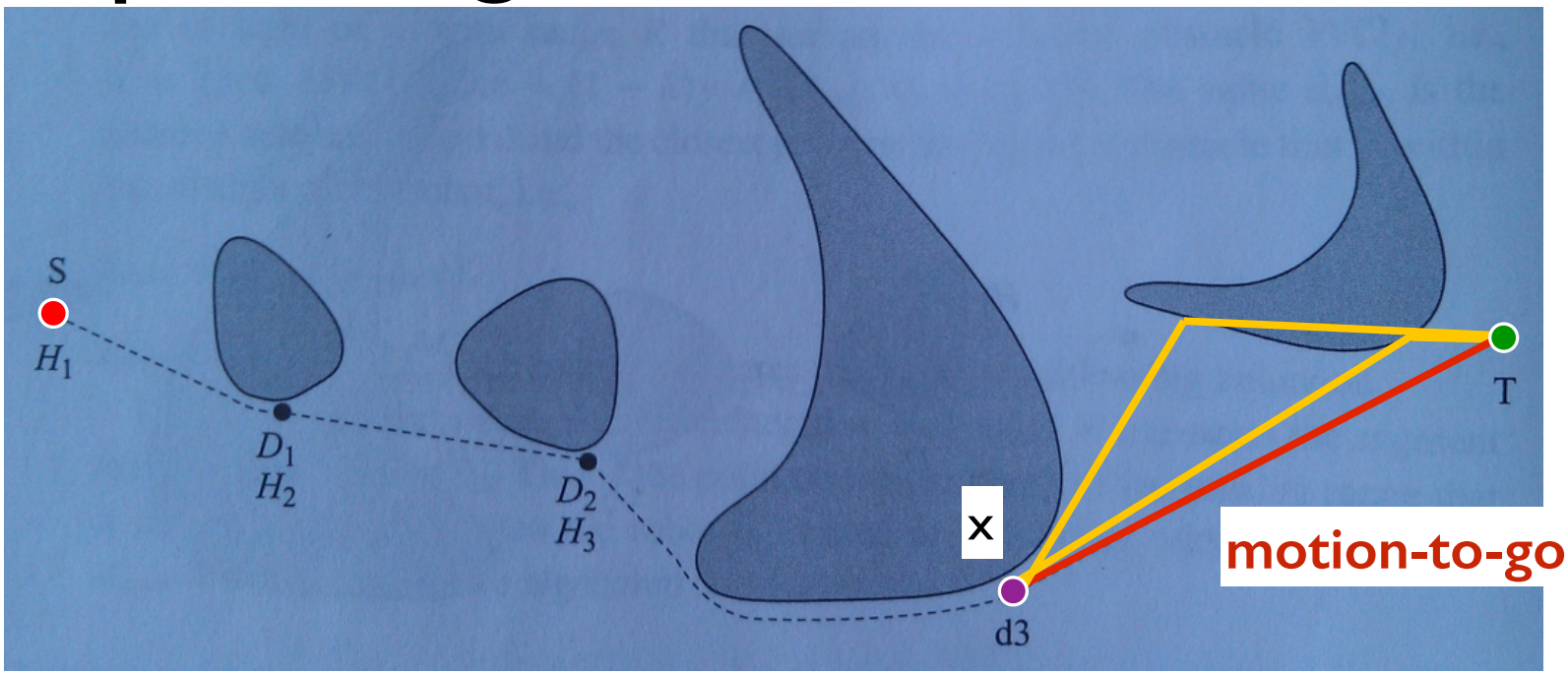


end following:

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

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

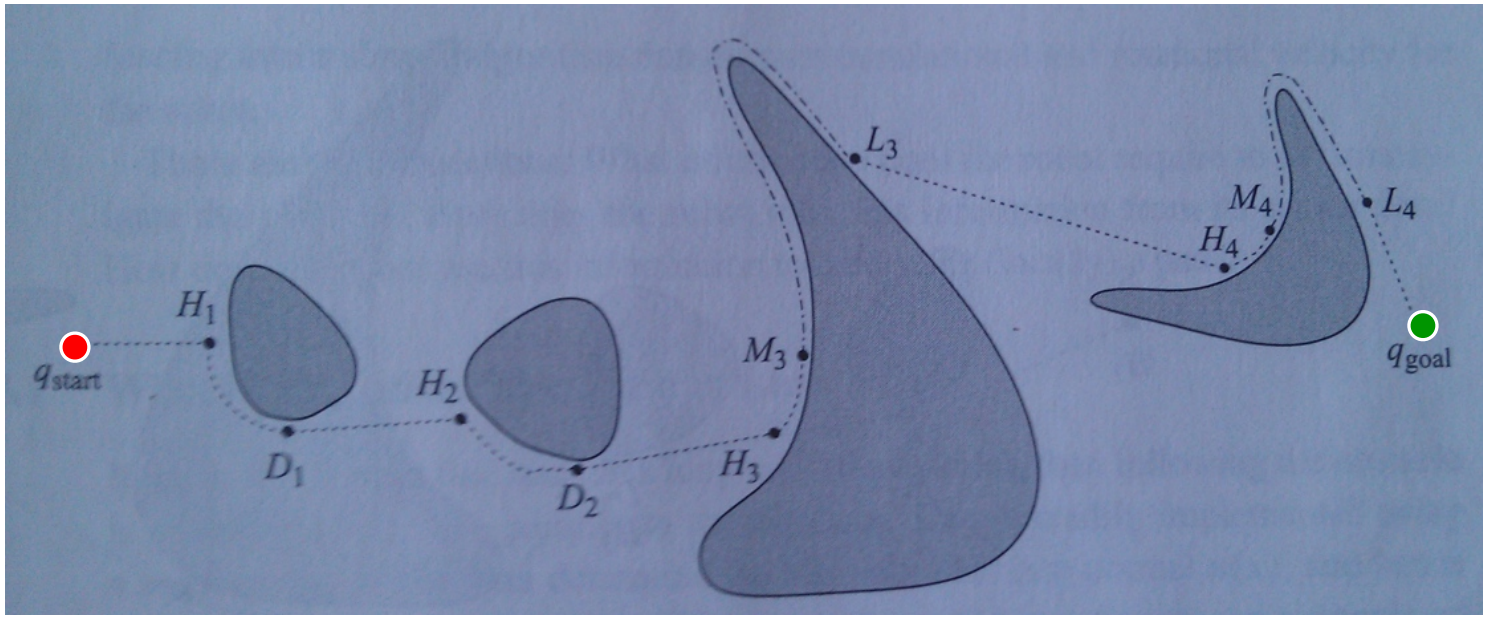
Example: range $R=\infty$



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

Example: range R=0

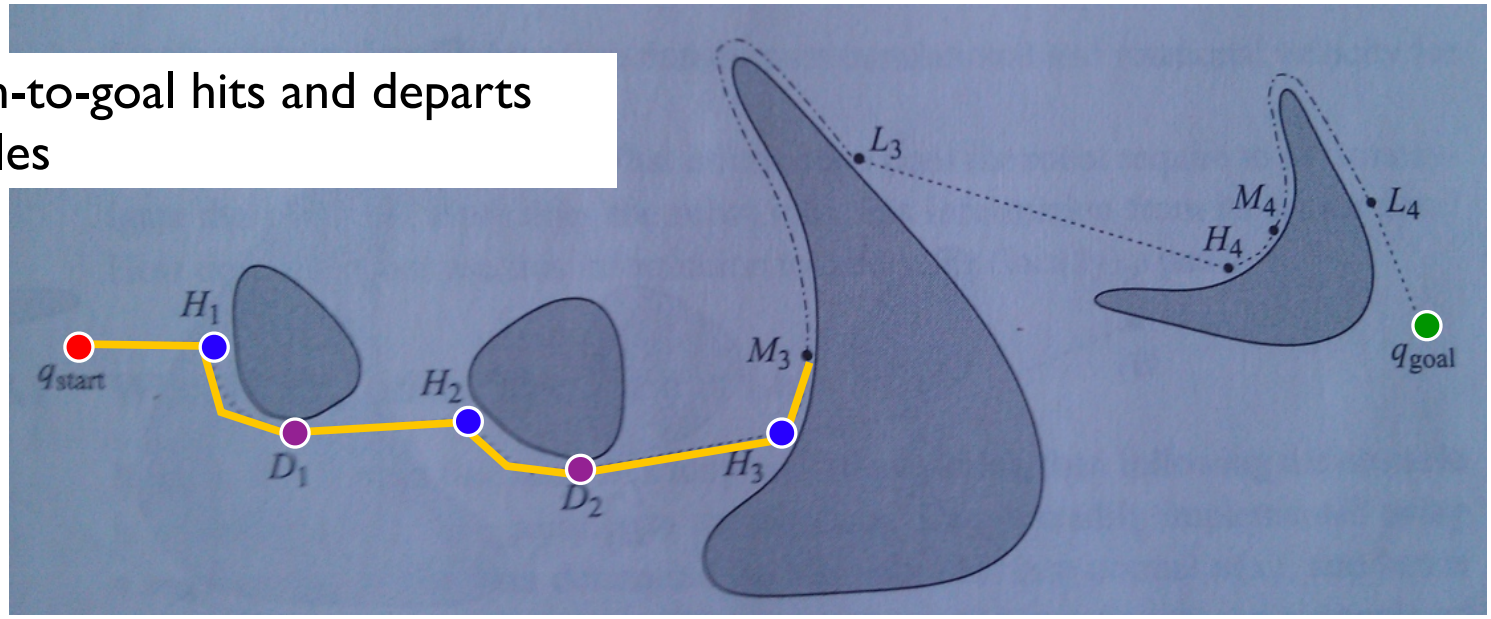
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



Example: range R=0

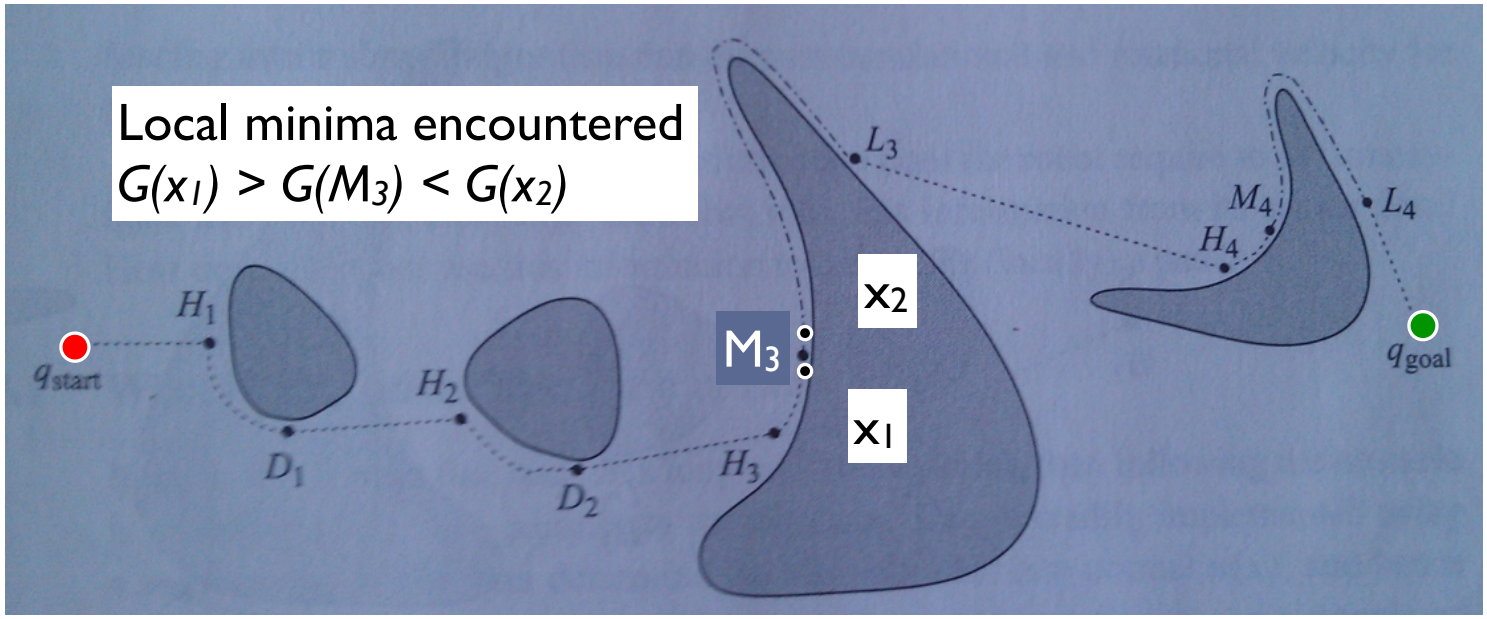
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

Motion-to-goal hits and departs obstacles



Example: range R=0

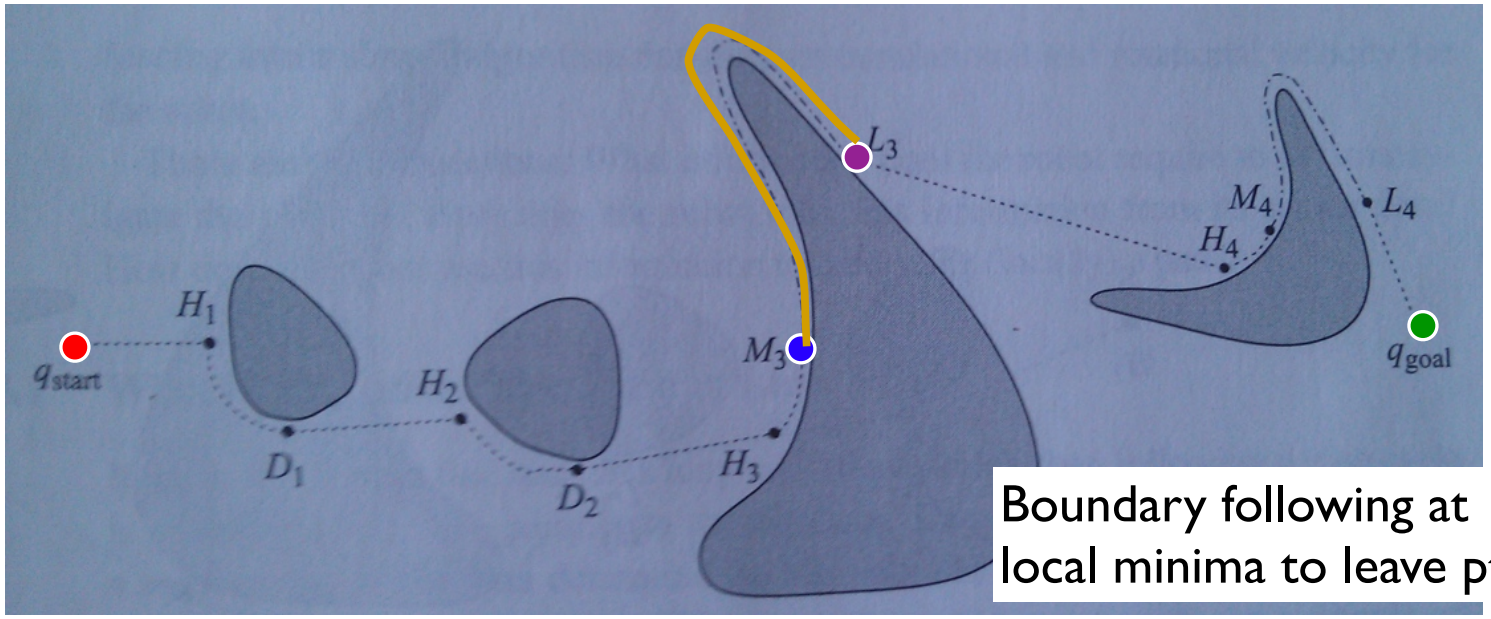
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



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

Example: range R=0

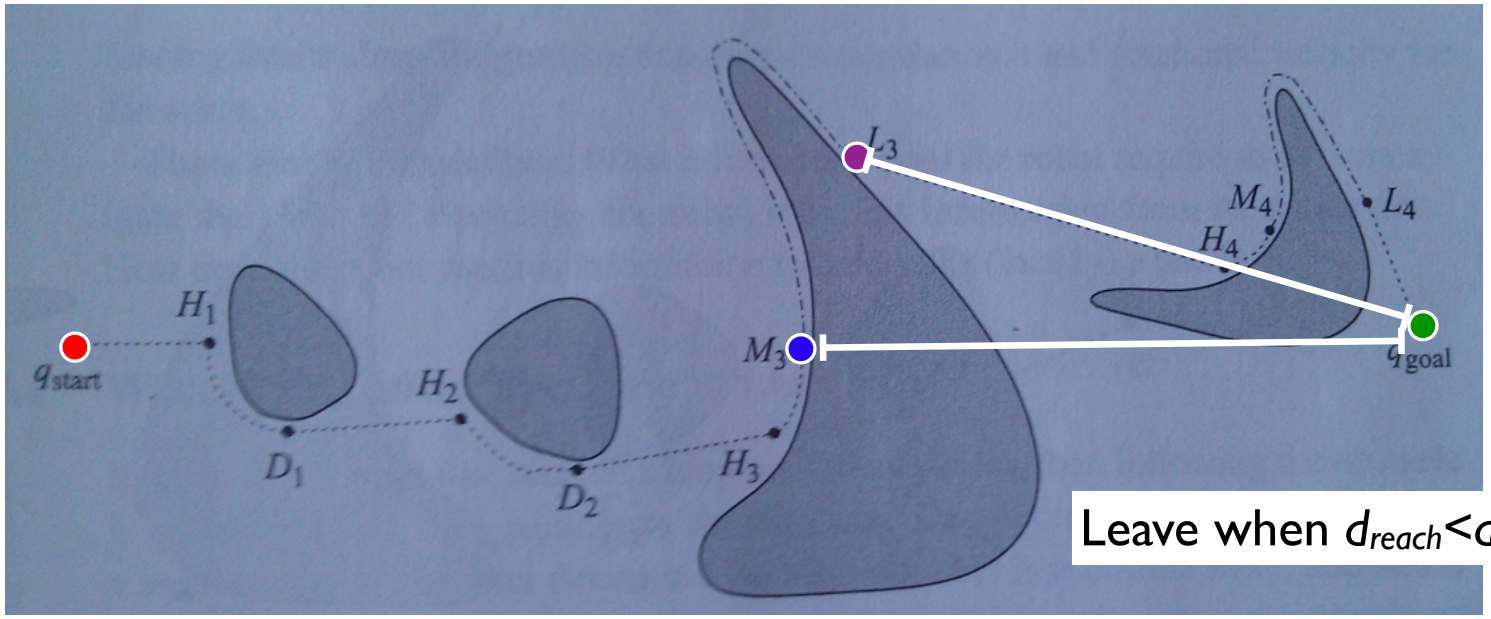
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima



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

Example: range R=0

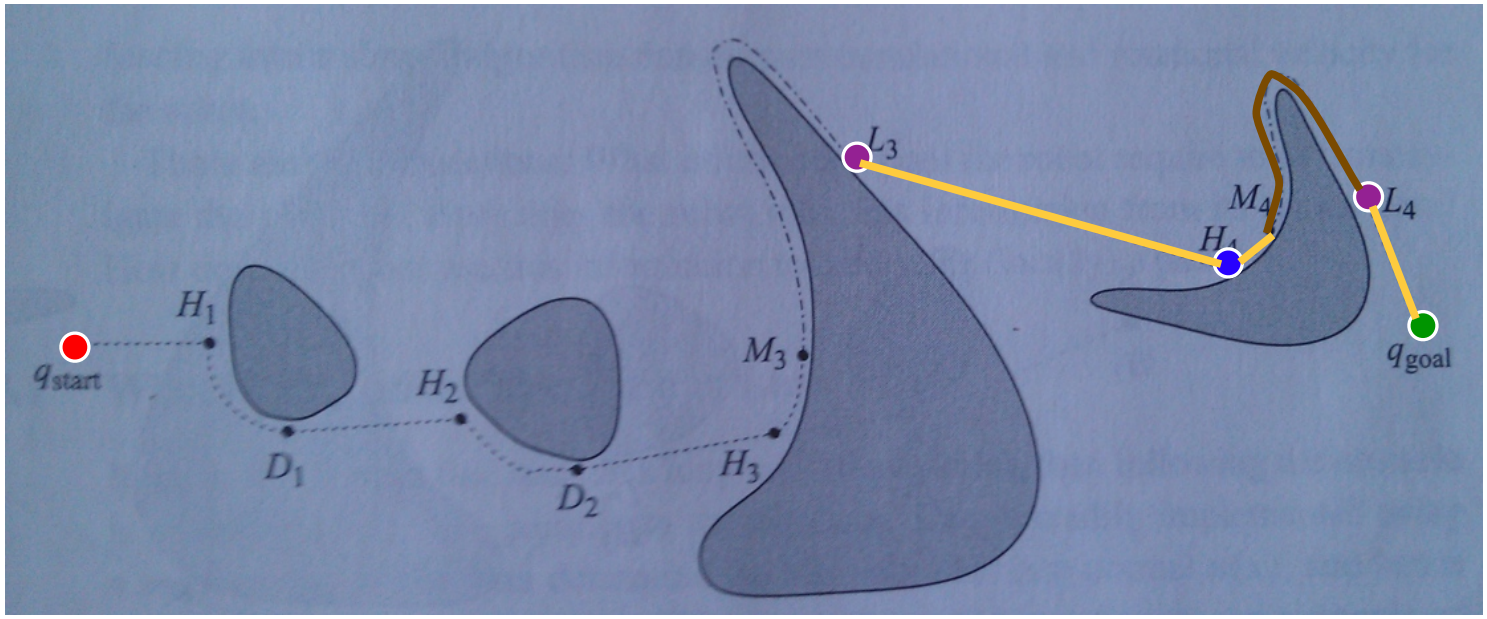
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

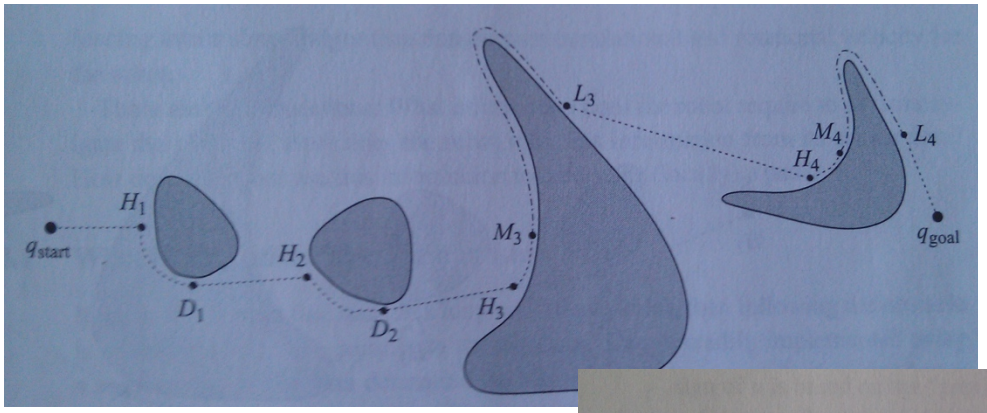


Local minima at increase of $G(x) = d(x,O_i)+d(O_i,q_{goal})$

Example: range R=0

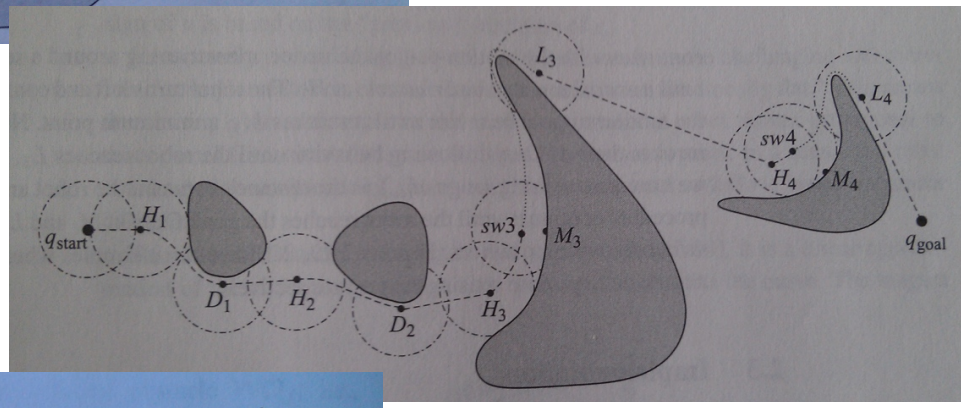
H_i: hit point
D_i: Depart point
L_i: Leave point
M_i: local minima

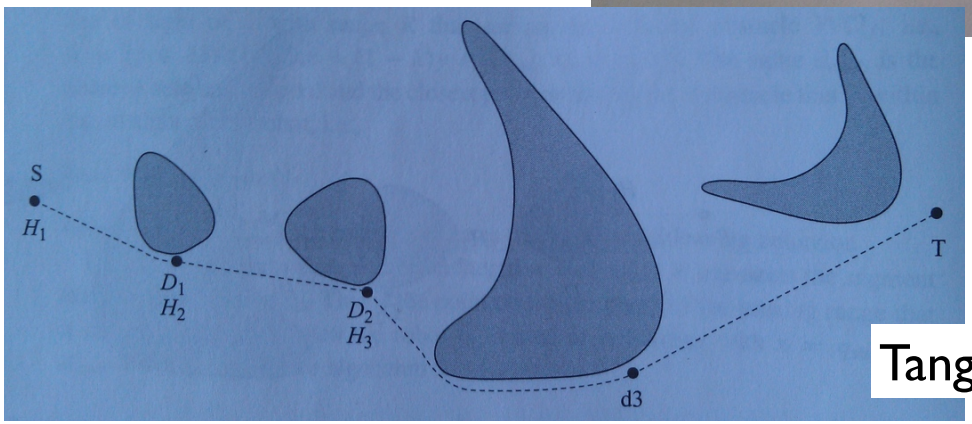




Tangent bug R=0

Tangent bug with limited radius





Tangent bug R=infinity

What does BugX assume that Random Walk does not?

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?

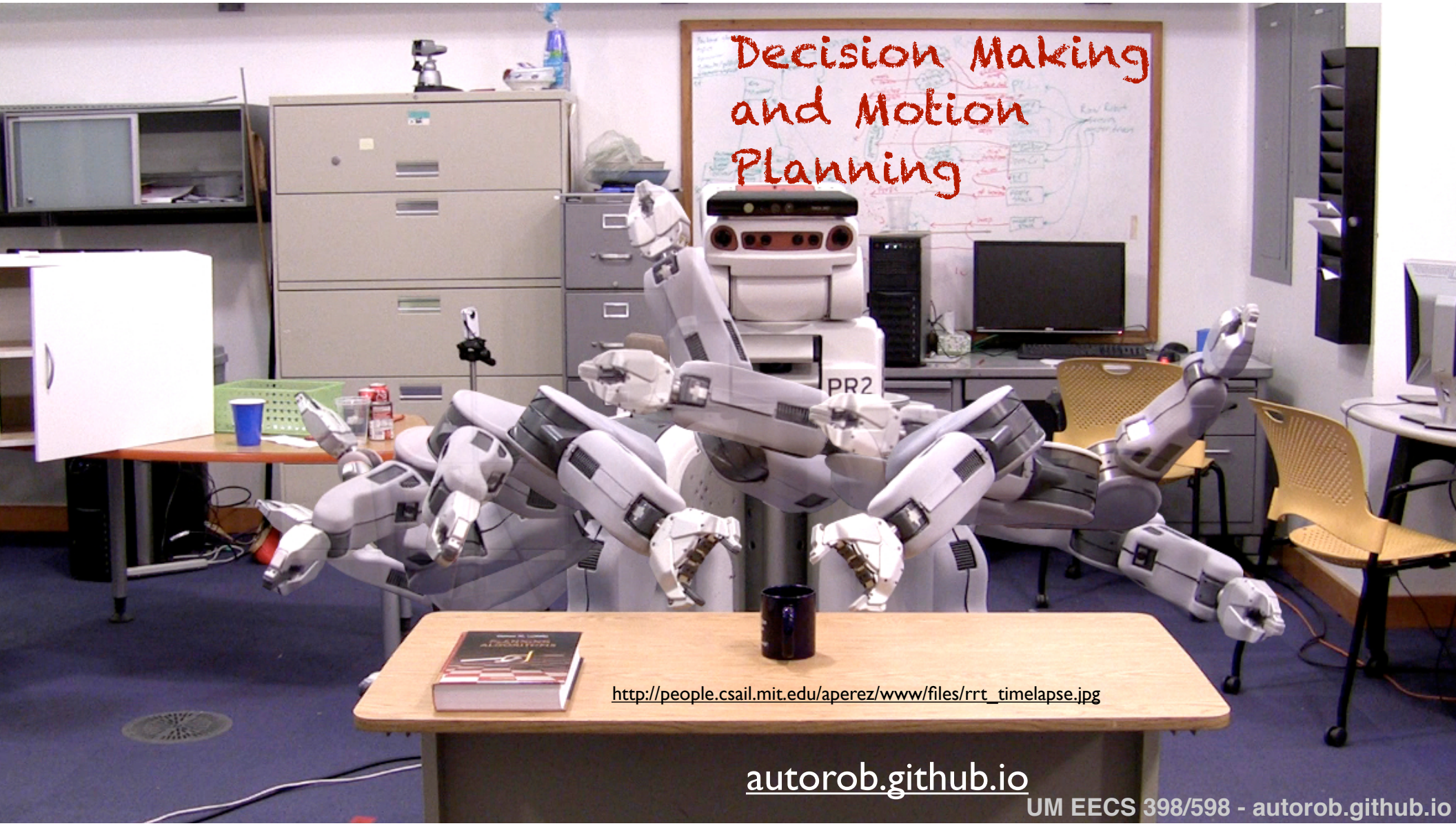
What does BugX assume that Random Walk does not?

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...



http://people.csail.mit.edu/aperez/www/files/rrt_timelapse.jpg

autorob.github.io